

The Preserve: Lehigh Library Digital Collections

The Value Function and the Efficient Frontier in Multiobjective Mixed Integer Linear Optimization

Citation

Fallah, Samira. *The Value Function and the Efficient Frontier in Multiobjective Mixed Integer Linear Optimization*. 2026, https://preserve.lehigh.edu/lehigh-scholarship/graduate-publications-theses-dissertations/theses-dissertations/value-function.

Find more at https://preserve.lehigh.edu/

This document is brought to you for free and open access by Lehigh Preserve. It has been accepted for inclusion by an authorized administrator of Lehigh Preserve. For more information, please contact preserve@lehigh.edu.

The Value Function and the Efficient Frontier in Multiobjective Mixed Integer Linear Optimization

by

Samira Fallah

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Doctor of Philosophy

in

Industrial and Systems Engineering

Lehigh University

May 2026

© 2026 Copyright

Samira Fallah

Dissertation Signature Sheet

Approved and recommended for acceptance as a dissertation in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Samira Fallah

The Value Function and the Efficient Frontier in Multiobjective Mixed Integer Linear Optimization

_____

Date

_____

Dissertation Director

_____

Accepted Date

Committee Members:

_____

Dr. Ted K. Ralphs, Committee Chair

_____

Dr. Lawrence V. Snyder

_____

Dr. Luis Nunes Vicente

_____

Dr. Stefano Coniglio

Acknowledgements

I want to express my heartfelt thanks to my advisor, Dr. Ted Ralphs, for his unwavering support, guidance, and mentorship during this dissertation journey. His valuable insights, encouragement, and patience were essential in shaping my research and helping me develop as a scholar. I feel truly fortunate to have such a dedicated mentor, and I appreciate the knowledge and wisdom I gained from him. I am also grateful to my colleague Federico for his generous help and support with the last chapter of this dissertation.

I would like to extend my sincere appreciation to my dissertation committee members, Dr. Lawrence Snyder, Dr. Natashia Boland, Dr. Luis Nunes Vicente, and Dr. Stefano Coniglio, for their guidance, feedback, and support throughout this research. Their constructive advice, expertise, and thoughtful insights were instrumental in improving the quality and rigor of this work. I also gratefully acknowledge Dr. Storer, who served as a committee member during my proposal stage; although he later retired and was unable to continue serving, his time and early feedback were greatly appreciated. I would especially like to thank Dr. Natashia Boland for her exceptional support, as well as the valuable vision and ideas she contributed, which significantly shaped this dissertation.

Lastly, I want to acknowledge the unwavering support and love from my family, especially my parents, Simin and Javad, and my husband, Saeed. Their continuous encouragement, understanding, and patience have been the foundation of my academic journey. Their sacrifices and unwavering belief in me were my main source of motivation. I am grateful for the countless hours of support, both emotional and practical, that they provided, without which this accomplishment would not have been possible.

Table of Contents

List of Tables x

List of Figures xiii

Abstract 1

1 Introduction 3

1.1 Related Work 8

1.1.1 Multiobjective Optimization 8

1.1.2 Value Function 10

1.2 Application of RVF 12

1.2.1 Dantzig-Wolfe Decomposition 12

1.2.2 Lagrangian Relaxation 15

1.2.3 Cutting-Plane Method 16

1.2.4 Two-stage Stochastic Linear Optimization 17

1.2.5 Mixed Integer Bilevel Linear Optimization Problems 18

1.3 Contributions 18

1.4 Outline of Thesis 19

2 The RVF and the EF of an MILP **21**

2.1 The Restricted Value Function . . . 21

2.1.1 Structure of the Restricted LP Value Function . . . 22

2.1.2 Structure of the Restricted Value Function . . . 26

2.2 The RVF and the EF . . . 35

2.3 Conclusions . . . 40

3 The RVF Algorithm **41**

3.1 Basic Framework . . . 41

3.1.1 Outline of the Algorithm . . . 42

3.1.2 Initialization and Termination . . . 44

3.1.3 Generating the Bounding Functions . . . 45

3.1.4 Converting Weakly Efficient Solutions . . . 51

3.1.5 Correctness of the RVF Algorithm . . . 52

3.1.6 Properties of the RVF Algorithm in the PILP Case . . . 53

3.2 Comparison to Other Algorithms . . . 55

3.2.1 Pure Integer (PILP) Case . . . 55

3.2.2 General Case . . . 57

3.3 Implementation of the RVF Algorithm . . . 59

3.3.1 Bounds on Variables . . . 60

3.3.2 Avoiding Cycling . . . 62

3.3.3 Algorithmic Variations . . . 64

3.4 Data Sets . . . 81

3.5 Computational Results . . . 87

3.5.1 Pure Integer Instances . . . 89
3.5.2 General Mixed Integer Instances . . . 110
3.6 Software . . . 118
3.7 Conclusions . . . 119

4 A Branch-and-Bound Algorithm for Constructing the RVF and EF 122

4.1 Introduction . . . 123
4.2 Bounding Functions from BB . . . 124
4.3 Tree-based Warm-Starting . . . 132
4.4 Finite Algorithm for Construction . . . 134
4.4.1 General Framework . . . 134
4.4.2 Methods for Generating Sample Points . . . 141
4.4.3 Algorithm Description . . . 142
4.5 Implementation . . . 149
4.6 Computational Experiment . . . 154
4.6.1 Data Sets . . . 155
4.6.2 A Comparative Study . . . 156
4.7 Software . . . 182
4.7.1 Dependencies required to run the program . . . 182
4.7.2 Data Files . . . 185
4.7.3 Running and Results . . . 186
4.8 Conclusions . . . 186

5 Future Research Directions 188

Bibliography 190

Appendix A An Instance of RLPVF 200

Appendix B The LP Frontiers of Example 1 202

Appendix C Proofs Regarding the Structure of RVF 205

Appendix D Theorem 2.2.1 with $C^{1:\ell}x = \zeta$ 212

Appendix E Proofs of Correctness for the RVF Algorithm 214

Appendix F Proof of the Nonempty Stability Region of the Integer Part in Each Iteration 217

Appendix G Directional Derivatives of the Minimum of Polyhedral Functions 220

Appendix H Polynomial Time Growth with Iterations 225

Appendix I Analysis of the Number of ILPs Solved in DCM 227

Appendix J Comparison of RVF Algorithm and BLM Variants with MILP Example 230

Appendix K Tables from the Computational Experiment in Chapter 3 233

Appendix L Base Performance Profile for the Computational Experiments in Chapter 3 252

Appendix M Tables from the Computational Experiment in Chapter 4 258

Appendix N Base Performance Profile for the Computational Experiments in Chapter 4 272

Vita **274**

List of Tables

3.1 Summary of RVF Implementations and Their Parameter Settings 65

3.2 Benchmarks for BOSPP 84

3.3 Benchmarks for Knapsack Problem 85

3.4 Performance comparison of implementations (all statistics reported per NDP) 104

4.1 Average execution time (in seconds) for different RVF call frequencies and initial sample sizes 173

4.2 Parameters and their corresponding values that need to be fixed within the warm solve functionality in SYMPHONY 185

K.1 Running time and iterations of the nonlinear and linear versions of the RVF Algorithm for SPP 234

K.2 Running time and iterations of the nonlinear and linear versions of the RVF Algorithm for KP 235

K.3 Running time in seconds for all the SPP using the RVF Algorithm with Implementation 2236

K.4 Running time in seconds for all the KP using the RVF Algorithm with Implementation 2237

K.5 Running time in seconds for all the randomly generated IPs using the RVF Algorithm with Implementation 2 238

K.6 Running times (in seconds) for all KP instances solved with the RVF Algorithm under Implementation 3, using different numbers of feasible solutions 239

K.7 Running times (in seconds) for all SPP instances solved with the RVF Algorithm under Implementation 3, using different numbers of feasible solutions 240

K.8 Running times (in seconds) for all KP instances solved with the RVF Algorithm under Implementation 4, using different cut ratios - Part I 241

K.9 Running times (in seconds) for all KP instances solved with the RVF Algorithm under Implementation 4, using different cut ratios - Part II 242

K.10 Running times (in seconds) for all SPP instances solved with the RVF Algorithm under Implementation 4, using different cut ratios - Part I 243

K.11 Running times (in seconds) for all SPP instances solved with the RVF Algorithm under Implementation 4, using different cut ratios - Part II 244

K.12 Running time in seconds for multiobjective KP using DCM and MOMIX 245

K.13 Running time in seconds for biobjective SPP using DCM 246

K.14 Running time in seconds for multiobjective Knapsack using MOA packages 247

K.15 Running time in seconds for biobjective SPP using MOA packages 248

K.16 Running times (in seconds) for all MILP instances solved with the RVF Algorithm under Implementation 3, using different numbers of feasible solutions 249

K.17 Running times (in seconds) for all MILP instances solved with the RVF Algorithm under Implementation 4, using different cut ratios 250

K.18 Running time (in seconds) for all MILP instances using the RVF Algorithm with Implementations 0, 1, 2, and 6 251

M.1 Running time in seconds for the SPP using the BB/sample-based Algorithm for different initial sample sizes (denoted as n) 259

M.2 Running time in seconds for the KP using the BB/sample-based Algorithm for different initial sample sizes (denoted as n) 260

M.3 Running time in seconds for the random PIP instances using the BB/sample-based Algorithm for different initial sample sizes (denoted as n) - part I 261

M.4 Running time in seconds for the random PIP instances using the BB/sample-based Algorithm for different initial sample sizes (denoted as n) - part II 262

M.5 Running time in seconds for the SPP using the BB/batch-based Algorithm for different initial sample sizes (denoted as n) and batch sizes 263

M.6 Running time in seconds for the KP using the BB/batch-based Algorithm for different initial sample sizes (denoted as n) and batch sizes 264

M.7 Running time in seconds for the random PIP instances using the BB/batch-based Algorithm for different initial sample sizes (denoted as n) and batch sizes - part I . . 265

M.8 Running time in seconds for the random PIP instances using the BB/batch-based Algorithm for different initial sample sizes (denoted as n) and batch sizes - part II . 266

M.9 Average running time (in seconds) for MILP instances solved by the BB/sample-based algorithm with an RVF call frequency of 5 and varying initial sample sizes 267

M.10 Average running time (in seconds) for MILP instances solved by the BB/sample-based algorithm with an RVF call frequency of 10 and varying initial sample sizes 268

M.11 Average running time (in seconds) for MILP instances solved by the BB/sample-based algorithm with an RVF call frequency of 50 and varying initial sample sizes 269

M.12 Average running time (in seconds) for MILP instances solved by the BB/sample-based algorithm with an RVF call frequency of 100 and varying initial sample sizes 270

M.13 Summary of all data used in the deep dive analysis of MILP cases 271

List of Figures

1.1 The portion of the RVF and the EF associated with Example 1 with $\zeta \geq -15$. Note that the frontier extends to the left up to $\zeta = -57\frac{2}{3}$; the finite domain of z is $\mathcal{C} = [-57\frac{2}{3}, +\infty)$. The complete EF is shown in Figure B.1. 7

2.1 Stability regions and corresponding members of $\mathcal{S}_I$ in Example 1, for the region $\zeta \geq -15$, stability regions from left to right correspond to $(x_1, x_2) \in \{(1,0), (1,1), (0,1), (1,0), (0,0)\}$. 30

2.2 Stability regions of the instance in Example 3. The area shaded in red, including the red boundary lines, is not in the stability region for $x_1 = 1$, which is thus neither convex nor closed. The line segment shaded in purple is in the stability region of both $x_1 = 0$ and $x_1 = 1$. 32

2.3 EF and RVF for the MILP in Example 4. 38

3.1 RVF and upper approximation in iterations of the RVF Algorithm with Example 1. The RVF z is illustrated on the range $\zeta \in [-15, 5]$. Note that the first cut occurs at $\zeta = -55.5$, which is outside of this range. Please refer to Appendix B to view the full EF or RVF. 45

3.2 An example with two objectives, four elements of $\mathcal{S}_{\min}$, all with disconnected stability regions, having a total of 24 linear pieces in the EF, shown with thicker lines. The LP EFs for the four elements of $\mathcal{S}_{\min}$ are shown as red solid, blue dashed, sepia solid, and green dotted lines, respectively. Numerical details are given in Appendix J. Close-up of the areas in the top-left and bottom-right parts are given in Figure 3.3. 58

3.3 Close-ups of the regions in the top-left and bottom-right of Figure 3.2 (Note that these have been asymmetrically scaled to allow better visibility of LP frontier intersections). 59

3.4 Normalized approximation gap vs. iteration number for the nonlinear version of the RVF Algorithm and RVF Algorithm With Implementation 1 on SPP instances. . . . 68

3.5 Normalized approximation gap vs. iteration number for the nonlinear version of the RVF Algorithm and RVF Algorithm With Implementation 1 on Knapsack instances. 68

3.6 Performance profile comparing execution time of naive linear and nonlinear version of the RVF Algorithm for PILPs. 98

3.7 Performance profile comparing execution time of naive linear version of the RVF Algorithm and Implementation 1 for PILPs. 98

3.8 Performance profile comparing execution time of Implementation 3 for PILPs. 100

3.9 Performance profile comparing execution time of Implementation 4 using different percentages of cuts for PILPs. 105

3.10 Performance profile comparing the execution time of the naive linear version and different implementations of the RVF Algorithm for PILPs. 105

3.11 Performance profile comparing the execution time of the RVF Algorithm (Implementation 2) with all existing algorithms for the biobjective SPP. 107

3.12 Performance profile comparing the execution time of the RVF Algorithm (Implementation 2) with all existing algorithms for the multiobjective KP. 108

3.13 Performance profile comparing execution time of RVF and DCM for SPP. 109

3.14 Performance profile comparing execution time of RVF and DCM for KP. 109

3.15 Performance profile comparing execution time of RVF and DCM for all PILPs. . . . 110

3.16 Hypervolume and Adjusted Hypervolume. 111

3.17 Volume. 112

3.18 Performance profile comparing execution time of Implementation 3 for MILPs. . . . 116

3.19 Performance profile comparing execution time of Implementation 4 for MILPs. . . . 117

3.20 Performance profile comparing the execution time of the naive version and different implementations of the RVF Algorithm for MILPs. 118

4.1 $z(\zeta)$ and RVFLB^1. 137

4.2 $z(\zeta)$ and RVFLB^2. 138

4.3 $z(\zeta)$ and RVFLB^3. 140

4.4 $z(\zeta)$ and RVFLB^4. 141

4.5 Performance profile comparing execution time of the algorithm with and without warm-starting for SPP and KP instances. 157

4.6 Performance profile comparing the execution times of the BB/sample-based Algorithm for different initial sample size parameters across all PILP instances. 162

4.7 Performance profile comparing the execution times of the BB/batch-based algorithm, with a batch size of 3, for different initial sample size parameters across all PILP instances. 163

4.8 Performance profile comparing the execution times of the BB/batch-based algorithm, with a batch size of 4, for different initial sample size parameters across all PILP instances. 164

4.9 Performance profile comparing the execution times of the BB/batch-based algorithm, with a batch size of 5, for different initial sample size parameters across all PILP instances. 165

4.10 Performance profile comparing the execution times of the BB/batch-based algorithm for each category across all PILP instances. 166

4.11 Performance profile comparing the execution times of the BB/sample-based and BB/batch-based algorithms across all PILP instances. 167

4.12 Performance profile comparing the execution times of the BB Algorithm and the RVF Algorithm (with Implementation 2) for all PILP instances, including SPP, KP, and randomly generated PILP instances. 167

4.13 Performance profile comparing the number of iterations of the RVF subproblem and SYMPHONY within the BB Algorithm for all PILP instances. 168

4.14 Performance profile comparing the total time of the RVF subproblem and SYMPHONY within the BB Algorithm for all PILP instances. 168

4.15 Performance profile comparing the average time (total time divided by total iterations) of the RVF subproblem and SYMPHONY within the BB Algorithm for all PILP instances. 169

4.16 Performance profile comparing the total time of solving MILPs with SYMPHONY, the total time of building the dual function, and the total time of evaluating the dual function within the BB Algorithm for all PILP instances. 169

4.17 Performance profile comparing the average time (total time divided by total nodes) of the RVF subproblem and SYMPHONY within the BB Algorithm for all PILP instances. 170

4.18 A graph illustrating the average number of new nodes generated by the RVF subproblem and SYMPHONY within the BB Algorithm for SPP. 170

4.19 A graph illustrating the average number of new nodes generated by the RVF subproblem and SYMPHONY within the BB Algorithm for KP. 171

4.20 A graph illustrating the average number of new nodes generated by the RVF subproblem and SYMPHONY within the BB Algorithm for randomly generated PILP instances. 171

4.21 Performance profiles comparing the execution times of the BB/sample-based algorithm under different RVF call frequencies and initial sample sizes across all MILP instances. 174

4.22 Performance profile comparing the execution times of RVF and BB for all MILPs. . 176

4.23 Performance profile comparing the number of iterations of the RVF subproblem and SYMPHONY within the BB Algorithm for all MILPs. 179

4.24 Performance profile comparing the total time of the RVF subproblem and SYMPHONY within the BB Algorithm for all MILPs. 179

4.25 Performance profile comparing the average time (total time divided by total iterations) of the RVF subproblem and SYMPHONY within the BB Algorithm for all MILPs. . 180

4.26 Performance profile comparing the total time of solving MILPs with SYMPHONY, the total time of building the dual function, and the total time of evaluating the dual function within the BB Algorithm for all MILPs. 180

4.27 Performance profile comparing the average time (total time divided by total nodes) of the RVF subproblem and SYMPHONY within the BB Algorithm for all MILPs. . 181

4.28 A graph illustrating the average number of new nodes generated by the RVF subproblem and SYMPHONY within the BB Algorithm for MILPs. 181

4.29 A graph illustrating the average number of new nodes generated by the RVF subproblem and SYMPHONY within the BB Algorithm for MILPs (log scale). 182

A.1 The RLPVF associated with Example 1 when $(x_1, x_2) = (0, 0)$. 201

B.1 The EF associated with Example 1. 203

B.2 The LP frontiers. Brown is for $(x_1, x_2) = (0, 0)$. Green is for $(x_1, x_2) = (1, 0)$. Red is for $(x_1, x_2) = (1, 1)$. Blue is for $(x_1, x_2) = (0, 1)$. Note that the axes are unequally scaled. 203

B.3 The (truncated at the left) LP frontiers. Brown is for $(x_1, x_2) = (0, 0)$. Green is for $(x_1, x_2) = (1, 0)$. Red is for $(x_1, x_2) = (1, 1)$. Blue is for $(x_1, x_2) = (0, 1)$. 204

H.1 Execution time vs. iteration for biobjective SPP. 225

H.2 Execution time vs. iteration for multiobjective Knapsack. 226

J.1 The complete LP frontier for the parts shown in red in Figure 3.2, together with the complete LP frontier for the part shown in green. 231

J.2 The complete LP frontier for the parts shown in blue in Figure 3.2, together with the complete LP frontier for the part shown in green. 232

J.3 The complete LP frontier for the parts shown in brown in Figure 3.2, together with the complete LP frontier for the part shown in green. 232

L.1 Base performance profile comparing execution time of Implementation 3 for PIPs. . 253

L.2 Base performance profile comparing execution time of Implementation 4 using different percentages of cuts for PIPs. 253

L.3 Base performance profile comparing the execution time of the naive linear version and different implementations of the RVF algorithm for PIPs. 254

L.4 Base performance profile comparing the execution time of the RVF Algorithm (Implementation 2) with all existing algorithms for the biobjective SPP. 254

L.5 Base performance profile comparing the execution time of the RVF Algorithm (Implementation 2) with all existing algorithms for the multiobjective KP. 255

L.6 Base performance profile comparing execution time of RVF and DCM for all PIPs. . 255

L.7 Base performance profile comparing execution time of Implementation 3 for MILPs. 256

L.8 Base performance profile comparing execution time of Implementation 4 for MILPs. 256

L.9 Base performance profile comparing the execution time of the naive version and different implementations of the RVF algorithm for MILPs. 257

N.1 Base performance profile comparing the execution times of the BB Algorithm and the RVF Algorithm (with Implementation 2) for all PIP instances, including SPP, KP, and randomly generated PIP instances. 273

N.2 Base performance profile comparing the execution times of RVF and BB for all MILPs. 273

Abstract

In this dissertation, we investigate the connection between the efficient frontier (EF) of a general multiobjective mixed integer linear program (MILP) and the so-called *restricted value function* (RVF) of a closely related single-objective MILP. We present two distinct methodologies for constructing the EF and RVF of an MILP based on information derived from the value functions. The first approach is detailed in the second chapter, with its implementations discussed in the third chapter. The second method is introduced and elaborated upon in the fourth chapter, and the final chapter summarizes the main contributions and outlines directions for future research.

In the first chapter of this dissertation, we examine concepts from duality theory and value functions, as well as from multiobjective optimization, exploring their practical implications and applications. Furthermore, we review existing research in these two areas, which have traditionally been investigated in separate bodies of the literature.

In the second chapter, we demonstrate that the EF of the multiobjective MILP consists of points on the boundary of the epigraph of the RVF, so that any description of the EF suffices to describe the RVF and vice versa. First, we describe the mathematical structure of the RVF, including characterizing the set of points at which it is differentiable, the gradients at such points, and the subdifferential at all nondifferentiable points. Because of the close relationship of the RVF to the EF, we observe that methods for constructing value functions and methods for constructing the EF, which have been developed in distinct research communities, are effectively interchangeable. By exploiting this relationship, we propose a generalized cutting-plane algorithm called the RVF Algorithm for constructing the EF of a multiobjective MILP based on a generalization of an existing algorithm for constructing the classical value function. We prove that the algorithm is finite under a standard boundedness assumption and comes with a performance guarantee if terminated early.

In the third chapter, we present computational implementations of the RVF Algorithm previously examined in detail in Chapter 2. These implementations play a central role in computational frameworks, as the RVF Algorithm has a wide range of applications. Through computational experiments on pure integer and mixed integer instances drawn from various classes of optimization problems, we demonstrate the superiority of one implementation over both a naive version of the RVF Algorithm and existing algorithms, according to several commonly used performance metrics.

In the fourth chapter, we present a unified branch-and-bound (BB) framework for constructing both the RVF and the EF associated with a multiobjective MILP by utilizing the SYMPHONY framework. We enhance two key aspects of the RVF Algorithm outlined in the second and third chapters. Firstly, we incorporate the warm-start strategy implemented in SYMPHONY to obviate the need to solve the related subproblems arising during the algorithms from scratch, a limitation present in the approaches outlined in the second and third chapters. Secondly, we introduce a lower approximation method for constructing the RVF. At each iteration, based on the dual solutions obtained thus far, we update the lower approximation. The algorithm terminates when the approximation becomes exact, thereby obviating the necessity to solve a nonlinear problem in the primary phase, as necessitated by the methodology described in the second and third chapters, where the gap between the upper approximation and the true value function required solving nonlinear subproblems. Instead, our approach involves solving a linear problem in each iteration. However, we still invoke the RVF Algorithm at both intermediate steps and at the conclusion to determine whether the termination condition has been met. Furthermore, we conduct extensive computational experiments on both pure integer and mixed integer programs, comparing the BB algorithms with the RVF Algorithms outlined in Chapters 2 and 3.

In the final chapter, we summarize the main contributions of the dissertation and discuss directions for future research.

Chapter 1

Introduction

In this dissertation, we consider the relationship between the efficient frontier (EF) of a multiobjective mixed integer linear program (MILP) and a certain value function (VF), which we refer to as the *restricted value function* (RVF), associated with a closely related MILP. Our main result is that any description of the EF suffices to describe the RVF, and vice versa. Specifically, we demonstrate that the EF is composed of a subset of the points on the boundary of the epigraph of the RVF (also referred to as the *graph* of the RVF), and that any point on the boundary and *not* in the EF has (i) the same objective value as a point on the EF that dominates it, and (ii) a directional derivative of zero, in the direction towards the dominating point.

Although our results demonstrate that the EF and the RVF are closely related and indeed effectively interchangeable, their relationship seems not to have been previously observed, mainly because of the disparate ways in which these mathematical objects have been described in the separate literatures in which the concepts have been developed. Upon closer examination, however, the relationship between the RVF and the EF is quite intuitive. While we believe this work is the first to formally and explicitly establish the relationship, it can be seen implicitly in the results of several previous works, such as those by [Trapp et al., 2013], [Ralphs and Hassanzadeh, 2014], and [Bodur et al., 2022]. The so-called minimal tenders utilized in Trapp et al.'s algorithm for constructing the value function (VF) of a pure integer linear optimization problem (PILP) can be seen as the points on the EF of a related multiobjective problem. Ralphs and Hassanzadeh [2014] generalized this concept in their work on the structure of the VF of a general MILP. More recently, Bodur et al. [2022]

observed that, in block-structured problems, the solution to the column generation subproblem can be viewed as equivalent to evaluating a certain VF, and solutions can thus be restricted only to the nondominated points on the EF of an associated multiobjective optimization problem.

The relationship described in the remainder of this dissertation has some apparently broad-ranging implications, including that algorithms designed for the construction and/or approximation of the EF are effectively interchangeable with algorithms for the construction and/or approximation of the RVF and VFs in general. Because algorithms for these two tasks have so far been used in very different application domains and for very different purposes, there are likely many possibilities for the cross-pollination of ideas. To illustrate this, we propose a generalized cutting-plane algorithm for constructing both the RVF and the EF. The approach we suggest is finitely convergent, exploits the discrete structure of the RVF, and provides a performance guarantee if terminated early. It is a modified version of an existing algorithm for constructing the full VF, and to the best of our knowledge, the approach is entirely different from existing algorithms for the construction of the EF. Additionally, our algorithm is one of the few algorithms developed to date that addresses multiobjective MILPs in the presence of continuous variables with any number of objectives, and it yields improved bounds on the number and size of subproblems to be solved to determine the discrete structure of the EF. These bounds are also comparable to existing algorithms for the PILP case (without continuous variables).

In the remainder of this section, we set the stage by formally defining the important terms and concepts. We first describe the terminology and basic properties related to multiobjective MILPs and their associated EFs before introducing the concept of the RVF. Although the RVF that we introduce is closely related to the classical VF of a single-objective MILP (it can be viewed as a generalization), we are not aware of any previous study of it. Its properties are much more difficult to characterize than those of the classical VF.

Multiobjective Optimization. The multiobjective MILP that serves as the focus of our study is defined as

$$\operatorname{vinf}_{(x_I,x_C)\in X_{\mathrm{MO}}} C_I x_I + C_C x_C, \tag{MO-MILP}$$

where

$$X_{\mathrm{MO}} = \left\{(x_I, x_C) \in \mathbb{Z}^r_+ \times \mathbb{R}^{n-r}_+ \ : \ A_I x_I + A_C x_C = b\right\},$$

is the feasible region; $A \in \mathbb{Q}^{m \times n}$ is the coefficient matrix of the constraints; $b \in \mathbb{Q}^m$ is the right-hand side (RHS) of the constraints; and the rows of matrix $C \in \mathbb{Q}^{(\ell+1) \times n}$ are the multiple objectives of the problem and denoted by $\{c^0, c^1, \ldots, c^\ell\}$. The vinf operator indicates that this is a vector minimization (multiobjective) problem, which means that there is not a single optimal value; the vinf operator returns a set of nondominated vectors of objective values, as described below [1]. A_I and C_I are the submatrices of A and C consisting of columns associated with the integer variables (indexed by set $I = \{0, \ldots, r-1\}$), as opposed to A_C and C_C, which are the submatrices corresponding to the columns associated with the continuous variables (indexed by set $C = \{r, \ldots, n-1\}$). We assume that the feasible region X_{MO} is bounded.

The aim of multiobjective optimization is to characterize the trade-offs inherent in optimizing multiple objectives simultaneously. This analysis is most naturally done in the $(\ell+1)$-dimensional space known as the *criterion space*, which contains the vectors of objective values associated with points in the n-dimensional *decision space*, the space containing the feasible region X_{MO}. While solving an MILP with a single objective means determining its unique optimal value, solving a multiobjective MILP means generating the set of all vectors in criterion space associated with the so-called efficient solutions, those for which there is no other solution for which the objective value is at least as good for every objective and strictly better for at least one objective.

We briefly review some concepts in multiobjective optimization, referring interested readers to [Ehrgott, 2005] for more details. An important concept in this context is that of dominance. The point $C_I x_I + C_C x_C \in \mathbb{R}^{\ell+1}$ in criterion space, associated with $(x_I, x_C) \in X_{\text{MO}}$, *dominates* $C_I x'_I + C_C x'_C \in \mathbb{R}^{\ell+1}$, associated with $(x'_I, x'_C) \in X_{\text{MO}}$, if $C_I x_I + C_C x_C \lneqq C_I x'_I + C_C x'_C$, i.e., $(C_I x_I + C_C x_C)_j \leq (C_I x'_I + C_C x'_C)_j$ for all $j = \{0, 1, \ldots, l\}$ and $(C_I x_I + C_C x_C)_j < (C_I x'_I + C_C x'_C)_j$ for at least one index $j \in \{0, 1, \ldots, l\}$. A point in criterion space that is not dominated by any other point is called a *nondominated point* (NDP). The set of all NDPs is the aforementioned EF. A preimage of an NDP in the decision space is referred to as an *efficient solution*. A point $(x_I, x_C) \in X_{\text{MO}}$ that is not necessarily efficient but for which there does not exist $(x'_I, x'_C) \in X_{\text{MO}}$ such that $C_I x'_I + C_C x'_C < C_I x_I + C_C x_C$ is called *weakly efficient*, and the associated point $C_I x_I + C_C x_C$ in criterion space is referred to as a *weakly nondominated point* or a *weak NDP*. It is important to note that a weak NDP that is not also an NDP is, in fact, a *dominated* point.

[1] We use inf rather than min because the definition of the RVF below introduces the possibility of MILPs with irrational right-hand sides for which attainment of the optimal value is not guaranteed

Restricted Value Function. What we call the RVF provides another way of analyzing the trade-offs in the multiobjective optimization problem. Specifically, we consider the following related MILP with a single objective obtained by imposing all but one of the objectives in (MO-MILP) as constraints, which we refer to as the *parametric constraints*. This MILP can be written as follows:

$$\inf_{(x_I,x_C)\in X} c_I^0 x_I + c_C^0 x_C, \tag{MILP}$$

where

$$X = \left\{(x_I, x_C) \in \mathbb{Z}^r_+ \times \mathbb{R}^{n-r}_+ \ : \ C_I^{1:\ell} x_I + C_C^{1:\ell} x_C \leq f, \ A_I x_I + A_C x_C = b\right\},$$

is the feasible region; c^0 is the first row of the matrix C; $C^{1:\ell}$ is the submatrix consisting of the remaining rows of C, and $f \in \mathbb{Q}^\ell$ is a fixed vector to be replaced shortly by a parameter to obtain the aforementioned RVF. The particular objective that is chosen as c^0 is arbitrary, and the results hold no matter what objective is chosen.

We now define the RVF $z : \mathbb{R}^\ell \to \mathbb{R} \cup \{\pm\infty\}$ associated with (MILP) to be the function

$$z(\zeta) = \inf_{(x_I,x_C)\in \mathcal{S}(\zeta)} c_I^0 x_I + c_C^0 x_C, \tag{RVF}$$

that returns the optimal solution value of (MILP) as a function of a RHS parameter $\zeta \in \mathbb{R}^\ell$, where

$$\mathcal{S}(\zeta) = \left\{(x_I, x_C) \in \mathbb{Z}^r_+ \times \mathbb{R}^{n-r}_+ \ : \ C_I^{1:\ell} x_I + C_C^{1:\ell} x_C \leq \zeta, \ A_I x_I + A_C x_C = b\right\}.$$

Note that the classical VF [Blair and Jeroslow, 1977, 1979, Güzelsoy and Ralphs, 2007b, Ralphs and Hassanzadeh, 2014] is closely related and is the special case of the RVF in which the only constraints with a fixed right-hand side are the bound constraints ($m = 0$). As usual, we let $z(\zeta) = +\infty$ for $\zeta \notin \mathcal{C}$, where

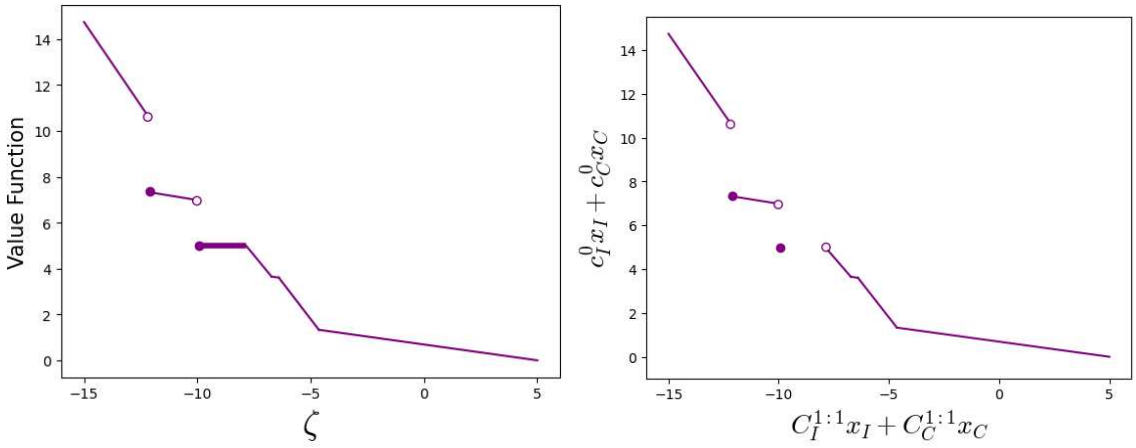
$$\mathcal{C} = \left\{\zeta \in \mathbb{R}^\ell \ : \ \mathcal{S}(\zeta) \neq \emptyset\right\}.$$

The function z is always bounded from below because of our assumption that the feasible region X_{MO} of the multiobjective MILP is bounded.

Example 1. *Here, we illustrate the concepts discussed so far. Consider an RVF defined by*

$$
\begin{aligned}
z(\zeta) = \quad & \min 2x_1 + 5x_2 + 7x_4 + 10x_5 + 2x_6 + 10x_7 \\
& \text{s.t. } -x_1 - 10x_2 + 10x_3 - 8x_4 + x_5 - 7x_6 + 6x_7 \leq \zeta \\
& \quad -x_1 + 4x_2 + 9x_3 + 3x_4 + 2x_5 + 6x_6 - 10x_7 = 4 \\
& \quad 5x_2 + x_4 + x_8 = 5 \\
& \quad 5x_2 + x_7 + x_9 = 5 \\
& \quad x_j \in \{0, 1\}, \quad \forall j \in \{1, 2\} \\
& \quad x_j \in \mathbb{R}_+, \quad \forall j \in \{3, 4, \ldots, 9\},
\end{aligned}
$$

for all $\zeta \in \mathbb{R}$. *This is not precisely in our standard form because the fixed upper bounds on the integer variables are embedded in their definition. However, it can easily be converted to our standard form. Figure 1.1a below shows the VF for the MILP, while Figure 1.1b shows the EF for the associated multiobjective optimization problem. Note that the graph of the RVF and the EF are identical except for the horizontal line segment between the points* $(-10, 5)$ *and* $(-7\frac{5}{6}, 5)$, *which has been thickened in the figure for emphasis.*



(a) RVF *(b) EF*

Figure 1.1: The portion of the RVF and the EF associated with Example 1 with $\zeta \geq -15$. *Note that the frontier extends to the left up to* $\zeta = -57\frac{2}{3}$; *the finite domain of* z *is* $\mathcal{C} = [-57\frac{2}{3}, +\infty)$. *The complete EF is shown in Figure B.1.*

1.1 Related Work

Methods for constructing both the EF of a multiobjective optimization problem and for constructing the VF of an MILP have been extensively studied in the open literature. As the literature is vast, we focus here on the most closely related works.

1.1.1 Multiobjective Optimization

Multiobjective optimization, the analysis of trade-offs between multiple conflicting objective functions, has numerous applications across various fields, as most real-world problems arising in practice do have multiple objectives. In this study, we address the generation of the exact EF for multiobjective MILPs. For a recent survey and a comprehensive overview of algorithms, we recommend [Halffmann et al., 2022].

Algorithms in the multiobjective area can be roughly classified into two categories: scalarization-based and non-scalarization-based methods. Scalarization techniques are the most common and involve solving a sequence of single-objective problems, each one producing a single NDP or otherwise yielding information about a local region of the EF. Non-scalarization methods, on the other hand, take a more global view, using, e.g., an outer approximation to bound the EF by convex or nonconvex bounding functions. In the MILP case, some variant of branch-and-bound is typically used to implement these algorithms.

A wide variety of scalarization-based methods have been proposed, including the *weighted sum* method [Zadeh, 1963], the *perpendicular search* method [Chalmet et al., 1986], the *weighted Tchebycheff* method [Bowman, 1976, Yu, 1973, Zeleny, 1973], the ϵ-*constraint* method [Haimes, 1971], the Hybrid method [Guddat et al., 1985], Benson's method [Benson, 1978], and the Pacoletti-Serafini's method [Pascoletti and Serafini, 1984]. These methods typically involve an iterative process in which a list of discovered NDPs and a list of unexplored regions of the criterion space are maintained. At each iteration, the algorithm produces a new NDP within an unexplored region of the criterion space. The process is repeated until there are no more unexplored regions. For a comprehensive review of scalarization methods in the multiobjective optimization field, the reader is referred to [Ehrgott, 2006].

The most straightforward scalarization method is the weighted sum method. The single objective created by this method is a weighted sum of the original objectives. When the weights are all positive, the solution to the weighted sum problem is guaranteed to be nondominated. On the other hand, not all nondominated solutions can be generated as a solution to some weighted sum problem in the MILP case—only the so-called *supported NDPs* can be generated in this way. The NDPs that can be found via weighted sum-based scalarization are called *supported* NDPs.

Several methods have been developed to address unsupported NDPs. The augmented weighted Tchebycheff method [Bowman, 1976] seeks to find NDPs within the exploration region by minimizing the distance to the ideal point, which is defined as the point whose components are obtained by minimizing the objective functions. Ralphs et al. [2006] proposed a weighted Tchebycheff scalarization algorithm for constructing the EF of a biobjective integer programming problem.

The ϵ-constraint method, which involves minimizing a primary objective while restricting the other objectives through inequality constraints, is widely used with many existing variants. This algorithmic approach has an obvious connection to the RVF. Tamby and Vanderpooten [2021] has recently expanded the use of the ϵ-constraint method to address problems involving two or more objectives. The proposed algorithm divides the search area into segments that can be individually explored by solving an integer program.

To the best of our knowledge, the GoNDEF algorithm developed by [Rasmi and Türkay, 2019] is currently the only algorithm that utilizes scalarization techniques to address MILPs with more than two objectives. GoNDEF first finds all integer parts of NDPs (as our proposed algorithm also does) and then attempts to describe the non-dominated facets of the stability regions associated with each integer part.

Recently, a number of non-scalarization-based methods have been proposed. Forget et al. in 2022b and 2022a focus on 0-1 problems and employ outer approximation for computing the linear relaxation to generate lower bound sets using a Benson-like algorithm. Most other non-scalarization-based algorithms are focused on the more general multiobjective mixed integer convex and nonconvex optimization settings with any number of objectives. In particular, the MOMIX [De Santis et al., 2020] and HyPaD [Eichfelder and Warnow, 2023, 2021] algorithms address multiobjective mixed integer convex optimization problems, while the algorithms proposed in [Eichfelder et al., 2021, 2023, Link and Volkwein, 2022] address both the convex and nonconvex cases. All of these algorithms

compute some kind of enclosure of the EF, which is iteratively refined until a convergence criterion is achieved. The RVF algorithm proposed herein similarly maintains a kind of enclosure, maintaining a global upper bounding function, but the upper bounding function converges to the EF without the need for a lower bounding function. On the other hand, this is why the RVF algorithm requires the solutions of a mixed integer nonlinear optimization problem in each iteration.

More recently, Dunbar et al. [2023] considered the optimization of multiobjective integer programs through various relaxation and duality approaches, including continuous, convex hull, and Lagrangian relaxations, as well as Lagrangian duals and set- and vector-valued superadditive duals, offering alternative solution methods. For more on approaches to approximating the EF using outer approximations, we refer the interested reader to [Benson, 1998, Hamel et al., 2014, Csirmaz, 2016, Löhne and Weißing, 2015, Ruzika and Wiecek, 2005]. A comprehensive survey of branch-and-bound methods for multiobjective linear integer problems can be found in [Przybylski and Gandibleux, 2017]. For a detailed review of the literature on multiobjective optimization, we refer the interested reader to [Ehrgott and Gandibleux, 2000, Ehrgott and Wiecek, 2005, Ehrgott et al., 2016].

1.1.2 Value Function

The classical VF of an MILP is well-studied, and understanding its structure is crucial for many applications due to its role as a core ingredient in optimality conditions used in a variety of algorithms for solving optimization problems. These optimality conditions are also employed in formulating and solving important classes of multistage and multilevel optimization problems, in which optimality conditions are embedded as constraints in a larger optimization problem. Additionally, optimality conditions are also the basis for techniques used for warm-starting and sensitivity analysis, which are the areas in which the connection to multiobjective optimization is most apparent.

There have been several studies investigating the structure of the VF in MILPs. Blair and Jeroslow [1977] and Blair and Jeroslow [1979] identified fundamental properties of the VF, including that it is comprised of a minimum of a finite number of polyhedral functions. Blair and Jeroslow [1982] showed that the VF of a PILP is a *Gomory function*, which is the maximum of subadditive functions known as *Chvátal functions*. Blair and Jeroslow [1984] extended this result to general MILPs, demonstrating that they are the maximum of Gomory functions. Finally, Blair [1995] identified what was then referred to as a "closed-form" representation of the MILP VF, the so-called *Jeroslow formula*, though

this did not lead to what could be considered a practical representation. Güzelsoy and Ralphs [2007a] further studied the properties of the VF as it is related to methods of warm-starting and sensitivity analysis, and also suggested a method of construction for the case of an MILP with a single constraint. Ralphs and Hassanzadeh [2014] extended this work by providing further details on the structure and properties of the VF for a general MILP and suggesting a practical representation.

Most methods for constructing the VF have focused on the case of PILP, where the discrete structure is the most evident and a finite representation is the easiest to achieve. Wolsey [1981] used a cutting-plane method to derive a sequence of Chvátal functions that leads to constructing the full VF for a PILP. Conti and Traverso [1991] employed reduced Gröbner bases and modified the classical Buchberger's algorithm to solve PILPs. Later, Schultz et al. [1998] used Gröbner basis methods to solve two-stage stochastic programs with complete integer recourse and different RHSs. The authors identified a countable set known as the candidate set of the first-stage variables in which the optimal solution is contained. Then Ahmed et al. [2004] developed a global optimization algorithm for solving general two-stage stochastic programs with integer recourse and discrete distributions by exploiting the structure of the second-stage integer problem VF. The authors demonstrated that their algorithm avoids enumerating the search space. Kong et al. [2006] considered a two-stage PILP and presented a superadditive dual formulation that exploits the VF in both stages, solving that reformulation by a global branch-and-bound or level-set approach. Trapp and Prokopyev [2015] proposed a constraint aggregation-based approach to alleviate the memory requirement for storing the VF. Zhang and Özaltın [2021] first generalized the complementary slackness theorem to bilevel IP (BIP) and showed that it can be an advantage for constructing the VFs of BIP. The authors also demonstrated that the VFs of BIPs can be constructed by bilevel minimal RHS vectors and presented a dynamic programming algorithm for constructing the BIP VF. Finally, Brown et al. [2021] used a Gilmore-Gomory approach to construct the IP VF.

There have been relatively few algorithmic advances in finding the VF of a general MILP. Bank et al. [1982] studied the qualitative and quantitative stability properties of mixed integer multiobjective optimization problems, which can also be considered an MILP VF. Güzelsoy and Ralphs [2006] proposed algorithms for constructing the VF of an MILP with a single constraint. The properties of the VF and a method for constructing the VF in the case of a general MILP were discussed in Ralphs and Hassanzadeh [2014]. In the current work, we generalize the work in Ralphs and Hassanzadeh [2014] to the multiobjective setting. Our hope is that methods of constructing the

VF will now quickly be advanced by the adoption of techniques developed in the multiobjective community.

1.2 Application of RVF

The value function, or EF, is a concept that has numerous applications in various fields. In this section, we focus on several examples of its use in different contexts in Subsections 1.2.1-1.2.5. These examples illustrate the versatility and utility of the value function as a tool for understanding and analyzing decision-making processes.

1.2.1 Dantzig-Wolfe Decomposition

The Dantzig-Wolfe Decomposition [Dantzig and Wolfe, 1960] is a mathematical optimization technique for solving MILPs, particularly useful for solving large and complex MILP problems. It involves decomposing the MILP problem into a series of smaller subproblems that can be solved independently. The solutions to these subproblems are then combined to form a solution to the original MILP problem. Consider the MILP problem as follows.

$$\begin{aligned} &\text{maximize} && c^\top x && \text{(MILP-D)}\\ &\text{subject to} && A'x \leq b' && \text{(1.1a)}\\ &&& A''x \leq b'' && \text{(1.1b)}\\ &&& x \in \mathbb{Z}^n. && \text{(1.1c)} \end{aligned}$$

We hereby define the following sets.

$$\begin{aligned} \mathcal{Q}' &= \{x \in \mathbb{R}^n \mid A'x \leq b'\},\\ \mathcal{Q}'' &= \{x \in \mathbb{R}^n \mid A''x \leq b''\},\\ \mathcal{Q} &= \mathcal{Q}' \cap \mathcal{Q}'',\\ \mathcal{S} &= \mathcal{Q} \cap \mathbb{Z}^n,\\ \mathcal{S}_R &= \mathcal{Q}' \cap \mathbb{Z}^n. \end{aligned}$$

Assuming that optimizing over $\mathcal{S}_R$ is tractable, we relax the constraints in Equation (1.1a). As a result, we can express any point in $\text{conv}(\mathcal{S}_R)$ as a convex combination of the extreme points of $\text{conv}(\mathcal{S}_R)$. Based on this observation, we can formulate the Dantzig-Wolfe linear programming (DWLP) as follows.

$$\text{maximize} \quad c^\top x \tag{DW-LP1}$$

$$\text{subject to} \quad \sum_{s \in \varepsilon} \lambda_s s = x \tag{1.2a}$$

$$A'' x \leq b'' \tag{1.2b}$$

$$\sum_{s \in \varepsilon} \lambda_s = 1 \tag{1.2c}$$

$$\lambda \in \mathbb{R}_+^{|\varepsilon|}, \tag{1.2d}$$

where ε is the set of extreme points of $\text{conv}(\mathcal{S}_R)$. It can be demonstrated that equation (DW-LP1) represents a relaxation and upper bound of equation (MILP-D). The following formulation can be obtained by substituting the variable x.

$$\text{maximize} \quad \sum_{s \in \varepsilon} \left(c^\top s\right) \lambda_s \tag{DW-LP2}$$

$$\text{subject to} \quad \sum_{s \in \varepsilon} \left(A'' s\right) \lambda_s \leq b'' \tag{1.3a}$$

$$\sum_{s \in \varepsilon} \lambda_s = 1 \tag{1.3b}$$

$$\lambda \in \mathbb{R}_+^{|\varepsilon|}. \tag{1.3c}$$

The column generation technique, first proposed by [Dantzig, 2016], is commonly used to solve (DW-LP2), also known as the *master problem*. The basic steps of this algorithm are as follows:

1. Start with a subset of promising columns and solve the restricted master problem, which is the master problem with a subset of columns.
2. Price the remaining columns and add those with positive reduced costs (in the maximization problem) that can improve the objective function value.
3. Iterate until no column with a positive reduced cost is found that can be added to the restricted master problem, at which point optimality is reached.

This procedure allows for a solution to the master problem by avoiding the need to consider all possible columns from the start.

We demonstrate the concept of reduced cost in the context of a linear programming (LP) problem. The LP problem can be defined as follows:

$$\begin{aligned} \text{maximize} \quad & c^\top x && \text{(LP)} \\ \text{subject to} \quad & Ax = b && \text{(1.4a)} \\ & x \geq 0. && \text{(1.4b)} \end{aligned}$$

In this analysis, we utilize the notation of B and x^B to represent the basis and basic variables, respectively. We consider the scenario in which a nonbasic variable, denoted as x^k, can be introduced into the basis with a value of λ, such that $x^k = \lambda$. In order to maintain feasibility, the constraint $Ax = b$ must also be satisfied by the inclusion of this new variable. This can be expressed as follows:

$$\begin{aligned} B(x^B + d^B) + A^k\lambda &= b, \\ Bd^B + A^k\lambda &= 0, \\ d^B &= -\lambda B^{-1}A^k. \end{aligned}$$

If the nonbasic variable x^k is increased from 0 to λ, the objective function value will increase by $c^k\lambda$, while at the same time, the changes in the basic variables will decrease the objective function value by $\lambda c_B B^{-1}A^k$. If the net change resulting from these two effects is positive, then the addition of the nonbasic variable x^k to the basis will improve the objective function value. This net change can be represented as $\lambda\bar{c}^k$ where $\bar{c}^k = c^k - c_B B^{-1}A^k$ is the reduced cost of the k-th nonbasic variable. In the context of the column generation technique, the reduced cost associated with the nonbasic variable s can be expressed as follows:

$$c^\top s - u(A''s) - \alpha = (c^\top - uA'')s - \alpha,$$

where α represents the dual multiplier on the convexity constraint (1.3b), and u is a vector of dual multipliers associated with the constraint (1.3a). Based on this information, the column generation

subproblem can be defined as follows:

$$z_{\mathrm{DW}}(u) = -\alpha + \max_{s \in \varepsilon}\{(c^\top - uA'')s\}.$$

Dantzig-Wolfe decomposition can be interpreted as solving a multiobjective optimization problem with a weighted objective function. In this problem, there are two sets of objectives: the original objective c with coefficient 1 and the objectives that are the rows of the constraint matrix A'' with coefficient $-u$. In addition to the column generation technique for adding the new column or variable, we can utilize the proposed RVF algorithm, which is designed to construct the EF for multiobjective problems.

1.2.2 Lagrangian Relaxation

The Lagrangian relaxation of the mixed integer linear programming problem in equation (MILP-D) can be defined as follows, under the assumption that optimizing over the set $\mathcal{S}_R$ is computationally tractable. For a Lagrange multiplier $u \geq 0$, the Lagrangian function is given by:

$$z_{\mathrm{LR}}(u) = \max_{x \in \mathcal{S}_R}\{c^\top x - u(A''x - b'')\} = ub'' + \max_{x \in \mathcal{S}_R}\{(c^\top - uA'')x\}.$$

In order to obtain a tighter upper bound on the optimal value of the original MILP in (MILP-D), it is desirable to find the value of $u \geq 0$ that yields the lowest upper bound when used in the Lagrangian relaxation. We call this problem the Lagrangian dual (LD). Hence we have

$$z_{\mathrm{LD}} = \min_{u \geq 0} z_{\mathrm{LR}}(u).$$

Thus, the LD of the MILP in (MILP-D) can be written as follows:

$$z_{\mathrm{LD}} = \min_{x \in \mathcal{S}_R, u \geq 0, \alpha}\{\alpha + ub'' : \ \alpha \geq (c^\top - uA'')x\}. \tag{LD}$$

It can be shown that the LD, as defined above, is equivalent to the LP dual of (DW-LP2). Hence z_{DW} and z_{LD} are equal at optimality. This equivalence suggests that solving LD using a cutting-plane algorithm is equivalent to solving the DWLP via column generation.

For a fixed value of the Lagrange multiplier $u \geq 0$, $z_{\mathrm{LR}}(u)$ can be interpreted as a multiobjective optimization problem with two objectives: the original objective function c, with weight 1, and the constraint A'', with weight $-u$.

The relationship between the Lagrangian relaxation algorithm and the RVF algorithm can be formalized as follows.

Proposition 1.2.1. *If an optimal solution $\hat{x}$ to the MILP in (MILP-D) exists, it can be shown that there exists a value of the Lagrange multiplier $\hat{u}$ in the (LD) such that $\hat{x}$ corresponds to one of the NDPs of the maximization multiobjective problem defined by the original objective function c and the constraint A'', with weights 1 and $-\hat{u}$, respectively.*

1.2.3 Cutting-Plane Method

In this section, we describe the relationship between the value function and the cutting-plane method, a solution approach for MILP that involves iteratively improving the upper bound on the optimal value of the original problem by adding valid inequalities to the relaxed problem. The relationship between the value function and the cutting-plane method is not as straightforward as the relationship between the value function and the Dantzig-Wolfe decomposition or the Lagrangian relaxation, as discussed in Subsections 1.2.1 and 1.2.2. First, we define the term valid inequality.

Definition 1.2.2 (Valid Inequality)**.** *The inequality denoted by (π, π_0) is valid for a polyhedron Q if $\pi x \leq \pi_0 \ \forall x \in Q$.*

Hence a valid inequality (π, π_0) is valid if and only if

$$\pi_0 \geq \max_{x \in Q} \pi x. \tag{1.5}$$

It is clear that solving the RHS of equation (1.5), which involves finding the maximum value of the objective function over the set Q, is as complex as solving the original MILP. This presents a challenge in using valid inequalities to improve the upper bound on the optimal value of the original problem, as it requires solving a problem that is equally difficult. One way to address this issue is to find a dual bound on the RHS of equation (1.5), which can be used in place of the actual value of the objective function. Hence, the RHS of any valid inequality can be thought of as arising from a

value function, which yields a tight dual bound. Therefore, we can have $\pi_0 \geq F(b)$, where F is a dual function with respect to the optimization problem $\max_{x \in Q} \pi x$.

For more details on decomposition-based algorithms and their application to MILPs, we refer the interested reader to the work of [Galati, 2009].

1.2.4 Two-stage Stochastic Linear Optimization

In two-stage stochastic linear optimization, the decision-making process is divided into two epochs, separated by a probabilistic event. In the first stage, we must make decisions before the realization of the uncertainty, and these decisions are referred to as the *first-stage decisions*. After the realization of the uncertainty becomes available, the *second-stage decisions* are made by solving an appropriate optimization problem. The general formulation of a two-stage stochastic linear optimization can be written as follows, which is presented in detail in [Hassanzadeh, 2015].

$$\min_{x \in S_1} c^\top x + \Xi(x),$$

where $S_1 = \{x = (x_I, x_C) \in \mathbb{Z}_+^{r_1} \times \mathbb{R}_+^{n_1 - r_1} : Ax = b\}$ is the first-stage feasible region; $A \in \mathbb{Q}^{m_1 \times n_1}$ is the matrix of coefficients of the constraints; $b \in \mathbb{Q}^{m_1}$ is the RHS of the constraints. The objective function of the first-stage formulation consists of two terms. The first term $c^\top x$ is the immediate cost of the first-stage solution, and the additional cost $\Xi(x)$ is incurred due to the uncertainty. Conventionally, the function Ξ is defined as

$$\Xi(x) = \mathbb{E}_{\omega \in \Omega}[z(h_\omega - T_\omega x)],$$

where $T_\omega \in \mathbb{Q}^{m_2 \times n_1}$ and $h_\omega \in \mathbb{Q}^{m_2}$. The function z is referred to as a second-stage value function. We rewrite the value function (RVF) for the case of the stochastic problem as follows.

$$z(b) = \inf\{q^\top y : y \in S_2(b)\},$$

where $S_2(b) = \{y = (y_I, y_C) \in \mathbb{Z}_+^{r_2} \times \mathbb{R}_+^{n_2 - r_2} : Wy = b\}$; $W \in \mathbb{Q}^{m_2 \times n_2}$ is the matrix of coefficients of the constraints; $q \in \mathbb{Q}^{n_2}$ is the second-stage objective function. Generally, we may have stochasticity in the constraint matrix W and the objective vector q. In this case, there are $|\Omega|$ second-stage value

functions.

1.2.5 Mixed Integer Bilevel Linear Optimization Problems

The value function can also arise in mixed integer bilevel linear optimization problems (MIBLP). The general formulation of an MIBLP can be written as follows:

$$\min_{x \in S_1} c^\top x + \Xi(x), \tag{MIBLP}$$

where

$$\Xi(x) = \min \left\{ d^1 y \mid y \in \mathcal{P}_1(x), y \in \operatorname{argmin} \left\{ d^2 y \mid y \in \mathcal{P}_2(x) \cap Y \right\} \right\},$$

$$\mathcal{P}_1(x) = \left\{ y \in \mathbb{R}^{n_2}_+ \mid G^1 y \geq b^1 - A^1 x \right\},$$

$$\mathcal{P}_2(x) = \left\{ y \in \mathbb{R}^{n_2}_+ \mid G^2 y \geq b^2 - A^2 x \right\}.$$

Here, all matrices and vectors are of compatible dimensions. One can show that there exists an alternative formulation that uses the value function of the second-stage problem, as follows:

$$\min \left\{ cx + d^1 y \mid x \in X, y \in \mathcal{P}_1(x) \cap \mathcal{P}_2(x) \cap Y, d^2 y \leq \phi\left(b^2 - A^2 x\right) \right\},$$

where

$$\phi(\beta) = \min \left\{ d^2 y \mid G^2 y \geq \beta, y \in Y \right\} \quad \forall \beta \in \mathbb{R}^{m_2},$$

defines the value function of the second-stage problem. This value function is expressed in the context of bilevel optimization problems. For more details on MIBLPs and their applications, we refer the interested reader to Tahernejad [2019].

1.3 Contributions

In this dissertation, we investigate the relationship between the RVF and the EF of an MILP, which, to the best of our knowledge, has not been previously studied in the literature.

Building upon this relationship, we introduce two distinct algorithms for constructing both the

EF and the value function of an MILP. Firstly, we propose the RVF Algorithm, a cutting-plane approach for this purpose. Secondly, we present a unified branch-and-bound approach within the SYMPHONY framework to construct either the RVF or the EF of an MILP.

Our contributions can be summarized as follows:

- We formalize the relationship between the RVF and the EF of an MILP.
- We propose a cutting-plane algorithm called the RVF Algorithm for constructing both the EF and the RVF of an MILP.
- We introduce computational implementations of the RVF algorithm.
- We conduct an extensive computational study using instances from different classes of optimization problems, including the set packing problem (SPP), Knapsack problem (KP), and randomly generated pure integer and mixed integer instances. This study demonstrates the effectiveness of our proposed algorithm in terms of several common metrics when compared to existing algorithms.
- We introduce a callback-based integration into the RVF Algorithm that mitigates computational issues—particularly numerical instability and solver limitations—encountered in MILP instances.
- We propose a unified branch-and-bound framework to construct both the EF and the RVF of an MILP using SYMPHONY.
- We perform an extensive computational experiment to fine-tune the proposed algorithm and compare it to the best-performing RVF Algorithm using instances from the SPP, KP, and randomly generated pure and mixed integer instances.

1.4 Outline of Thesis

The remainder of this dissertation is structured as follows. In Chapter 2, we delve into the relationship between the RVF and the EF. Within Section 2.1, we outline fundamental properties of the RVF. Subsequently, in Section 2.2, we formally establish the relationship between the RVF and the EF. Finally, in Section 2.3, we conclude this chapter.

In Chapter 3, we begin with the naive version of our cutting-plane algorithm and describe the problem solved at each iteration. We then explore several variants of the RVF Algorithm, obtained by adjusting key parameters, and evaluate their performance through a detailed computational study. Section 3.1 introduces the cutting-plane algorithm for constructing both the EF and the value function, showing that both are finite under the assumption that X_{MO} is bounded. Section 3.2 provides a comparison between the RVF Algorithm and other existing algorithms in the literature. Section 3.3 presents the alternative implementations we propose. The data sets used in the computational experiments are described in Section 3.4, with their outcomes analyzed in Section 3.5. Section 3.6 outlines how the RVF Algorithm can be applied in practice, including software dependencies and data input requirements. Finally, Section 3.7 concludes the chapter by summarizing the findings.

In Chapter 4, we introduce a unified branch-and-bound framework for constructing the EF and the RVF of an MILP using the open-source framework SYMPHONY. Section 4.1 provides background information, reviews related work, and outlines the objectives of the chapter. Section 4.2 develops the theoretical foundations, establishing how branch-and-bound trees define valid lower and upper approximations of the RVF. Section 4.3 introduces the concept of tree-based warm-starting and explains how a single branch-and-bound tree can be iteratively refined for different right-hand sides. The algorithmic framework is presented in Section 4.4, where we describe strategies for selecting right-hand sides and ensuring convergence. Section 4.5 discusses the practical implementation of the proposed methods within SYMPHONY. In Section 4.6, we present computational experiments evaluating the performance of the proposed approach and compare it with the RVF Algorithm introduced in Chapters 2 and 3. Section 4.7 provides details of the software developed for this work, including usage and reproducibility. Finally, Section 4.8 concludes the chapter by summarizing the main findings.

Finally, Chapter 5 summarizes the main contributions of the dissertation and discusses directions for future research.

Chapter 2

The RVF and the EF of an MILP

In this chapter, we discuss the structure of the restricted value function (RVF) and its relationship with the efficient frontier (EF). Specifically, in Section 2.1, we characterize the RVF in terms of a discrete set of integer parts of nondominated points (NDP). We also examine the structure of the Restricted LP Value Function, which forms the foundation for understanding the Restricted Value Function. These structures are analyzed in detail, followed by an examination of the relationship between the RVF and the EF in Section 2.2. Building on this theoretical foundation, the next chapter presents the design of the RVF Algorithm and introduces its different implementations. Finally, in Section 2.3, we provide a summary and concluding remarks.

2.1 The Restricted Value Function

Before presenting the main result detailing the relationship between the RVF and the EF in Section 2.2, we first describe some basic properties of the RVF. In doing this, we aim to demonstrate that certain properties of the EF arise more naturally by considering the structure of the RVF. For example, the gradients and subdifferentials of the RVF provide a means to understand the trade-offs between objectives in a local region of the EF and we show that these can be fully characterized by considering the dual feasible region of the so-called *continuous restriction*, whose own value function is the so-called *restricted LP value function* (RLPVF), described next in Section 2.1.1. In Section 2.1.2, we show that the RVF can be described as the minimum of a set of translations of the

RLPVF and characterize its gradients and subdifferentials.

2.1.1 Structure of the Restricted LP Value Function

The RLPVF is the special case of the RVF when there are no integer variables ($r = 0$). We define the RLPVF $z_{\text{LP}} : \mathbb{R}^\ell \to \mathbb{R} \cup \{\pm\infty\}$ by

$$z_{\text{LP}}(\zeta) = \inf \; c_C^0 x_C \tag{RLPVF-a}$$
$$\text{s.t.} \; C_C^{1:\ell} x_C \leq \zeta \tag{RLPVF-b}$$
$$A_C x_C = \beta \tag{RLPVF-c}$$
$$x_C \in \mathbb{R}_+^{n-r}, \tag{RLPVF-d}$$

for all $\zeta \in \mathbb{R}^\ell$. As with the RVF, we define z_{LP} to take values in the extended reals rather than restricting it to its finite domain, which affects the analysis in significant ways that we discuss below. In this section, we consider $\beta \in \mathbb{Q}^m$ to be a fixed vector throughout, while in Section 2.1.2, we consider a parametric class of functions of this form, with different values of β arising from fixing the integer part of the solution in (RVF). As previously, we assume that z_{LP} is bounded from below.

The structure of the classical LP value function, where the entire right-hand side is parametric ($m = 0$), is well-studied. It is well established that in this special case, the function z_{LP} is a polyhedral function whose epigraph is a polyhedral cone described by facets associated with the extremal elements of the dual feasible region $\mathcal{P}_D$, defined below. In the more general case of the RLPVF, the function is instead a *slice* of this full LP VF, and its epigraph is hence the intersection of a hyperplane with a polyhedral cone.

In order to analyze the structure of this function, we consider the dual of the LP that arises in the evaluation of $z_{\text{LP}}(\hat{\zeta})$ for $\hat{\zeta} \in \mathbb{R}^\ell$, which is

$$\sup_{(u,v)\in\mathcal{P}_D} \hat{\zeta}^\top u + \beta^\top v, \tag{D-RLP}$$

where u is the vector of dual variables associated with the parametric constraints (RLPVF-b), v is the vector of dual variables associated with the nonparametric constraints (RLPVF-c), and the

feasible region is

$$\mathcal{P}_D = \left\{(u, v) \in \mathbb{R}^{\ell}_{-} \times \mathbb{R}^m \ : \ {C_C^{1:\ell}}^\top u + A_C^\top v \leq c_C^0\right\}.$$

By assumption, $\mathcal{P}_D$ is nonempty, since z_{LP} is bounded below. Note that the feasible region $\mathcal{P}_D$ of (D-RLP) does not depend on ζ.

Next, let $\mathcal{E}$ and $\mathcal{R}$ be the sets of extreme points and extreme rays of $\mathcal{P}_D$, respectively. Recall that $\mathcal{R}$ represents the set of extreme elements of the recession cone

$$\left\{(e, h) \in \mathbb{R}^{\ell} \times \mathbb{R}^m \ : \ {C_C^{1:\ell}}^\top e + A_C^\top h \leq 0\right\}.$$

By Farkas' lemma [Farkas, 1902], $z_{\text{LP}}(\zeta)$ is finite (the problem (D-RLP) has an optimal solution) if and only if $\zeta \in \mathcal{C}_{\text{LP}}$, where

$$\mathcal{C}_{\text{LP}} = \left\{\zeta \in \mathbb{R}^{\ell} \ : \ \zeta^\top e + \beta^\top h \leq 0 \ \ \forall (e, h) \in \mathcal{R}\right\}.$$

Otherwise, we have $z_{\text{LP}}(\zeta) = +\infty$. Note that $\mathcal{C}_{\text{LP}}$ is a polyhedron.

Putting all of this together, we have the following proposition characterizing z_{LP} and yielding a finite combinatorial description.

Proposition 2.1.1. *z_{LP} is a polyhedral function over $\mathcal{C}_{\text{LP}}$ and we have that*

$$z_{\text{LP}}(\zeta) := \max_{(u,v)\in\mathcal{P}_D} (\zeta^\top u + \beta^\top v) = \max_{(u,v)\in\mathcal{E}} (\zeta^\top u + \beta^\top v), \qquad \forall \zeta \in \mathcal{C}_{\text{LP}}. \tag{D-RLPVF}$$

Proof. By (D-RLPVF), the epigraph of RLPVF over $\mathcal{C}_{\text{LP}}$ is

$$\left\{(\zeta, w) \in \mathcal{C}_{\text{LP}} \times \mathbb{R} \ : \ \zeta^\top u + \beta^\top v \leq w, \ \forall (u, v) \in \mathcal{E}\right\},$$

which is a polyhedron ($\mathcal{E}$ must be finite, since it is the set of extreme points of a polyhedron). Therefore, z_{LP} is a polyhedral function over $\mathcal{C}_{\text{LP}}$. The characterization (D-RLPVF) follows from strong duality and the well-known fact that when an LP has a finite optimum, that optimum is achieved at an extreme point. ■

For an illustration, see Appendix A, where we detail the RLPVF obtained by setting $(x_1, x_2) = (0, 0)$ in Example 1. Since all polyhedral functions are convex, it follows that z_{LP} is convex over $\mathcal{C}_{\text{LP}}$.

In the remainder of this section, we characterize the directional derivatives and the subdifferentials of the RLPVF. For the classical VF associated with an LP, it is well-known that the subdifferential at a given right-hand side vector within the finite domain of the VF is the set of all optimal solutions to the dual of the LP associated with that given right-hand side. A similar result can be obtained for the RLPVF by projecting this set of optimal dual solutions onto the subspace of the dual variables associated with the parametric constraints (RLPVF-b). The projection will also be useful in characterizing the directional derivatives.

We start by describing the face of all optimal solutions of the LP dual (D-RLP) for a given $\zeta \in \mathcal{C}_{\text{LP}}$. By adding an optimality constraint to the constraints of $\mathcal{P}_D$, we get that the set of optimal solutions with respect to the objective vector (ζ, β) is the face

$$OPT(\zeta) = \{(u, v) \in \mathbb{R}^{\ell}_{-} \times \mathbb{R}^{m} \ : \ {C_C^{1:\ell}}^{\top} u + A_C^{\top} v \leq c_C^0, \ \zeta^{\top} u + \beta^{\top} v = z_{\text{LP}}(\zeta)\} \tag{2.1}$$

of $\mathcal{P}_D$ (itself a polyhedron). It is well-known how to project a polyhedron into a subspace (see, e.g., Theorem 3.46 of Conforti et al. [2014]). Applying this procedure, we get

$$\begin{aligned} \text{proj}_u(\text{OPT}(\zeta)) &= \{u \in \mathbb{R}^{\ell}_{-} \ : \ \exists v \in \mathbb{R}^m \text{ s.t. } A_C^{\top} v \leq c_C^0 - {C_C^{1:\ell}}^{\top} u, \ \beta^{\top} v = z_{\text{LP}}(\zeta) - \zeta^{\top} u\} \\ &= \{u \in \mathbb{R}^{\ell}_{-} \ : \ (C_C^{1:\ell} r + s\zeta)^{\top} u \leq r^{\top} c_C^0 + s z_{\text{LP}}(\zeta), \ \forall (r, s) \in \mathcal{Q}\}, \end{aligned} \tag{2.2}$$

where $\mathcal{Q}$ are the extreme rays of $\{(x_C, t) \in \mathbb{R}^{n-r}_{+} \times \mathbb{R} \ : \ A_C x_C + t\beta = 0\}$.

Note that for $(\hat{x}_C, \hat{t})$ with $\hat{t} < 0$, we can scale the resulting ray so without loss of generality, $\hat{t} = -1$ and then $A_C \hat{x}_C = \beta$, so that $\hat{x}_C$ is an extreme point of $\{x_C \in \mathbb{R}^{n-r}_{+} \ : \ A_C x_C = \beta\}$, the feasible region of the relaxation obtained by relaxing (RLPVF-b). Then we can rewrite the condition for u to be in the projection as

$$(c_C^0 - {C_C^{1:\ell}}^{\top} u)^{\top} r \geq z_{\text{LP}}(\zeta) - \zeta^{\top} u, \tag{2.3}$$

which has an interpretation in terms of the reduced costs associated with u. Note that the above implies that if we precompute the set $\mathcal{Q}$, we can easily obtain a description of $\text{proj}_u(\text{OPT}(\zeta))$ for any $\zeta \in \mathcal{R}^{\ell}$.

We now state the main result of this section, which is that the set $\text{proj}_u(\text{OPT}(\zeta))$ is precisely the subdifferential of the RLPVF at $\zeta \in \mathcal{C}_{\text{LP}}$. We apply the standard definition for the subdifferential of

a convex function [Rockafellar, 1997], which states that the subdifferential of z_{LP} at $\hat{\zeta} \in \mathcal{C}_{\text{LP}}$ is[1]

$$\partial z_{\text{LP}}(\hat{\zeta}) = \left\{ g \in \mathbb{R}^{\ell} \ : \ z_{\text{LP}}(\zeta) \geq g^{\top}(\zeta - \hat{\zeta}) + z_{\text{LP}}(\hat{\zeta}), \ \forall \zeta \in \mathcal{C}_{\text{LP}} \right\}.$$

Proposition 2.1.2. *For all $\zeta \in \mathcal{C}_{\text{LP}}$,*

$$\partial z_{\text{LP}}(\zeta) = \text{proj}_u(\text{OPT}(\zeta)).$$

The proof relies on Lemmas C.0.2, C.0.3, and C.0.4, all stated and proven in Appendix C. A consequence of this result is that z_{LP} is differentiable at points in the interior of its domain at which the dual LP has a unique solution. We formalize this in Corollary C.0.5 in Appendix C.

We now examine the properties of the directional derivatives to highlight how the function behaves at the boundaries of the finite domain $\mathcal{C}_{\text{LP}}$. From the definition of $\mathcal{C}_{\text{LP}}$, the following characterization of its interior and boundary can be easily derived.

- $\zeta \in \text{int}\, \mathcal{C}_{\text{LP}}$ if and only if $\zeta^{\top} e + \beta^{\top} h < 0 \quad \forall (e, h) \in \mathcal{R}$.
- $\zeta \in \mathcal{C}_{\text{LP}}$ is on the boundary of $\mathcal{C}_{\text{LP}}$ if and only if $\zeta^{\top} e + \beta^{\top} h \leq 0 \quad \forall (e, h) \in \mathcal{R}$ and there exists $(\hat{e}, \hat{h}) \in \mathcal{R}$ such that $\zeta^{\top} \hat{e} + \beta^{\top} \hat{h} = 0$.
- $\zeta \notin \mathcal{C}_{\text{LP}}$ if and only if there exists $(\hat{e}, \hat{h}) \in \mathcal{R}$ such that $\zeta^{\top} \hat{e} + \beta^{\top} \hat{h} > 0$.

Now consider a given $\hat{\zeta}$ on the boundary of $\mathcal{C}_{\text{LP}}$. By the characterization above, there exists $(\hat{e}, \hat{h}) \in \mathcal{R}$ such that $\hat{\zeta}^{\top} \hat{e} + \beta^{\top} \hat{h} = 0$. As an aside, it is interesting to observe that this means that specifically for points on the boundary of $\mathcal{C}_{\text{LP}}$, we have $z_{\text{LP}}(\hat{\zeta}) < +\infty$ (there is a finite optimal value), while the set of optimal solutions to the LP (D-RLP) is unbounded. This is because the rays of the optimal face have an objective value of zero.

Let $d \in \mathbb{R}^{\ell}$ be such that $d^{\top} \hat{e} > 0$. Then we have that $\hat{\zeta} + \epsilon d \notin \mathcal{C}_{\text{LP}}$ for all $\epsilon > 0$, since $(\hat{\zeta} + \epsilon d)^{\top} \hat{e} + \beta^{\top} \hat{h} > 0$. Thus, we can interpret d as a direction pointing out of $\mathcal{C}_{\text{LP}}$. Per the above discussion, we define the set

$$\delta^{-}(\zeta) = \text{cone}(\{ d \in \mathbb{R}^{\ell} \ : \ \exists (e, h) \in \mathcal{R} \text{ such that } \zeta^{\top} e + \beta^{\top} h = 0, \ d^{\top} e > 0 \}) \setminus \{\mathbf{0}\},$$

[1] This is obviously equivalent to $\partial z_{\text{LP}}(\hat{\zeta}) = \{ g \in \mathbb{R}^{\ell} \ : \ z_{\text{LP}}(\zeta) \geq g^{\top}(\zeta - \hat{\zeta}) + z_{\text{LP}}(\hat{\zeta}), \ \forall \zeta \in \mathbb{R}^{\ell} \}$, since $z_{\text{LP}}(\zeta) = +\infty$ for $\zeta \notin \mathcal{C}_{\text{LP}}$.

to be the set of all directions pointing out of $\mathcal{C}_{\text{LP}}$ at $\zeta \in \mathcal{C}_{\text{LP}}$. Note that with this definition, we have $\delta^-(\zeta) = \emptyset$ for $\zeta \in \operatorname{int} \mathcal{C}_{\text{LP}}$, as expected.

In what follows, we consider several results characterizing the directional derivatives of both the RLPVF and the RVF. For general $f : \mathbb{R}^n \to \mathbb{R}$, we take the directional derivative of f at $\bar{x}$ in direction d to be

$$\nabla_d f(\bar{x}) = \lim_{t \searrow 0} \frac{f(\bar{x} + td) - f(\bar{x})}{t}.$$

For both the RLPVF and the RVF considered in the next section, this limit may go to $+\infty$ at points of discontinuity, and we take the directional derivative to have the value $+\infty$ in such cases. For z_{LP}, we have continuity over the finite domain $\mathcal{C}_{\text{LP}}$, but discontinuities at points on the boundary of $\mathcal{C}_{\text{LP}}$, since we define z_{LP} over the extended reals. Then the directional derivative $\nabla_d z_{\text{LP}}(\zeta)$ of z_{LP} at $\zeta \in \mathcal{C}_{\text{LP}}$ in direction d is finite if and only if $d \notin \delta^-(\zeta)$. For $\zeta \in \operatorname{int} \mathcal{C}_{\text{LP}}$, $\delta^-(\zeta) = \emptyset$ and the directional derivative is finite in all directions.

From the properties of convex functions and subdifferentials, we can alternatively characterize the directional derivative as

$$\nabla_d z_{\text{LP}}(\zeta) = \max_{u \in \partial z_{\text{LP}}(\zeta)} u^\top d,$$

where $\partial z_{\text{LP}}(\zeta)$ denotes the subdifferential of z_{LP} at $\zeta \in \mathbb{R}^\ell$. As a result, we have that for $\zeta \in \mathcal{C}_{\text{LP}}$, we have that

$$\nabla_d z_{\text{LP}}(\zeta) = \begin{cases} \max_{u \in \operatorname{proj}_u(\text{OPT}(\zeta))} u^\top d, & \text{if } d \notin \delta^-(\zeta), \\ +\infty, & \text{otherwise,} \end{cases}$$

for all $d \in \mathbb{R}^\ell$. Proposition C.0.1 in Appendix C formally establishes this from first principles, though it can also be seen as a corollary of our more general characterization of the subdifferentials of the RLPVF, which is the main result of this section.

2.1.2 Structure of the Restricted Value Function

We now characterize the structure of the RVF by observing that the RVF is the minimum of a finite number of translations of functions of the form (RLPVF) for different values of the (previously) non-parametric RHS β. Each of these translated functions defines a *stability region* over which the integer part of all solutions defining points on the graph of the associated RLPVF is fixed.

To further develop our characterization of the VF, we define the following sets of integer parts of solutions by projecting $\mathcal{S}(\zeta)$ onto the space of the integer variables:

$$\mathcal{S}_I(\zeta) = \operatorname{proj}_I \mathcal{S}(\zeta) = \left\{x_I \in \mathbb{Z}^r_+ \ : \ (x_I, x_C) \in \mathcal{S}(\zeta)\right\}, \text{ and}$$

$$\mathcal{S}_I = \bigcup_{\zeta \in \mathcal{C}} \mathcal{S}_I(\zeta).$$

Thus, $\mathcal{S}_I$ is the set of all integer parts of points in $\mathcal{S}(\zeta)$ for some $\zeta \in \mathcal{C}$. For a given $\hat{x}_I \in \mathcal{S}_I$, the *continuous restriction* (CR) with respect to $\hat{x}_I$ is the function[2]

$$\bar{z}(\zeta;\ \hat{x}_I) = c^0_I \hat{x}_I + z_{\text{LP}}(\zeta - C^{1:\ell}_I \hat{x}_I;\ b - A_I \hat{x}_I) \quad \forall \zeta \in \mathcal{C}, \tag{CR}$$

which is defined in terms of the previously defined function z_{LP}, but with an additional secondary parameter. In particular, $z_{\text{LP}}(\cdot;\ \beta)$ is similar to the previously defined z_{LP} except that we now allow for a parametric family of such functions with different vectors for the non-parametric RHS β in (RLPVF). The form shown above is precisely a translation of a function of the form (RLPVF) for $\beta = b - A_I \hat{x}_I$.

This motivates the following formal definition of bounding functions for the RVF, which will be justified in Proposition 2.1.4.

Definition 2.1.3. *A function $\bar{z} : \mathcal{C} \to \mathbb{R} \cup \{+\infty\}$ is called an* upper bounding function *(or* primal bounding function*) for the restricted value function z if*

$$z(\zeta) \leq \bar{z}(\zeta), \qquad \forall \zeta \in \mathcal{C}.$$

Similarly, a function $\underline{z} : \mathcal{C} \to \mathbb{R} \cup \{-\infty\}$ is called a lower bounding function *(or* dual bounding function*) for z if*

$$\underline{z}(\zeta) \leq z(\zeta), \qquad \forall \zeta \in \mathcal{C}.$$

These bounding functions play a central role in the algorithms developed in Chapters 3 and 4. In particular, the RVF Algorithm constructs upper (primal) bounding functions, while the branch-and-bound framework derives lower (dual) bounding functions and combines them with primal

[2] In the multiobjective optimization literature, the corresponding multiobjective LP is sometimes referred to as a *slice problem* [Belotti et al., 2013].

evaluations to assess convergence of the RVF approximation.

In the remainder of the study, we refer to functions $\bar{z}(\cdot;\ x_I)$ for $x_I \in \mathcal{S}_I$ as *bounding functions*, since they bound the RVF from above, as demonstrated in Proposition 2.1.4.

Proposition 2.1.4. *For any $\hat{x}_I \in \mathcal{S}_I$, $\bar{z}(\cdot;\ \hat{x}_I)$ bounds z from above.*

Proof. For $\hat{x}_I \in \mathcal{S}_I$, we have

$$\begin{aligned}\bar{z}(\zeta;\ \hat{x}_I) = c_I^0 \hat{x}_I + \quad & z_{\mathrm{LP}}(\zeta - C_I^{1:\ell}\hat{x}_I;\ b - A_I\hat{x}_I) \geq \\ & \min_{x_I \in \mathcal{S}_I} \left(c_I^0 x_I + z_{\mathrm{LP}}(\zeta - C_I^{1:\ell} x_I;\ b - A_I x_I)\right) = z(\zeta), \quad \forall \zeta \in \mathcal{C}.\end{aligned}$$

■

Proposition 2.1.4 shows that any collection of points from $\mathcal{S}_I$ yields an upper approximation of z simply by taking the minimum of the associated set of bounding functions, previously defined as $\bar{z}(\cdot;\ x_I)$ for $x_I \in \mathcal{S}_I$. The algorithm described in Section 3.1 constructs a subset of $\mathcal{S}_I$ that fully describes the RVF by iteratively approximating it from above.

We first focus on characterizing the RVF as the minimum of a set of bounding functions, as defined in Proposition 2.1.4. When X_{MO} is bounded, $\mathcal{S}_I$ is finite, so the number of such functions required is finite. As such, the main result of this section is the following discrete characterization.

Theorem 2.1.5. *Let $\mathcal{S}$ be any subset of $\mathcal{S}_I$ with the property that for any $\zeta \in \mathcal{C}$, there exist $x_I \in \mathcal{S}$ and $x_C \in \mathbb{R}_+^{n-r}$ such that $C_I^{1:\ell} x_I + C_C^{1:\ell} x_C \leq \zeta$, $A_I x_I + A_C x_C = b$, and $c_I^0 x_I + c_C^0 x_C = z(\zeta)$. Then for any $\zeta \in \mathcal{C}$ we have*

$$z(\zeta) = \min_{x_I \in \mathcal{S}_I} \bar{z}(\zeta;\ x_I) = \min_{x_I \in \mathcal{S}} \bar{z}(\zeta;\ x_I).$$

Proof. Let $\zeta \in \mathcal{C}$. Since $\mathcal{S} \subseteq \mathcal{S}_I$, then by Proposition 2.1.4, we have that

$$z(\zeta) \leq \min_{x_I \in \mathcal{S}} \bar{z}(\zeta;\ x_I).$$

Now by definition of $\mathcal{S}$, there exists $\hat{x}_I \in \mathcal{S}$ and $\hat{x}_C \in \mathbb{R}_+^{n-r}$ such that $C_I^{1:\ell}\hat{x}_I + C_C^{1:\ell}\hat{x}_C \leq \zeta$, $A_I\hat{x}_I + A_C\hat{x}_C = b$, and $c_I^0\hat{x}_I + c_C^0\hat{x}_C = z(\zeta)$. So

$$\min_{x_I \in \mathcal{S}} \bar{z}(\zeta;\ x_I) = \min_{x_I \in \mathcal{S}} \left(c_I^0 x_I + z_{\mathrm{LP}}(\zeta - C_I^{1:\ell} x_I;\ b - A_I x_I)\right)$$

$$\leq c_I^0 \hat{x}_I + z_{\text{LP}}(\zeta - C_I^{1:\ell} \hat{x}_I;\ b - A_I \hat{x}_I)$$
$$\leq c_I^0 \hat{x}_I + c_C^0 \hat{x}_C = z(\zeta),$$

where the first equation follows from (CR), the subsequent inequality follows since $\hat{x}_I \in \mathcal{S}$, the next since, by its definition, $\hat{x}_C$ is feasible for $z_{\text{LP}}(\zeta - C_I^{1:\ell} \hat{x}_I;\ b - A_I \hat{x}_I)$, and the final equation follows from the definition of $\hat{x}_I$ and $\hat{x}_C$. We thus have both that $z(\zeta)$ is no greater and no less than $\min_{x_I \in \mathcal{S}} \bar{z}(\zeta;\ x_I)$, and the result follows. ■

We call a set $\mathcal{S}$ with the properties described in the above theorem a *description* of the RVF. Of course, what constitutes a full "description" of this function is largely a philosophical question. Given the complexity of the function itself, there is obviously a trade-off between the time needed to construct a given description and the ease with which information about the function can be extracted. Determining the "best" description is itself a multiobjective problem. The description proposed here allows for z to be evaluated in time polynomial in the size of the original problem data and $|\mathcal{S}|$. We also show below that with a little additional computation, we can also describe the function's gradients and subdifferential (the latter are polyhedra).

It is clearly desirable to construct a description that is minimal with respect to the properties of Theorem 2.1.5. Although such a minimal description is clearly not necessarily unique, we denote any such minimal description by the notation $\mathcal{S}_{\min}$, since the particular one chosen does not affect our results. Each member $\hat{x}_I \in \mathcal{S}_{\min}$ is associated with its own *stability region*, denoted by $\mathcal{C}(\hat{x}_I)$, and defined to be

$$\mathcal{C}(\hat{x}_I) = \{\zeta \in \mathcal{C}\ :\ z(\zeta) = \bar{z}(\zeta;\ \hat{x}_I)\}.$$

Informally, this is the subset of $\mathcal{C}$ for which the bounding function associated with $\hat{x}_I$ agrees with the RVF. We discuss the properties of stability regions below, but generally, stability regions do not have to be closed or connected, as exemplified in the following example.

Example 2. *The stability regions in Example 1 are depicted in Figure 2.1 for the domain* $\zeta \geq -15$. *Here,* $\mathcal{S}_I = \mathcal{S}_{\min} = \{(0,0),(0,1),(1,0),(1,1)\}$. *The stability regions are*

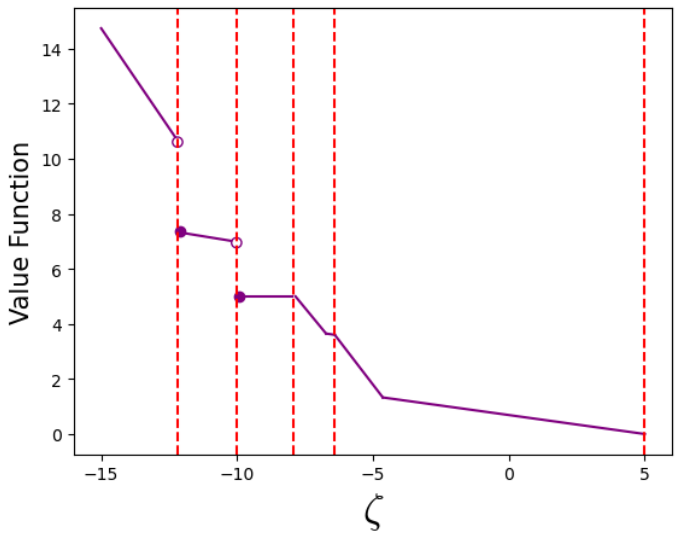


Figure 2.1: Stability regions and corresponding members of $\mathcal{S}_I$ in Example 1, for the region $\zeta \geq -15$, stability regions from left to right correspond to $(x_1, x_2) \in \{(1,0),(1,1),(0,1),(1,0),(0,0)\}$.

$$\mathcal{C}((1,1)) = [-12.167, -10)$$

$$\mathcal{C}((0,1)) = [-10, -7.833]$$

$$\mathcal{C}((1,0)) = [-55.464, -12.167) \cup [-7.833, -6.4]$$

$$\mathcal{C}((0,0)) = [-6.4, +\infty).$$

Note that the stability region for $(1,0)$ consists of two disjoint intervals, one of which is open. The stability region for $(0,0)$ is a single interval, closed at both ends. The stability region for $(0,1)$ is a single closed interval on which the RVF has a zero gradient. Further insights can be obtained by examining the bounding functions associated with each element of $\mathcal{S}_I$, provided in Appendix B.

Even when connected, a stability region may be nonconvex and open, as illustrated in Example 3.

Example 3. *Consider the RVF instance given by*

$$\begin{aligned} z(\zeta) = \min \quad & x_2 \\ \text{s.t.} \quad & x_3 \leq \zeta_1 \\ & x_4 \leq \zeta_2 \\ & 2(1-x_1) \leq x_j \leq 2(1-x_1) + 5x_1, \quad j = 2,3,4 \\ & x_2 + x_3 \geq 5x_1 \\ & x_1 \in \{0,1\} \\ & x_j \leq 5, \quad x_j \in \mathbb{R}_+, \quad j = 2,3,4. \end{aligned}$$

As in previous examples, although this is not in the standard form as in (RVF)*, it can easily be made so with the addition of slack and surplus variables. Since these do not change the RVF or stability regions, we keep the instance in its natural form. The VF for this instance can be written explicitly as*

$$z(\zeta) = \begin{cases} 2, & \zeta_1 \in [2,3] \text{ and } \zeta_2 \geq 2, \\ 5 - \zeta_1, & \zeta_1 \leq 5 \text{ and } \zeta_2 < 2 \text{ or } \zeta_1 \in [0,2) \cup (3,5], \\ 0, & \text{otherwise}, \end{cases}$$

for $\zeta = (\zeta_1, \zeta_2) \in \mathcal{C} = \mathbb{R}^2_+$*. The stability region for* $x_I = (x_1) = (1)$ *is*

$$\{\zeta \in \mathbb{R}^2_+ \ : \ \zeta_1 < 2 \text{ or } \zeta_1 \geq 3 \text{ or } \zeta_2 < 2\},$$

which is shown in Figure 2.2, which displays both stability regions for this example. The stability region for $x_I = x_1 = 1$ *is connected but is neither convex nor closed.*

In general, we do not expect there to be a minimal description that is unique; it is possible for multiple integer solutions $x_I^1, x_I^2 \in \mathcal{S}_I$ with $x_I^1 \neq x_I^2$ to satisfy $\bar{z}(\cdot;\ x_I^1) = \bar{z}(\cdot;\ x_I^2)$, e.g., in the trivial case of duplicate variables. Furthermore, for any point $(\zeta, z(\zeta))$ on the graph of the RVF, there may be more than one integer part x_I with this point lying on the RVF for its associated bounding function. In other words, it is possible that stability regions may overlap, even for integer parts in a given minimal description. Such a case occurs in Example 1: points in the purple-colored line segment seen in Figure 2.2 are in the stability regions of two different integer parts, both of which are in $\mathcal{S}_{\min}$. Of course, for each integer part $x_I \in \mathcal{S}_{\min}$, there must exist at least *one* point

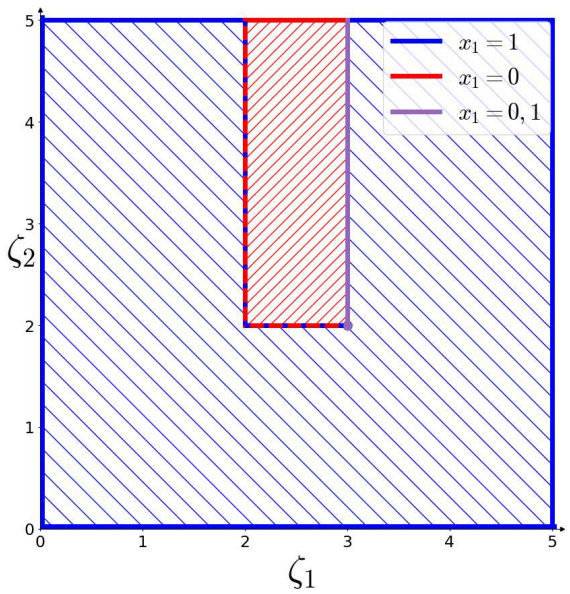


Figure 2.2: Stability regions of the instance in Example 3. The area shaded in red, including the red boundary lines, is not in the stability region for $x_1 = 1$, which is thus neither convex nor closed. The line segment shaded in purple is in the stability region of both $x_1 = 0$ and $x_1 = 1$.

on the boundary of the epigraph of the associated bounding function that is on the boundary of the epigraph of the RVF itself and is not on the boundary of the epigraph of any other bounding function; otherwise, $\mathcal{S}_{\min}$ would not be minimal. All we can say for sure is that elements of $\mathcal{S}_I$ that have an empty stability region will not be present in any minimal description.

We now turn our attention from the structure of stability regions and properties of minimal descriptions to properties of the RVF itself arising from the finite description of Theorem 2.1.5. Figure 1.1a illustrates the RVF of the mixed integer linear program (MILP) described in Example 1. The RVF is piecewise polyhedral, nonincreasing, and lower semi-continuous (here, nonincreasing means that $\zeta_1 \geq \zeta_2,\ \zeta_1, \zeta_2 \in \mathbb{R}^\ell \Rightarrow z(\zeta_1) \leq z(\zeta_2)$). We formalize this in Proposition C.0.6, which can be viewed as a generalization of the previous results by [Nemhauser and Wolsey, 1988] and [Bank et al., 1982] for the full MILP VF. We articulate this within Proposition C.0.6 found in Appendix C.

The RVF may be discontinuous due to the fact that individual bounding functions themselves are discontinuous at the boundaries of their finite domains. When the RVF is discontinuous, the discontinuities occur at (some of) the boundaries between stability regions. At such points, as well as at points on the boundary of $\mathcal{C}$, there may be directions in which the directional derivative of z is infinite, exactly as can occur at points on the boundary of $\mathcal{C}_{\text{LP}}$ in the RLPVF.

Next, we describe the directional derivative of z at a point $\zeta \in \mathcal{C}$ in terms of the faces of optimal solutions to the duals of the LPs associated with the bounding functions in directions for which the directional derivative is finite. To characterize these directions, we need the analogues of the sets of optimal extreme points and extreme rays characterizing the optimal faces of solutions to the LP dual, as we had in the RLPVF case, but with the integer part of the solution fixed. For $x_I \in \mathcal{S}_I$ and $\zeta \in \mathbb{R}^\ell$, we define

$$\text{proj}_u(\text{OPT}(\zeta; x_I)) = \{u \in \mathbb{R}^\ell_- \ : \ \exists v \in \mathbb{R}^m \text{ s.t. } A_C^\top v \leq c_C^0 - {C_C^{1:\ell}}^\top u,\ \beta^\top v = \bar{z}(\zeta;\ x_I) - (\zeta - C_I^{1:\ell} x_I)^\top u\}$$

$$= \{u \in \mathbb{R}^\ell_- \ : \ (C_C^{1:\ell} r + s(\zeta - C_I^{1:\ell} x_I))^\top u \leq r^\top c_C^0 + s\bar{z}(\zeta;\ x_I),\ \forall (r,s) \in \mathcal{Q}\}, \tag{2.4}$$

where $\mathcal{Q}$ is defined as previously to be the extreme rays of $\{(x_C, t) \in \mathbb{R}^{n-r}_+ \times \mathbb{R} \ : \ A_C x_C + t\beta = 0\}$. We further need to know the set of integer vectors associated with the bounding functions that agree with the RVF at a given point, which we denote as

$$\begin{aligned} S_I^*(\zeta) \ &= \left\{x_I \in S_I(\zeta) \ : \ c_I^0 x_I + c_C^0 x_C = z(\zeta),\ \exists x_C \text{ s.t. } (x_I, x_C) \in S(\zeta)\right\} \\ &= \left\{x_I \in S_I(\zeta) \ : \ \bar{z}(\zeta;\ x_I) = z(\zeta)\right\}. \end{aligned}$$

With all of this notation established, let us now consider a point $\hat{\zeta} \in \mathbb{R}^\ell$ at which the RVF is discontinuous. As with the RLPVF, the directional derivative of a bounding function associated with $x_I \in \mathcal{S}_I$ is infinite in direction $d \in \mathbb{R}^\ell$ at $\hat{\zeta}$ if and only if d is contained in the set

$$\begin{aligned} \delta^-(\hat{\zeta};\ x_I) = \text{cone}(\{d \in \mathbb{R}^\ell \ : \ &\exists (e,h) \in \mathcal{R} \text{ such that} \\ &(\hat{\zeta} - C_I^{1:\ell} x_I)^\top e + (b - A_I x_I)^\top h = 0,\ d^\top e > 0\}) \setminus \{\mathbf{0}\}. \end{aligned}$$

Finally, for the directional derivative of z itself to be infinite in direction $d \in \mathbb{R}^\ell$ at $\hat{\zeta} \in \mathbb{R}^\ell$, the derivatives of *all* bounding functions that agree with z at ζ must also be infinite, so we have the following proposition, whose proof appears in Appendix C.

Proposition 2.1.6. *For $\zeta \in \mathcal{C}$ and $d \in \mathbb{R}^\ell$,*

$$\nabla_d z(\zeta) = \min_{x_I \in \mathcal{S}_I^*(\zeta) \cap \mathcal{S}_{\min}} \nabla_d \bar{z}(\zeta;\ x_I)$$

$$= \begin{cases} \min\limits_{x_I \in \mathcal{S}_I^*(\zeta) \cap \mathcal{S}_{\min}} (\max_{u \in \mathrm{proj}_u(\mathrm{OPT}(\zeta;\ x_I))} u^\top d), & \text{if } d \notin \cap_{x_I \in \mathcal{S}_I^*(\zeta) \cap \mathcal{S}_{\min}} \delta^-(\zeta;\ x_I), \\ +\infty, & \text{otherwise,} \end{cases}$$

where $\mathcal{S}_{\min}$ is any minimal description of the RVF.

Even though the RVF is not convex, nor expected to be continuous, it is still possible to define a notion of subdifferential by considering the local structure of the VF at a given point, which is inherited from the bounding functions that are active at the point and necessary to a minimal description of the VF.

Definition 2.1.7 (Local Subdifferential)**.** *Let $f : \mathbb{R}^\ell \to \mathbb{R}$ and $\zeta \in \mathbb{R}^\ell$. Then $q \in \mathbb{R}^\ell$ is a* local subgradient *of f at $\zeta' \in \mathbb{R}^\ell$ if there exists $\epsilon > 0$ such that*

$$f(\zeta) \geq f(\zeta') + q^\top(\zeta - \zeta'), \quad \forall \zeta \in \mathbb{R}^\ell \text{ such that } \|\zeta - \zeta'\| \leq \epsilon. \tag{2.5}$$

The local subdifferential *of f at ζ, denoted $\partial_L f(\zeta)$, is defined as the set of all local subgradients of f at ζ.*

Our definition of a local subgradient is closely related to that of the proximal subgradient given in [Rockafellar and Wets, 2009] (Definition 8.45). The main difference is that we drop the quadratic term $\sigma\|\zeta' - \zeta\|^2$ in the definition of the proximal subgradient, since $\sigma = 0$ is always valid in our setting. In particular, for functions that are piecewise affine or convex, our local subdifferential and the proximal subdifferential are identical.

Finally, we are ready to state the main result of this section, that the intersection of the subdifferentials of these active and necessary bounding functions at a point yields the local subdifferential of the VF at that point. The proof of this result depends crucially on the following observation, stated formally as Lemma C.0.7 in Appendix C. Since (by Proposition C.0.6) z is comprised of a minimum of a finite number of polyhedral functions, each associated with a member of some minimal description $\mathcal{S}_{\min}$, as per Theorem 2.1.5, then z must agree with the upper bounding function associated with *some* member of $\mathcal{S}_{\min}$ in the local neighborhood of any $\zeta \in \mathcal{C}$. The remaining details of the proof appear in Appendix C.

Proposition 2.1.8. *The local subdifferential of z at $\zeta \in \mathcal{C}$ is given by*

$$\partial_L z(\zeta) = \bigcap_{x_I \in S_I^*(\zeta) \cap \mathcal{S}_{\min}} \text{proj}_u(\text{OPT}(\zeta - C_I^{1:\ell} x_I;\ b - A_I x_I)),$$

where $\mathcal{S}_{\min}$ is a minimal description of the RVF.

2.2 The RVF and the EF

We now formalize the relationship between the RVF and the EF. Informally, Theorem 2.2.1 below states that the boundary of the epigraph of the RVF and the EF effectively encode the same information, with the boundary of the epigraph including weak NDPs, while the EF does not.

Theorem 2.2.1. *The EF associated with (MO-MILP) is a (possibly strict) subset of the boundary of the epigraph of the RVF. In particular, the following statements hold.*

1. *If $C_I x_I + C_C x_C$ belongs to the EF for some $(x_I, x_C) \in X_{\text{MO}}$, then $(C_I^{1:\ell} x_I + C_C^{1:\ell} x_C, c_I^0 x_I + c_C^0 x_C)$ is a point on the boundary of the epigraph of z.*
2. *If $(\zeta, z(\zeta))$ is a point on the boundary of the epigraph of z, then there exists an efficient solution $(x_I, x_C) \in X_{\text{MO}}$ such that $c_I^0 x_I + c_C^0 x_C = z(\zeta)$ and $C_I^{1:\ell} x_I + C_C^{1:\ell} x_C \leq \zeta$. Further, $C_I^{1:\ell} x_I + C_C^{1:\ell} x_C = \zeta$ if and only if $\nabla_d z(\zeta) > 0$ for all $d \in \mathbb{R}^\ell_- \setminus \{\mathbf{0}\}$.*

Proof. 1. To prove statement 1, let $(\hat{x}_I, \hat{x}_C) \in X_{\text{MO}}$ be a given efficient solution and let $\hat{\zeta} = C_I^{1:\ell} \hat{x}_I + C_C^{1:\ell} \hat{x}_C$. Now $(\hat{x}_I, \hat{x}_C) \in X_{\text{MO}}$ and $\hat{\zeta} = C_I^{1:\ell} \hat{x}_I + C_C^{1:\ell} \hat{x}_C$ implies, from definitions, that $(\hat{x}_I, \hat{x}_C) \in \mathcal{S}(\hat{\zeta})$. We want to show that $(\hat{\zeta}, c_I^0 \hat{x}_I + c_C^0 \hat{x}_C)$ is a point on the boundary of the epigraph of the RVF z. Since $z(\hat{\zeta}) = \min_{(x_I, x_C) \in \mathcal{S}(\hat{\zeta})} c_I^0 x_I + c_C^0 x_C$, by definition, and since $(\hat{x}_I, \hat{x}_C) \in \mathcal{S}(\hat{\zeta})$, it must be that $z(\hat{\zeta}) \leq c_I^0 \hat{x}_I + c_C^0 \hat{x}_C$. Assume, for the sake of contradiction, that $c_I^0 \hat{x}_I + c_C^0 \hat{x}_C \neq z(\hat{\zeta})$. Then it must be that $c_I^0 \hat{x}_I + c_C^0 \hat{x}_C > z(\hat{\zeta})$. Now, by the definition of $z(\hat{\zeta})$, there must exist $(x_I, x_C) \in \mathcal{S}(\hat{\zeta})$ with $z(\hat{\zeta}) = c_I^0 x_I + c_C^0 x_C$. Furthermore, $(x_I, x_C) \in \mathcal{S}(\hat{\zeta})$ implies $C_I^{1:\ell} x_I + C_C^{1:\ell} x_C \leq \hat{\zeta} = C_I^{1:\ell} \hat{x}_I + C_C^{1:\ell} \hat{x}_C$, while $c_I^0 x_I + c_C^0 x_C = z(\hat{\zeta}) < c_I^0 \hat{x}_I + c_C^0 \hat{x}_C$, so $C_I x_I + C_C x_C \lneqq C_I \hat{x}_I + C_C \hat{x}_C$. This contradicts the hypothesis that $(\hat{x}_I, \hat{x}_C)$ is an efficient solution, and the result follows.

2. To prove the first part of 2, let $\zeta \in \mathcal{C}$ be given, so that $(\zeta, z(\zeta))$ is a point on the boundary of the epigraph of z. Then there exists $(\tilde{x}_I, \tilde{x}_C) \in \mathcal{S}(\zeta)$ such that $c_I^0 \tilde{x}_I + c_C^0 \tilde{x}_C = z(\zeta)$. There are two cases. If $(\tilde{x}_I, \tilde{x}_C)$ is an efficient solution, the result follows trivially. Otherwise, there must exist an efficient solution $(x_I, x_C) \in X_{\text{MO}}$ that dominates $(\tilde{x}_I, \tilde{x}_C)$ at least weakly, i.e., such that $C_I x_I + C_C x_C \leq C_I \tilde{x}_I + C_C \tilde{x}_C$. Then $C_I^{1:\ell} x_I + C_C^{1:\ell} x_C \leq C_I^{1:\ell} \tilde{x}_I + C_C^{1:\ell} \tilde{x}_C \leq \zeta$, which means $(x_I, x_C) \in \mathcal{S}(\zeta)$ and we also have $c_I^0 x_I + c_C^0 x_C \leq c_I^0 \tilde{x}_I + c_C^0 \tilde{x}_C = z(\zeta)$. But by the optimality of $z(\zeta)$, we must also have that $c_I^0 x_I + c_C^0 x_C \geq z(\zeta)$, so we conclude that $c_I^0 x_I + c_C^0 x_C = z(\zeta)$. Since (x_I, x_C) is an efficient solution, the result follows.

   For the second part, there are two directions.

   $\Leftarrow$ Suppose that $\nabla_d z(\zeta) > 0$ for all $d \in \mathbb{R}^\ell_- \setminus \{\mathbf{0}\}$ and assume for the sake of contradiction that $\zeta \gneqq C_I^{1:\ell} x_I + C_C^{1:\ell} x_C$. Let $\tilde{\zeta} = C_I^{1:\ell} x_I + C_C^{1:\ell} x_C$. Now $\mathcal{S}(\tilde{\zeta}) \subseteq \mathcal{S}(\zeta') \subseteq \mathcal{S}(\zeta)$ for all ζ' with $\tilde{\zeta} \leq \zeta' \leq \zeta$. Thus, since (x_I, x_C) minimizes the objective over $\mathcal{S}(\zeta)$ and $(x_I, x_C) \in \mathcal{S}(\tilde{\zeta})$, it must be that $z(\zeta') = c_I^0 x_I + c_C^0 x_C$ for all $\zeta' \in [\tilde{\zeta}, \zeta]$. So $z(\zeta') = z(\zeta)$ for all $\zeta' \in [\tilde{\zeta}, \zeta]$. Recalling that $\zeta \gneqq \tilde{\zeta}$, it must thus be that $\hat{d} = \tilde{\zeta} - \zeta \in \mathbb{R}^\ell_- \setminus \{\mathbf{0}\}$ and $\nabla_{\hat{d}} z(\zeta) = 0$, which is a contradiction to the initial hypothesis. Therefore, $\zeta = C_I^{1:\ell} x_I + C_C^{1:\ell} x_C$.

   $\Rightarrow$ We prove the contrapositive. Therefore, suppose there exists $d \in \mathbb{R}^\ell_- \setminus \{\mathbf{0}\}$ such that $\nabla_d z(\zeta) = 0$. Then, by Proposition C.0.6, there must exist $\tilde{\zeta} \lneqq \zeta$ with $z(\tilde{\zeta}) = z(\zeta)$. By the proof of the first part of 2, above, there must exist $(\tilde{x}_I, \tilde{x}_C)$ an efficient solution with $(\tilde{x}_I, \tilde{x}_C) \in \mathcal{S}(\tilde{\zeta})$ and $c_I^0 \tilde{x}_I + c_C^0 \tilde{x}_C = z(\tilde{\zeta})$. Now the former condition implies $C_I^{1:\ell} \tilde{x}_I + C_C^{1:\ell} \tilde{x}_C \leq \tilde{\zeta} \lneqq \zeta$ while the latter implies that $c_I^0 \tilde{x}_I + c_C^0 \tilde{x}_C = z(\zeta)$, since $z(\tilde{\zeta}) = z(\zeta)$. The result follows.

■

Since the theorem is somewhat technical, we now further explain the intuition of the result. Part 1 is a straightforward statement that every point on the EF is also a point on the boundary of the epigraph of the RVF. The technicalities in Part 2 arise from the aforementioned fact that there are points on the boundary of the epigraph of the RVF that are not in the EF. A point on the boundary of the epigraph that is not contained in the EF corresponds to a weak NDP and can thus be associated with one or more NDPs that weakly dominate it.

For illustration, consider $(\zeta, z_{\text{LP}}(\zeta))$ on the boundary of the epigraph of z. If the function value

strictly increases whenever any component of the argument ζ decreases, then the directional derivative is positive in all negative directions, and this point must correspond to an NDP. For example, in Example 1, $\nabla_d z(-11) > 0$ for $d = -1$ (see Figure 2.1) and thus $(-11, z(-11))$ corresponds to an NDP.

On the other hand, if there is a direction d in which the directional derivative is zero at ζ, then the point $(\zeta, z_{\text{LP}}(\zeta))$ must be a weak NDP, since moving in the direction d from ζ corresponds to strictly improving the value of one or more of the multiple objectives that correspond to the parametric constraints, while the objective of (MILP) remains unchanged. For example, in Example 1, points in the interior of the stability region associated with $(0, 1)$ (e.g., $\zeta = -9$) correspond to weak NDPs since they are dominated by the point $(-10, 5)$.

The conditions involving directional derivatives also have another interpretation that is possibly more intuitive. Recall from the previous section that when $\nabla_d z(\hat{\zeta})$ is finite for $\hat{\zeta} \in \mathbb{R}^\ell$, we have that $\nabla_d z(\hat{\zeta}) = d^\top \hat{u}$ for some optimal solution $\hat{u} \in \mathcal{P}_D$ to (D-RLP). This allows us to re-interpret the above conditions involving directional derivatives in terms of solutions to (D-RLP). In particular, the condition $\nabla_d z(\hat{\zeta}) > 0$ for all $d \in \mathbb{R}^\ell_- \setminus \{\mathbf{0}\}$ is equivalent to $u < \mathbf{0}$ for all alternative optimal solutions $(u, v) \in \mathcal{P}_D$ for (D-RLP) associated with $\hat{\zeta}$, while a zero directional derivative implies that the dual variable associated with one of the constraints is zero. This makes sense, as a zero dual value implies that the constraint can be tightened without changing the optimal solution, and this is exactly the condition that would indicate a given solution is not nondominated in the multiobjective context.

Note that when all directional derivatives in directions $d \in \mathbb{R}^\ell_- \setminus \{\mathbf{0}\}$ are strictly positive (the function is strictly decreasing everywhere), then the boundary of the epigraph of the RVF and the EF exactly coincide. This case is illustrated in Example 4 below.

Example 4. *We consider the following instance of* (RVF)*:*

$$
\begin{aligned}
z(\zeta) = \min \quad & x_1 + \frac{1}{4}x_2 + \frac{1}{2}y_1 - \frac{3}{4}y_2 \\
\text{s.t.} \quad & \frac{4}{5}x_1 + \frac{1}{2}x_2 + \frac{1}{3}y_1 + 0y_2 \leq \zeta \\
& \frac{3}{5}x_1 + \frac{1}{3}x_2 + \frac{1}{4}y_1 - \frac{1}{5}y_2 = 4 \\
& x_i \in \mathbb{Z}_+, \quad \forall i \in \{1,2\} \\
& y_j \in \mathbb{R}_+, \quad \forall j \in \{1,2\}.
\end{aligned}
$$

Figure 2.3 shows the EF in Example 4, which exactly coincides with the boundary of the epigraph of the associated RVF.

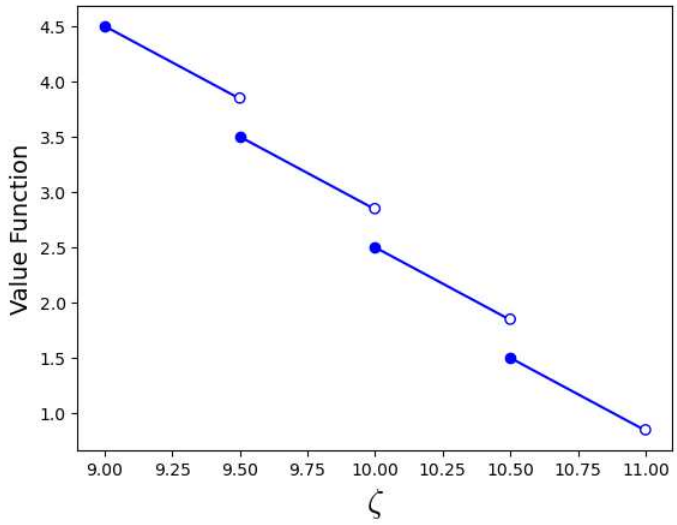


Figure 2.3: EF and RVF for the MILP in Example 4.

It is worth noting that it is possible to avoid the difficulty of the weak NDPs that aren't part of the EF by changing from "$\leq$" to equality for the constraints associated with the objectives of the multiobjective version of the problem. However, in that case, a different difficulty is introduced—there may then be parts of the RVF that are *increasing* (strictly positive directional derivative in the direction $d \in \mathbb{R}^{\ell}_{+}$), and we then have that the boundary of the epigraph for *those* parts of the RVF is not part of the EF. As such, this approach therefore does not make the statement of the theorem any cleaner. In Appendix D, we state the equivalent Theorem D.0.1, noting that the only deviation from the original statement is replacing $\leq$ with $=$ for the parametric constraints.

Before closing this section, we use the results of Theorem 2.2.1 to connect the description of the RVF, from Theorem 2.1.5, and the EF of the (MO-MILP). Specifically, we will see that not only does a minimal subset $\mathcal{S}_{\min}$ give us a description of the VF but also a description of the EF (under the previous caveat that what constitutes a "description" is context dependent). In doing so, we provide some insight into the role of the objective vector c^0, which is distinguished in the (RVF) problem but is indistinguishable from other objectives in the (MO-MILP).

First, suppose that the (MO-MILP) has been solved in some way to obtain a set $\mathcal{F}$ that is the set of points on the EF. We wish to determine the RVF. The first part of Theorem 2.2.1 says

$$z(\zeta) = \min_{\gamma \in \mathcal{F}} \left\{ \gamma_0 \ : \ \gamma^{1:\ell} \leq \zeta \right\}, \qquad \forall \zeta \in \mathcal{C}.$$

Alternatively, the (MO-MILP) may be solved by providing some set of integer parts of efficient solutions, denoted as $\mathcal{S}_I^E \subseteq \mathcal{S}_I$, so that for all $\gamma \in \mathcal{F}$ there must exist an efficient solution $(x_I, x_C) \in X_{\text{MO}}$ with $x_I \in \mathcal{S}_I^E$ and $\gamma = C_I x_I + C_C x_C$. Then, by Theorem 2.1.5, there is a minimal description $\mathcal{S}_{\min} \subseteq \mathcal{S}_I^E$.

Now we take the opposite perspective: suppose that we can determine a RVF (by finding $\mathcal{S}_{\min}$ as per Theorem 2.1.5) and wish to determine a solution to a given (MO-MILP) that has p objectives, encoded as rows $1, \ldots, p$ in the matrix $D \in \mathbb{R}^{p \times n}$. It is well known in multiobjective optimization (and straightforward to prove) that adding an objective of the form $\lambda^\top D$ to the (MO-MILP) does not change the set of efficient solutions, provided $\lambda \geq 0$, nor does removing any duplicate objective. We may thus consider any restricted value problem with (i) $c^0 = D^{k:k}$, the kth objective, and $C^{1:\ell}$ consisting of D with the kth row deleted, so $\ell = p - 1$, for any $k = 1, \ldots, p$, or (ii) $c^0 = \lambda^\top D$ and $C^{1:\ell} = D$, so $\ell = p$, for any $\lambda \geq 0$. In any case of (i) or (ii), the set of efficient solutions of $\operatorname{vinf}\{Dx \ : \ x \in X_{\text{MO}}\}$ is precisely the set of efficient solutions of $\operatorname{vinf}\{Cx \ : \ x \in X_{\text{MO}}\}$. We may thus determine the EF of the former by finding $\mathcal{S}_{\min}$ for the RVF associated with the latter. We are assured, by the first part of Theorem 2.2.1, that if $\gamma \in \mathcal{F}$ where (x'_I, x'_C) is an efficient solution with $\gamma = C_I x'_I + C_C x'_C$, then there exists $\zeta \in \mathcal{C}$ with $\zeta = \gamma^{1:\ell}$ and $z(\zeta) = \gamma_0$. Now Theorem 2.1.5 ensures that for such ζ, there must exist $(x_I, x_C) \in X_{\text{MO}}$ with $x_I \in \mathcal{S}_{\min}$ so that $\gamma = C_I x_I + C_C x_C$. Thus, integer parts in $\mathcal{S}_{\min}$ are sufficient to describe the EF, $\mathcal{F}$.

In this chapter, we discussed the structure of the Restricted LP Value Function, which forms the foundation for understanding the structure of the Restricted Value Function. We examined these

structures in detail and analyzed the relationship between the VF and the EF of the corresponding multiobjective optimization problem. Building on this theoretical foundation, the next chapter presents the design of the RVF Algorithm and introduces its different implementations.

2.3 Conclusions

In this study, we discussed the relationship between the RVF and the EF of a multiobjective integer linear program (MO-MILP). We demonstrated that the EF lies on a subset of the boundary of the epigraph of the RVF. In so doing, we highlight an important relationship that connects two parts of the literature that have been considered distinct. We also demonstrated that the RVF is the minimum of polyhedral functions associated with RLPVF and discussed the structure of the RVF, including its continuity and differentiability.

These results provide the foundation for the algorithmic developments presented in the next chapter. In particular, an important question is how to construct the RVF in practice. This is explored in the next chapter, where we develop and implement a generalized cutting-plane algorithm for iteratively constructing the RVF. In addition, we investigate several implementation variants obtained by varying key components of the algorithm along different dimensions. The implementation of this framework reveals several computational challenges, in particular the need to incorporate callback mechanisms within the solver to dynamically enforce additional constraints and eliminate repeated or undesirable solutions during the search process.

Chapter 3

The RVF Algorithm

In this chapter, we present the RVF Algorithm, a generalized cutting-plane algorithm for constructing the restricted value function (RVF) introduced in Chapter 2, and describe in detail the "separation" problem to be solved at each iteration. We then discuss details of the implementation of this algorithm and specify a number of variants obtained by adjusting the parameters. Finally, we conduct an extensive computational study to evaluate the effectiveness of these variants. The outline is as follows. In Section 3.1, we present our cutting-plane algorithm for constructing both the efficient frontier (EF) and the value function. Both the value function representation and the cutting-plane algorithm are finite under our assumption that X_{MO} is bounded. Section 3.2 compares the RVF Algorithm with alternatives. Section 3.3 discusses different variants of the basic algorithm that we have proposed for testing. The data sets used in the computational experiments are described in Section 3.4. The results of experiments with these implementations are analyzed in Section 3.5. Section 3.6 describes how to use the available implementation of the RVF Algorithm, including what dependencies are needed and how to input data. Finally, Section 3.7 summarizes what was discussed.

3.1 Basic Framework

In this section, we present the basic framework for the RVF Algorithm. The purpose of the proposed algorithm is to construct a subset of $\mathcal{S}_I$ that comprises a description of the RVF, as codified in

Theorem 2.1.5. The proposed algorithm is a generalized cutting-plane method that iteratively improves an upper approximation of the RVF until it converges to the true function. The "cuts" in this context refer to the convex bounding functions described in (RLPVF). In Section 3.1.1, we first provide a basic outline of the algorithm. This is followed by a discussion of the algorithm's correctness in Section 3.1.5. A key step of the RVF Algorithm is the optimization of this upper approximating function, which can be formulated as a mixed integer nonlinear optimization problem (MINLP), as discussed in detail in Subsection 3.1.3.

3.1.1 Outline of the Algorithm

At a high level, the algorithm iteratively improves the approximation of an upper bounding function until convergence. At iteration k, the RVF Algorithm maintains an upper bounding function (primal bounding function) of the RVF, denoted by $\bar{z}^k$, given by

$$\bar{z}^k(\zeta) = \min\left\{\min_{x_I \in S^k} \bar{z}(\zeta;\ x_I),\ U\right\} \quad \forall \zeta \in \mathcal{C}, \tag{3.1}$$

where $\mathcal{S}^k$ is the set of points in $\mathcal{S}_I$ identified so far and $U \in \mathbb{R}$ is an upper bound on the VF, i.e. $z(\zeta) \leq U$ for all $\zeta \in \mathcal{C}$. The selection of U will be addressed in Section 3.1.2.

The algorithm proceeds by identifying in iteration k the point $\zeta^* \in \mathcal{C}$ at which the difference between the approximate value $\bar{z}^k(\zeta^*)$ and the true value $z(\zeta^*)$ is maximized. This point is given by

$$\zeta^* = C_I^{1:\ell} x_I^{k+1} + C_C^{1:\ell} x_C^{k+1},$$

where

$$(x_I^{k+1}, x_C^{k+1}) \in \underset{(x_I, x_C) \in X_{\mathrm{MO}}}{\arg\max} \left(\bar{z}^k(C_I^{1:\ell} x_I + C_C^{1:\ell} x_C) - (c_I^0 x_I + c_C^0 x_C)\right). \tag{3.2}$$

Here $\bar{z}^k(C_I^{1:\ell} x_I^{k+1} + C_C^{1:\ell} x_C^{k+1}) = \bar{z}^k(\zeta^*)$ is the value of the upper approximation at ζ^* and—as we shall argue later—the true value at ζ^* is $z(\zeta^*) = c_I^0 x_I^{k+1} + c_C^0 x_C^{k+1}$. In the algorithm, this difference is denoted as

$$\begin{aligned} \theta^{k+1} &= \bar{z}^k(C_I^{1:\ell} x_I^{k+1} + C_C^{1:\ell} x_C^{k+1}) - (c_I^0 x_I^{k+1} + c_C^0 x_C^{k+1}) \\ &= \max_{(x_I, x_C) \in X_{\mathrm{MO}}} \left(\bar{z}^k(C_I^{1:\ell} x_I + C_C^{1:\ell} x_C) - (c_I^0 x_I + c_C^0 x_C)\right). \end{aligned}$$

The steps of the algorithm are outlined formally in the RVF Algorithm.

Algorithm 1 RVF Algorithm

Input: X_{MO}, $C \in \mathbb{Q}^{(\ell+1)\times n}$, U an upper bound on $z(\zeta)$ over $\zeta \in \mathcal{C}$.
Output: $\mathcal{S}^k$ such that $z(\zeta) = \bar{z}^k(\zeta) = \min\limits_{x_I \in \mathcal{S}^k} \bar{z}(\zeta;\ x_I), \quad \forall \zeta \in \mathcal{C}$.

Initialize $\bar{z}^0(\zeta) = U$ for all $\zeta \in \mathbb{R}^\ell$, $k = 0$, $\mathcal{S}^0 = \emptyset$, $\theta^0 = +\infty$.
while $\theta^k > 0$ **do**
  Determine $(x_I^{k+1}, x_C^{k+1}) \in \underset{(x_I, x_C) \in X_{\text{MO}}}{\arg\max} \left(\bar{z}^k(C_I^{1:\ell} x_I + C_C^{1:\ell} x_C) - (c_I^0 x_I + c_C^0 x_C)\right)$.
  Convert (x_I^{k+1}, x_C^{k+1}) to an efficient solution (see Section 3.1.4).
  Set $\mathcal{S}^{k+1} \leftarrow \mathcal{S}^k \cup \{x_I^{k+1}\}$.
  Set $\theta^{k+1} = \bar{z}^k(C_I^{1:\ell} x_I^{k+1} + C_C^{1:\ell} x_C^{k+1}) - (c_I^0 x_I^{k+1} + c_C^0 x_C^{k+1})$.
  $\bar{z}^{k+1}(\zeta) = \min\left\{\bar{z}^k(\zeta),\ \bar{z}(\zeta;\ x_I^{k+1})\right\}$ for all $\zeta \in \mathcal{C}$.
  $k \leftarrow k + 1$.
end

It is important to note that (x_I^{k+1}, x_C^{k+1}) obtained during the k-th iteration of the RVF Algorithm may only be *weakly* efficient, while only (strongly) efficient solutions are required to describe the RVF. Therefore, before adding the integer part to $\mathcal{S}^k$, we must "convert it" to a (strong) efficient solution using a process common in multiobjective optimization. The details of this are discussed later in Section 3.1.4.

It is easily proved that any optimal solution to (3.3) must also solve (3.2). (See Lemma F.0.2 in Appendix F.) In the case of multiple optimal solutions for (3.2), solving (3.3) ensures an efficient solution for the multiobjective problem is selected.

$$\underset{(x_I, x_C) \in X_{\text{MO}}}{\arg\min} \left\{\mathbf{1}^\top (C_I x_I + C_C x_C) \ :\ C_I x_I + C_C x_C \leq C_I x_I^{k+1} + C_C x_C^{k+1}\right\}. \tag{3.3}$$

Provided that the maximum difference between the upper approximation to the VF and the true VF is strictly positive, indicated by $\theta^{k+1} > 0$, this yields a new stability region associated with a new member of $\mathcal{S}_I$, which we add to obtain $\mathcal{S}^{k+1}$.

The algorithm is designed to iterate until the approximation is exact, detected by reaching $\theta^k = 0$. However, it's worth mentioning that although setting the stopping condition as "while $\theta > 0$," seems reasonable in theory, in practice, it might be better to opt for "while $\theta > thetaGap$," where "thetaGap" should be determined based on the instance data or instance class data. In our research, we set thetaGap to 0.001, and it appears to function effectively in practice based on the instances we utilized in our computational experiment. This change is recommended because the algorithm deals with a nonlinear problem in each iteration, and to handle potential numerical challenges, it's

more reliable to use a very small value instead of zero. A high-level overview of the RVF Algorithm is provided below.

A few important details regarding the RVF Algorithm are worth noting. The RVF Algorithm is designed to iteratively improve an upper approximation of the value function (RVF) until it becomes exact. The algorithm generates a sequence of points in X_{MO} but only stores their integer parts in the set $\mathcal{S}^k$, which converges to the set referred to as $\mathcal{S}_{\text{min}}$ in Chapter 2. The upper approximation at iteration k, denoted $\bar{z}^k$, is initialized and then updated according to (3.1). It is important to note that $\bar{z}^k$ is not explicitly constructed; rather, (3.1) provides a description of the upper approximating function, while the set $\mathcal{S}^k$ is the object actually constructed by the algorithm.

An illustration of these steps is provided in Example 5.

Example 5. ***Illustrative Example for the RVF Algorithm:*** *The steps of the RVF Algorithm as applied to Example 1 are depicted graphically in Figure 3.1 below. In the first iteration, the RVF Algorithm identifies the point* $(\zeta, z(\zeta)) = (4\frac{4}{9}, 0)$*, with stability region corresponding to* $(x_1, x_2) = (0, 0)$*, and updates the upper approximation to the blue convex function. In the subsequent iteration, the algorithm searches for the point with the largest difference between the current upper approximation and the RVF (the red piecewise linear function). The point of largest difference actually occurs at the far left, outside of the region shown in the figure, at* $\zeta = -55.5$*, but we illustrate the gap at* $\zeta = -15$ *(refer to Appendix B to view the full extent of the linear programming EFs). This point lies within the stability region corresponding to* $(x_1, x_2) = (1, 0)$*. The RVF Algorithm then updates the upper approximation again and, in the subsequent iteration, finds the point with the most difference between the upper approximation and the RVF, which occurs at* $\zeta = -12\frac{1}{6}$*. The RVF Algorithm continues in this manner, next finding* $\zeta = -10$*, until, at the next iteration, there is no such point. Thus, the RVF Algorithm terminates in iteration 4 with* $\theta = 0$ *when the upper approximation and the VF are the same.*

3.1.2 Initialization and Termination

The most important aspect of algorithm initialization is determining an initial upper bound U. One possible approach is to solve the linear programming (LP) relaxation of the mixed integer linear program (MILP) $\max\{c_I^0 x_I + c_C^0 x_C \ : \ (x_I, x_C) \in X_{\text{MO}}\}$. This would provide a valid initial upper bound for the optimization problem.

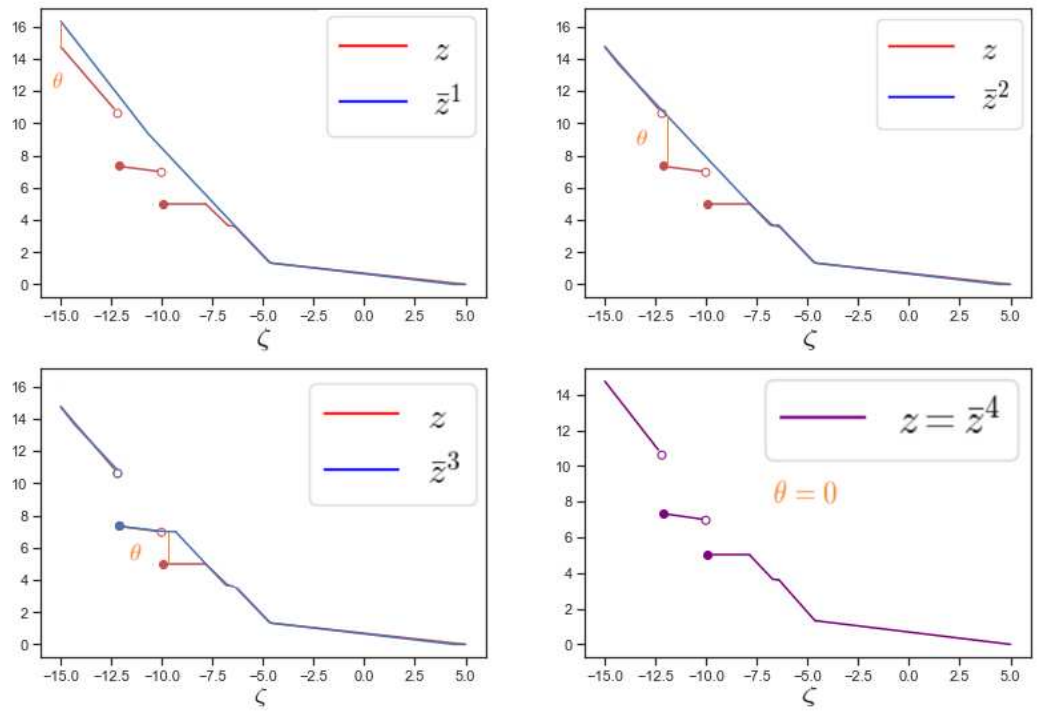


Figure 3.1: RVF and upper approximation in iterations of the RVF Algorithm with Example 1. The RVF z is illustrated on the range $\zeta \in [-15, 5]$. Note that the first cut occurs at $\zeta = -55.5$, which is outside of this range. Please refer to Appendix B to view the full EF or RVF.

It is important to note that in the final iteration, when θ^{k+1} takes the value 0, there is no need to convert the solution (x_I^{k+1}, x_C^{k+1}) into an efficient solution and add it to $\mathcal{S}^k$ *unless* its objective value $c_I^0 x_I^{k+1} + c_C^0 x_C^{k+1}$ happens to coincide with the initial upper bound, U.

Finally, observe that if the RVF Algorithm is terminated early, while $\theta^k > 0$, then the value of the final optimization problem solved in Step 3 provides a natural performance guarantee on the quality of the current upper approximation to the VF: it measures the maximum error between the true VF, z, and the upper approximation, $\bar{z}^k$, over any point in its domain.

3.1.3 Generating the Bounding Functions

In this section, we give the details of the "separation" problem (Step 3) in the main algorithm. While the RVF Algorithm was previously presented at a high level, its implementation requires solving a nontrivial subproblem at each iteration. We therefore describe how this subproblem can

be formulated explicitly as a standard mathematical optimization problem and discuss how it can be solved in practice.

General MILPs. The RVF Algorithm was presented in the previous section at a high level of abstraction to simplify the exposition. In this section, we start by clarifying how the subproblem that needs to be solved in each iteration can be solved in practice by formulating it as a standard mathematical optimization problem.

The optimization problem in (3.2), solved in Step 3 of the algorithm, can be formulated as a mixed integer nonlinear optimization problem (MINLP) as follows. First, we model it using an auxiliary variable θ to move $\bar{z}^k$ into the constraints:

$$\begin{array}{lll} \theta^{k+1} = \max & \theta & \\ \text{subject to} & \theta \leq \bar{z}^k(C_I^{1:\ell} x_I + C_C^{1:\ell} x_C) - (c_I^0 x_I + c_C^0 x_C) & \\ & (x_I, x_C) \in X_{\text{MO}} & \\ & \theta \in \mathbb{R}. & \end{array} \tag{3.4}$$

Next, we use (3.1) and (CR) to expand (3.4) to the $k+1$ constraints

$$\theta + c_I^0 x_I + c_C^0 x_C \leq U, \tag{3.5}$$

and

$$\theta + c_I^0 x_I + c_C^0 x_C \leq c_I^0 x_I^i + z_{\text{LP}}(C_I^{1:\ell} x_I + C_C^{1:\ell} x_C - C_I^{1:\ell} x_I^i;\ b - A_I x_I^i), \quad i = 1, \ldots, k. \tag{3.6}$$

(Recall $\mathcal{S}^k = \{x_I^1, \ldots, x_I^k\}$.) We can model the term involving z_{LP} for each i by using its LP dual problem, which via (D-RLP) is given by

$$\max_{(u^i, v^i) \in \mathcal{P}_D} (C_I^{1:\ell} x_I + C_C^{1:\ell} x_C - C_I^{1:\ell} x_I^i)^\top u^i + (b - A_I x_I^i)^\top v^i. \tag{3.7}$$

If $(x_I, x_C) \in X_{\text{MO}}$ satisfies (3.6) for some i, then by strong LP duality, there must exist $(u^i, v^i) \in \mathcal{P}_D$ (optimal for the dual of the LP after fixing the values of the integer variables to x_I^i) such that

$$\theta + c_I^0 x_I + c_C^0 x_C \leq c_I^0 x_I^i + z_{\text{LP}}(C_I^{1:\ell} x_I + C_C^{1:\ell} x_C - C_I^{1:\ell} x_I^i;\ b - A_I x_I^i)$$

$$= c_I^0 x_I^i + (C_I^{1:\ell} x_I + C_C^{1:\ell} x_C - C_I^{1:\ell} x_I^i)^\top u^i + (b - A_I x_I^i)^\top v^i.$$

Conversely, if there exists some $(u^i, v^i) \in \mathcal{P}_D$ such that $(x_I, x_C) \in X_{\text{MO}}$ satisfies

$$\theta + c_I^0 x_I + c_C^0 x_C \leq c_I^0 x_I^i + (C_I^{1:\ell} x_I + C_C^{1:\ell} x_C - C_I^{1:\ell} x_I^i)^\top u^i + (b - A_I x_I^i)^\top v^i,$$

then by weak duality, it must be that (x_I, x_C) satisfies (3.6) for this i. This shows that the following explicit MINLP is a valid formulation of the optimization problem in (3.2):

$$\begin{aligned}
\theta^{k+1} = \max \quad & \theta && && (3.8)\\
\text{subject to} \quad & \theta + c_I^0 x_I + c_C^0 x_C \leq c_I^0 x_I^i + \\
& (C_I^{1:\ell} x_I + C_C^{1:\ell} x_C - C_I^{1:\ell} x_I^i)^\top u^i + (b - A_I x_I^i)^\top v^i, && i = 1, \ldots, k && (3.9)\\
& \theta + c_I^0 x_I + c_C^0 x_C \leq U && && (3.10)\\
& (u^i, v^i) \in \mathcal{P}_D, && i = 1, \ldots, k && (3.11)\\
& (x_I, x_C) \in X_{\text{MO}} && && (3.12)\\
& \theta \in \mathbb{R}. && && (3.13)
\end{aligned}$$

Note that the constraint (3.9) makes the problem nonlinear: it has bilinear terms $(C_I^{1:\ell} x_I)^\top u^i$ and $(C_C^{1:\ell} x_C)^\top u^i$ linking the u^i and (x_I, x_C) variables for each i. However, the problem (3.8)–(3.13), which we refer to as the *MINLP subproblem*, can be solved either with an off-the-shelf nonconvex quadratic solver or possibly with a customized algorithm.

The discussion above pertains to general MILPs. In the remainder of this section, we discuss and analyze the RVF Algorithm when applied to pure integer linear programs (PILP), since the VF exhibits a special structure that allows the problem to be linearized.

PILPs. In the PILP case, the algorithm can be simplified. In this case, since there are no continuous variables, the continuous restriction has no variables, in principle. This seems problematic since it means we cannot dualize the continuous restriction, and the method we used in the general MILP case does not seem to go through. Naively writing the continuous restriction, we have

$$z_{\text{LP}}(\zeta;\ \hat{x}_I) = \min \left\{ 0 \ :\ 0 \leq \zeta - C_I^{1:\ell} \hat{x}_I \right\},$$

which is a trivial problem that either has optimal value 0 (when $C_I^{1:\ell}\hat{x}_I \leq \zeta$) or is infeasible. So the challenge is that some integer assignments may simply be infeasible for a given right-hand side, and there is no continuous restriction that can be dualized. This contrasts with the MILP setting, where fixing the integer part always yields a continuous LP whose dual remains feasible even when the associated primal is infeasible, ensuring that every integer part still gives rise to a valid dual solution and hence a valid upper bounding function.

To treat PILPs within the same RVF framework, the formulation in (3.14) introduces a dummy continuous variable with a sufficiently large cost whose sole purpose is to guarantee the existence of a feasible dual solution for every integer part. Once this device is introduced, the PILP case reduces to the MILP case from the standpoint of constructing bounding functions, and the core value-function formulation already captures the full structure of the linear case, with the following formulation included here for completeness.

We introduce a dummy variable as follows.

$$z_{\mathrm{LP}}(\zeta;\ \hat{x}_I) = \min\left\{-My\ :\ y \leq \zeta - C_I^{1:\ell}\hat{x}_I, y \leq 0\right\}.$$

From strong duality, we have that

$$\begin{aligned} z_{\mathrm{LP}}(\zeta;\ \hat{x}_I) &:= \max\left\{(\zeta - C_I^{1:\ell}\hat{x}_I)^\top u\ :\ \ u \in \mathbb{R}^\ell_-, u \geq -M\right\} \\ &= \begin{cases} 0, & \text{if } \zeta \geq C_I^{1:\ell}\hat{x}_I, \\ +\infty \to M(\zeta - C_I^{1:\ell}\hat{x}_I) \to M', & \text{otherwise.} \end{cases} \end{aligned} \tag{3.14}$$

We may now rewrite problem (3.8)–(3.13) for the PILP case explicitly as follows.

$$\theta^{k+1} = \max \quad \theta \tag{3.15}$$

$$\text{subject to} \quad \theta + c_I^0 x_I \leq c_I^0 x_I^i + (\zeta - C_I^{1:\ell} x_I^i)^\top u^i, \qquad i = 1, \ldots, k \tag{3.16}$$

$$\zeta = C_I^{1:\ell} x_I \tag{3.17}$$

$$\theta + c_I^0 x_I \leq U \tag{3.18}$$

$$u^i \in \mathbb{R}^\ell_-, \qquad i = 1, \ldots, k \tag{3.19}$$

$$A_I x_I = b \tag{3.20}$$

$$x_I \in \mathbb{Z}^r_+ \tag{3.21}$$

$$\theta \in \mathbb{R}. \tag{3.22}$$

Making use of (3.14) and (3.17), we can conceptually eliminate the variables u^i and ζ to obtain

$$\theta^{k+1} = \max \quad \theta \tag{3.23}$$

$$\text{subject to} \quad \theta + c^0_I x_I \leq c^0_I x^i_I, \quad \forall i \in 1, \ldots, k \text{ s.t. } C^{1:\ell}_I x_I \geq C^{1:\ell}_I x^i_I \tag{3.24}$$

$$(3.18),\ (3.20),\ (3.21) \text{ and } (3.22). \tag{3.25}$$

We refer to the problem given by (3.23)–(3.25) as PILP-NLP^k in what follows. Note that this formulation is not a standard mathematical optimization problem because enforcement of constraints (3.24) is conditional. This could potentially be handled by using indicator variables, which are a standard feature in modern MILP solvers, but we instead reformulate this directly by introducing auxiliary variables and big-M constraints, as follows.

$$\theta^k = \max \quad \theta \tag{3.26}$$

$$\text{subject to} \quad \theta + c^0_I x_I \leq (1-\beta^i) c^0_I x^i_I + \beta^i U, \qquad i = 0, 1, \ldots, k-1 \tag{3.27}$$

$$(C^{1:\ell}_I x_I)_j - (C^{1:\ell}_I x^i_I)_j + \epsilon^i_j \leq \overline{M}^j (1 - \alpha^i_j), \qquad i = 0, 1, \ldots, k-1,\ j = 1, \ldots, l \tag{3.28}$$

$$(C^{1:\ell}_I x^i_I)_j - (C^{1:\ell}_I x_I)_j \leq \underline{M}^j \alpha^i_j, \qquad i = 0, 1, \ldots, k-1,\ j = 1, \ldots, l \tag{3.29}$$

$$\beta^i \geq \alpha^i_j, \qquad i = 0, 1, \ldots, k-1,\ j = 1, \ldots, l \tag{3.30}$$

$$\beta^i \leq \sum_{j=1}^{\ell} \alpha^i_j, \qquad i = 0, 1, \ldots, k-1 \tag{3.31}$$

$$\alpha^i \in \{0,1\}^\ell, \qquad i = 0, 1, \ldots, k-1 \tag{3.32}$$

$$\beta^i \in \{0,1\}, \qquad i = 0, 1, \ldots, k-1 \tag{3.33}$$

$$(3.20),\ (3.21) \text{ and } (3.22), \tag{3.34}$$

where for each i and j, $\underline{M}^j$ and ϵ^i_j are set to sufficiently large and small positive values, respectively, so that for any possible choice of x_I having $(C^{1:\ell}_I x_I)_j < (C^{1:\ell}_I x^i_I)_j$ it must be that $\epsilon^i_j \leq (C^{1:\ell}_I x^i_I)_j - (C^{1:\ell}_I x_I)_j \leq \underline{M}^j$, and $\overline{M}^j$ is set to a sufficiently large positive value so that for any possible choice of x_I having $(C^{1:\ell}_I x_I)_j \geq (C^{1:\ell}_I x^i_I)_j$ it must be that $(C^{1:\ell}_I x_I)_j - (C^{1:\ell}_I x^i_I)_j \leq \overline{M}^j - \epsilon^i_j$. For $j = 1, \ldots, l$, it can be established that $\overline{M}^j = \underline{M}^j = \sum_{i=0}^{r-1} C^{1:\ell}_I[j][i]$. Note that in the case that $C^{1:\ell}$ is entirely

integer-valued, $\epsilon_j^i = 1$ for all i and j suffices. Furthermore, for each i, α^i is a vector of binary variables such that for all $j = \{1, 2, \ldots, l\}$

$$\alpha_j^i = \begin{cases} 0, & (C_I^{1:\ell} x_I)_j \geq (C_I^{1:\ell} x_I^i)_j, \\ 1, & (C_I^{1:\ell} x_I)_j < (C_I^{1:\ell} x_I^i)_j, \end{cases} \tag{3.35}$$

if and only if (3.28) and (3.29) are satisfied. Note that we substituted out (3.24) and (3.18) in the problem set (3.23)–(3.25) with (3.27), (3.28), and (3.29).

The binary variables α_j^i encode, for each previously discovered integer part x_I^i, which componentwise inequalities $(C_I^{1:\ell} x_I)_j \geq (C_I^{1:\ell} x_I^i)_j$ are satisfied by the current candidate x_I. The auxiliary binary β^i serves as the logical OR of the α_j^i variables and is used to switch the right-hand side in constraint (3.27) between $c_I^0 x_I^i$ and the global bound U. This switching is inherently *local to each index i*: for one index i, the inequality may hold in all components (so the tight bound $c_I^0 x_I^i$ applies), whereas for a different index i', it may be violated in at least one component (so the looser bound U applies).

The RVF Algorithm with the problem set (3.26)–(3.34) would be equivalent to the method given in [Sylva and Crema, 2004], which enumerates all nondominated points (NDP) of a PILP using a sequence of progressively more constrained MILP that generates a new solution at each iteration. In their study, Sylva and Crema employ a reformulation of disjunctive constraints using a big-M approach in conjunction with artificial binary variables. This modification results in the production of nondominated solutions, as opposed to just weakly nondominated solutions, by transforming the objective function into a weighted sum. However, this process also leads to an increase in computational effort as p binary variables and $p + 1$ constraints are added in each iteration, where $p = \ell + 1$. The authors only present complete representations for biobjective cases and limit their numerical examination for three objectives to the generation of incomplete representations.

The approach proposed by [Lokman and Köksalan, 2013] builds upon the methodology of [Sylva and Crema, 2004] by incorporating the other objectives into the objective function, scaled by a small constant. This modification results in the elimination of one constraint and one binary variable in each iteration. Despite this improvement, the algorithm still experiences a rapid increase in the number of constraints and binary variables.

3.1.4 Converting Weakly Efficient Solutions

As mentioned earlier, Step 3 of the RVF Algorithm requires determining whether (x_I^{k+1}, x_C^{k+1}) is only weakly efficient and, if so, converting it to an efficient solution. We consider two types of conversion. The first, referred to as the *one-stage conversion*, involves replacing (x_I^{k+1}, x_C^{k+1}) with an optimal solution of the following problem:

$$\min_{(x_I, x_C) \in X_{\text{MO}}} \{\mathbf{1}^\top (C_I x_I + C_C x_C) : C_I x_I + C_C x_C \leq C_I x_I^{k+1} + C_C x_C^{k+1}\}. \tag{3.36}$$

The second conversion method involves solving a *two-stage problem* to transform a possibly *weakly* efficient solution into a (strongly) efficient one as follows. In the first stage, we solve

$$\min \quad c_I^0 x_I + c_C^0 x_C \tag{3.37}$$

$$\text{subject to} \quad C_I^{1:l} x_I + C_C^{1:l} x_C \leq C_I^{1:l} x_I^{k+1} + C_C^{1:l} x_C^{k+1} \tag{3.38}$$

$$(x_I, x_C) \in X_{\text{MO}}, \tag{3.39}$$

and in the second stage, we solve

$$\min_{(x_I, x_C) \in X_{\text{MO}}} \{\mathbf{1}^\top (C_I x_I + C_C x_C) : C_I x_I + C_C x_C \leq C_I x_I^1 + C_C x_C^1\}, \tag{3.40}$$

where (x_I^1, x_C^1) is the optimal solution to the first stage problem, i.e., (3.37)–(3.39). For the optimal solutions found via branch-and-bound in each iteration of the RVF Algorithm, we can use the one-stage conversion, i.e., (3.36), whereas, for the feasible solutions, we can consider each of the conversions, i.e., (3.36) or (3.37)–(3.40). A preliminary investigation was conducted, wherein it was determined that the utilization of a one-stage conversion approach yields superior outcomes in comparison to a two-stage method. As a consequence of this finding, the subsequent computational experimentation phase exclusively adopts the one-stage conversion technique for the conversion of feasible solutions into efficient solutions.

3.1.5 Correctness of the RVF Algorithm

The proof of the correctness of the RVF Algorithm relies on the fact that $\bar{z}^k$ is a nonincreasing function for the same reason that the RVF is nonincreasing, namely that it is the minimum of a finite set of nonincreasing bounding functions by (3.1). Additionally, in iteration k of the algorithm, $\theta^{k+1} = \max_{\zeta \in \mathcal{C}} \bar{z}^k(\zeta) - z(\zeta)$ (see Lemma E.0.2 in Appendix E). Furthermore, any optimal solution (x_I^{k+1}, x_C^{k+1}) to the optimization problem in Equation (3.2) having optimal value $\theta^{k+1} > 0$ has the property that $x_I^{k+1} \notin \mathcal{S}^k$ (see Lemma E.0.3 in Appendix E.)

Now, with Theorem 3.1.1, we show the correctness of the RVF Algorithm by proving that it terminates finitely and returns the correct VF.

Theorem 3.1.1. *(Correctness of the RVF Algorithm) At termination, we have that $z(\zeta) = \bar{z}^k(\zeta)$ for all $\zeta \in \mathcal{C}$, so that $\mathcal{S}^k$ describes both the VF and the EF. Furthermore, the RVF Algorithm terminates in finitely many iterations under the assumption that X_{MO} is bounded.*

Proof. When X_{MO} is bounded, $\mathcal{S}_I$ is finite. Therefore, by Lemma E.0.3, which states that each iteration of the algorithm produces a new element of $\mathcal{S}_I$, the number of iterations must be finite. To show that $z = \bar{z}^k$ at termination, assume not for the sake of contradiction. Then there must exist $\zeta \in \mathbb{R}^\ell$ such that $z(\zeta) < \bar{z}^k(\zeta)$. But, by Lemma E.0.2, this is a contradiction to the assumption that $\theta^k = 0$ at termination. This completes the proof. ■

Proposition F.0.3 in Appendix F shows that the algorithm has one seemingly advantageous property that we briefly describe here. As the description is being constructed, each newly added integer part is guaranteed to be associated with a new nonempty stability region. In fact, it is not difficult to see that the added region must be *non-redundant* at the time it is added. In other words, we must have that $\bar{z}^k \neq \bar{z}^{k+1}$ at iteration k. Equivalently, there exists $\zeta \in \mathcal{C}$ such that $\bar{z}^{k+1}(\zeta) < \bar{z}^k(\zeta)$, namely ζ^{k+1} from the proof of Lemma E.0.2. Unfortunately, this property does not translate into a guarantee that the stability region associated with x^k is non-redundant in the end, so the algorithm is not guaranteed to produce a minimal description. It would be possible to postprocess the description to make it minimal if this were important for a particular application.

3.1.6 Properties of the RVF Algorithm in the PILP Case

It can be observed that the RVF Algorithm generates all NDPs of the EF in nondecreasing order of their $c_I^0 x_I$ value. We formalize this in Theorem 3.1.4.

The following result states that unless the optimal value of (3.15)–(3.22) is zero, the objective space image of its solution is new, meaning that it cannot already be in the objective space image of points in $\mathcal{S}^k$. Here, we define the set of objective space images of points in $\mathcal{S}^k$ by

$$F(\mathcal{S}^k) = \left\{ C_I x_I^i \ : \ i = 1, \ldots, k \right\}.$$

Lemma 3.1.2. *Any optimal solution (x_I, θ) to (3.15)–(3.22) (equivalently (3.23)–(3.25)) with $\theta > 0$ has the property that $C_I x_I \notin F(\mathcal{S}^k)$.*

Proof. If $C_I x_I \in F(\mathcal{S}^k)$ then there exists $i \in \{1, \ldots, k\}$ such that $C_I x_I = C_I x_I^i$. This means that $C_I^{1:\ell} x_I = C_I^{1:\ell} x_I^i$, so constraint (3.24) applies for this i, and $c_I^0 x_I = c_I^0 x_I^i$. But constraint (3.24) says $\theta + c_I^0 x_I \leq c_I^0 x_I^i$ so with $c_I^0 x_I^i = c_I^0 x_I$ it must be that $\theta \leq 0$. The results follow by contradiction. ■

We now consider the set of all NDPs of the (pure integer) multiobjective problem as

$$\operatorname{vinf} \left\{ C_I x_I : A_I x_I = b, \ x_I \in \mathbb{Z}_+^n \right\}.$$

Let $\mathcal{S}^E$ denote the set of all efficient solutions for the problem. Recall that in the RVF Algorithm, after solving the problem (3.15)–(3.22) to yield solution $\hat{x}_I$, an auxiliary problem is solved to "convert it" to a strongly efficient solution, x_I^k, to add to $\mathcal{S}^k$ to create $\mathcal{S}^{k+1}$. Thus, we may safely assume $\mathcal{S}^k \subseteq \mathcal{S}^E$ at any iteration k.

Lemma 3.1.3. *Suppose in the RVF Algorithm at iteration k, the elements of $\mathcal{S}^k$ are efficient solutions with the property that*

$$\min \left\{ c_I^0 y \ : \ y \in \mathcal{S}^E, \ C_I y \notin F(\mathcal{S}^k) \right\} \geq \max \left\{ c_I^0 y \ : \ y \in \mathcal{S}^k \right\}. \tag{3.41}$$

Then the optimal solution (x_I, θ) to (3.15)–(3.22) having $x_I \in \mathcal{S}^E$ and $\theta > 0$ satisfies $c_I^0 x_I = \min\{c_I^0 y \ : \ y \in \mathcal{S}^E, \ C_I y \notin F(\mathcal{S}^k)\}$.

Proof. Optimal solution (x_I, θ) to (3.15)–(3.22) having $x_I \in \mathcal{S}^E$ and $\theta > 0$ implies that $C_I x_I \notin F(\mathcal{S}^k)$ by Lemma 3.1.2. Now suppose that for some $i \in \{1, \ldots, k\}$, we have that $C_I^{1:\ell} x_I \geq C_I^{1:\ell} x_I^i$. Then since $x_I \in \mathcal{S}^E$, it must be that $c_I^0 x_I < c_I^0 x_I^i$, since otherwise $C_I x_I^i$ would dominate $C_I x_I$, contradicting $x_I \in \mathcal{S}^E$. But $c_I^0 x_I < c_I^0 x_I^i$ contradicts equation (3.41). Thus it must be that $C_I^{1:\ell} x_I \not\geq C_I^{1:\ell} x_I^i$ for all $i = 1, \ldots, k$. Hence (x_I, θ) must solve

$$\begin{array}{ll} \max & \theta' \\ \text{subject to} & \theta' + c_I^0 x'_I \leq U \\ & x'_I \in \mathcal{S}^E \\ & C_I x'_I \notin F(\mathcal{S}^k) \\ & \theta' \in \mathbb{R}. \end{array}$$

Now if any $x'_I \in \mathcal{S}^E$ with $C_I x'_I \notin F(\mathcal{S}^k)$ has $c_I^0 x'_I < c_I^0 x_I$, then (x'_I, θ') is feasible for this problem, where

$$\theta' := U - c_I^0 x'_I > U - c_I^0 x_I \geq \theta,$$

which contradicts optimality of (x_I, θ). Thus, it must be that

$$c_I^0 x_I = \min \left\{ c_I^0 y \ : \ y \in \mathcal{S}^E, \ C_I y \notin F(\mathcal{S}^k) \right\},$$

as required. ■

This lemma established that if the points in $\mathcal{S}^k$ induce a set of NDPs having the smallest value of the first objective, then the next NDP produced must also have that property. Hence, we can establish the final result by induction.

Theorem 3.1.4. *In the PILP case, the RVF Algorithm generates NDPs in the nondecreasing order of the first objective.*

Proof. We proceed by induction on k. In the case $k = 0$, the set $\mathcal{S}^k = \emptyset$, so any solution of (3.15)–(3.22) for $k = 0$ also solves $\min\{c_I^0 x_I \ : \ A_I x_I = b, \ x_I \in \mathbb{Z}_+^n\} = \min\{c_I^0 x_I \ : \ x_I \in \mathcal{S}^E\}$. The next iterate $x_I^1 \in \mathcal{S}^E$ is a solution of the latter and $\mathcal{S}^1 = \{x_I^1\}$. Thus, the base step for the induction, with $k = 1$, is proved. The inductive step is proved by Proposition 3.1.3. ■

3.2 Comparison to Other Algorithms

In this section, we examine related and recent algorithms, offering an in-depth analysis and presenting a review of the current state-of-the-art. While many algorithms exist for solving multiobjective MILPs, and some for finding the VF of an MILP, the RVF Algorithm has some distinctive features. First, it is very general, applying across both pure and mixed integer cases and any number of objective functions. This is in contrast to multiobjective algorithms, which typically apply in either the pure or mixed cases, but not both, and which often apply only to a specific number of objectives. Second, the RVF Algorithm fills a gap in the multiobjective MILP field in its applicability to problems that include continuous variables and any number of objectives. This area of multiobjective MILP is still in its infancy, and the RVF Algorithm constitutes a significant advance in it. Thirdly, the RVF Algorithm solves two optimization problems per stability region: one MINLP, which identifies a new stability region, and one "conversion" MILP, which converts a weakly nondominated solution to an NDP and is usually easily solved in practice. The size of the MINLP, in terms of its variables and constraints, grows linearly in the number of stability regions found: $l+m$ variables and $n-r+1$ extra constraints (one including l bilinear terms and the rest linear) are added per new stability region. This is comparable to existing multiobjective algorithms in the PILP case and offers advantages in the mixed case, even for small numbers of objectives. We provide more details for each of these cases below in comparison with well-known multiobjective algorithms.

3.2.1 Pure Integer (PILP) Case

In the PILP case, the stability regions are precisely the NDPs, and any minimal set of efficient solutions, denoted as $\mathcal{S}_{\min}$, consists of one efficient solution per NDP. Therefore, for an instance with N NDPs, the RVF Algorithm will solve N MINLPs and N conversion integer linear programs (ILP), with the size of the MINLPs increasing by l variables and only one constraint per NDP found, where the constraint added includes l bilinear terms. It is worth noting that none of these subproblems (MINLP or ILP) will be infeasible problems. In multiobjective algorithms, it is usually considered desirable to avoid infeasible ILPs, since these are often observed to be harder to solve in practice.

The RVF Algorithm performs comparably to earlier algorithms for multiobjective PILP with any number of objectives, $p = \ell + 1$. For example, in the theoretical analysis of their algorithm

performance, following on from Lemma 4.4 of [Özlen and Azizoğlu, 2009] suggest their algorithm solves on the order of $\binom{N}{p-1}$ ILPs. Similarly, Theorem 4 of [Kirlik and Sayın, 2014] suggests that their algorithm solves at most N^{p-1} "two-stage" ILPS, meaning an ILP minimizing one of the objectives over the ILP feasible set followed by a conversion ILP. The method of [Sylva and Crema, 2004] solves one ILP per NDP, but the size of the ILP increases by p binary variables and $p+1$ constraints per NDP found, which model the disjunctive requirement that ILP feasible points are not dominated by NDPs already found. The big-M values required in the model present challenges in addition to their size. Improvements were made in [Lokman and Köksalan, 2013], who suggest a variant limiting the ILP size to have at most $N(p-1)$ variables and Np constraints (reducing the objective dimension by one), and who propose using the disjunctions to define a search tree. Although some practical improvements are suggested, the number of ILPs solved in the search procedure is given in [Lokman and Köksalan, 2013] to be of the order of N^{p-1} in the worst case.

The RVF Algorithm can be related to the variant of the method proposed by [Sylva and Crema, 2004] and developed by [Lokman and Köksalan, 2013] in the PILP case by modeling its MINLP subproblem as an MILP. In [Lokman and Köksalan, 2013] variant, NDPs are found in nondecreasing order of one objective, more specifically, the c^0x objective, and the size of the subproblem grows in a similar manner to the RVF Algorithm, as we showed in Theorem 3.1.4.

More recent algorithms for the PILP case, such as the Disjunctive Constraints Method (DCM) proposed by [Boland et al., 2017] (given as Algorithm 1 in the paper), have shown good performance in practice, but their theoretical analysis in the paper provides a worst-case number of ILPs to be solved that is exponential in N. Furthermore, the ILPs increase in size through the need to model more disjunctions, and ILPs that do not yield a new NDP are infeasible. Hence, the worst-case analysis suggests that the number of infeasible ILPs solved could be exponential in N. The authors of [Boland et al., 2017] observe in computational experiments that, in practice, a small number of ILPs are solved per NDP and conjecture that the number is at most $pN+1$, with $(p-1)N+1$ of these being infeasible ILPs. In fact, a closer examination of DCM, as presented in Appendix I, reveals that irrespective of p, at most $2N+1$ ILPs need to be solved, with at most $N+1$ infeasible ILPs, and that the number of disjunctions modeled in any ILP is N, in the worst case. (In practice, this number is found to be several orders of magnitude smaller than N.)

The goal of this discussion is to demonstrate that the performance of the RVF Algorithm remains

within contention of existing multiobjective algorithms for the PILP case and any number of objectives, in theory. In practice, the state-of-the-art in MINLP solution versus ILP will play a part. While the MINLP solution is currently less well-developed, it is an active area of research. In addition to advances in MINLP, it is expected that future research on the RVF Algorithm will focus on extending the concept to improve its practical performance, as has been the case with earlier versions of multiobjective algorithms.

3.2.2 General Case

For the setting in which we have continuous variables, only the case of two objectives has existing algorithms with theoretical performance analysis. These are two variants of the Boxed Line Method (BLM), which were proposed by [Perini et al., 2020]. It is important to note that the performance analysis of the BLM variants depends on the number of line segments in the EF, not on the number of stability regions. Let L denote the number of line segments in the frontier. Then the basic iterative variant of BLM solves at most $3L + 2$ two-stage MILPs and at most $\frac{1}{2}L^2 + \frac{5}{2}L - 2$ scalarized (single-objective) MILPs. The recursive variant of BLM has better theoretical performance guarantees, with the number of two-stage MILPs remaining the same but the number of scalarized MILPs being at most $2L - 1$. This results in a total of $5L + 1$ MILPs plus $3L + 2$ conversion MILPs as a worst-case bound. It is worth noting that the size of the MILPs does not change as the algorithm progresses. In comparison, the RVF Algorithm solves increasingly larger MINLPs, which may be more challenging to solve, but may also be required to solve far fewer of them. If N denotes the number of stability regions found by the RVF Algorithm, so $N = \text{card}(\mathcal{S}_{\text{min}})$, then $N \leq L$ and the number of MINLPs it solves is bounded above by $N \leq L < 5L + 1$. However, each integer vector in $\mathcal{S}_{\text{min}}$ (each stability region) could contribute *many* line segments, so the RVF Algorithm may need to solve far fewer subproblems in practice. Overall, the RVF Algorithm offers a promising approach for solving multiobjective MILPs with any number of objectives.

To illustrate the difference in performance between the RVF Algorithm and the BLM variants, consider the example presented in Figures 3.2 and 3.3, with numerical details provided in Appendix J. The example has two objectives, and its EF has 24 line segments but only 4 stability regions. This means that the RVF Algorithm would be expected to solve only 4 MINLPs, with the first being solvable as an MILP, while the best variant of the BLM method would require 121—which is over

thirty times more—MILPs to be solved.

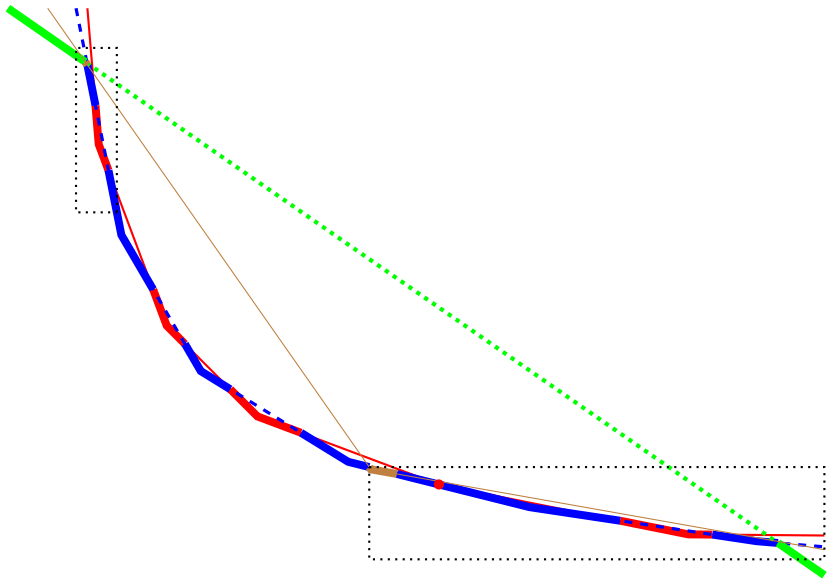
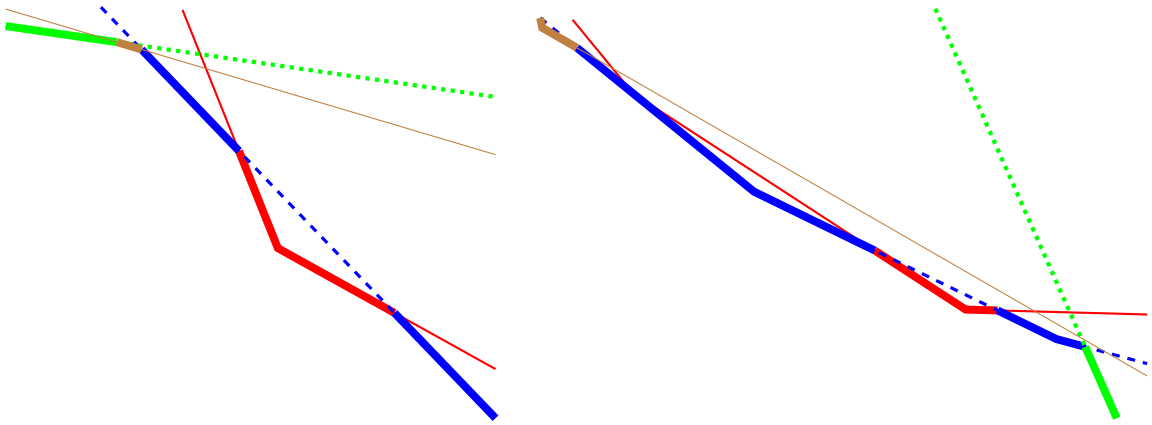


Figure 3.2: An example with two objectives, four elements of $\mathcal{S}_{\min}$, all with disconnected stability regions, having a total of 24 linear pieces in the EF, shown with thicker lines. The LP EFs for the four elements of $\mathcal{S}_{\min}$ are shown as red solid, blue dashed, sepia solid, and green dotted lines, respectively. Numerical details are given in Appendix J. Close-up of the areas in the top-left and bottom-right parts are given in Figure 3.3.

In a recent study, Rasmi and Türkay [2019] presented an algorithm for identifying all the integer vectors required to construct the frontier for a (MO-MILP) with an arbitrary number of objectives. They also provided a method for identifying the facets of multiobjective linear programming (MOLP) EF, which contribute at least one point to the overall (MO-MILP) frontier. For all pairs of extreme points of such facets that are adjacent in the facet, the authors demonstrated how to determine the parts of the line segment connecting the two points that are nondominated, if any. If only a portion of a facet of an MOLP EF appears in the (MO-MILP) frontier, Rasmi and Türkay [2019] do not attempt to describe the nondominated part explicitly. Their method for generating all integer vectors required for the (MO-MILP) frontier utilizes "no-good" constraints to eliminate integer vectors that have already been identified and disjunctive constraints, similar to those used by [Sylva and Crema, 2004], to eliminate parts of the objective space dominated by individual NDPs found. Thus, the size of the MILPs they solve grows in two ways as the algorithm progresses. No theoretical performance analysis is provided. It is noted that the algorithm may solve MILPs that yield integer solutions that do not contribute to the frontier, and many MILPs solved may find the same integer solutions repeatedly. Both of these issues are avoided in the RVF Algorithm.



(a) Top-left *(b) Bottom right*

Figure 3.3: Close-ups of the regions in the top-left and bottom-right of Figure 3.2 (Note that these have been asymmetrically scaled to allow better visibility of LP frontier intersections).

Computational results on instances with 2, 3, and 4 objectives are given by [Rasmi and Türkay, 2019]. These show that even for only 2 objectives, the algorithm was required to solve more than 25 MILPs per integer solution, and for one class of instances with 3 objectives, on average, more than 135 MILPs were solved per integer solution. No information is provided regarding the number of disjunctive constraints added to the MILPs in computation or how the size of these MILPs grows as the algorithm progresses. In another study, Pettersson and Ozlen [2019] developed an algorithm for identifying all supported and non-supported NDPs with an arbitrary number of objectives. The algorithm first identifies a super-set (which contains the EF) using Benson's method, then modifies the set of polytopes so that no two polytopes have a nonempty intersection, and finally, refines the super-set to exclude portions that are not part of the EF. The details of this algorithm are not provided, as it was presented in a short proceedings article, and no performance analysis is given.

3.3 Implementation of the RVF Algorithm

In this section, we expound upon the details of the implementations. However, prior to delving into that discussion, we first address a computational issue pertaining to setting bounds on the dual variables. While we devised a strategy to establish and update a secure bound on the dual variables for PILP, a robust approach proved elusive for MILP cases. Subsequently, we present our implementations.

3.3.1 Bounds on Variables

Some MINLP solvers require bounds on variables involved in nonlinear expressions. The only nonlinearity present in PILP-NLP^k is the term $\zeta^T u^i$, or equivalently $(C_I^{1:l} x_I)^T u^i$ in (3.16). In order to address this issue, we propose the use of safe bounds on the dual variable in PILP cases, which involves the initialization and updating of these bounds.

The initialization of upper and lower bounds on ζ can be achieved through the solution of ℓ separate MILPs, representing the maximization and minimization cases, respectively, as follows:

$$\overline{\zeta}_h = \max\{C_I^{h:h} x_I : x_I \in X_{\text{MO}-\text{I}}\} \qquad \text{and} \qquad \underline{\zeta}_h = \min\{C_I^{h:h} x_I : x_I \in X_{\text{MO}-\text{I}}\},$$

for each $h = 1, \ldots, \ell$ where $X_{\text{MO}-\text{I}} = \{x_I : A_I x_I = b, x_I \in \mathbb{Z}_+^r\}$. It is worth noting that, in order to reduce computational costs, the LP relaxation of $X_{\text{MO}-\text{I}}$ could be utilized in place of the MILPs mentioned above, although this may result in weaker bounds. Either way, we have the bounds

$$\underline{\zeta} \leq \zeta \leq \overline{\zeta},$$

satisfied for all $\zeta = C_I^{1:\ell} x_I$ with $x_I \in X_{\text{MO}-\text{I}}$.

As an alternative approach, we have $\zeta = C_I^{1:l} x_I$, so we can also derive bounds on ζ using bounds on x_I. While it is assumed that $X_{\text{MO}-\text{I}}$ is bounded, an additional computation may be necessary to obtain explicit bounds on x_I. It is not clear *a priori* which of these two approaches—bounding ζ or bounding x_I—will result in a tighter outer approximation within the MINLP solver.

It may also be necessary to establish valid bounds on dual variables u. Note that upper bounds on u of zero are already included via (3.19); the challenge is determining valid lower bounds, denoted as $\underline{u}_j^i$. It is assumed without loss of generality that u_j^i in an optimal solution for IP-NLP^k satisfies

$$u_j^i = \begin{cases} 0, & (C_I^{1:\ell} x_I^i)_j \leq \zeta_j, \\ \underline{u}_j^i, & \text{otherwise}, \end{cases}$$

for each $j = 1, \ldots, l$, where $(C_I^{1:\ell})_j$ denotes the jth row of $C_I^{1:\ell}$. In the case that for some $i \in \{0, 1, \ldots, k-1\}$ we have $C_I^{1:\ell} x_I^i \not\leq \zeta$, to avoid impact on the optimal value of MINLP^k in the

pure integer cases, we may require that

$$c_I^0 x_I^i + (\zeta - C_I^{1:\ell} x_I^i)^T u^i \geq U,$$

which can be assured if

$$(\zeta_j - (C_I^{1:\ell} x_I^i)_j) u_j^i \geq U - c_I^0 x_I^i,$$

or, equivalently

$$u_j^i \leq \frac{U - c_I^0 x_I^i}{(\zeta_j - (C_I^{1:\ell} x_I^i)_j)}, \tag{3.42}$$

In Equation (3.42), the only variable on the right-hand side is ζ. The objective is to determine a lower bound for the variable u that is sufficiently tight such that for any possible value of ζ, there's at least one value for u so that the equation holds. As a result, the lower bound we choose for u must not be greater than the right-hand side over all possible values of ζ; otherwise, we might remove some feasible solutions from the problem, compared to the case where we set the lower bounds to negative infinity. Thus we may require that

$$\underline{u}_j^i \leq \frac{U - c_I^0 x_I^i}{(\zeta_j - (C_I^{1:\ell} x_I^i)_j)}, \tag{3.43}$$

in which, if the denominator in the RHS of Equation (3.43) is non-negative, then the RHS becomes positive, and therefore any non-positive lower bound for u will suffice. However, if the denominator in the RHS of Equation (3.43) is negative, then the RHS becomes negative, and we are looking to see how small the RHS can get so that we set the lower bound for u to at least such a value, in order to avoid removing feasible solutions for x from the problem. This requires finding the largest negative value that the denominator in the RHS of Equation (3.43) can get, resulting in the RHS getting its smallest possible value. Thus, we set the lower bound for u accordingly as follows.

$$\underline{u}_j^i = \begin{cases} 0, & \text{if the problem (LB-u) is infeasible,} \\ \frac{U - c_I^0 x_I^i}{\eta - (C_I^{1:\ell} x_I^i)_j}, & \text{otherwise,} \end{cases}$$

where η represents the optimal value of the objective function for the problem (LB-u) defined as

below

$$\begin{aligned} \max \quad & (C_I^{1:\ell} x_I)_j \\ \text{subject to} \quad & (C_I^{1:\ell} x_I)_j \leq (C_I^{1:\ell} x_I^i)_j - \tau \\ & x_I \in X_{\text{MO-I}}, \end{aligned} \tag{LB-u}$$

where we set τ to a very small value. We can solve the LP relaxation of (LB-u) to obtain a looser bound on $\underline{u}_j^i$ if solving the MILP is too computationally expensive. In the pure integer case with the integer coefficient values of the objectives, in order to reduce computational costs, we can replace $\eta - (C_I^{1:\ell} x_I^i)_j$ to -1, although this may result in weaker bounds. In this case, we set the lower bound $\underline{u}_j^i$ as follows

$$\underline{u}_j^i = \begin{cases} 0, & \text{if the problem (LB-u) is infeasible,} \\ c_I^0 x_I^i - U, & \text{otherwise.} \end{cases}$$

3.3.2 Avoiding Cycling

In the MILP setting, it is possible for cycling to occur due to numerical difficulties, resulting in the same solution being produced more than once. This can be addressed by the addition of lazy cuts explicitly disallowing repeated integer solutions. Two cases are considered: binary variables and general integer variables.

Binary Case. When all decision variables x_j are binary, the lazy cut to eliminate a previously found solution x^* is simple and linear:

$$\sum_{j:x_j^*=1} (1 - x_j) + \sum_{j:x_j^*=0} x_j \;\geq\; 1. \tag{3.44}$$

This ensures that at least one variable must differ from its value in x^*, thereby excluding the repeated solution.

Note: In this research, all integer variables in our MILP instances are binary (i.e., capped at 1). We applied this approach to add lazy constraints whenever a repeated solution was encountered during the search.

General Integer Case (e.g., $x_j \in \{0, 1, 2, \ldots\}$). For general integer variables, the no-good cut is more complex. A natural extension is

$$Q(x; x^*) \;=\; \sum_{j: x_j^* = 0} x_j \;+\; \sum_{j: x_j^* > 0} (x_j^* - x_j)^2 \;\geq\; 1. \tag{3.45}$$

This formulation ensures that for any variable x_j^* taking a positive integer value, the squared deviation $(x_j^* - x_j)^2$ penalizes mismatches, while the zero case remains linear in x_j. However, the expression is quadratic.

Alternative Approaches for General Integer Variables. Several methods have been proposed to circumvent the quadratic form:

1. **Add all no-good cuts upfront.** Introduce all cuts of the form $Q(x; x^i) \geq 1$ directly in the main model. *Pros:* Easy to implement. *Cons:* Adds K additional constraints, which may significantly increase runtime, where K is the number of efficient integer solutions found so far.

2. **Deferred enforcement.** Define an auxiliary variable y_k:

$$y_k \;=\; Q(x; x^*), \tag{3.46}$$

   and enforce $y_k \geq 1$ only when a repeated solution is detected during the callback.

3. **Full linearization.** Suppose each integer variable x_j takes values in $\{0, 1, \ldots, U_j\}$. Introduce $(U_j + 1)$ new binary variables $(y_{j0}, y_{j1}, \ldots, y_{jU_j})$ for each x_j, indicating whether $x_j = k$. Then impose

$$x_j = \sum_{k=0}^{U_j} k \cdot y_{jk}, \tag{3.47}$$

$$\sum_{k=0}^{U_j} y_{jk} = 1. \tag{3.48}$$

   With this encoding, the lazy cut becomes

$$\sum_{j} y_{(j,\, x_j^*)} \;\leq\; n - 1, \tag{3.49}$$

where n is the dimension of the integer solution x. *Pros:* Fully linear; avoids quadratic constraints. *Cons:* Increases problem size due to the additional variables and constraints (total of $\sum_j (U_j + 1)$ binary variables).

4. **Binary encoding with polyhedral strengthening (Lee [2002]).** Instead of unary expansion, represent each general integer variable in binary, introducing only $\lceil \log_2(U_j + 1) \rceil$ binary variables for $x_j \in \{0, \ldots, U_j\}$. Feasibility is then enforced via the *all-different polytope*, whose facets are described by families of valid inequalities. This approach yields a parsimonious formulation (significantly fewer variables than unary encoding), while enabling the addition of strong cutting planes within a branch-and-cut framework.

In the next section, we introduce six different implementations of the RVF Algorithm. All of these implementations are essentially built on top of the naive version, with modifications made by adjusting certain parameters of the algorithm.

3.3.3 Algorithmic Variations

In this section, we detail the variations of the base algorithm that were implemented and tested in the computational experiments. Broadly speaking, these implementations can be understood as varying along several independent axes:

1. **The objective function in the subproblem**: In each iteration, we need to solve a subproblem. This subproblem can either be solved to optimality or formulated as a feasibility problem. In the feasibility case, the objective function can be arbitrary or based on θ, which is the objective in the optimality case.

2. **The number of solutions considered in each subproblem**: For example, we can solve the problem to optimality and consider only the optimal solution (the naive version), or we can consider the first feasible solution in the feasibility case. Alternatively, we can consider both the optimal solution and all feasible solutions when the subproblem is solved to optimality. Another possibility is to solve a feasibility problem with the objective function θ, while fixing the number of feasible solutions considered in each iteration to a specified value. Yet another variant is to solve a feasibility problem in each iteration and stop the algorithm as soon as a new stability region is found.

3. **The use of a cut pool**: In each iteration, one or more cuts are generated based on the efficient feasible solutions identified. The levers discussed above essentially determine how many solutions are added and according to what criteria. However, it may not always be necessary to consider all cuts found thus far, making the design of a cut pool an important aspect of some implementations. In fact, each implementation can broadly be interpreted as a variation of a cut pool mechanism, since every idea provides a way of identifying one or multiple cuts to be carried forward into subsequent iterations.

4. **Stopping Criteria**: In each iteration, we may choose either to solve the problem to optimality or to stop the solver after obtaining a specified number of feasible solutions. This threshold, denoted by x, can be set to any value greater than or equal to 1.

By varying along these axes, we obtain multiple implementations of the RVF Algorithm. Not all possible combinations of these levers are considered in this research; however, the most relevant and promising ones are studied in detail.

Here, we summarize all parameter settings for each implementation considered. The detailed descriptions for each implementation can be found in their corresponding sections.

The first column represents the implementation number. The second column specifies the objective function of the subproblem solved at each iteration. The third column indicates the number of solutions collected during the process and added to the efficient set if they are not dominated. The fourth column reports the percentage of cuts (corresponding to the stability regions identified so far) included in the next iteration. The final column describes the stopping criterion—namely, whether the subproblem is solved to optimality or terminated early once a specified feasibility condition is met.

Table 3.1: Summary of RVF Implementations and Their Parameter Settings

Implementation	Objective	# Solutions	Cut Pool Ratio	Stopping Criteria
0	θ	1	1	Optimality
1	θ	∞	1	Optimality
2	1	1	1	Feasibility
3	θ	n (tunable)	1	Feasibility
4	θ	∞	< 1	Optimality
6	1	1	< 1	Feasibility

In the following sections, we describe each implementation in detail. Before proceeding, it is important to clarify the distinction between stability regions in the PILP and MILP cases. In the PILP case, each stability region corresponds to a discrete point. Consequently, when an efficient solution is identified and its corresponding NDP is constructed, we can verify its uniqueness by checking the NDP against the existing EF. If the NDP is not present in the current EF list, it indicates that the efficient solution corresponds to a novel stability region.

In the MILP case, multiple efficient solutions may correspond to the same stability region. We classify an efficient solution as 'novel' if its integer part is unique relative to both the existing set of integer solutions and any other solutions identified within the current iteration—including the optimal solution, when applicable. While the optimal solution may not be utilized in every implementation, it must be accounted for in those where the subproblem is solved to optimality. Since we can simply verify the uniqueness of each efficient integer solution, we add a solution to the set whenever its integer vector has not been previously identified. However, this approach inherently allows for the possibility that multiple efficient solutions may correspond to the same stability region. Although additional post-processing could be employed to group solutions belonging to the same stability region, such procedures would significantly increase computational complexity and were therefore not implemented in this study

Implementation 1: Using feasible solutions found in each iteration of the branch-and-bound algorithm (Objective: θ, # Solutions: ∞, Cut Ratio: 1, Stopping: Optimality). In the RVF Algorithm, the problem solved in each iteration by the branch-and-bound (BB) algorithm aims to maximize the difference between the upper approximation and the true value function, denoted by θ. In iteration k, the optimal efficient solution, corresponding to the maximum value of θ, is added to the set $\mathcal{S}^k$. During the execution of the BB algorithm in each iteration, several feasible solutions that improve the lower bound on θ may be found. These feasible solutions, while not necessarily optimal, may correspond to new stability regions if they are converted to efficient solutions.

The implementation proposed here is to add the distinct efficient solutions corresponding to these feasible solutions to the set $\mathcal{S}^k$, which includes the integer parts of the NDPs found thus far. However, it is important to note that because this implementation aims to maximize θ without a strictly positive lower bound, the BB solver may identify feasible solutions where $\theta = 0$. Such solutions

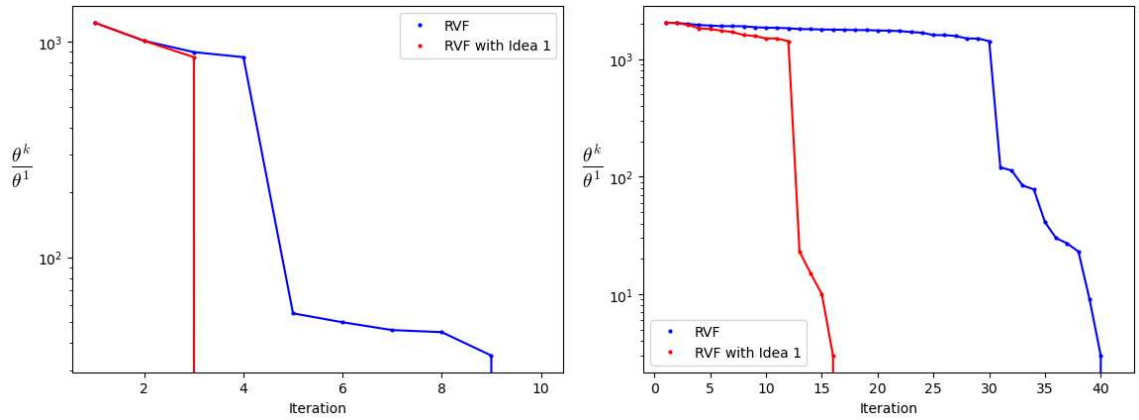
correspond to efficient solutions or stability regions that already exist in $\mathcal{S}^k$. Furthermore, even for solutions where $\theta > 0$, multiple feasible solutions identified within the same search tree may map to the same new stability region.

Therefore, on the one hand, we might convert a feasible solution to an efficient solution whose stability region already exists or is redundant relative to another solution found in the same iteration. If a constraint requiring θ to be greater than a small positive threshold (i.e., $\theta \geq \delta$) were added to the subproblem, the solver would be prevented from rediscovering these $\theta = 0$ regions. Without such a constraint, we may incur the extra work of converting these solutions and adding unnecessary cuts, resulting in a more complex model. On the other hand, we might be able to reduce the total number of iterations in the algorithm by adding the cuts corresponding to new stability regions. Therefore, there is a tradeoff that should be examined computationally on whether adding the cuts corresponding to all efficient solutions (obtained while solving each iteration and whose efficient solutions do not already exist) improves the overall running time.

In the pure integer cases, since each stability region has only one unique NDP, instead of adding the distinct efficient integer solution to the set $\mathcal{S}_F^{k+1}$, we would better add the efficient integer solutions with corresponding different NDPs. This avoids adding multiple optimal solutions corresponding to each stability region, resulting in a less complex model. An outline of the improved RVF Algorithm is provided in RVF Algorithm: Implementation 1.

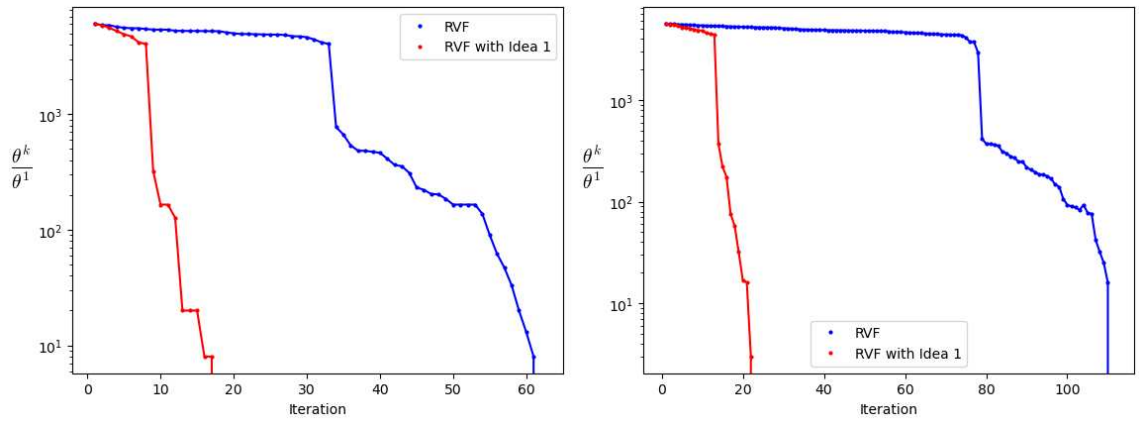
It should be noted that for MILP cases in Implementation 1, a constraint was added to ensure that θ remains greater than a small positive threshold. This was done to prevent the solver from identifying feasible solutions for which $\theta = 0$.

Figures 3.4 and 3.5 illustrate the change in relative θ (i.e., θ at each iteration divided by the θ of the first iteration) for the nonlinear version of the RVF Algorithm as the algorithm proceeds for both the RVF Algorithm and the RVF Algorithm With Implementation 1. The results demonstrate that the RVF Algorithm with Implementation 1 can achieve a zero value for θ with fewer iterations. The figures for the linear version, comparing the relative θ for the RVF Algorithm and the RVF Algorithm with Implementation 1, are similar to those for the nonlinear version. In the labels of these two figures, Idea 1 refers to Implementation 1.



(a) 2spp100_300A *(b) 2mis101_300D*

Figure 3.4: Normalized approximation gap vs. iteration number for the nonlinear version of the RVF Algorithm and RVF Algorithm With Implementation 1 on SPP instances.



(a) KP_p-3_n-20_ins-5 *(b) KP_p-5_n-20_ins-5*

Figure 3.5: Normalized approximation gap vs. iteration number for the nonlinear version of the RVF Algorithm and RVF Algorithm With Implementation 1 on Knapsack instances.

Algorithm RVF Algorithm: Implementation 1 : Algorithm for constructing the RVF and the associated EF using all identified efficient solutions

Input: X_{MO}, $C \in \mathbb{Q}^{(l+1)\times n}$, U an upper bound on $z(\zeta)$ over $\zeta \in \mathcal{C}$, *thetaGap*.
Output: $\mathcal{S}^k$ such that $z(\zeta) = \bar{z}^k(\zeta) = \min_{x_I \in \mathcal{S}^k} \bar{z}(\zeta;\ x_I), \quad \forall \zeta \in \mathcal{C}$.

Initialize $\bar{z}^0(\zeta) = U$ for all $\zeta \in \mathbb{R}^l$, $k = 0$, $\mathcal{S}^0 = \emptyset$, $\theta^0 = +\infty$.
while $\theta^k >$ *thetaGap* **do**
  Determine $(x_I^{k+1}, x_C^{k+1}) \in \underset{(x_I, x_C) \in X_{\text{MO}}}{\arg\max} \ \bar{z}^k \left(C_I^{1:l} x_I + C_C^{1:l} x_C\right) - c_I^0 x_I - c_C^0 x_C$.
  Convert (x_I^{k+1}, x_C^{k+1}) and all feasible solutions found during the optimization process into efficient solutions using the one-stage (3.36) or two-stage (3.37)–(3.40) conversion; store them in $\mathcal{S}_{\text{tmp}}$.
  if *PILP Case* **then**
    Filter $\mathcal{S}_{\text{tmp}}$: retain solutions only if the NDP is unique relative to the current EF and other solutions in the batch.
  end
  else if *MILP Case* **then**
    Filter $\mathcal{S}_{\text{tmp}}$: retain solutions only if the integer vector is unique relative to $\mathcal{S}^k$ and other solutions in the batch.
  end
  Set $\mathcal{S}^{k+1} \leftarrow \mathcal{S}^k \cup \{x_I^* \mid x^* \in \mathcal{S}_{\text{tmp}}\}$.
  Set $\theta^{k+1} = \bar{z}^k \left(C_I^{1:l} x_I^{k+1} + C_C^{1:l} x_C^{k+1}\right) - c_I^0 x_I^{k+1} - c_C^0 x_C^{k+1}$.
  $\bar{z}^{k+1}(\zeta) = \min\left(\{\bar{z}^k(\zeta)\} \cup \{\bar{z}(\zeta; x_I^*) \mid x^* \in \mathcal{S}_{\text{tmp}}\}\right)$ for all $\zeta \in \mathcal{C}$.
  $k \leftarrow k + 1$.
end

Implementation 2: Retrieving the first feasible solution with a positive θ in each iteration of the BB algorithm with the feasibility problem (MINLP^k-Feas) **(Objective: 1, # Solutions: 1, Cut Ratio: 1, Stopping: Feasibility).** The optimization problem (3.8)–(3.13) to be solved with the BB algorithm maximizes the θ, which is the difference between the upper approximation and the real value function. While maximizing θ is justified if the RVF Algorithm is viewed as an approximate method for constructing the value function or the EF, it may not be necessary if the algorithm is intended to be used as an exact method for these purposes. One strategy involves terminating the nonlinear solver as soon as a new stability region is identified.

Theoretically, in both the PILP and MILP contexts, any feasible solution yielding a positive θ value should identify a novel stability region. While the linearized structure of the PILP subproblem ensures this property holds, the nonlinear nature of the MILP subproblem—combined with inherent numerical sensitivities—can result in "false positives". In such instances, a solution with $\theta > 0$ may, upon conversion to an efficient solution, map to a stability region that has already been identified. To resolve this, we implement a callback mechanism utilizing lazy cuts, as detailed in Section 3.5.2.

Within this framework, every identified solution is converted and verified against the current set of efficient solutions. If the solution is found to be redundant, the solver applies a lazy cut to exclude that specific feasible point and resumes the search. This mechanism prevents the algorithm from entering an infinite loop and ensures that only genuinely novel efficient solutions are incorporated. The incorporation of a callback strategy with lazy cuts is not specific to Implementation 2; rather, it is applied to all implementations in the MILP case.

The specific feasibility problem solved during each iteration of the BB algorithm under this approach is presented below.

$$
\begin{aligned}
\theta^{k+1} = \max \quad & 1 \\
\text{subject to} \quad & \theta + c_I^0 x_I + c_C^0 x_C \leq c_I^0 x_I^i + \\
& (C_I^{1:l} x_I + C_C^{1:l} x_C - C_I^{1:l} x_I^i)^\top u^i + (b - A_I x_I^i)^\top v^i, \quad i = 1, \ldots, k \\
& \theta + c_I^0 x_I + c_C^0 x_C \leq U \\
& (u^i, v^i) \in \mathcal{P}_D, \qquad i = 1, \ldots, k \\
& (x_I, x_C) \in X_{\text{MO}} \\
& \theta \geq \delta \\
& \theta \in \mathbb{R},
\end{aligned}
\tag{MINLP^k-Feas}
$$

where the θ is not smaller than the parameter δ; we set δ to a very small positive value to avoid getting the previous solution. We encounter a similar feasibility problem in both the nonlinear and linear versions of the PILP cases, which we refer to as PILP-NLP^k-Feas and PILP-LP^k-Feas, respectively. These are obtained from PILP-NLP^k and PILP-LP^k by substituting the objective function, changing from $\max \theta$ to $\max 1$, and adding a new constraint $\theta \geq \delta$, where δ is a small number.

In iteration k of the RVF Algorithm: Implementation 2, the feasibility problem defined in PILP-LP^k-Feas (or MINLP^k-Feas and PILP-NLP^k-Feas) is solved. If this problem is feasible, the resulting solution is converted into an efficient solution using either the one-stage conversion in (3.36) or the two-stage conversion defined in (3.37)–(3.40). In the PILP case, this process is guaranteed to yield an efficient solution corresponding to a novel stability region. However, in the MILP case, a callback mechanism is required. If the conversion yields a solution that corresponds to an existing efficient solution, a lazy cut is added to exclude that feasible point from the search space. The solver then resumes the search until the conversion process identifies a novel solution, relative to the current set

of identified efficient solutions. If the feasibility problem is infeasible, the algorithm terminates and returns the set $\mathcal{S}^k$, which is required to construct the entire value function or EF. The effectiveness of this approach should be evaluated computationally to determine whether it leads to improvements in the running time of the RVF Algorithm. A high-level description of the algorithm is provided in RVF Algorithm: Implementation 2. Note that in the pseudocode we refer to PILP-LP^k-Feas, but it may also correspond to MINLP^k-Feas or PILP-NLP^k-Feas.

Algorithm RVF Algorithm: Implementation 2 : Algorithm for constructing the RVF and the associated EF using feasibility-based termination

Input: X_{MO}, $C \in \mathbb{Q}^{(l+1)\times n}$, U an upper bound on $z(\zeta)$ over $\zeta \in \mathcal{C}$, δ a small positive threshold.
Output: $\mathcal{S}^k$ such that $z(\zeta) = \bar{z}^k(\zeta) = \min_{x_I \in \mathcal{S}^k} \bar{z}(\zeta;\ x_I), \quad \forall \zeta \in \mathcal{C}$.

Initialize $\bar{z}^0(\zeta) = U$ for all $\zeta \in \mathbb{R}^l$, $k = 0$, $\mathcal{S}^0 = \emptyset$.
while *True* **do**
  Solve the feasibility problem (PILP-LP^k-Feas, PILP-NLP^k-Feas, or MINLP^k-Feas)
  if *the problem is infeasible* **then**
    Break
  end
  if *PILP Case* **then**
    Retrieve the first feasible solution (x_I^{k+1}, x_C^{k+1}) where $\theta \geq \delta$
  end
  else if *MILP Case* **then**
    Execute a **callback** to convert feasible solutions found during the search
    while *converted solution exists in* $\mathcal{S}^k$ **do**
      Add a **lazy cut** to exclude the current feasible point
      Resume the search for a new feasible solution
    end
    Retrieve the first novel feasible solution (x_I^{k+1}, x_C^{k+1})
  end
  Convert (x_I^{k+1}, x_C^{k+1}) to an efficient solution using one-stage (3.36) or two-stage (3.37)–(3.40) conversion
  Set $\mathcal{S}^{k+1} \leftarrow \mathcal{S}^k \cup \{x_I^{k+1}\}$
  $\bar{z}^{k+1}(\zeta) = \min\{\bar{z}^k(\zeta), \bar{z}(\zeta; x_I^{k+1})\}$ for all $\zeta \in \mathcal{C}$
  Set $k \leftarrow k + 1$
end

Implementation 3: Retrieving the first n feasible solution with a positive θ in each iteration of the BB algorithm with maximizing θ as the objective (Objective: θ, # Solutions: n, Cut Ratio: 1, Stopping: Feasibility). This implementation is similar to Implementation 2 but involves maximizing θ as the objective function rather than using an arbitrary objective. We halt the solving process once it identifies n feasible solutions (n is a

number in the list $[1, 2, 3, 4, 5]$) featuring a positive θ value. The parameter n will be fine-tuned in the computational experiments, specifically in Section 3.5.1 for PILPs and Section 3.5.2 for MILPs. Please refer to Figure 3.8 and Figure 3.18 for the performance profiles comparing the execution times of Implementation 3 for PILPs and MILPs, respectively.

After solving the problem in each iteration, two scenarios may occur. If the solution is infeasible, it indicates that the solving procedure must be halted as the algorithm has discovered all the necessary integer parts. However, if the solution is feasible, the feasible solutions are transformed into an efficient solution, and the efficient integer parts are incorporated into the set $\mathcal{S}^k$. Subsequently, the algorithm advances to the next iteration utilizing the expanded set $\mathcal{S}^{k+1}$. Note that when $n > 1$, we must verify whether the corresponding feasible integer solutions represent distinct NDPs in the PILP setting. This is done by comparing each solution's associated NDP against the EF constructed thus far and evaluating them individually. For instance, when n = 3, it is possible that two feasible solutions map to the same NDP. For MILPs, this process is more intricate. In such cases, we can only determine whether an integer-feasible solution matches any element of the current efficient set; however, we may fail to detect when multiple integer-feasible solutions belong to the same stability region. Consequently, this may introduce additional complexity, as we could end up adding feasible solutions that actually correspond to the same stability region.

The main difference between Implementation 2 and Implementation 3 is the objective function used to solve the problem in each iteration. In Implementation 3, the use of the main objective function of maximizing θ may lead to the discovery of better efficient integer solutions with larger values of θ. The pseudocode for Implementation 3 is similar to that of RVF Algorithm: Implementation 2, with the objective function in the feasibility problem set to θ rather than an arbitrary objective.

Implementation 4: Cut Pool (Objective: θ, # Solutions: ∞, Cut Ratio: < 1, Stopping: Optimality). In this concept, we are introducing a methodology to identify a subset of the cuts acquired thus far, rather than considering all the cuts. This approach aims to reduce the size and complexity of the problem. One potential approach involves selecting a subset of cuts that relate to stability regions positioned sufficiently far from one another.

To accomplish this, we can calculate the Euclidean distance between the points on the VF graph obtained thus far. To elaborate further, we denote these points as $(c_I^0 x_I^k + c_C^0 x_C^k,\ C_I^{1:l} x_I^k + C_C^{1:l} x_C^k)$,

where (x_I^k, x_C^k) represents the optimal solution specific to iteration k.

To achieve this goal, during the k-th iteration, we consider $\min(\text{int}(a \times \max(\text{totalIter}, \text{num_ndp})), \text{num_ndp})$ cuts from those identified thus far. Here, a can be assigned a value less than 1, totalIter denotes the total number of iterations conducted so far, and num_ndp represents the number of efficient integer solutions found.

The reason for this behavior is that the number of NDPs typically increases more quickly than the total number of iterations, since in each iteration, in addition to the optimal solution, we also consider the cuts generated for all feasible solutions that have not yet been dominated. However, if the total number of iterations grows faster than the number of NDPs—indicating a large number of unsuccessful iterations—then we scale the ratio by the total number of iterations. This adjustment increases the number of cuts considered and helps reduce the probability of additional unsuccessful iterations.

The objective could be to maximize the cumulative distances between the corresponding points on the VF graph (i.e., the objective space). This constitutes an optimization problem; nevertheless, we can employ a greedy approach to find a solution. The ratio of cuts to be considered, which is a tunable parameter, was evaluated through experiments in Section 3.5.1 for PILPs and Section 3.5.2 for MILPs. The performance profile comparing the execution time of Implementation 4 with different cut ratios for PILPs is shown in Figure 3.9, and for MILPs, it is shown in Figure 3.19.

Firstly, the RVF Algorithm is executed for 5 iterations without considering the subset selection. Subsequently, the subset selection process begins. The initiation of this process involves choosing any point in the objective space, specifically, the NDP set, as the initial element of our subset. In each subsequent iteration, we commence with the NDP identified in the previous iteration to ensure consecutive selection of distinct NDPs. Within the iteration, we iterate over the remaining points that haven't yet been incorporated into the subset. For each of these points, the distance is calculated (using methods like the Euclidean distance) from all the points already encompassed within the subset.

Regarding points not yet present in the subset, the minimum distance from the points already included is computed. Finally, the point to be added to the subset is chosen based on the criterion that its minimum distance is the greatest among all the candidate points. This process continues until $\min(\text{int}(a \times \max(\text{totalIter}, \text{num_ndp})), \text{num_ndp})$ elements are identified, where a is a tunable

parameter assigned a value less than 1. Based on our computational study, we selected a from the set $[0.1, 0.2, \ldots, 0.9]$ for PILPs and $[0.1, 0.5, 0.9]$ for MILPs. While this approach does not guarantee an optimal solution, it provides a nearly optimal subset of points that are relatively well-separated.

Next, using the set obtained from this procedure, the algorithm is executed with the new set of integer parts, denoted as $\mathcal{S}$. If the new integer part is already present in the list of all cuts obtained so far, it is not added to the set. In the subsequent iteration, all the cuts obtained so far are added, and the RVF Algorithm is executed. However, if the new integer part is not within the set, it is added, and the subset generator is invoked for the following iteration.

It is important to note that this procedure applies identically to both PILP and MILP cases. For PILPs, to determine whether the integer part is connected to the new stability region, we examine whether the corresponding NDP is present within the EF. In PILP instances, each stability region corresponds to a single NDP, which is not necessarily the case for MILP scenarios.

This iterative process continues until θ reaches a sufficiently small value, and in the preceding iteration, all the cuts—not just a subset—were considered.

Therefore, there is a tradeoff that should be examined computationally on whether this strategy improves the overall running time. The concept of the cut pool can be integrated with any of the aforementioned implementations to create an optimized version of the RVF Algorithm. However, we will implement the cut pool in Implementation 1 (solving to optimality and considering both optimal and feasible solutions for inclusion in our efficient set) of the RVF Algorithm to observe the potential improvements it can provide to the overall runtime. A high-level description of the algorithm is provided in RVF Algorithm: Implementation 4.

Algorithm RVF Algorithm - main ($\mathcal{S}_{\text{it}}$)

$\bar{z}^k(\zeta) = \min_{x_I^i \in \mathcal{S}_{\text{it}}} \{\bar{z}(\zeta; x_I^i)\}$ for all $\zeta \in \mathcal{C}$.
Determine $(x_I^{k+1}, x_C^{k+1}) \in \underset{(x_I, x_C) \in X_{\text{MO}}}{\arg\max} \left(\bar{z}^k \left(C_I^{1:l} x_I + C_C^{1:l} x_C\right) - (c_I^0 x_I + c_C^0 x_C)\right)$.
Set $\theta^{k+1} = \bar{z}^k \left(C_I^{1:l} x_I^{k+1} + C_C^{1:l} x_C^{k+1}\right) - (c_I^0 x_I^{k+1} + c_C^0 x_C^{k+1})$ for the optimal solution (x_I^{k+1}, x_C^{k+1}).
return θ^{k+1}, (x_I^{k+1}, x_C^{k+1}), set of all feasible solutions found.

Algorithm genSubIntLst($\mathcal{S}^k_{\text{all}}$, initPoint, initPointIdx, totalIter, k, a) : Generate a Subset of Integer Parts List

num_ndp = len ($\mathcal{S}^k_{\text{all}}$)
if *num_ndp* $\leq$ *5* **then**
  return $\mathcal{S}^k_{\text{all}}$.
end
Set tempAllIntPartDict $\leftarrow$ allIntPartDict.copy().
Set subIntPartDict $\leftarrow$ {}.
Set subIntPartDict[initPointIdx] $\leftarrow$ initPoint.
Delete the item corresponding to the initial point (initPoint) from tempAllIntPartDict.
for *i* ***in*** *range*(min(*int*($a \times$ max(*totalIter*, *num_ndp*)) $-$ 1, *num_ndp*) **do**
  Set max_min_distance $\leftarrow$ **-inf**.
  for *k_all, v_all in tempAllIntPartDict* **do**
    Set min_distance $\leftarrow$ **inf**.
    for *k_subset in subIntPartDict* **do**
      Set distance $\leftarrow$ Calculate the Euclidean norm (L2 norm) between the points on the graph of VF corresponding to the indices k_all and k_subset.
      if *distance* $<$ *min_distance* **then**
        Set min_distance $\leftarrow$ distance.
    end
    if *min_distance* $>$ *max_min_distance* **then**
      Set max_min_distance $\leftarrow$ min_distance.
      Set selected_point $\leftarrow$ k_all.
    end
  end
  Add the integer part corresponding to the selected_point to subIntPartDict and delete that from tempAllIntPartDict.
end
return the list of values of subIntPartDict.

Algorithm RVF Algorithm: Implementation 4 : Algorithm for constructing the RVF using a Cut Pool strategy and case-specific filtering

Input: X_{MO}, $C \in \mathbb{Q}^{(l+1)\times n}$, U an upper bound on $z(\zeta)$ over $\zeta \in \mathcal{C}$, timeLimit, thetaGap.
Output: $\mathcal{S}^k$ such that $z(\zeta) = \bar{z}^k(\zeta) = \min_{x_I \in \mathcal{S}^k} \bar{z}(\zeta; x_I),\ \forall \zeta \in \mathcal{C}$.

Initialize $\bar{z}^0(\zeta) \leftarrow U$ for all $\zeta \in \mathbb{R}^l$, $k \leftarrow 0$, totalIter $\leftarrow 0$, $\mathcal{S}^0 = \emptyset$, $\mathcal{S}^0_{\text{all}} = \emptyset$, $\theta^0 = \infty$, shouldCallGenSubIntLst $\leftarrow$ **True**, allIntPart $\leftarrow$ **False**, initPoint $\leftarrow \emptyset$, initPointIdx $\leftarrow \emptyset$, $a \leftarrow$ some value from $\{0.1, 0.2, \ldots, 0.9\}$, EF (set of NDPs) $\leftarrow \emptyset$.
while ***True*** **do**
  if *shouldCallGenSubIntLst* **then**
    $\mathcal{S}^k \leftarrow$ **genSubIntLst**$(\mathcal{S}^k_{\text{all}}, \text{initPoint}, \text{initPointIdx}, \text{totalIter}, k, a)$ $allIntPart \leftarrow$ **False**
  else
    $\mathcal{S}^k \leftarrow \mathcal{S}^k_{\text{all}}$ $shouldCallGenSubIntLst \leftarrow$ **True** $allIntPart \leftarrow$ **True**
  end
  $\theta, (x_I, x_C) =$ **RVF Algorithm - main**$(\mathcal{S}^k)$
  if $(\theta < thetaGap$ ***and*** $allIntPart)$ ***or*** $(elapsedTime > timeLimit)$ **then**
    return $\mathcal{S}^k_{\text{all}}$
  end
  Convert all feasible solutions found during the optimization process into efficient solutions via one-stage (3.36) or two-stage (3.37)–(3.40) conversions; store in $\mathcal{S}_{\text{tmp}}$
  if *PILP Case* **then**
    Filter $\mathcal{S}_{\text{tmp}}$: retain solutions only if their NDP is unique relative to EF and the current batch
  end
  else if *MILP Case* **then**
    Filter $\mathcal{S}_{\text{tmp}}$: retain solutions only if their integer vector is unique relative to $\mathcal{S}^k_{\text{all}}$ and the current batch
  end
  if $|\mathcal{S}_{tmp}| = 0$ ***and*** *not allIntPart* **then**
    $shouldCallGenSubIntLst \leftarrow$ **False** **continue**
  end
  For each remaining solution in $\mathcal{S}_{\text{tmp}}$, update EF with its NDP and $\mathcal{S}^k_{\text{all}}$ with its integer part
  Set $initPoint \leftarrow \mathcal{S}_{\text{tmp}}[-1]$ Set $k \leftarrow k + |\mathcal{S}_{\text{tmp}}|$ Set $initPointIdx \leftarrow k$ $totalIter \leftarrow totalIter + 1$
end

Implementation 5: Considering the lower bound of the RVF instead of the true RVF to construct the RVF. At every iteration in the naive version of the RVF Algorithm, our objective is to determine the largest difference between the true value function and its upper approximation. By reducing the gap to zero between the true value function and its upper approximation, we can construct the VF. Consequently, during the intermediate stages of iteration, we opt to work with the lower approximation of the VF instead of the true VF, simplifying the problem-solving process. However, in the final iteration, it becomes necessary to consider the true value function to ensure that the approximation is exact.

Each relaxation of the value function would constitute a valid lower bound. This relaxation involves either the integrality constraints or the constraints with fixed right-hand sides, denoted as $A_I x_I + A_C x_C = b$ in the context of the constraint within the model (RVF). We opted for the latter choice. We refer to the relaxed feasible region as $X_{\text{MO}}^{\text{relax}}$. To generate a subset of the primal constraints, we call the function "genSubPrimalConst(primalRatio, numConst, randSeed)" with the following inputs: "primalRatio", which represents the ratio of primal constraints to be considered; "numConst", indicating the number of original primal constraints; and "randSeed", serving as the fixed random seed.

In this concept, we replace the true value function with a lower bound by considering a superset, which involves removing certain constraints with fixed RHSs. The upper bounding functions (RHS of (3.9)) remain unchanged, maintaining the original coefficient matrix A and RHS vector b. Concerning the integer parts acquired through the substitution of the true value function with a lower bound, if they indeed reside within the set $\mathcal{S}_I$, that is advantageous. Nevertheless, if they do not, it should not present an issue. In situations where the CR is infeasible, the dual problem becomes unbounded. Nonetheless, due to the presence of a global upper bound, i.e., U, on theta, this situation is not expected to be problematic. This implies that it is possible to identify some integer parts that do not belong to $\mathcal{S}_I$, but such instances are deemed acceptable. To locate any missing integer parts, the reintroduction of the previously removed primal constraints to the problem is necessary. This concept can be implemented in various ways, but we will consider two versions of it here. The first version involves the following steps:

1. Removal of a specified percentage of primal constraints ("primalRatio" in the pseudocode is a parameter that is subject to tuning).

2. Execution of the RVF Algorithm for a fixed number of iterations called batch_num (which is another parameter subject to tuning) with the same primal constraints.
3. After completing Step 2, the reintegration of all constraints ($Ax = b$) and solving the problem to optimality.
4. Monitor how theta has changed relative to its initial value.
5. Repeat steps 1 to 4 iteratively until theta reaches a small value, such as (1/10) of the initial theta value, i.e., $\theta^k/\theta^1 \leq relativeGap$. Please note that we obtain the value of theta during the iterations when we invoke the RVF with all the original primal constraints.
6. Subsequently, the algorithm continues with the original primal constraints from that point onward.

A high-level description of this algorithm is provided in RVF Algorithm: Implementation 5 - v1.

The second version involves executing the RVF Algorithm using a modified problem where certain primal constraints are removed while monitoring the theta value. When the theta value exhibits minimal change, denoted by $(|\theta^k - \theta^{k-1}|)/\theta^{k-1} \leq relativeGap$, it indicates that the algorithm has converged to the minimum attainable value of theta, considering the lower bound instead of the true value function. Consequently, the algorithm transitions back to the original problem containing all primal constraints. It's essential to note that before switching to the original problem, theta assumes a slightly different meaning; here, it signifies the largest difference between the upper approximation and the lower approximation, distinct from its original interpretation, which relates to the largest difference between the upper approximation and the true value function. For a high-level overview of this algorithm, refer to RVF Algorithm: Implementation 5 - v2.

In these two pseudocodes of Implementation 5, there are two parameters called "allPrimalAfterThisIter" and "allPrimal" that need to be defined. We set the former one to True when we want to switch to considering all the primal constraints from that iteration onward; otherwise, we set it to False. When the latter parameter is True, it means that all the primal constraints were considered in the previous iteration; otherwise, they were not. Additionally, we set the "relativeGap" to 0.1 in both versions of Implementation 5.

During the iterations where we consider the LB of the VF, one question may arise: whether to consistently employ the same subset of primal constraints or a different subset of primal constraints

during each iteration. This is an empirical question that should be addressed computationally. However, to ensure a nonincreasing value of theta throughout the entire algorithm, the subset of primal constraints must remain unchanged.

Algorithm RVF Algorithm: Implementation 5 - v1 : Algorithm for constructing the RVF and the associated EF using lower bound approximations

Input: X_{MO}, $X_{\text{MO}}^{\text{relax}}$, $C \in \mathbb{Q}^{(l+1)\times n}$, U an upper bound on $z(\zeta)$ over $\zeta \in \mathcal{C}$, batch_num $= 3$, thetaGap, relativeGap.

Output: $\mathcal{S}^k$ such that $z(\zeta) = \bar{z}^k(\zeta) = \min_{x_I \in \mathcal{S}^k} \bar{z}(\zeta; x_I), \quad \forall \zeta \in \mathcal{C}$.

Initialize $\bar{z}^0(\zeta) = U$, $k = 0$, $\mathcal{S}^0 = \emptyset$, $\theta^0 = \infty$, $\theta^1 = 0$, allPrimalAfterThisIter = **False**, $EF = \emptyset$
while ***True*** **do**
  if *allPrimalAfterThisIter* ***or*** *($k > 0$* ***and*** *k* (mod *batch_num*) $== 0$*)* **then**
    Set $X'_{\text{MO}} \leftarrow X_{\text{MO}}$ allPrimal $\leftarrow$ **True**
  else
    Set $X'_{\text{MO}} \leftarrow X_{\text{MO}}^{\text{relax}}$ allPrimal $\leftarrow$ **False**
  end
  Determine $(x_I^{k+1}, x_C^{k+1}) \in \underset{(x_I, x_C) \in X'_{\text{MO}}}{\arg\max} \; \bar{z}^k \left(C_I^{1:l} x_I + C_C^{1:l} x_C\right) - c_I^0 x_I - c_C^0 x_C$
  Set $\theta^{k+1} = \bar{z}^k \left(C_I^{1:l} x_I^{k+1} + C_C^{1:l} x_C^{k+1}\right) - c_I^0 x_I^{k+1} - c_C^0 x_C^{k+1}$
  if $\theta^{k+1} <$ *thetaGap* ***and*** *allPrimal* **then**
    Break
  end
  Convert (x_I^{k+1}, x_C^{k+1}) and any feasible solutions found into efficient solutions via one-stage (3.36) or two-stage (3.37)–(3.40) conversion; store in $\mathcal{S}_{\text{tmp}}$
  if *PILP Case* **then**
    Filter $\mathcal{S}_{\text{tmp}}$: keep only solutions with a unique **NDP** relative to EF and the current batch
  end
  else if *MILP Case* **then**
    Filter $\mathcal{S}_{\text{tmp}}$: keep only solutions with a unique **integer vector** relative to $\mathcal{S}^k$ and the current batch
  end
  For each solution in $\mathcal{S}_{\text{tmp}}$, update EF with its NDP and $\mathcal{S}^{k+1}$ with its integer part
  $\bar{z}^{k+1}(\zeta) = \min\left(\{\bar{z}^k(\zeta)\} \cup \{\bar{z}(\zeta; x_I^*) \mid x^* \in \mathcal{S}_{\text{tmp}}\}\right)$ for all $\zeta \in \mathcal{C}$
  if *allPrimal* ***and*** *($\theta^{k+1}/\theta^1 \leq$ relativeGap)* **then**
    Set allPrimalAfterThisIter = **True**
  end
  Set $k \leftarrow k + 1$
end

Implementation 6: Combining Implementation 4 (Cut Pool) with Implementation 2 (Objective: 1, # Solutions: 1, Cut Ratio: < 1, Stopping: Feasibility). We have suggested various implementations thus far, but it is also worth considering the combination of

Algorithm RVF Algorithm: Implementation 5 - v2 : Algorithm for constructing the RVF and the associated EF using lower bound approximations

Input: X_{MO}, $X_{\text{MO}}^{\text{relax}}$, $C \in \mathbb{Q}^{(l+1)\times n}$, U an upper bound on $z(\zeta)$ over $\zeta \in \mathcal{C}$, *thetaGap*, *relativeGap*.
Output: $\mathcal{S}^k$ such that $z(\zeta) = \bar{z}^k(\zeta) = \min\limits_{x_I \in \mathcal{S}^k} \bar{z}(\zeta; x_I), \quad \forall \zeta \in \mathcal{C}$.

Initialize $\bar{z}^0(\zeta) = U$, $k = 0$, $\mathcal{S}^0 = \emptyset$, $\theta^0 = \infty$, $allPrimalAfterThisIter$ = **False**, $EF = \emptyset$, Set $X'_{\text{MO}} \leftarrow X_{\text{MO}}^{\text{relax}}$
while ***True*** **do**
  if *allPrimalAfterThisIter* **then**
    Set $X'_{\text{MO}} \leftarrow X_{\text{MO}}$
  else
    Set $X'_{\text{MO}} \leftarrow X_{\text{MO}}^{\text{relax}}$
  end
  Determine $(x_I^{k+1}, x_C^{k+1}) \in \underset{(x_I, x_C) \in X'_{\text{MO}}}{\arg\max} \; \bar{z}^k \left(C_I^{1:l} x_I + C_C^{1:l} x_C\right) - c_I^0 x_I - c_C^0 x_C$
  Set $\theta^{k+1} = \bar{z}^k \left(C_I^{1:l} x_I^{k+1} + C_C^{1:l} x_C^{k+1}\right) - c_I^0 x_I^{k+1} - c_C^0 x_C^{k+1}$
  if $\theta^{k+1} <$ *thetaGap* ***and*** *allPrimalAfterThisIter* **then**
    Break
  end
  Convert (x_I^{k+1}, x_C^{k+1}) and all feasible solutions found during the optimization into efficient solutions; store in $\mathcal{S}_{\text{tmp}}$
  if *PILP Case* **then**
    Filter $\mathcal{S}_{\text{tmp}}$: keep solutions only if the **NDP** is unique relative to EF and the current batch
  end
  else if *MILP Case* **then**
    Filter $\mathcal{S}_{\text{tmp}}$: keep solutions only if the **integer vector** is unique relative to $\mathcal{S}^k$ and the current batch
  end
  For each solution in $\mathcal{S}_{\text{tmp}}$, update EF with its NDP and $\mathcal{S}^{k+1}$ with its integer part
  $\bar{z}^{k+1}(\zeta) = \min\left(\{\bar{z}^k(\zeta)\} \cup \{\bar{z}(\zeta; x_I^*) \mid x^* \in \mathcal{S}_{\text{tmp}}\}\right)$ for all $\zeta \in \mathcal{C}$
  if $k > 0$ ***and not*** *allPrimalAfterThisIter* **then**
    if $(|\theta^{k+1} - \theta^k|)/\theta^k \leq relativeGap$ **then**
      Set $allPrimalAfterThisIter$ = **True**
    end
  end
  Set $k \leftarrow k + 1$
end

implementations that do not conflict with each other. We will conduct computational experiments for each individual implementation and compare them to the naive version, as we aim to observe the specific effects of each implementation. However, we can also explore the potential combinations of implementations that can work together, conducting experiments on the results of these combined implementations in addition to the individual ones and the naive version. It's possible that combining these implementations could yield the best-performing version among all the options.

Based on preliminary analysis, we found that combining Implementation 4 with Implementation 2 is particularly promising. Implementation 4 introduces a cut pool in which only a subset of previously generated cuts is used. Specifically, during the k-th iteration, we select $\min\big(\text{int}(a \times \text{totalIter}),\ \text{num_ndp}\big)$ cuts from those identified so far, where $a < 1$ is a tuning parameter, totalIter is the number of iterations completed, and num_ndp is the number of efficient integer solutions obtained.

Implementation 6 applies this cut-pool selection rule while adopting the feasibility-based strategy of Implementation 2. Thus, in each iteration, rather than solving the problem to optimality, we solve a feasibility problem using only the selected subset of cuts, incorporating the first feasible solution with positive θ into $\mathcal{S}_{\min}$. This combined approach retains the strengths of both methods while aiming for improved computational performance.

3.4 Data Sets

In this section, we describe the data sets used in the experiments. The comparative study on these data sets is presented in Subsection 3.5. We consider benchmark instances for the SPP and the Knapsack problem, as well as randomly generated PILP and MILP instances. The SPP and Knapsack problems are used to evaluate the performance of the proposed algorithm on well-known benchmark problems, while the randomly generated PILP and MILP instances are used to evaluate the performance of the algorithm on more general problems.

Set Packing Problem. The biobjective set packing problem considered in this data set can be mathematically formulated as follows:

$$
\begin{aligned}
\text{maximize} \quad & \sum_{i=1}^{n} c_1^i x_I^i && && \text{(BOSPP)}\\
\text{maximize} \quad & \sum_{i=1}^{n} c_2^i x_I^i && &&\\
\text{subject to} \quad & \sum_{i=1}^{n} t_{ij} x_I^i \leq 1, && j = 1, ..., m &&\\
& x_I^i \in \{0,1\}, && i = 1, ..., n, &&
\end{aligned}
$$

where m is the number of constraints, n is the number of variables, c_1 and c_2 are the coefficient vectors of the first and second objective functions, respectively, and $T = (t_{ij})$ is an $n \times m$ matrix with entries that are either 0 or 1. The j-th column of the matrix T is said to be covered by row i if $t_{ij} = 1$.

In this data set, we use instances of the biobjective set packing problem (BOSPP) that have been constructed previously in [Delorme et al., 2010]. This data set includes six families of instances, with the objective coefficients generated in different ways. The six families of instances are described as follows:

- A: two randomly generated objectives with

$$c_1^i = \text{Uniform}[1, 100],\ c_2^i = \text{Uniform}[1, 100],\ \forall i \in \{1, \ldots, n\}$$

- B: one randomly generated objective and one with the same coefficients in the reverse order

$$c_1^i = \text{Uniform}[1, 100],\ c_2^{n-i+1} = c_1^i,\ \forall i \in \{1, \ldots, n\}$$

- C: two randomly generated objectives with patterns

$$
\begin{aligned}
l_1 = \text{Uniform}[1, \frac{n}{10}],\ l_2 = \text{Uniform}[1, \frac{n}{10}],\ \ldots\\
c_1^1 = c_1^2 = \ldots = c_1^{l_1} = \text{Uniform}[1, 100]\\
c_1^{l_1+1} = c_1^{l_1+2} = \ldots = c_1^{l_1+l_2} = \text{Uniform}[1, 100],\ \ldots
\end{aligned}
$$

  and similarly for $c_2^i,\ \forall i \in \{1, \ldots, n\}$.

- D: one randomly generated objective with patterns given in C and one with the same coefficient

in reverse order. So, the same as C with

$$c_2^{n-i+1} = c_1^i, \ \forall i \in \{1, \ldots, n\}$$

- E: one unicost and one randomly generated objective, the same as A.
- F: one unicost and one randomly generated objective with patterns the same as C.

Each family in the data set consists of 20 instances, for a total of 120 instances. Table 3.2 provides some information about the parameters of the 20 instances within each family. Each row in the table corresponds to an instance within a particular family. The data set also includes two types of problems, referred to as "mis" and "spp", which correspond to the Maximum Independent Set and the Set Packing Problem, respectively. The MIS is a well-known problem in graph theory that seeks to find a set of disconnected nodes with the maximum cardinality (or total weights). The SPP with at most two non-zero coefficients in each constraint can be transformed into an MIS problem by interpreting each set in the SPP as a node in the graph and each constraint as an edge. More details on how the instances were constructed can be found in [Delorme et al., 2010]. We present the number of NDPs for each SPP instance in Table K.3.

Knapsack Problem. The multiobjective Knapsack problem (MKP) considered in this data set is constructed in [Kirlik and Sayın, 2014] and can be formulated as follows:

$$\begin{aligned} \text{maximize} \quad & \sum_{r=1}^{n} v_{jr} x_r \quad j = 1, \ldots, p \\ \text{subject to} \quad & \sum_{r=1}^{n} \omega_r x_r \leq W \\ & x_r \in \{0, 1\}, \qquad r = 1, \ldots, n, \end{aligned} \tag{MKP}$$

where n is the number of variables, p is the number of objectives, v_{jr} is the coefficient of the objective $j \in \{1, \ldots, p\}$ for each item $r \in \{1, \ldots, n\}$, and ω_r is the weight of item r in the capacity constraint. Instances in this data set are constructed as follows. v_{jr} and ω_r are random integers drawn from

Table 3.2: Benchmarks for BOSPP

No	Instances family	# Variables	# Constraints	Density (%)	Max-One
1	2mis100_300	100	300	2.00	2
2	2mis100_500	100	500	2.00	2
3	2mis101_300	100	300	2.00	2
4	2mis101_500	100	500	2.00	2
5	2mis200_1000	200	1000	1.00	2
6	2mis200_600	200	600	1.00	2
7	2mis201_1000	200	1000	1.00	2
8	2mis201_600	200	600	1.00	2
9	2spp100_300	100	300	3.08	4
10	2spp100_500	100	500	2.96	4
11	2spp101_300	100	300	2.97	4
12	2spp101_500	100	500	3.03	4
13	2spp200_1000	200	1000	1.49	4
14	2spp200_600	200	600	1.5	4
15	2spp201_1000	200	1000	1.49	4
16	2spp201_600	200	600	1.49	4
17	2spp202_1000	200	1000	2.48	8
18	2spp202_600	200	600	2.48	8
19	2spp203_1000	200	1000	2.49	8
20	2spp203_600	200	600	2.56	8

the interval $[1, 1000]$ and the capacity of the Knapsack is calculated as

$$W = \left\lceil 0.5 \left(\sum_{r=1}^{n} \omega_r \right) \right\rceil .$$

The instances in this data set are named according to the following format: KP_p-X_n-Y_ins-Z. "KP" stands for the Knapsack problem, X represents the number of objectives in the list $[3, 4, 5]$, Y represents the number of items in the list $[10, 20, \ldots, 100]$, and Z represents the instance number in the list $[1, 2, \ldots, 10]$. The size of the instances is reported in Table 3.3. We present the number of NDPs for each KP instance in Table K.4. (We excluded "KP" from the beginning of the instances for the sake of space.)

Pure Integer Instances. In this dataset, the IP instances are created through a random generation process described in [Mavrotas and Diakoulaki, 1998]. However, we made a modification

Table 3.3: Benchmarks for Knapsack Problem

Instance ID	# Objectives	# Variables	# Instances
KP_p-3_n-10_ins-i	3	10	$i \in \{1, 2, \ldots, 10\}$
KP_p-3_n-20_ins-i	3	20	$i \in \{1, 2, \ldots, 10\}$
KP_p-3_n-30_ins-i	3	30	$i \in \{1, 2, \ldots, 10\}$
KP_p-3_n-40_ins-i	3	40	$i \in \{1, 2, \ldots, 10\}$
KP_p-3_n-50_ins-i	3	50	$i \in \{1, 2, \ldots, 10\}$
KP_p-3_n-60_ins-i	3	60	$i \in \{1, 2, \ldots, 10\}$
KP_p-3_n-70_ins-i	3	70	$i \in \{1, 2, \ldots, 10\}$
KP_p-3_n-80_ins-i	3	80	$i \in \{1, 2, \ldots, 10\}$
KP_p-3_n-90_ins-i	3	90	$i \in \{1, 2, \ldots, 10\}$
KP_p-3_n-100_ins-i	3	100	$i \in \{1, 2, \ldots, 10\}$
KP_p-4_n-10_ins-i	4	10	$i \in \{1, 2, \ldots, 10\}$
KP_p-4_n-20_ins-i	4	20	$i \in \{1, 2, \ldots, 10\}$
KP_p-4_n-30_ins-i	4	30	$i \in \{1, 2, \ldots, 10\}$
KP_p-4_n-40_ins-i	4	40	$i \in \{1, 2, \ldots, 10\}$
KP_p-5_n-10_ins-i	5	10	$i \in \{1, 2, \ldots, 10\}$
KP_p-5_n-20_ins-i	5	20	$i \in \{1, 2, \ldots, 10\}$

by excluding the continuous elements. Therefore, the format of these instances is as follows:

$$
\begin{aligned}
\text{vmin} \quad & C_I x_I \\
\text{s.t.} \quad & \textstyle\sum_{j=0}^{n-1} a_{ij} [x_I]_j \leq b_i, \quad i = 0, 1, \ldots, n-1 \\
& \textstyle\sum_{j=0}^{n-1} [x_I]_j \leq \frac{n}{3} \\
& x_I \in \mathbb{Z}_+^r,
\end{aligned}
$$

where the number of constraints is equal to the number of variables, all the variables are assigned as integer variables. The coefficients of the instances are generated using the following approach: the objective function coefficients corresponding to integer variables, the RHS of the constraints, and the matrix coefficients for integer variables are all drawn from uniform random integer distributions in the ranges $[-200, 200]$, $[50, 100]$, and $[-1, 20]$, respectively.

This instance set of random PILP instances was created to demonstrate the scalability of the RVF Algorithm across a broader range of objective dimensions. We generated an additional suite of instances with a higher number of objectives. Note that we did not test all implementations of RVF or existing algorithms on this dataset; we only tested Implementation 2. In these cases, the number of objectives (a) was increased to $\{10, 15, 20, 25, 30\}$, with the variable count (b) fixed at 10, the non-zero ratio (c) varying within $\{30\%, 40\%, 50\%, 60\%, 70\%\}$, and the upper bound ($d$) set to

either 1 or 2. These instances were solved using the best-performing implementation, and the results are summarized in Table K.5. This experiment highlights the capability of the RVF Algorithm to handle an arbitrary number of objectives, significantly exceeding the range typically evaluated in the literature.

We have provided a link to a GitHub page [Fallah, 2025b] where the randomly generated IP instances can be accessed for reference.

Mixed Integer Instances. In this data set, the MILP instances are generated randomly according to the procedure outlined in [Mavrotas and Diakoulaki, 1998], which has been used previously by [Boland et al., 2015b] and [Mavrotas and Diakoulaki, 2005]. The structure of these instances is as follows:

$$
\begin{aligned}
\text{vmin} \quad & C_I x_I + C_C x_C \\
\text{s.t.} \quad & \textstyle\sum_{i=r}^{n-1} a_{ij}[x_C]_i + a'_j[x_I]_j \leq b_j, \quad j = 0, 1, \ldots, r-1 \\
& \textstyle\sum_{i=r}^{n-1} a''_{ij}[x_C]_i \leq b_j, \quad j = r, \ldots, m-2 \\
& \textstyle\sum_{j=0}^{r-1}[x_I]_j \leq \frac{r}{3} \\
& (x_I, x_C) \in \mathbb{Z}_+^r \times \mathbb{R}_+^{n-r},
\end{aligned}
$$

where the number of constraints is equal to the number of variables, denoted by $m = n$, a subset of the variables is assigned as integer variables based on the integer ratio, while the remaining variables are considered continuous. The coefficients of the instances are generated using the following approach: the objective function coefficients corresponding to continuous variables, the objective function coefficients corresponding to integer variables, the RHS of the constraints, and the matrix coefficients for both continuous and integer variables are all drawn from uniform random integer distributions in the ranges $[-10, 10]$, $[-200, 200]$, $[50, 100]$, and $[-1, 20]$, respectively.

Each instance is identified by the nomenclature: "MILP_obj_{a}_var_{b}_nzr_{c}_intR_{d}_u_{e}_ins_{f}". Here, a denotes the number of objectives, b is the total number of variables, and c represents the non-zero ratio of the coefficient matrix. Additionally, d indicates the integer ratio, e is the upper bound on integer variables, and f refers to the specific instance index within its category. For the instances provided, the number of objectives (a) is 2, indicating biobjective problems. The variable count (b) belongs to the set $\{30, 40\}$, the non-zero ratio (c) is selected from

$\{40\%, 50\%, 60\%, 70\%, 80\%, 90\%\}$, and the integer ratio (d) is either 90% or 95%. The upper bound (e) is fixed at 1 for all instances.

We have provided a link to a GitHub page [Fallah, 2025d] where the randomly generated MILP instances can be accessed for reference.

3.5 Computational Results

In this section, the results of a computational experiment are presented to evaluate the implementations of the RVF Algorithm presented in Section 3.3. The aim of the experiment is to assess the effectiveness of the proposed implementations in terms of computational efficiency and solution quality. The details of the experimental setup, including the test problems used and the performance measures, are described below. The results are then presented and discussed in terms of their implications for the proposed implementations of the RVF Algorithm.

The RVF Algorithm was implemented in Python 3, with the Gurobi 10 optimizer used to solve the problem at each iteration. The experiments were conducted on a machine with dual AMD Opteron 6128 processors and 32 GB of RAM running the Linux (Debian 9) operating system. We compare the performance of the RVF Algorithm to several well-known methods from the literature. To this end, we consider biobjective SPP, multiobjective Knapsack (with up to 5 objectives), and randomly generated PILP and MILP instances, and we solve each instance once using our proposed algorithm and once using a selected method from the literature. For your reference, all the results of the computational experiment are presented in tables within Appendix K.

Prior to presenting the computational results, we discuss the RVF Algorithm in terms of the running time for both PILPs and MILPs, and the assessment of approximation quality in each iteration, specifically for MILPs. Note that the RVF Algorithm is an exact algorithm; however, if the algorithm is terminated before completion, the result obtained constitutes an approximation of the EF. It is also worth noting that the approximation quality comparison is not applicable to PILPs. This limitation arises from the inherent behavior of the RVF Algorithm in the pure integer case, where it generates NDPs in a nondecreasing order based on the first objective, as substantiated by Theorem 3.1.4.

As described in (3.8)–(3.13), the RVF Algorithm solves an MINLP in each iteration in order to find

the integer part associated with a newly discovered stability region. This optimization problem determines the maximum distance between the upper approximation and the real value function (θ in the formulation). In each iteration of the algorithm, a new bounding function (cut) is generated and added to the formulation, which increases the size of the formulation as the algorithm progresses (the number of variables is linear in the number of iterations), potentially making the problem more difficult to solve as the iterations continue. Assuming that we have N stability regions, the algorithm must solve $2N$ optimization problems (N MINLPs and N MILPs) to construct the entire EF. Therefore, the number of iterations in the RVF algorithm may be significantly smaller, representing a tradeoff that should be analyzed computationally.

Regarding the quality of the approximation, we consider an approximation formed from a subset of $\mathcal{S}_{\text{min}}$ in each iteration as an approximation and then compare the approximate EFs generated by the algorithms based on the metrics in the literature. The RVF Algorithm directly targets the so-called θ. In other words, the RVF Algorithm is designed to find the maximum violation between the upper approximation and the real value function by finding the point at which the approximation is farthest from the true value. In contrast, other existing algorithms simply explore a new region in each iteration and attempt to refine the approximation without specifically targeting the maximum violation. Therefore, in the mixed integer case, we expect the RVF Algorithm to perform better than the other existing algorithms in terms of the metrics defined in the literature for comparing the approximate EF in each iteration. The θ is being decreased during the iterations, so the volume measure (explained in Figure 3.16), which is one way to compare the approximate EF in a biobjective case, follows the same pattern and is being decreased. Similarly, for the multiobjective case, the RVF Algorithm performs better than the existing algorithms in terms of the metrics for comparing the approximate EFs. However, in the pure integer case, the RVF Algorithm generates NDPs in the nondecreasing (nonincreasing) order for the first minimization (maximization) objective. Therefore, for PILPs, we expect that, regarding the quality of the approximation, the RVF Algorithm may not be as competitive in terms of approximation quality compared to scalarization and non-scalarization algorithms.

In the following two subsections, we showcase the computational experiment involving the naive version of the RVF Algorithm, along with its different implementations for both PILP (covered in Section 3.5.1) and MILP (discussed in Subsection 3.5.2).

3.5.1 Pure Integer Instances

3.5.1.1 Experimental Setup

In this study, we impose a time constraint of 24 hours on the execution of each algorithm in order to generate the entire EF or the value function. If the time limit is exceeded, the algorithm is terminated.

We compared the proposed RVF Algorithm with multiple existing algorithms, which are discussed in detail below. For biobjective instances, we compared RVF with DCM [Boland et al., 2017], Chalmet [Chalmet et al., 1986], EpsilonConstraint [Haimes, 1971], DominguezRios [Domínguez-Ríos et al., 2021], KirlikSayin [Kirlik and Sayın, 2014], and TambyVanderpooten [Tamby and Vanderpooten, 2021]. For multiobjective instances, we compared RVF with DCM, MOMIX [De Santis et al., 2020], DominguezRios, KirlikSayin, and TambyVanderpooten. In the following subsection, we discuss the methodologies of these algorithms one by one.

The algorithms Chalmet, EpsilonConstraint, DominguezRios, KirlikSayin, and TambyVanderpooten are implemented in the MultiObjectiveAlgorithms.jl package [Dowson et al., 2025], also referred to as MOA.jl. This open-source Julia package, built on top of JuMP and the MathOptInterface standard, is the successor to the vOptSolver project [Gandibleux et al., 2017]. It provides a unified and extensible framework for modeling, implementing, and comparing exact and approximate algorithms for multiobjective optimization. MOA.jl reformulates vector-valued objective functions into a sequence of single-objective subproblems through iterative scalarization and leverages a wide range of commercial and open-source solvers to handle linear, integer, conic, semidefinite, and nonlinear problems. The package includes several classical and modern exact algorithms, each differing in the type of efficient or representative set they return. In this study, we employ the exact methods for biobjective and multiobjective optimization.

3.5.1.2 Algorithms Compared

Chalmet Algorithm. Chalmet et al. proposed one of the earliest exact algorithms for generating all nondominated points of a bi-criterion integer programming problem. Their method is a variant of the weighting (scalarization) approach, which solves a series of single-objective integer programs

with modified constraints to systematically explore nondominated regions. Starting with the lexicographic maxima of each objective, the algorithm iteratively introduces additional constraints to exclude previously found nondominated points and the regions they dominate. By repeatedly solving weighted-sum subproblems within these restricted feasible regions, the method ensures that each new solution corresponds to a distinct nondominated point. The authors implemented both multi-start and single-tree implicit enumeration versions: the multi-start version restarts the search for each subproblem independently, while the single-tree version reuses partial branch-and-bound information from previous subproblems, greatly improving computational efficiency. Although originally developed for bi-objective 0–1 linear programs, the authors also discussed a generalization to higher dimensions. The method guarantees validity, completeness, and finite convergence under boundedness and remains a foundational approach for multiobjective integer programming.

EpsilonConstraint Algorithm. In their seminal work, Haimes formulated the joint problem of *system identification* and *system optimization* as a bicriterion optimization problem. The two components are conceptually distinct yet mathematically linked: system identification seeks to estimate model parameters a such that the predicted system output $H(X, U, a)$ matches observed data Y^j by minimizing the least-squares error

$$G(a) = \sum_{j=1}^{J} [H(X_j, U_j, a) - Y^j]^T W_j [H(X_j, U_j, a) - Y^j],$$

whereas system optimization aims to determine decision or control variables U (and associated states X) that minimize a performance function $f(X, U, a)$ subject to system constraints $g(X, U, a) \leq 0$. Since the optimization objective f depends on parameters a obtained from the identification problem, the two tasks are inherently coupled. To integrate them, the authors proposed a bicriterion formulation

$$\min_{X,U,a} [f(X, U, a), G(a)],$$

and its ε-constraint equivalent:

$$\min_{X,U,a} f(X, U, a) \quad \text{s.t.} \quad G(a) \leq \varepsilon,\ g(X, U, a) \leq 0,$$

where ε bounds the acceptable model error. They further proved an *equivalence theorem* showing that under mild conditions, solving this ε-constrained problem yields an efficient solution to the original bicriterion formulation. This framework provided one of the earliest mathematical foundations for integrating model accuracy and operational optimality within a unified multiobjective optimization structure.

DominguezRios Algorithm. Domínguez-Ríos et al. proposed an *exact anytime algorithm* for multiobjective combinatorial optimization (MOCO) problems, designed to progressively generate well-distributed efficient solutions at any moment during the search. Their approach builds upon the *full m-split* framework of Dächert and Klamroth [2015], which iteratively partitions the objective space into p-dimensional regions (or boxes) and solves a scalarized optimization subproblem within each region to identify new NDPs. The proposed algorithm, called *TPA (Tchebycheff Partition Algorithm)*, introduces three methodological innovations: (1) an *alternation of directions* mechanism to diversify the exploration of regions, (2) a new *p-partition scheme* that ensures the subdivisions of the objective space are non-overlapping, and (3) a *reduced-scaled priority function* that guides which regions to explore next, prioritizing those with higher potential to contain unexplored efficient solutions.

At the core of TPA lies a scalarization model based on the weighted augmented Tchebycheff norm:

$$\min_{x \in X} \max_{i=1,\ldots,p} \lambda_i \, |f_i(x) - z_i^I| \; + \; \epsilon \sum_{i=1}^{p} \lambda_i \, |f_i(x) - z_i^I|,$$

where $f_i(x)$ denotes the i-th objective function, z_i^I is the ideal (best attainable) value of that objective, $\lambda_i > 0$ are normalization or weighting coefficients, and $\epsilon > 0$ is a small scalar ensuring numerical stability and smoothness. The *decision variable* is $x \in X$, representing a feasible solution (for example, a binary or integer vector in combinatorial problems). The objective function considered is

$$F(x) = \max_i \lambda_i |f_i(x) - z_i^I| + \epsilon \sum_i \lambda_i |f_i(x) - z_i^I|.$$

Although the first term involves a maximum over the objectives, the algorithm *minimizes* $F(x)$ overall. In other words, it seeks a solution x that *reduces the largest deviation from the ideal point across all objectives*. The inner "max" identifies the worst-performing objective, and the outer

"min" chooses the solution that makes this worst-case deviation as small as possible, implementing a min–max strategy that balances trade-offs among objectives. The second term, $\epsilon \sum_i \lambda_i |f_i(x) - z_i^I|$, acts as a tiebreaker and smoothing factor, encouraging convergence toward the EF by favoring solutions that are closer to the ideal across all objectives. By minimizing $F(x)$, the algorithm effectively searches for NDPs nearest to the ideal point in a multiobjective sense.

KirlikSayin Algorithm. The methodology proposed by Kirlik and Sayın provides an exact algorithm for generating all nondominated solutions of multiobjective discrete optimization problems (MODO) with any number of objectives. The general MODO can be formulated as

$$\min_{x \in X} f(x) = (f_1(x), f_2(x), \ldots, f_p(x)),$$

where $X \subseteq \mathbb{Z}^n$ is the discrete feasible set, and $f_j(x)$ represents the j-th objective to be minimized. The proposed method is based on the ε-constraint scalarization, in which one objective $f_k(x)$ is retained as the main objective while the remaining $(p-1)$ objectives are converted into constraints of the form $f_j(x) \leq \varepsilon_j$ for $j \neq k$. To ensure that only strictly efficient (not weakly efficient) solutions are obtained, the authors introduce a two-stage optimization scheme. In the first stage, the algorithm solves

$$P_k(\varepsilon): \quad \min f_k(x) \quad \text{s.t.} \quad f_j(x) \leq \varepsilon_j,\ j \neq k,\ x \in X,$$

yielding an optimal objective value z^*. In the second stage, this value is fixed, and the algorithm solves

$$Q_k(\varepsilon): \quad \min \sum_{j=1}^{p} f_j(x) \quad \text{s.t.} \quad f_j(x) \leq \varepsilon_j,\ j \neq k,\ f_k(x) = z^*,\ x \in X.$$

The solution of $Q_k(\varepsilon)$ eliminates weakly efficient points and guarantees efficiency. The search for nondominated solutions is conducted over the $(p-1)$-dimensional ε-space, which is partitioned into axis-aligned hyperrectangles $R(l,u) = \{\varepsilon \in \mathbb{R}^{p-1} : l \leq \varepsilon \leq u\}$. In each iteration, the algorithm evaluates the upper vertex of a rectangle by solving the two-stage problems, adds the resulting nondominated point to the set, and removes all rectangles corresponding to dominated or infeasible regions. This process continues until all feasible regions are exhausted. The authors prove that the algorithm terminates in finitely many steps and enumerates the complete nondominated set.

TambyVanderpooten Algorithm. Tamby and Vanderpooten [2021] present a generic exact algorithm for enumerating the complete set of nondominated points in multiobjective discrete optimization problems. The method generalizes the classical ε-constraint technique—originally designed for two objectives—to any number of objectives p. The problem is formulated as

$$\min(f_1(x), f_2(x), \ldots, f_p(x)) \quad \text{s.t. } x \in X,$$

where X is a discrete feasible set. Each feasible x corresponds to a performance vector $y = (f_1(x), \ldots, f_p(x))$ in the objective space $\mathbb{R}^p$, and the aim is to generate all nondominated points Y_{ND}.

The algorithm explores the objective space through a dynamically updated *search region* $S(N)$, representing areas that may still contain nondominated points given the set N already found. This region is expressed as a union of *search zones* bounded by local upper vectors $u = (u_1, \ldots, u_p)$:

$$S(N) = \bigcup_{u \in U(N)} z(u), \quad z(u) = \{y \in \mathbb{R}^p \mid y_i < u_i, \ \forall i\}.$$

When a new nondominated point y^* is discovered in $z(u)$, that zone is split into p subzones

$$u^{(k)} = (u_1, \ldots, u_{k-1}, y_k^*, u_{k+1}, \ldots, u_p),$$

removing the region dominated by y^*.

Each zone is explored by solving a *lexicographic integer program*

$$\begin{aligned} \text{lexmin} \quad & (y_k, \sum_{i \neq k} y_i) \\ \text{s.t.} \quad & y \in Y, \\ & y_i < u_i, \quad \forall i \neq k, \end{aligned}$$

or equivalently, a single weighted-sum formulation

$$\min \ \Delta y_k + \sum_{i \neq k} y_i,$$

with a sufficiently large Δ ensuring lexicographic dominance. Each feasible program yields one NDP,

while reduction rules identify and discard redundant or empty zones without solving additional models. In summary, the TambyVanderpooten algorithm provides a framework for the exact enumeration of NDPs. By systematically partitioning the objective space and exploring each subregion through lexicographic optimization, it guarantees completeness of the EF while avoiding redundant or infeasible searches.

DCM Algorithm. Boland et al. [2017] proposed an exact algorithm for optimizing a linear function over the efficient set of a multiobjective integer program (MOIP). The method extends the L-shape search algorithm to the full-dimensional criterion space and is built upon a *criterion space decomposition* approach. At each iteration, the algorithm partitions the criterion space into subregions (or "objects"), represented by sets of disjunctive constraints that restrict the search to unexplored regions while ensuring that each subregion contains at most one nondominated point. The algorithm repeatedly solves single-objective integer programs of the form

$$\min_{x \in X} \sum_{j=1}^{p} z_j(x)$$

subject to disjunctive constraints that remove dominated regions of the criterion space:

$$z_j(x) \leq (y_j - \varepsilon - M_j)b_j + M_j, \qquad \sum_{j=1}^{p} b_j = 1, \qquad b_j \in \{0,1\},$$

where M_j is a sufficiently large constant and $\varepsilon > 0$ is a small tolerance parameter. The vector $y = (y_1, \ldots, y_p)$ denotes a *reference point* in the criterion space, typically corresponding to a previously discovered nondominated point. The binary variables b_j implement a logical "OR" condition, ensuring that at least one objective must improve upon y by at least ε. Specifically, exactly one $b_{j^*} = 1$ forces $z_{j^*}(x) \leq y_{j^*} - \varepsilon$, while all other inequalities are relaxed via the big-M constants. After each nondominated point $z(x^n)$ is found, the criterion space is decomposed so that at most one new subproblem is added to the search queue, significantly reducing computational complexity. The algorithm maintains dynamic lower and upper bounds on the linear objective to guide the search process and can also be adapted to compute the *nadir point* of a MOIP.

MOMIX Algorithm. De Santis et al. [2020] proposed MOMIX, a deterministic branch-and-bound algorithm for solving multiobjective mixed integer convex optimization problems (MOMICs)

of the form

$$\min_{x \in B} \; F(x) = (f_1(x), \ldots, f_m(x))^T \quad \text{s.t. } g(x) \leq 0, \; x_i \in \mathbb{Z} \; \forall i \in I,$$

where the objective functions f_j and the constraint functions g_k are convex and continuously differentiable. Due to the presence of integrality constraints, the feasible region is nonconvex and the corresponding image set may be disconnected.

MOMIX explores the decision space within a branch-and-bound framework while constructing and refining linear outer approximations of the image set (criterion space). At each node associated with a subregion $\tilde{B}$, the algorithm derives lower bounds via convex relaxations of the integrality constraints and linear approximations of $\text{conv}(F(\tilde{B}))$, while upper bounds are obtained from feasible integer solutions corresponding to potential NDPs. These bounds are iteratively updated using local upper bound sets, enabling the identification and pruning of dominated regions in the objective space.

Formally, MOMIX maintains lower and upper bounding approximations $L_{\tilde{B}}$ and $U_{\tilde{B}}$ such that

$$F(\tilde{B}) \subseteq L_{\tilde{B}} + \mathbb{R}^m_+, \qquad U_{\tilde{B}} \subseteq F(B),$$

and any node is pruned whenever

$$L_{\tilde{B}} + \mathbb{R}^m_+ \subseteq U_{\tilde{B}} + \mathbb{R}^m_+.$$

Two variants are proposed: MOMIXLIGHT, which relies solely on continuous convex relaxations, and MOMIX, which additionally solves mixed integer subproblems to obtain tighter bounds. The algorithm is guaranteed to compute an ε-approximation of the EF and the nondominated set, providing a deterministic, non-scalarization-based approach for multiobjective mixed integer convex optimization.

In contrast to MOMIX, the proposed RVF Algorithm does not employ a lower approximation of the EF. Instead, it relies exclusively on an upper approximation, which is iteratively refined until exactness is achieved. This refinement is performed by solving a nonconvex problem at each iteration and exploiting dual information to obtain tight updates that are exact within the corresponding stability region. While the RVF Algorithm may yield tighter upper approximations, MOMIX is

designed to establish a coverage of the EF. Consequently, direct time-based comparisons between MOMIX and RVF are not meaningful.

In the following, we present the results of our study on the effectiveness of the proposed RVF Algorithm and its implementations across the selected data sets. Our investigation aims to assess the impact of the algorithm and its implementations on each data set to evaluate their performance and potential utility in solving the corresponding optimization problems. Subsequently, we compare the best-performing version of the RVF Algorithm with other existing algorithms.

3.5.1.3 Evaluating the Efficacy

We conducted a comparative study to assess the impact of linearization and the implementations of the RVF Algorithm for PILP. Initially, we compared the naive linear version of the RVF Algorithm with the naive nonlinear version of the RVF Algorithm. Subsequently, we examined the effects of Implementations 1–6, except 5, in comparison to the naive linear version of the RVF Algorithm and each other. All the implementations are based on the linear version, as we observed that the naive linear version of the RVF Algorithm significantly reduced the processing time compared to the naive nonlinear version of the RVF Algorithm. Regarding Implementation 5, as previously mentioned in Section 3.3, where we discussed the implementations of the RVF Algorithm, this particular implementation could not be implemented within the linear version. Therefore, we excluded the experiment for this implementation because it can be developed based on the nonlinear version, and the naive nonlinear version is not comparable to the naive linear version.

The results of this study for PILP cases are presented in Tables K.1–K.15, which compare the running time and the total number of iterations for the naive RVF Algorithm and Implementations 1 to 6. Note that "Imp 0 (L)" and "Imp 0 (NL)" refer to the naive linear and nonlinear implementations of the RVF Algorithm. In this study, we considered 39 out of 120 SPP instances and 42 out of 160 KP instances introduced in Section 3.4. In these tables, "Imp" stands for implementation.

We additionally employed Implementation 2 to 30 instances of randomly generated IP with a specified number of objectives listed as $[10, 15, 20, 25, 30]$. We have provided the number of NDPs and the corresponding running times in Table K.5. The primary objective of this analysis was to demonstrate the RVF Algorithm's capability to solve problems with an arbitrary number of

objectives.

In the following, we present detailed computational experiments. One of the comparison tools used is the performance profile. The performance profiles reported in this section follow the standard Dolan–Moré framework, in which the performance of each method is compared against the best-performing method on each instance. In addition to the standard performance profile, an alternative variant—referred to as the baseline performance profile—can also be considered. In this variant, the performance of each algorithm is evaluated relative to a fixed reference (baseline) method rather than the per-instance best.

The conclusions drawn from the computational experiments using the standard and baseline performance profiles are consistent. Therefore, to avoid overloading the main text, we do not present the baseline performance profiles here. For completeness, the corresponding baseline performance profiles for the computational experiments are reported in Appendix L.

In this study, we first compared the naive linear and nonlinear versions of the RVF Algorithm. Figure 3.6 presents the performance profile comparing the execution times of these two approaches, where "idea_0_L" and "idea_0_NL" denote the linear and nonlinear implementations, respectively. The results indicate that the linear version is significantly superior in terms of computational efficiency; specifically, the linear implementation reduced the average execution time by approximately 95.6%.

We also compared the naive linear variant of the RVF Algorithm (Implementation 0) with Implementation 1. For clarity, Implementation 1 uses the following configuration: objective parameter θ, number of solutions set to ∞, cut ratio equal to 1, and stopping criterion based on optimality. Figure 3.7 presents the performance profile. Implementation 1 outperformed the naive linear version by approximately 28% and 6% in terms of running time for the KP and SPP, respectively. On average, Implementation 1 improved the running time by about 17%.

These results were expected, since Implementation 1 builds upon Implementation 0 with one key modification. In Implementation 0, the algorithm may generate feasible solutions corresponding to new stability regions, but these solutions are not added to the list. In contrast, Implementation 1 verifies whether the NDPs corresponding to the new feasible solutions are dominated by the EF. If they are not dominated, they are added to the list. Consequently, Implementation 1 reaches termination with fewer iterations, leading to an overall improvement in running time.

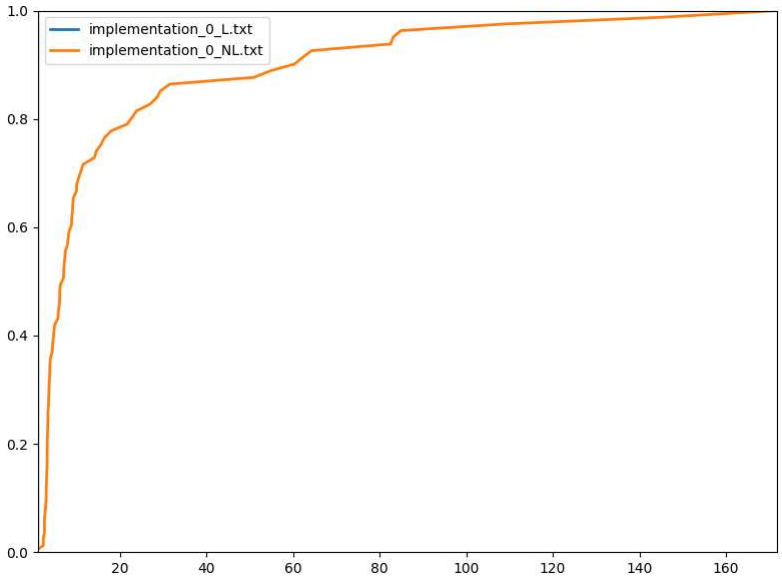


Figure 3.6: Performance profile comparing execution time of naive linear and nonlinear version of the RVF Algorithm for PILPs.

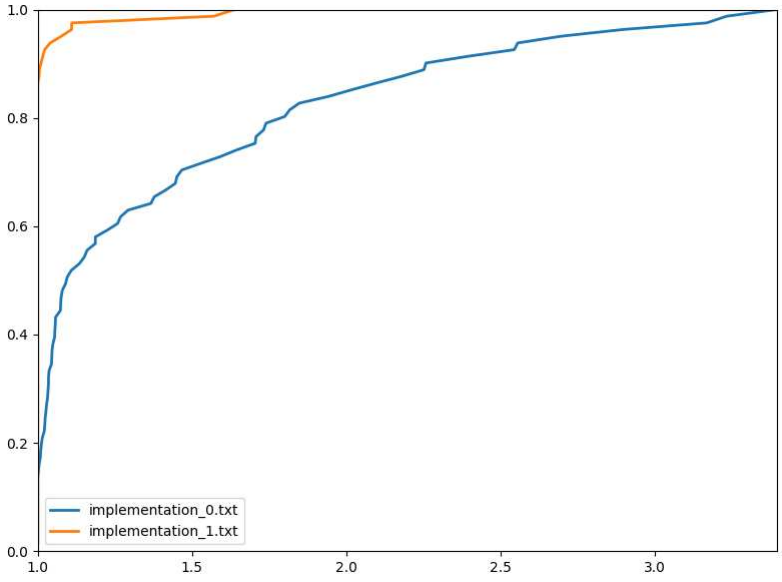


Figure 3.7: Performance profile comparing execution time of naive linear version of the RVF Algorithm and Implementation 1 for PILPs.

Implementation 3 requires fine-tuning of its parameter. For completeness, the parameter configuration of Implementation 3 is as follows: objective parameter θ, number of solutions set to n, cut ratio equal to 1, and stopping criterion based on feasibility. In this particular approach, the algorithm terminates its search upon encountering feasible solutions. We've established a range of permissible quantities of feasible solutions, denoted by the list $[1, 2, 3, 4, 5]$. For each value of i within this range $i \in [1, 2, 3, 4, 5]$, we designate the resultant algorithm as "implementation_3_n_i," representing Implementation 3 while i signifies the number of feasible solutions considered.

Figure 3.8 presents the performance profile comparing the execution time of the RVF Algorithm with implementation_3_n_i, where i ranges from 1 to 5. Although none of the configurations consistently outperforms the others across all instances, the settings $i = 4$ and $i = 5$ generally demonstrate superior performance relative to the remaining values. Considering the area under the curve, these two configurations also exhibit comparable behavior. The average running times for $i = 1$ through $i = 5$ are 166.57, 146.74, 151.33, 144.39, and 145.58 seconds, respectively. Furthermore, comparing the best and worst configurations, specifically $i = 4$ versus $i = 1$, we observe that Implementation 3 with four feasible solutions outperforms the configuration with a single feasible solution, yielding an average runtime improvement of approximately 13.3%.

The corresponding results for execution time and number of iterations are reported in Table K.6 and Table K.7, for the KP and SPP instances, respectively.

In contrast, Implementation 2 does not require parameter tuning. For completeness, the parameter configuration of Implementation 2 is as follows: objective value set to 1, number of solutions equal to 1, cut ratio equal to 1, and stopping criterion based on feasibility. In this case, the solver terminates its search upon encountering the first feasible solution, as the objective function consistently holds a value of 1.

Implementation 4 also requires parameter tuning. For clarity, the parameter configuration of Implementation 4 is as follows: objective parameter θ, number of solutions set to ∞, cut ratio strictly less than 1, and stopping criterion based on optimality. In this implementation, each iteration involves a procedure that generates a subset of the cuts discovered thus far. This subset is considered in the subsequent iteration to facilitate the solve of the subproblem in intermediate iterations. We denote these configurations as implementation_4_p_x, where $x \in \{10, 20, 30, 40, 50, 60, 70, 80, 90\}$. The parameter x specifies the proportion of cuts retained, expressed as $x/100$.

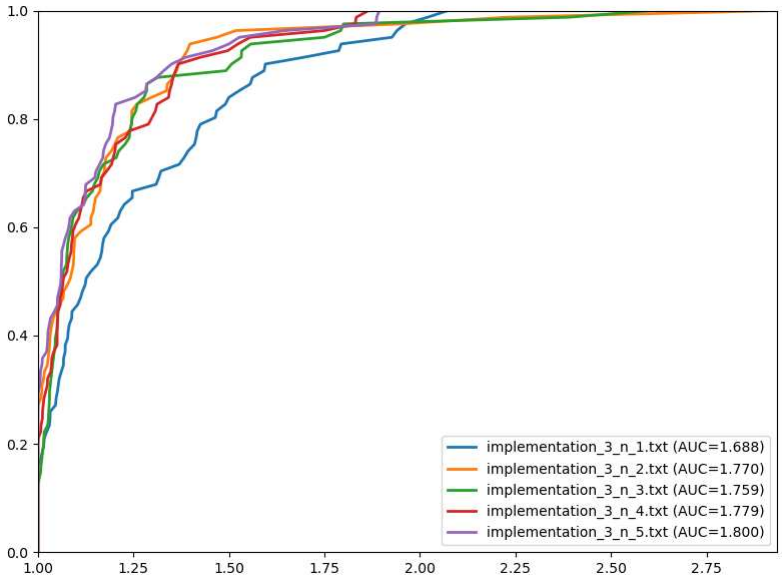


Figure 3.8: Performance profile comparing execution time of Implementation 3 for PILPs.

Implementation 6, in contrast, does not require parameter tuning. Its parameter configuration is as follows: objective value set to 1, number of solutions equal to 1, cut ratio strictly less than 1, and stopping criterion based on feasibility.

Figure 3.9 illustrates how various cut ratios influence the performance of the algorithm. In each iteration, the number of cuts selected for inclusion is dynamically determined by the formula: $\min(\text{int}(a \times \max(\text{totalIter}, \text{num_ndp})), \text{num_ndp})$, where a represents the chosen cut ratio. The rationale behind this scaling logic is to maintain a well-distributed set of cuts in the objective space. To identify the optimal ratio, we conducted a parameter-tuning study evaluating values from $\{0.01, 0.05, 0.1\}$ through additional increments up to 0.9.

The behavior observed in this experiment is quite insightful and, in several ways, counter to our initial expectations. Intuitively, one would expect scenarios with a lower cut ratio to require a larger number of iterations than those with a higher cut ratio, with each iteration being computationally lighter. However, our results reveal the opposite: the low-ratio scenarios not only exhibit better overall running times but also fewer iterations, which is quite remarkable.

We examined several instances in detail and analyzed the outcomes of all iterations individually. When comparing, for example, the 10% and 90% cut-ratio scenarios, we observed that during the

first half of the execution—that is, before roughly half of all NDPs are identified—both scenarios progress quickly, producing a similar number of iterations. In this phase, the 10% scenario runs faster due to simpler subproblems, yet, interestingly, it still identifies a relatively large number of new NDPs per iteration.

In the second half of the execution, however, the behaviors diverge substantially. In the 10% scenario, many of the cut-based iterations become unsuccessful (i.e., they do not yield a new NDP). As a consequence, the iteration immediately following an unsuccessful one adds all cuts, typically resulting in the identification of multiple new NDPs at once. By contrast, in the 90% scenario, most cut-based iterations remain successful but yield only a small number of new NDPs per iteration. For instance, within the second half of the run, the 10% scenario may require only two iterations (one unsuccessful and one full-cut iteration) to find three new NDPs, whereas the 90% scenario may require three successful iterations to identify the same number.

These observations explain why the 10% scenario results in both fewer iterations and shorter overall running times: one unsuccessful iteration with only 10% of cuts followed by one full-cut iteration is computationally cheaper than three successful iterations each containing 90% of the cuts.

The aggregated numerical results reinforce these insights. The average running times for ratios ranging from 0.01 to 0.9 are 170.64, 170.84, 170.37, 177.12, 182.75, 187.9, 201.74, 197.46, 214.89, 214.02, and 223.85 seconds, respectively. From an AUC perspective, the ratios 0.01, 0.05, and 0.1 occupy the same high-performance range, with their AUC values significantly exceeding those of the alternative versions. Given that these ratios also demonstrate the most favorable computation times, the 10% ratio was selected as the representative configuration for Implementation 4 in the subsequent comparative analysis with other implementations.

This intuition is also consistent with the computational experiments for this implementation of the MILP.

The corresponding results for running time and the number of iterations are reported in Tables K.8–K.9 for KP, and Tables K.10–K.11 for SPP, respectively.

Figure 3.10 illustrates the performance profile, which compares the execution times of both the naive linear version and the implementations of the RVF Algorithm. It's important to note that for Implementations 3 and 4, we selected the best-performing algorithms, specifically implementation_3_n_4

and implementation_4_p_10. In terms of the average running time, Implementations 0, 1, 2, 3, 4, and 6 have running times of 175.24, 149.12, 120.52, 144.39, 170.37, and 250.09, respectively. The best-performing algorithm, Implementation 2, achieves an average running time improvement of approximately 51.8% compared to the worst-performing algorithm, Implementation 6. This improvement is calculated as the difference between the running times of Implementations 6 and 2, divided by the running time of Implementation 6. Additionally, Implementation 2 outperforms the naive approach (Implementation 0) by 31.23%.

Based on the average running time, Implementation 2, which addresses the feasibility problem, demonstrates the best performance among all configurations. The next tier of performance is achieved by Implementations 1 and 3, followed by Implementation 4 and the naive version. At the bottom, Implementation 6—combining the cut pool approach with the feasibility problem—ranks as the least effective configuration.

Based on the performance profile analysis, most implementations exhibit superior performance relative to the naive version (Implementation 0). However, Implementations 4 and 6, both of which employ a cut-pool strategy, display more nuanced behavior.

In the standard performance profile (Figure 3.10), Implementation 4 achieves a higher AUC than the naive version (2.297 versus 2.045), despite an intersection between their profiles, indicating that neither implementation uniformly dominates the other across all instances. In contrast, Implementation 6 is the only variant that is consistently outperformed by the naive approach, as reflected by lower AUC values in both the standard and base performance profiles.

The observed intersections in both the standard and base performance profiles (Figure L.3) indicate instance-dependent performance. While the cut-pool strategy in Implementation 4 yields better performance on a large subset of instances, the naive approach demonstrates slightly stronger baseline performance in the base profile, as evidenced by its marginally higher base AUC (2.047 versus 2.020).

The intersections observed in both the standard and base performance profiles (Figure L.3) indicate instance-dependent behavior. In particular, the cut-pool strategy employed in Implementation 4 achieves superior performance on a substantial fraction of the tested instances. However, in the base performance profile, the naive approach exhibits slightly stronger baseline performance, as reflected by its marginally higher base AUC (2.047 versus 2.020), suggesting that its simpler structure can be

advantageous when the cut-pool provides limited benefit.

Here, we provide an intuitive explanation for why Implementation 2, which solves a feasibility problem, performs better than Implementation 1, which solves to optimality at each iteration while also generating all feasible solutions. The total computation time can be divided into two components: (i) the time required to convert feasible solutions into efficient solutions, and (ii) the time spent by the solver on each iteration of the algorithm. The conversion time is the same for both Implementation 1 and Implementation 2, as it depends only on the number of NDPs (or stability regions). In Implementation 2, each iteration is guaranteed to yield a new solution corresponding to a new stability region by design. In Implementation 1, however, every feasible solution must first be checked against the existing EF; its conversion is performed only if it is not dominated. Thus, conversion time is equivalent across both approaches, and the difference arises from the time spent solving subproblems.

To illustrate, we examined several instances and report here on instance 2spp203_1000A, which has 16 NDPs. In this case, the average subproblem time per NDP is 17.5 seconds under Implementation 2, compared to 51 seconds under Implementation 1. Conversion times are similar: 42 seconds for Implementation 1 and 40 seconds for Implementation 2. When looking at solver effort, Implementation 1 requires about 11.7k nodes per NDP, while Implementation 2 requires only 7.3k. Moreover, Implementation 1 needs 8 iterations to recover all 16 NDPs. Overall, the higher time per NDP in Implementation 1 can be attributed to the increasing complexity of the search tree, which requires exploring more nodes for each new NDP.

Here, we compare Implementation 2 and Implementation 3 and explain why Implementation 2 outperforms Implementation 3 in the PILP setting. Overall, Implementation 2 achieves an improvement in running time of approximately 25%. Implementation 2 solves a feasibility problem at each iteration using an arbitrary objective (e.g., a constant value of 1), whereas Implementation 3 solves the same feasibility problem using θ as the objective function.

It is important to note that in PILP cases, every feasible solution with a positive value of θ is guaranteed to induce a new stability region. As a result, regardless of whether an arbitrary objective is used (as in Implementation 2) or θ is used as the objective (as in Implementation 3), each feasible solution corresponds to a distinct stability region. Consequently, in the PILP setting, Implementation 2 is guaranteed to generate new stability regions without explicitly optimizing over

θ.

Our extensive computational experiments confirm that, for PILP instances, Implementation 2 consistently outperforms Implementation 3 in terms of running time. In contrast, for MILP instances (see Section 3.2), we observed that Implementation 3 outperforms Implementation 2. This difference can be attributed to the additional computational complexities present in the MILP setting, where using θ as the objective helps guide the solver more effectively. In the PILP setting, however, the problem is already fully linearized and does not suffer from such computational difficulties, rendering the explicit optimization of θ unnecessary and potentially detrimental to performance.

To illustrate this behavior, we present an example based on the SPP instance 2spp203_600B. The results for this instance across both implementations are reported below.

Table 3.4: Performance comparison of implementations (all statistics reported per NDP)

Implementation	**Gurobi Time**	**Gurobi Nodes**	**Conversion Time**
Implementation 3	59.5	15,165.2	17.2
Implementation 2	10.33	7,641.1	12.66

The intuition behind this discrepancy lies in the solver's behavior when handling feasibility problems. When an explicit objective is specified, the solver does not solely prioritize finding feasibility; instead, its branching and node selection strategies are influenced by the objective function. In contrast, using an arbitrary constant objective allows the solver to focus more directly on feasibility. We also experimented with solver parameters, such as MIPFocus, to encourage feasibility-oriented search in Implementation 3; however, these adjustments did not yield improvements on average.

3.5.1.4 Comparative Analysis of Implementation 2

We conducted a comparative analysis by solving each instance in the given dataset using several distinct approaches: our proposed RVF Algorithm, the DCM package [Boland et al., 2017], the MOMIX solver [De Santis et al., 2020], and the exact algorithms implemented in the MultiObjectiveAlgorithms (MOA) package [Dowson et al., 2025]. Specifically, for the MOA package, we considered the following algorithms for biobjective instances in this computational study: Chalmet [Chalmet et al., 1986], EpsilonConstraint [Haimes, 1971], DominguezRios [Domínguez-Ríos et al., 2021], KirlikSayin [Kirlik and Sayın, 2014], and TambyVanderpooten [Tamby and Vanderpooten, 2021]. For multiobjective

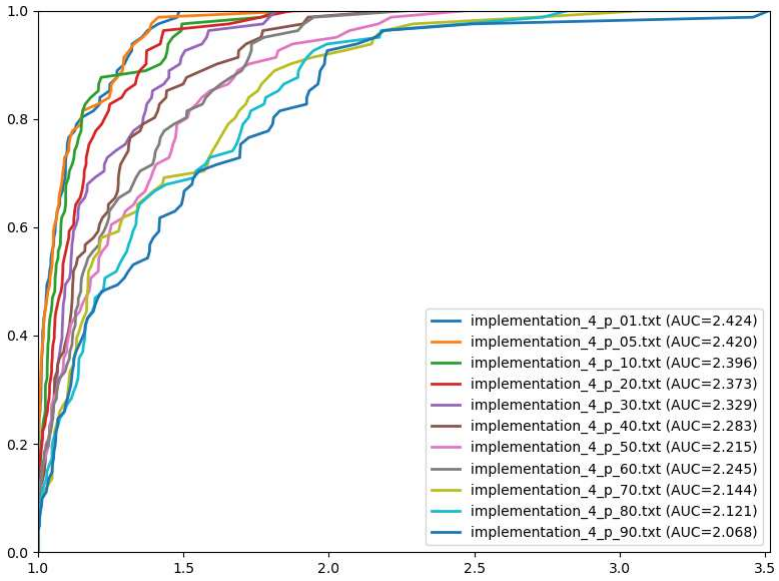


Figure 3.9: Performance profile comparing execution time of Implementation 4 using different percentages of cuts for PILPs.

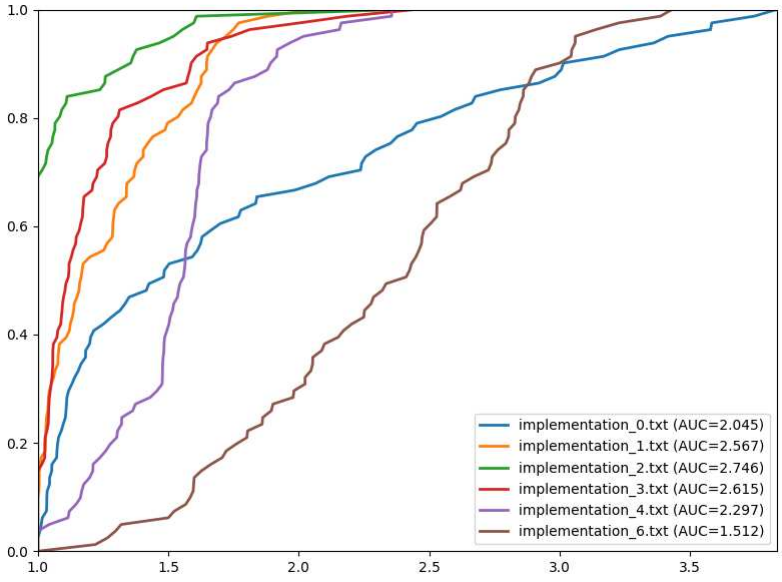


Figure 3.10: Performance profile comparing the execution time of the naive linear version and different implementations of the RVF Algorithm for PILPs.

instances, we considered DominguezRios, KirlikSayin, and TambyVanderpooten.

For the purpose of comparison, we specifically employed the linear version of RVF with Implementation 2. This version was selected based on the findings of our study, which demonstrated its superior performance compared to alternative versions of the RVF Algorithm.

Figure 3.11 presents the performance profile comparing the execution times of the RVF Algorithm with all other existing algorithms considered in our study for the biobjective SPP instances. Based on the performance profile, both RVF and DCM outperform all other algorithms. Moreover, RVF and DCM exhibit nearly identical AUC values. The average execution times (in seconds) for RVF, DCM, EpsilonConstraint, KirlikSayin, TambyVanderpooten, DominguezRios, and Chalmet are 159.67, 462.42, 801.42, 1235.44, 1325.11, 1438.67, and 1479.24, respectively. None of the SPP instances could be solved by the MOMIX method.

When comparing RVF and DCM in terms of average execution time, RVF outperforms DCM by approximately 65.4%, calculated as the ratio of the difference between their average times to the average time of DCM. All existing algorithms implemented using the MOA package perform within a similar range but are significantly less efficient compared to RVF and DCM.

The detailed results for the execution time and the number of NDP for algorithms from the MOA package are presented in Table K.15 for the SPP instances. For RVF, which corresponds to Implementation 2, the results are provided in Table K.3. The corresponding results for DCM are reported in Table K.13.

Figure 3.12 presents the performance profile comparing the execution times of the RVF Algorithm with all other existing algorithms considered in our study for the multiobjective KP instances. Based on the performance profile, both RVF and DCM outperform all other algorithms. Moreover, RVF and DCM exhibit nearly identical AUC values. The average execution times (in seconds) for RVF, DCM, TambyVanderpooten, DominguezRios, KirlikSayin, and MOMIX are 84.17, 11.67, 110.15, 98.47, 469.38, and 1600.08, respectively.

When comparing RVF and DCM in terms of average execution time, DCM outperforms RVF by approximately 86%, calculated as the ratio of the difference between their average times to the average time of RVF. All existing algorithms implemented using the MOA package perform within a similar range but are significantly less efficient compared to RVF and DCM. In addition, MOMIX

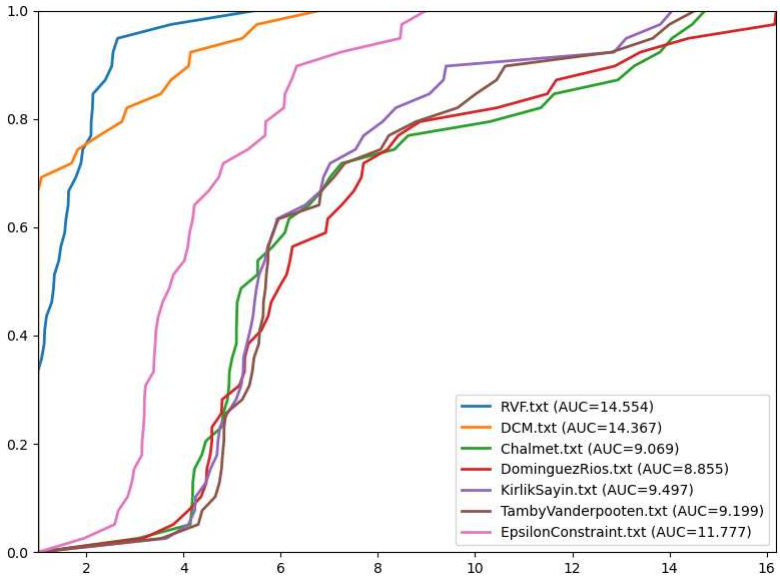


Figure 3.11: Performance profile comparing the execution time of the RVF Algorithm (Implementation 2) with all existing algorithms for the biobjective SPP.

performs the worst among all algorithms.

Table K.14 presents the detailed results for the execution time and the number of NDP obtained using algorithms from the MOA package for the KP instances. The results for RVF (Implementation 2) are shown in Table K.4, while the corresponding outcomes for DCM and MOMIX are summarized in Table K.12.

Please note that for all computational experiments across all algorithms, we report both execution time and the number of NDPs. In rare cases, the number of NDPs may not be identical across different versions; however, the difference is typically within 1–2 NDPs. This small variation is considered acceptable because linear solvers may occasionally encounter computational issues that prevent them from producing an optimal solution. The likelihood of such occurrences is very low, which explains why, in a few rare cases, the reported number of NDPs differs slightly. For the KP and SPP instances, the number of NDPs is known from the literature, and our RVF implementation is consistent with these values. The minor differences mentioned above occur only for the other existing algorithms.

Here, we compare the RVF and DCM methods in more detail. Our experimental results show that, within the prescribed time limit, both RVF and DCM consistently outperform all other existing

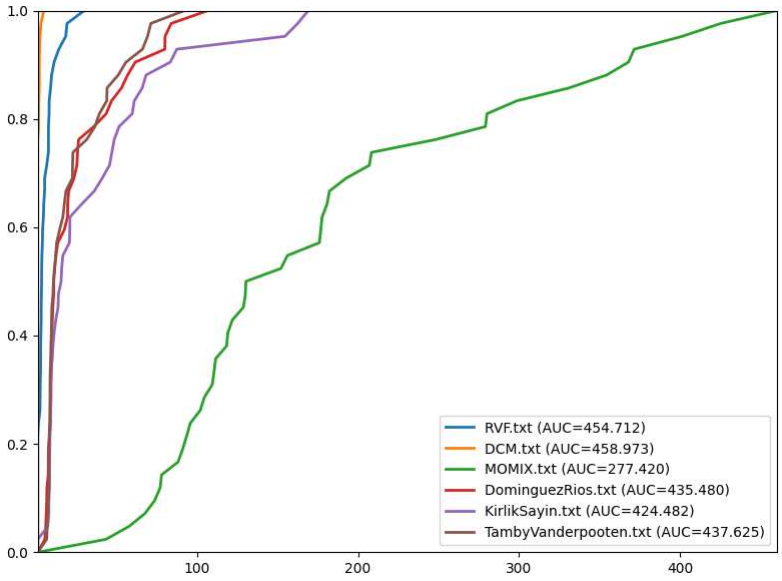


Figure 3.12: Performance profile comparing the execution time of the RVF Algorithm (Implementation 2) with all existing algorithms for the multiobjective KP.

algorithms in terms of runtime. A more nuanced comparison reveals that their relative performance depends on instance complexity. Specifically, KP instances are generally smaller and easier, while SPP instances are more complex, as reflected by their higher average solution times across all solvers.

On the easier KP instances, DCM outperforms RVF by approximately 86%, whereas on the more complex SPP instances, RVF is faster than DCM by about 65%. This behavior indicates that RVF scales more favorably with instance complexity, while DCM is more effective on simpler problems. Overall, despite DCM achieving a slightly higher AUC in the performance profile, RVF outperforms DCM in terms of average runtime by approximately 47%. In particular, the average runtime across all PILP instances is 120 seconds for RVF, compared to 228 seconds for DCM.

Figures 3.13 and 3.14 present the performance profiles comparing the execution times of RVF and DCM for SPP and KP instances, respectively, while Figure 3.15 summarizes the results across all PILP instances.

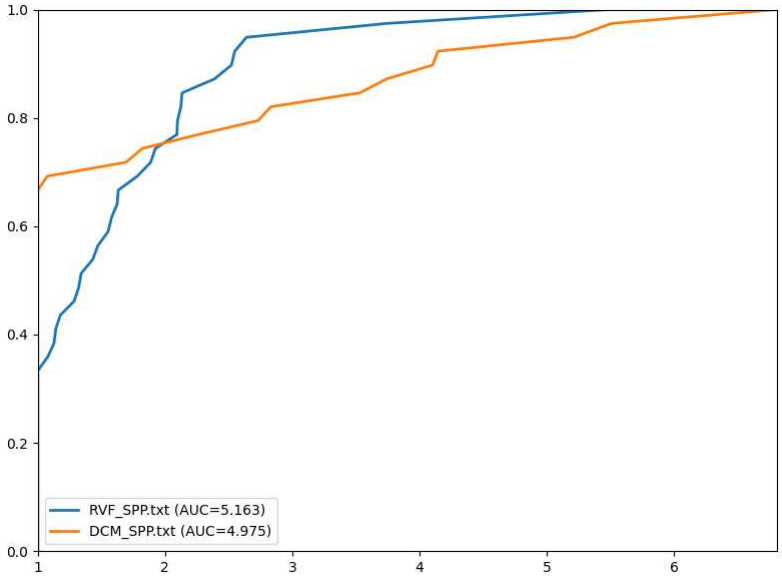


Figure 3.13: Performance profile comparing execution time of RVF and DCM for SPP.

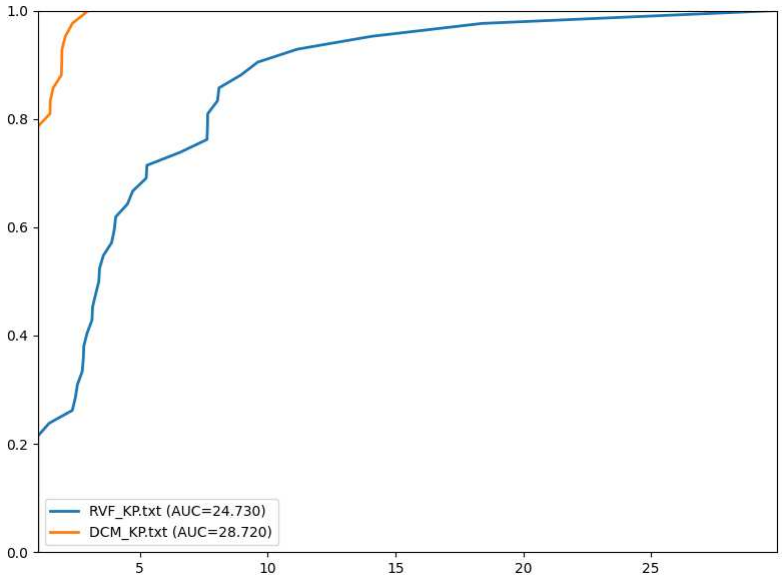


Figure 3.14: Performance profile comparing execution time of RVF and DCM for KP.

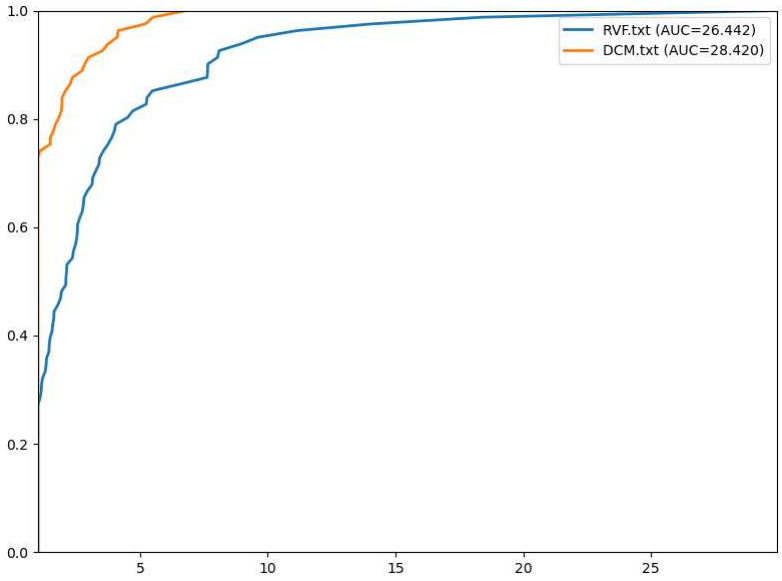


Figure 3.15: Performance profile comparing execution time of RVF and DCM for all PILPs.

3.5.2 General Mixed Integer Instances

3.5.2.1 Experimental Setup

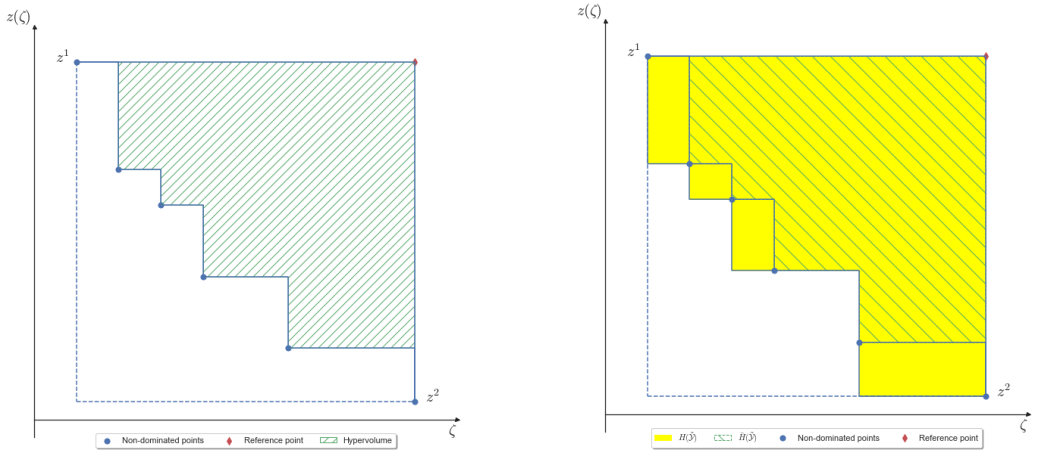
In this study, we impose a time constraint of 4 hours on the execution of each algorithm in order to generate the entire EF. If the time limit is exceeded, the algorithm is terminated.

To the best of our knowledge, there are limited approaches for generating the EF for a (MO-MILP) problem with both integer and continuous variables and an arbitrary number of objective functions. Furthermore, these methodologies do not guarantee access to the information required to construct the exact EF.

One such method is the GoNDEF algorithm recently proposed by Rasmi and Türkay [2019]. The GoNDEF, which is a two-stage algorithm, generates the NDPs, including the line segments or facets consisting of NDPs for MO-MILP. First, the GoNDEF applies a subroutine to fix the efficient integer parts that result in NDPs. Accordingly, fixing the efficient integer parts results in a multiobjective linear program (MOLP) which includes its set of NDPs, but not all the NDPs by MOLP are in the set of NDPs of the (MO-MILP). In summary, the GoNDEF has four main steps; in the first step, it finds a new efficient integer solution; in the second step, it solves the MOLP by fixing the

efficient integer solutions; in the third and fourth steps, it finds all the ND segments and ND facets of the (MO-MILP), respectively, and at the end, the GoNDEF terminates when there are no such new efficient integer solutions. Additionally, for removing the newly found efficient integer solution in the subsequent iterations, GoNDEF uses no-good constraints [Hooker, 1994, 2011]. Due to the unavailability of an open-source implementation of GoNDEF from Rasmi and Türkay [2019], we faced limitations in directly comparing the RVF Algorithm with their method.

Here, we focus on the metrics used to compare multiobjective algorithms for MILPs. The hypervolume indicator introduced by [Zitzler et al., 2003] is a measure used to evaluate the quality of approximate EFs in the biobjective case. It is defined as the area of the intersection of dominated points with $(z_1^2, z_2^1) - \mathbb{R}^2_+$. This is illustrated in Figure 3.16a. A higher hypervolume indicates a better approximation of the true EF. The adjusted hypervolume indicator, introduced by [Boland et al., 2015a], is another measure for evaluating the quality of approximate EFs. It is defined as the union of the hypervolume with rectangles between every two consecutive NDPs. This is illustrated in Figure 3.16b. A lower adjusted hypervolume indicates a better approximation of the true EF. The difference between the adjusted hypervolume and the hypervolume can be used to compare the quality of different approximate EFs.



(a) Hypervolume *(b) Adjusted Hypervolume*

Figure 3.16: Hypervolume and Adjusted Hypervolume.

In this context, we use the same idea as in [Boland et al., 2015a] but slightly differently. We consider the summation of all the areas of the rectangles between every two NDPs in the approximate EF as an indicator, as illustrated in Figure 3.17, and we call it the *volume indicator*. As more NDPs are

found, the rectangles are broken into smaller ones, which results in a smaller total rectangle area. Therefore, as the algorithm progresses and finds new NDPs at each iteration, the value of the volume indicator decreases. This measure can be used to compare the quality of different approximate EFs and track the progress of the algorithm.

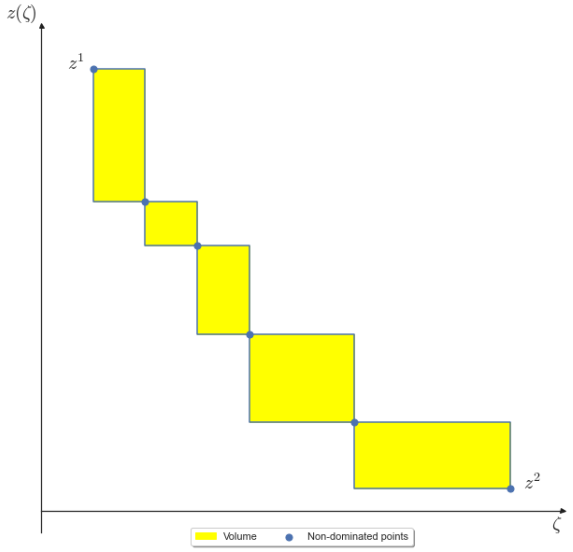


Figure 3.17: Volume.

In multiobjective optimization problems with more than two objectives, it is not possible to define the volume indicator for EFs. Therefore, in this study, we utilize two alternative metrics for comparing approximate EFs in such problems. These metrics, called coverage and uniformity, were introduced in [Sayın, 2000] as indicators of the quality of the discrete representation of EFs for multiobjective optimization problems with any number of objectives. The coverage indicator measures the extent to which a subset of the true EF (i.e., the set of NDPs on the approximate EF) covers the true EF. In other words, a better approximate EF will contain NDPs from every part of the true EF. The uniformity indicator measures the degree of uniformity in the approximate EF, with a better approximate EF being one that does not include discrete NDPs that are very close to each other, thus reducing redundancy. The coverage error is defined by $\epsilon = \max_{z \in Z} \min_{x \in D} d(z, x)$, where d measures the distance (which can be any arbitrary norm), Z is the set of the true EF i.e., NDPs, and D is a subset of Z (i.e., the set of NDPs on the approximate EF). Similarly, the uniformity precision is defined by $\delta = \min_{x,y \in D, x \neq y} d(x, y)$. A smaller value of the coverage error and a larger value of the uniformity precision indicate a better approximate EF.

In order to implement the RVF Algorithm for MILPs, we experimented with various nonlinear solvers or frameworks, namely Gurobi ([Gurobi Optimization, LLC, 2022]), Couenne ([Belotti, 2009]), and SCIP ([Bestuzheva et al., 2021]), and successfully obtained working code for each of them.

However, due to the inherent nonlinearity of the problem in each iteration, we were unable to utilize all of these solvers for our extensive computational experiments. Among the solvers we tested, Gurobi demonstrated superior performance compared to the others, but it still encountered difficulties in solving all of our complex problem instances. This was primarily attributed to the presence of bilinear terms, specifically, $(C_I^{1:l}x_I)^\top u^i$ and $(C_C^{1:l}x_C)^\top u^i$ for $i = 0, 1, \ldots, k-1$, which contribute to the nonlinearity. In Gurobi, it is necessary to set bounds for the terms within the bilinear expressions. Gurobi relies on the McCormick relaxation technique to handle such problems. While the primal variables x_I and x_C were assigned bounds based on the assumption of problem boundedness, we did not have advanced knowledge of the bounds on $u^i,\ i = 0, 1, \ldots, k-1$, i.e., the dual variables corresponding to the parametric constraints. It should be noted that the problem sets the upper bound on $u^i,\ i = 0, 1, \ldots, k-1$ to 0, so we were primarily concerned with establishing a lower bound for $u^i,\ i = 0, 1, \ldots, k-1$. For the PILP cases, we devised a strategy to construct the lower bound on $u^i,\ i = 0, 1, \ldots, k-1$ based on specific properties exhibited by the RVF Algorithm in such cases. This strategy is detailed in Subsection 3.3, specifically in the paragraph labeled as "Setting and Updating Safe Bounds on ζ and Dual Variables ($u^i,\ i = 0, 1, \ldots, k-1$) for PILP cases". For MILPs, we set the lower bound of the dual variables to a very low value, -1×10^{20}, and incorporated a callback with lazy cuts, which we explain in detail in the following section.

3.5.2.2 Evaluating the Efficacy

The MILP cases posed considerable numerical challenges, and we developed several strategies to mitigate these issues. In this section, we first present the general procedure used to avoid numerical difficulties, focusing on the role of callbacks with lazy constraints. After establishing this foundation, we outline the specific nuances introduced in each implementation.

At a high level, we maintain a record of all feasible solutions encountered during the BB search. Each feasible solution has two forms: (i) the *raw* unconverted integer solution reported by the solver, and (ii) the corresponding *converted* efficient solution. Conceptually, this can be viewed as a

map where the keys are unconverted feasible solutions, and the values are their converted efficient counterparts.

During the BB process, whenever the solver identifies a new feasible solution, the callback is triggered. At this point, we perform the following checks:

1. **Duplicate check:** If the new solution already appears as a key or value in the map (i.e., it has been processed before, either in its raw or converted form), we add a lazy constraint to exclude it from further consideration. This prevents the solver from repeatedly exploring the same solution.
2. **Conversion check:** If the new solution does not appear in the map, it is converted into an efficient feasible solution. We then check whether this converted solution already exists in the map.
    - If it does, we add a lazy constraint to eliminate the raw solution.
    - If it does not, we have identified a new efficient solution that corresponds to a new stability region. In this case, we update our list of efficient solutions and extend the EF.
    - In both cases, the map of unconverted and converted solutions is updated accordingly after the conclusion of each iteration.

This procedure describes the general callback mechanism in our framework. Below, we summarize how it is applied and adapted in different implementations:

- **Implementation 0 (Naive version):** For every feasible solution found, we follow the above procedure, but only update the EF and the efficient integer set with the best (optimal) solution at the end.
- **Implementation 1:** In this version, we follow the same procedure for all feasible solutions and add every newly discovered efficient solution to the efficient integer list, rather than only the final optimal solution.
- **Implementation 2 (Feasibility problem):** Here, once a new efficient solution is identified using the above procedure, the BB process is immediately terminated.

- **Implementation 3 (Feasibility problem with θ-objective):** The procedure is applied as above, but the solver is stopped once a designated number of new efficient feasible solutions have been found.
- **Implementation 4 (Cut-pool method):** In each iteration, only a subset of cuts is considered. Accordingly, the map of unconverted-to-converted solutions is restricted to this subset. The same procedure is then applied to the filtered map. At the end, if a solution is determined to be new relative to the entire map, it is added to the efficient integer set and the EF is updated.
- **Implementation 6:** This combines Implementation 4 (cut-pool) with Implementation 2 (feasibility). The solver is stopped as soon as a new efficient integer solution is found.

3.5.2.3 Comparing Methods for Avoiding Cycling

In the following, we discuss the computational experiments conducted with the different implementations discussed in Section 3.3.2. We compare tuned versions with the naive version of the algorithm. Figure 3.18 presents the performance profile comparing the execution time of the RVF Algorithm with implementation_3_n_i, where i ranges from 1 to 5. All versions exhibit similar performance in terms of the AUC value. Therefore, the next metric that allows for a fairer comparison is the average execution time of each version. Note that the average time is computed based only on the instances that were successfully solved by all versions. The average times for $i = 1$ to 5 are 623.10, 412.79, 709.26, 854.91, and 435.98 seconds, respectively. Based on these results, version $i = 2$ achieves the lowest average execution time and can thus be considered the best-performing variant among the *implementation 3* versions, outperforming the worst-performing version $i = 1$ by approximately 33.7%. The corresponding results for the running time and the number of stability regions for all MILP instances using Implementation 3 are reported in Table K.16.

Throughout this chapter, both performance profiles and average execution times are considered for analysis. Each metric has its own advantages and limitations. For instance, performance profiles do not account for the magnitude of differences in execution time across algorithms—for example, the difference between 0.5 and 1 second is treated the same as that between 500 and 1000 seconds, giving both cases equal weight. In contrast, the average time metric naturally weights each instance according to its actual running time, so such differences are not treated equally. However, the

average time metric is sensitive to outliers, meaning that extreme values can disproportionately affect the results. On the other hand, performance profiles are more robust to outliers. Therefore, both metrics have complementary strengths and weaknesses, and it is important to consider them together when evaluating algorithmic performance.

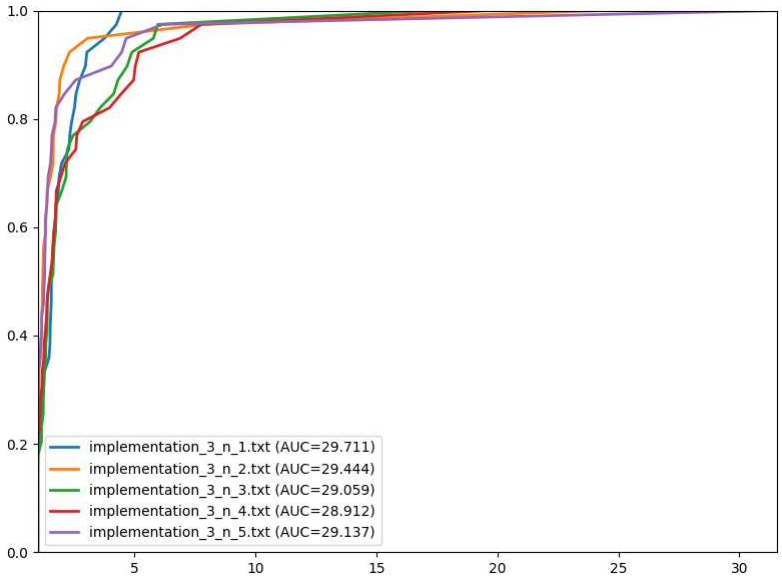


Figure 3.18: Performance profile comparing execution time of Implementation 3 for MILPs.

Similar to Implementation 3, for Implementation 4, we tested several versions. Instead of fixed percentages, the number of cuts added in each iteration was determined by the formula $\min(\text{int}(a \times \max(\text{totalIter}, \text{num_ndp})), \text{num_ndp})$, where a is a scaling factor, totalIter is the total number of iterations, and num_ndp is the current number of NDPs. Figure 3.19 presents the performance profile comparing the execution time of the RVF Algorithm with implementation_4_p_i, where p takes values in the set $\{10, 50, 90\}$. There is no substantial difference among these versions in terms of tuning performance. Based on the AUC values, the 10% version performs slightly better; however, since the differences are marginal, it is also important to examine their average execution times. The average execution times for the 10%, 50%, and 90% versions are 841.4, 893.63, and 1255.34 seconds, respectively. Therefore, based on execution time, the 10% version performs better compared to the others.

We examined several instances and compared the cut ratios of 10%, 50%, and 90% to understand why the 10% version outperformed the other two. The key reason is that during roughly the first

half of the iterations, all three versions behave similarly. However, in the second half, the behavior diverges. In the 10% version, when only a small number of cuts are added, the algorithm often fails to find a new stability region. As a result, it eventually adds all available cuts, which guarantees the discovery of new cuts. In contrast, the 50% and 90% versions add cuts according to their respective ratios, but the number of new cuts discovered in their successful iterations is relatively small. They accumulate a few cuts across multiple iterations, yet the number of cuts obtained in these multiple iterations is often similar to the set that the 10% version obtains in a single iteration when it adds all cuts. This observation supports the intuition behind why the 10% version outperforms the other two.

Table K.17 presents the results for the running time and the number of stability regions across all MILP instances obtained with Implementation 4.

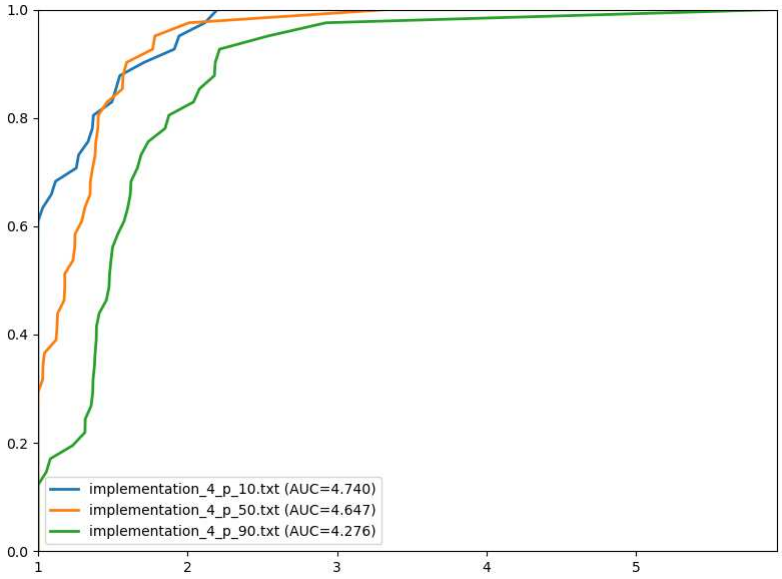


Figure 3.19: Performance profile comparing execution time of Implementation 4 for MILPs.

Figure 3.20 illustrates the performance profile comparing the execution times of the naive version and the various implementations of the RVF Algorithm for MILPs. It is important to note that for Implementations 3 and 4, we selected the best-performing variants, specifically implementation_3_n_2 and implementation_4_p_10. The average running times for Implementations 0, 1, 2, 3, 4, and 6 are 1005.39, 416.05, 1786.98, 330.24, 572.14, and 1784.81 seconds, respectively. In terms of AUC, all implementations perform within a similar range, although Implementation 3 achieves the highest AUC.

Regarding execution time, implementation_3_n_2 again outperforms the other variants. Compared to the naive version (Implementation 0), Implementation 3 improves execution time by approximately 67%. Table K.18 presents the results for the running time and the number of stability regions across all MILP instances obtained using Implementations 0 (naive), 1, 2, and 6.

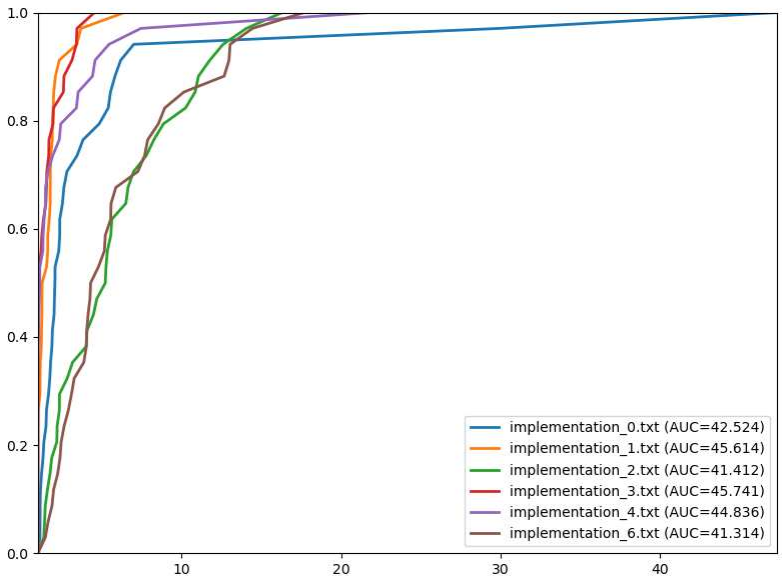


Figure 3.20: Performance profile comparing the execution time of the naive version and different implementations of the RVF Algorithm for MILPs.

3.6 Software

We have developed a Python package implementing our RVF Algorithm, available on GitHub. Separate links are provided for the IP Fallah [2025a] and MILP Fallah [2025c] instances, each containing all versions of the enhanced RVF algorithms.

This package is tailored to enumerate all NDPs for multiobjective integer programs and to generate efficient integer solutions necessary for constructing multiobjective mixed integer programs with any number of objective functions. To run the code successfully, Python 3 and the Gurobi solver must be installed on your machine.

We will provide guidance on how to input your instance into the package. Our approach assumes that the RVF Algorithm solves the multiobjective problem formulated as (MO-MILP). The aim is

to minimize each objective function. For maximization problems, all objective function coefficients need to be multiplied by -1.

To prepare the input, "OBJ" denotes a list of coefficients for the primary objective (represented as c^0 in (RVF)). "Mat" serves as a dictionary for the coefficients of the objectives considered in the constraint set ($C^{1:l}$). "MatFixed" represents a dictionary for the coefficients of the left-hand side of constraints with fixed right-hand sides (A). "RHS" is a dictionary containing the right-hand side values of the constraints (i.e., b in the equation $Ax = b$). For the linear version of IP cases, ensure to set the parameters "M" and "eps" to sufficiently large and small positive values, respectively.

In mixed integer cases where the problem structure involves $A_I x_I + A_C x_C \leq b$, it should be transformed into equality constraints by introducing slack variables. Similarly, for pure integer cases with $A_I x_I \leq b$, slack should be added after ensuring that the coefficients of matrix A and the RHS vector b are integers, given that the slack variable in the pure integer case is treated as an integer in the code.

To run the project, navigate to the directory where the project is saved and execute the Python files by typing "python3 RVF_<version>.py" in the command line. Upon successful completion of the problem, the phrase "Finished!" will be displayed. The RVF Algorithm will generate results in a text file. All NDPs will be presented in a list of lists format, along with the elapsed time shown as "Total Running Time" in seconds. It's worth noting that you can set the maximum runtime using the parameter "timeLimit". If the RVF Algorithm fails to complete within this time limit, it will output "Finished due to time limit!" and display an approximate EF in the result text file. If the "debug_print" parameter is set to True, the variable values will be logged in detail in respective detail files.

3.7 Conclusions

We introduced the RVF algorithm to identify all efficient integer solutions required for constructing the RVF or EF in general multiobjective optimization settings. Unlike existing frontier-construction methods, our approach applies to an arbitrary number of objectives, supports continuous variables, and offers early-termination guarantees. We also provide a Python package implementing the algorithm.

This chapter presents new computational implementations of the RVF Algorithm. The naive version in Section 3.1 constructs the EF or RVF but is not runtime-competitive. We therefore develop several implementations that retain the algorithm's generality while significantly enhancing efficiency.

Section 3.2 compares how the RVF Algorithm and existing approaches scale with respect to subproblem size across iterations.

Section 3.3 introduces six implementations:

- Implementation 1 leverages both feasible and optimal solutions encountered in each BB iteration; nondominated feasible solutions are converted into efficient integer solutions and added to $\mathcal{S}_{\min}$.
- Implementation 2 retrieves the first feasible solution with positive θ encountered while solving a feasibility problem with an arbitrary objective, converts it into an efficient integer solution, and adds it to $\mathcal{S}_{\min}$.
- Implementation 3 is similar but uses θ as the objective and varies the number of feasible solutions examined per iteration.
- Implementation 4 (Cut Pool) selects only a subset of previously generated cuts at iteration k, using $\min\big(\text{int}(a \times \max(\text{totalIter},\ \text{num_ndp})),\ \text{num_ndp}\big)$, to reduce problem difficulty during intermediate iterations.
- Implementation 5 replaces the true value function with its lower bound in selected iterations by temporarily relaxing some primal constraints, restoring them later to recover any missing cuts.
- Finally, Implementation 6 combines nonconflicting improvements (Implementations 2 and 4), employing a cut pool while solving a feasibility problem at each iteration.

For the computational experiments on PILP, we compared multiple implementations of the PILP and then compared the best-performing implementation with existing algorithms. When comparing the naive version with the linearized version of PILP, we observed that linearization improved the running time by 95.6%. For Implementation 3, we tuned the number of solutions examined at each iteration and found that $n = 4$ yielded the best performance. For Implementation 4, we tuned the cut ratio parameter, setting it to 10%. Among all implementations, Implementation 2 outperformed

the others, as expected, since it solves only a feasibility problem, and it outperformed the naive version by 31.23%.

We also compared the best-performing version of the RVF Algorithm for PILP with existing algorithms, including the DCM package and the exact algorithms available in the MOA package. From MOA, we considered both biobjective and multiobjective exact methods. The algorithms implemented in MOA were not competitive with either RVF or DCM. When considering all PILP instances, the RVF Algorithm outperformed DCM by 47%.

The MILP cases posed significant numerical challenges, which were addressed using a branch-and-bound framework with callbacks and lazy constraints. Each feasible solution encountered during the search was tracked in both its raw and converted (efficient) forms to avoid duplicates and dynamically update the EF. Similar to PILP cases, various implementations were tested.

For Implementation 3, we evaluated several parameterized versions ($i = 1$ to 5) to optimize the algorithm's performance, with the parameter controlling aspects of the branch-and-bound search. Among these, version $i = 2$ achieved the fastest average runtime. Similarly, Implementation 4 was tested with different percentages of cuts considered per iteration (10%, 50%, and 90%), where the cut-pool parameter determines the subset of constraints included. The 10% cut version proved most efficient, performing slightly better than the 50% and 90% variants.

Comparing the best-tuned versions of each implementation, Implementation 3 ($i = 2$) consistently achieved the shortest runtime across all MILP instances, outperforming the naive version (Implementation 0) by approximately 67%.

We anticipate that advancements in nonlinear programming will address the computational challenges encountered in this research. Leveraging evolving methodologies and algorithms holds promise for enhancing the efficacy of our proposed RVF Algorithm and contributing to the overall progress in optimization and decision-making processes.

Chapter 4

A Branch-and-Bound Algorithm for Constructing the RVF and EF

In this chapter, we present a branch-and-bound (BB) framework for constructing the restricted value function (RVF) and the efficient frontier (EF) of a multiobjective mixed integer linear program (MILP), along with its implementation using the open-source framework SYMPHONY. Section 4.1 provides an introduction, including background information, related work, and an overview of the chapter.

Section 4.2 develops the theoretical foundations, establishing the relationship between BB trees and valid lower and upper approximations of the RVF. Section 4.3 introduces the concept of tree-based warm-starting and explains how a single BB tree can be iteratively refined for different right-hand sides (RHS).

The algorithmic framework is presented in Section 4.4, where we describe strategies for selecting right-hand sides and ensuring convergence. Section 4.5 discusses the practical implementation of the proposed methods within SYMPHONY.

In Section 4.6, we present computational experiments evaluating the performance of the proposed approach and compare it with the RVF Algorithm introduced in Chapters 2 and 3. Section 4.7 provides details of the software developed for this work, including usage and reproducibility. Finally,

conclusions are presented in Section 4.8.

4.1 Introduction

As discussed in earlier chapters, a number of methods are available for constructing the classical value function of an MILP. These can generally be adapted for constructing the RVF and include both approximation techniques and exact methodologies, typically relying on schemes for iteratively improving lower and/or upper approximations of the RVF (see Section 1.1.2). Similarly, a wide variety of methods are employed to construct the EF in multiobjective optimization, utilizing both scalarization and non-scalarization techniques (see Section 1.1.1). The methods used in multiobjective optimization for constructing the EF are, in general, of a decidedly different nature than those used for constructing the RVF, often relying on the solution of a sequence of subproblems.

In Chapter 3, we described the RVF Algorithm, a generalized cutting-plane algorithm for constructing the RVF based on iteratively reducing the gap between an upper approximation and the true value function. The algorithm refines an approximation until it becomes exact by solving a mixed integer nonlinear program (MINLP) at each step. Here, we propose an alternative algorithm that works entirely within a single BB tree using a virtually unmodified BB algorithm with only the usual machinery of BB. This study builds on and generalizes the framework of [Hassanzadeh and Ralphs, 2014] in which an existence result showed the possibility for an algorithm constructing the classical value function based on BB, but did not specify key elements needed for an actual implementation.

To our knowledge, Forget et al. [2022b] is the only author to propose a strategy for the construction of the EF using an algorithm that works entirely within a single BB tree. The algorithm works by using information from the parent node to warm-start a Benson-like algorithm in each branching node. The authors address the difficulties that arise when non-binary integer variables are introduced in the models and test their algorithm on problems containing non-binary integer variables but no continuous ones.

We propose an alternative that utilizes a pure branch-and-bound approach while addressing the computational challenges posed by the RVF algorithm and its requirement of solving a MINLP at every iteration. The algorithm we propose below differs from the RVF algorithm in two primary ways. First, a lower approximation on the RVF is constructed in addition to the upper approximation.

Second, rather than determining where the gap between the approximation and the true function is maximized, we introduce the option of improving the approximation at randomly sampled points in the function's domain during some iterations. This latter difference eliminates the nonlinearity but makes it more difficult to assess the quality of the approximation and detect convergence. We draw upon insights presented by Hassanzadeh and Ralphs [2014], showing that the full value function can, in principle, be constructed through the exploration of a single BB tree, but they only employ this result in the narrow context of an algorithm for solving two-stage stochastic optimization problems with recourse and do not provide a convergent algorithm for performing this exploration efficiently in the context of multiobjective optimization.

A key innovation first introduced in [Güzelsoy, 2009] and exploited both in [Hassanzadeh and Ralphs, 2014] and in the work presented here is that of warm-starting the solution algorithm rather than initiating it from scratch when solving a sequence of related MILPs. More precisely, the algorithm we introduce in Section 4.4 can be viewed as solving a sequence of MILPs with differing right-hand sides, with each solve conducted from a warm-start, as described in Section 4.2. This procedure allows us to construct a single BB tree that encodes the components required to represent the RVF. The implementation with SYMPHONY [Ralphs and Güzelsoy, 2005] is described in Section 4.5 and computation results are presented in Section 4.6.

In the remainder of this chapter, we use the notation and terminology established in Chapter 1, so we refer the reader there for a refresher on the basic definitions related to the RVF and multiobjective optimization.

4.2 Bounding Functions from BB

In this section, we establish the theoretical foundation for the BB framework developed in this chapter. Our goal here is not yet to describe the algorithmic details of how the tree is refined, but rather to show that BB trees naturally yield both primal and dual bounding functions for the RVF, and that, in principle, there exists a single tree whose associated bounding functions provide the components required to represent the RVF.

The starting point is the observation of Wolsey [1981] that solving a single MILP by BB yields not only an optimal solution value for that instance, but also a dual function that bounds the

corresponding value function from below. This dual function, together with a corresponding primal solution, provides a certificate of optimality by establishing matching upper and lower bounds. In the present setting, we apply this idea to the parametric MILP defining the RVF. Within this framework, both lower and upper approximations are required: the dual function provides a global lower bound, while primal evaluations provide corresponding upper bounds that are essential for detecting convergence.

Extending beyond the solution of a single instance, since the dual feasible region of the linear programming (LP) relaxation at a node does not depend on the right-hand side, the affine lower-bounding functions obtained from the leaf nodes of a BB tree remain valid when the right-hand side is changed. This observation is the key theoretical basis for the tree-based warm-starting procedure developed later in the chapter.

Finally, building on the existence results of Hassanzadeh and Ralphs [2014], we show that there exists a BB tree whose associated bounding functions provide the components required to represent the RVF. This establishes the existence of such a tree. While this result establishes existence, it does not provide a direct method for constructing such a tree, which is the focus of the subsequent sections.

We now introduce the BB framework used to construct such bounding functions. To formalize the bounding functions derived from BB, we briefly introduce the node-level notation used throughout this chapter. The problem of evaluating $z(\hat{\zeta})$ for a given $\hat{\zeta} \in \mathcal{C}$ (a right-hand side feasible for (MILP)) is an MILP and can be solved by BB. Assuming that branching is done by modifying the upper and lower bounds on the variables, the BB tree from evaluating $z(\hat{\zeta})$ consists of a collection of MILP subproblems that differ from the original MILP only by node-specific upper and lower bounds on the integer variables. We denote the variable bounds in node t by $lb^t, ub^t \in \mathbb{Z}^r_+$.

Note that in order to derive globally valid bounding functions, we include nodes that are either infeasible or pruned by bounds for a given instance. In general, the collection of upper and lower bounds must provide a partition of $\mathcal{S}(\zeta)$ for *all* $\zeta \in \mathcal{C}$ in order for the functions we define below to provide valid global bounds.

Definition 4.2.1. *A collection* $\{lb^t, ub^t\}_{t \in T}$ *indexed by set* T *such that* $lb^t, ub^t \in \mathbb{Z}^r_+$ *for* $t \in T$ *is a*

(valid) global disjunction *if*

$$S(\zeta) \subseteq \bigcup_{t \in T} [lb^t, ub^t] \qquad \forall \zeta \in \mathcal{C}.$$

Unless specified otherwise, we assume throughout that the leaf nodes of all BB trees include pruned nodes and that the associated collection of lower and upper bounds constitutes a global disjunction.

Let us now formally consider the problem of evaluating $z(\hat{\zeta})$ for a given $\hat{\zeta} \in \mathcal{C}$ by BB, and let T index the leaf nodes of the resulting tree. Since the subproblem associated with node $t \in T$ is itself an MILP, we can associate with node t the same functions associated with (MILP) in Chapter 2. As such, the *nodal RVF* z^t is defined by

$$z^t(\zeta) = \min \left\{ c^0 x : C^{1:\ell} x \leq \zeta,\ Ax = b,\ lb^t \leq x_I \leq ub^t,\ x \in \mathbb{Z}^r_+ \times \mathbb{R}^{n-r}_+ \right\} \qquad \forall \zeta \in \mathcal{C}, \qquad (\text{RVF}^t)$$

differing from (RVF) only by the introduction of the variable bounds. Similarly, the *nodal RLPVF* z^t_{LP} is an RLPVF arising from the LP relaxation of the subproblem associated with node t, defined by

$$z^t_{LP}(\zeta) = \min \left\{ c^0 x : C^{1:\ell} x \leq \zeta,\ Ax = b,\ lb^t \leq x_I \leq ub^t,\ x \in \mathbb{R}^n_+ \right\} \qquad \forall \zeta \in \mathcal{C}. \qquad (\text{RLP}^t)$$

When the subproblem is infeasible, we define the value of this function to be ∞.

As noted in the previous chapters, although the *primal* feasible region of the subproblem associated with node t depends on ζ, the *dual* feasible region does not, and we can thus utilize the dual of the LP relaxation in node t to alternatively write z^t_{LP} as

$$z^t_{LP}(\zeta) = \sup_{(u,v,\underline{\pi},\overline{\pi}) \in \mathcal{P}^{BB}_D} \zeta^\top u + b^\top v + {lb^t}^\top \underline{\pi} + {ub^t}^\top \overline{\pi} \qquad \forall \zeta \in \mathcal{C}, \qquad (\text{D-RLP-t})$$

where, as before, $u \in \mathbb{R}^\ell_-$ is the vector of dual variables associated with the parametric constraints (RLPVF-b); $v \in \mathbb{R}^m$ is the vector of dual variables associated with the nonparametric constraints (RLPVF-c); and $\overline{\pi} \in \mathbb{R}^n_+$ and $\underline{\pi} \in \mathbb{R}^n_-$ are the dual variables associated with the bound constraints. The dual feasible region for all subproblems is thus

$$\mathcal{P}^{BB}_D = \left\{ (u, v, \underline{\pi}, \overline{\pi}) \in \mathbb{R}^\ell_- \times \mathbb{R}^m \times \mathbb{R}^n_+ \times \mathbb{R}^n_- \ :\ {C^{1:\ell}_C}^\top u + A_C^\top v + \underline{\pi} + \overline{\pi} \leq c^0_C \right\}. \qquad (\text{DF-BB})$$

By weak duality, when the LP relaxation of the subproblem in node t is primal feasible, any dual

solution $(u^t, v^t, \underline{\pi}^t, \overline{\pi}^t) \in \mathcal{P}_D^{BB}$ of the LP relaxation associated with node t defines an affine function bounding z_{LP}^t from below, so that

$$\zeta^\top u^t + b^\top v^t + {lb^t}^\top \underline{\pi}^t + {ub^t}^\top \overline{\pi}^t \leq z_{LP}^t(\zeta)$$

for all $\zeta \in \mathcal{C}$. When the subproblem is primal infeasible, then the dual is unbounded, and we can utilize a dual solution with sufficiently high cost to derive an affine lower bound. The minimum of such affine functions over all leaf nodes of a BB tree yields the dual bounding function, as formalized in the next result.

Theorem 4.2.2. *[Wolsey, 1981] Let $\hat{\zeta} \in \mathcal{C}$ and suppose T indexes the set of terminating nodes of a BB tree resulting from the evaluation of $z(\hat{\zeta})$, including infeasible nodes and nodes pruned by bound, and that $\{lb^t, ub^t\}_{t \in T}$ is a global disjunction.*

Then the function $\underline{z}$, defined by

$$\underline{z}(\zeta) = \min_{t \in T} \left(\zeta^\top u^t + \alpha_t\right) \qquad \forall \zeta \in \mathcal{C},$$

where

$$\alpha_t = b^\top v^t + {lb^t}^\top \underline{\pi}^t + {ub^t}^\top \overline{\pi}^t,$$

and where $(u^t, v^t, \underline{\pi}^t, \overline{\pi}^t) \in \mathcal{P}_D^{BB}$ is either an optimal solution to the LP arising from evaluating $z_{LP}^t(\hat{\zeta})$ or a dual solution with sufficiently high cost for each $t \in T$, is a valid lower bound on z, i.e.,

$$\underline{z}(\zeta) \leq z(\zeta) \qquad \forall \zeta \in \mathcal{C}.$$

Moreover,

$$\underline{z}(\hat{\zeta}) = z(\hat{\zeta}).$$

Proof. As in the statement of the theorem, let T index the leaf nodes of the final BB tree (including nodes that are infeasible or pruned by bound) resulting from evaluating $z(\hat{\zeta})$ for a given $\hat{\zeta} \in \mathcal{C}$ using BB and let $\{lb^t, ub^t\}_{t \in T}$ be the global disjunction arising from the variable bounds for those leaf nodes. For any $\zeta \in \mathcal{C}$, the collection $\{lb^t, ub^t\}_{t \in T}$ defines a partition of the feasible region $\mathcal{S}(\zeta)$ and

we thus have that

$$z(\zeta) = \min_{t \in T} z^t(\zeta) \qquad \forall \zeta \in \mathcal{C}.$$

It then follows immediately that

$$\underline{z}(\zeta) = \min_{t \in T} \left(\zeta^\top u^t + \alpha^t\right) = \min_{t \in T} z^t_{LP}(\zeta) \leq \min_{t \in T} z^t(\zeta) = z(\zeta) \qquad \forall \zeta \in \mathcal{C}.$$

Finally, because the lower bound provided by the BB tree must be optimal for $\hat{\zeta}$, we have that

$$z(\hat{\zeta}) = \min_{t \in T} z^t_{LP}(\hat{\zeta}) = \underline{z}(\hat{\zeta}).$$

■

This result establishes that each BB tree implicitly defines a global lower approximation of the RVF.

Theorem 4.2.2 has an important consequence for the present setting. Since the dual feasible region $\mathcal{P}^{BB}_D$ does not depend on the right-hand side, the affine lower-bounding functions associated with the leaf nodes of a BB tree remain valid when the right-hand side changes. Thus, a tree built while evaluating $z(\hat{\zeta})$ for $\hat{\zeta} \in \mathcal{C}$ can be interpreted as providing a valid lower approximation of the RVF for all $\zeta \in \mathcal{C}$. This is precisely the property on which tree-based warm-starting operates in our framework.

Theorem 4.2.2 makes clear that the lower bounding function $\underline{z}$ arising from a BB tree is defined entirely by a collection of variable bounds constituting a global disjunction, together with a collection of dual solutions in $\mathcal{P}^{BB}_D$. More generally, we can thus represent the lower bounding function associated with a given tree more succinctly by the combination of a global disjunction arising from the collection of variable bounds of the leaf nodes, along with a collection

$$\{(u^t, v^t, \underline{\pi}^t, \overline{\pi}^t)\}_{t \in T}, \tag{4.1}$$

where $(u^t, v^t, \underline{\pi}^t, \overline{\pi}^t) \in \mathcal{P}^{BB}_D$ for each $t \in T$.

In fact, since the feasible region of the dual of the LP relaxation at node t does not depend on the branching bounds, it is not necessary to associate the dual solutions in the collection with specific nodes. We get a potentially stronger dual function by not doing so, as specified in the following

theorem, which makes explicit the role of both primal and dual information in the BB framework. This result shows that any partition of the feasible region, along with any collection of members of $\mathcal{P}_D^{BB}$, naturally induces both primal and dual bounding functions.

Theorem 4.2.3. *Let $\mathcal{V} = \{lb^t, ub^t\}_{t\in T}$ constitute a global disjunction indexed by T and let $\mathcal{D} \subseteq \mathcal{P}_D^{BB}$. Then $\underline{z}_{\mathcal{D}}^{\mathcal{V}}$ defined by*

$$\underline{z}_{\mathcal{D}}^{\mathcal{V}}(\zeta) = \min_{t\in T} \max_{d\in\mathcal{D}} \left(\zeta^\top u_d + b^\top v_d + {lb^t}^\top \underline{\pi}_d + {ub^t}^\top \overline{\pi}_d\right) \qquad \forall \zeta \in \mathcal{C}, \tag{4.2}$$

is a dual bounding function. Further, let $\mathcal{F} \subseteq \mathcal{S}_I$ (as defined in (2.1.2)). Then

$$\min_{x_I\in\mathcal{F}} \bar{z}^{\mathcal{F}}(\zeta; x_I) \qquad \forall \zeta \in \mathcal{C} \tag{4.3}$$

is a primal bounding function.

Proof. The proof that $\underline{z}_{\mathcal{D}}^{\mathcal{V}}$ is a dual function follows the same outline as that of Theorem 4.2.2. The fact that $\bar{z}^{\mathcal{F}}$ follows from Proposition 2.1.4. ■

This result extends Theorem 4.2.2 by highlighting the role of both primal and dual information in the BB framework. It implies that any global disjunction, along with any collection of primal and dual solutions, no matter how they are derived, can be used to derive valid primal and dual bounding functions. The method of warm-starting that will be introduced in Section 4.3 iteratively refines a given global disjunction by solving instances with differing right-hand sides using the global disjunction arising from the leaf nodes of an existing BB tree, along with a collection of all primal and dual solutions produced so far as a warm start.

The refinement process consists of further partitioning some of the leaf nodes of the given tree, and thus results in a collection with a finer partition. The process proceeds by further partitioning one node at a time. The following result formalizes how refinement affects the strength of the dual bounding function.

Theorem 4.2.4. *Let $\mathcal{V}_1 = \{lb^t, ub^t\}_{t\in T_1}$ and $\mathcal{V}_2 = \{lb^t, ub^t\}_{t\in T_2}$ be global disjunctions such that $T_2 = (T_1 \setminus \{t_1\}) \cup \{t_2, t_3\}$ where $lb^{t_1} = lb^{t_2} \leq ub^{t_2} \leq lb^{t_3} \leq ub^{t_3} = ub^{t_1}$. Also, let $\mathcal{D}_1 \subseteq \mathcal{D}_2 \subseteq \mathcal{P}_D^{BB}$. Then the dual bounding function induced by $\mathcal{V}_2$ and $\mathcal{D}_2$ is at least as strong as that induced by $\mathcal{V}_1$*

and $\mathcal{D}_1$*, i.e.,*

$$\underline{z}^{\mathcal{V}_2}_{\mathcal{D}_2}(\zeta) \geq \underline{z}^{\mathcal{V}_1}_{\mathcal{D}_1}(\zeta) \qquad \forall \zeta \in \mathcal{C}.$$

Proof. By construction, we have

$$\min\{z^{t_2}(\zeta), z^{t_3}(\zeta)\} \geq z^{t_1}(\zeta) \qquad \forall \zeta \in \mathcal{C}.$$

The result follows immediately. ■

The theorem shows that if we branch on a single node, no matter how this is done, the result is a strengthening of the dual function, provided we retain and augment the collection of dual solutions. This property provides the key intuition behind tree-based warm-starting and the iterative refinement strategies developed in the following sections because it guarantees that as we branch within the same tree, we are guaranteed to strengthen our function globally. Although this result is stated in terms of branching on a single node, the clear implication is that the same property holds even if we refine multiple nodes.

The following example illustrates the evaluation of the RVF at a single right-hand side and the associated dual bounding function obtained from the resulting BB tree.

Example 6. *Consider the MILP instance used throughout this chapter (see Example 7 for full details). Fix* $\hat{\zeta} = 0$ *and evaluate* $z(0)$ *using BB. At the root node, the LP relaxation has an optimal primal solution*

$$x_1 = 0, \quad x_2 = 0, \quad x_3 = 0.2276, \quad x_4 = 0, \quad x_5 = 0, \quad x_6 = 0.3252, \quad x_7 = 0,$$

with optimal value 0.6504*.*

The corresponding dual solution yields the affine function

$$\underline{z}(\zeta) = -0.1463\,\zeta + 0.6504,$$

which defines a valid lower bound on $z(\zeta)$ *for all* $\zeta \in \mathcal{C}$*.*

At the evaluated right-hand side $\zeta = 0$*, this bound is exact for the root-node LP relaxation, since*

$\underline{z}(0) = 0.6504$.

This example illustrates how solving the LP relaxation at a node yields both a primal solution value and a dual affine function, with the latter providing a valid lower bound on the RVF.

To state the existence result for an exact tree, we next recall the continuous restriction and the minimal set of integer vectors needed to represent the RVF.

Theorem 4.2.5. *There exists a global partition $\mathcal{V}$ arising from the leaf nodes of a single BB tree and a collection $\mathcal{D} \subseteq \mathcal{P}_D^{BB}$ such that $z_{\mathcal{D}}^{\mathcal{V}} = z$ and such that the collection of integer parts of solutions to the LP relaxations associated with the leaf nodes is a description of z, as defined in Theorem 2.1.5.*

Proof. We show that whenever the current global disjunction and collection of dual solutions does not yield the RVF exactly, then there is a way to refine the tree, as in Theorem 4.2.4, and that this process must converge.

As such, let some starting global disjunction $\mathcal{V} = \{lb^t, ub^t\}_{t \in T}$ be given.

Let $\mathcal{D} = \{(u^t, v^t, \underline{\pi}^t, \overline{\pi}^t)\}_{t \in T} \subseteq \mathcal{P}_D^{BB}$ be the collection of optimal dual solutions associated with the LP relaxations at the nodes indexed by T. Let us assume that there exists $\hat{\zeta} \in \mathcal{C}$ such that $z_{\mathcal{D}}^{\mathcal{V}}(\hat{\zeta}) < z(\hat{\zeta})$. Then there must exist $t \in T$ such that $z_{LP}^t(\hat{\zeta}) < z(\hat{\zeta})$. Let x^t be the solution to the LP associated with evaluating $z^t(\hat{\zeta})$. There must be $1 \leq i \leq r$ such that $x_i^t \notin \mathbb{Z}$ and hence we must also have $lb_i^t < ub_i^t$. This means we can further refine node t to obtain a strengthened dual function, as in Theorem 4.2.4.

This process must eventually terminate, since X_{MO} is bounded, so the number of times we can refine before the values of all integer variables are fixed in all nodes must also be finite. ■

The process described in the above proof is roughly the process that the algorithm we specify in the remainder of the section follows, so this proof is not only descriptive, but also prescriptive.

Note that the above proof does not address whether the process yields a collection of integer parts of solutions constituting a description of the RVF, as in Theorem 2.1.5, but we note that this must also be the case. For a given $\hat{x}_I \in \mathcal{S}_I$, the continuous restriction was defined in (CR). As shown in

Theorem 2.1.5, if $S^{\min}$ is a minimal description of z, then for every $\zeta \in \mathcal{C}$, we have

$$z(\zeta) = \min_{x_I \in \mathcal{S}_I} \bar{z}(\zeta; x_I) = \min_{x_I \in \mathcal{S}^{\min}} \bar{z}(\zeta; x_I).$$

Recall that the stability region associated with a given $\hat{x}_I \in \mathcal{S}_I$, denoted by $\mathcal{C}(\hat{x}_I)$, was defined earlier to be

$$\mathcal{C}(\hat{x}_I) = \{\zeta \in \mathcal{C} \ : \ z(\zeta) = \bar{z}(\zeta; \hat{x}_I)\}.$$

That is, $\mathcal{C}(\hat{x}_I)$ is the subset of the domain on which the bounding function associated with $\hat{x}_I$ coincides with the RVF. We now state the main existence result of this section.

Let $\mathcal{S}^{\min}$ be a minimal description of z, as in Theorem 2.1.5. Any given initial BB tree can also be refined so that for each $x_I \in \mathcal{S}^{\min}$, there exists a leaf node t satisfying $lb^t = ub^t = x_I$. Let $\mathcal{V} = \{lb^t, ub^t\}_{t \in T}$ be a global disjunction indexed by T with this property, and let $T^{\min}$ be the indices of nodes associated with members of $\mathcal{S}^{\min}$ in this way. For $t \in T^{\min}$, we have

$$z^t_{LP}(\zeta) = \bar{z}(\zeta; x_I) \qquad \forall \zeta \in \mathcal{C}.$$

The results of this section establish only the existence of such a tree and the validity of the lower-bounding functions derived from BB. The next section addresses the mechanism by which a tree can be refined through warm-starting as the right-hand side changes. After that, Section 4.4 develops the algorithmic framework for choosing the sequence of right-hand sides used to construct the tree in practice.

4.3 Tree-based Warm-Starting

To construct the RVF, one approach is to evaluate (RVF) at multiple right-hand sides ζ by solving corresponding problems of the form (MILP) from scratch and simply combining the resulting lower bounding functions by taking their maximum until convergence. Rather than solving each instance independently, we can instead iteratively refine a single BB tree, reusing information across solves, as indicated in Theorem 4.2.4. In particular, the BB tree constructed for one value of ζ can be used as a starting point for solving the problem at a different right-hand side. This is referred to as

tree-based warm-starting.

To explain this mechanism, consider solving (MILP) for a given value ζ_1 using BB. This produces a tree whose leaf nodes correspond to subproblems defined by branching decisions. Now, suppose we wish to evaluate the problem at a different right-hand side ζ_2. Instead of constructing a new tree from scratch, we can reuse the existing tree structure.

Specifically, for each leaf node, we update the right-hand side of the LP relaxation from ζ_1 to ζ_2 and re-solve the corresponding LP. If the LP relaxation at a node becomes infeasible under the new right-hand side, then the node can be pruned. Otherwise, the node remains feasible and can be further processed, potentially requiring additional branching. In this way, the existing tree is refined to accommodate the new right-hand side.

When infeasibility occurs, a certificate of infeasibility can be obtained via Farkas' lemma. In particular, there exists a dual ray that proves that no feasible solution satisfies the constraints at that node. This justifies pruning the node in the BB tree. In practice, this is handled directly by the LP solver.

From the perspective of the dual function described in Section 4.2, this process systematically strengthens the lower-bounding function. Each time the tree is refined, new dual solutions may be generated, leading to tighter affine lower bounds. In particular, if the current dual function is not exact at a given ζ, solving the MILP at that value of ζ using warm-starting refines the tree in a way that improves the lower bound at that point.

This observation implies that there exists, in principle, a sequence of right-hand sides $\{\zeta^k\}$ such that repeatedly applying tree-based warm-starting yields a BB tree whose associated dual function matches the RVF at the evaluated right-hand sides and provides the components required to represent the RVF. The main challenge, which is addressed in the next section, is to determine how to select such a sequence of right-hand sides efficiently.

The warm-starting methodology described here is implemented natively in the SYMPHONY solver on which our algorithm is built. For additional details on warm-starting and sensitivity analysis in BB methods, particularly in the context of SYMPHONY, we refer the interested reader to [Güzelsoy, 2009, Ralphs and Güzelsoy, 2005, Hassanzadeh and Ralphs, 2014].

4.4 Finite Algorithm for Construction

In this section, we develop algorithmic procedures for constructing the RVF (or equivalently the EF) by iteratively refining a single BB tree using the warm-starting mechanism described in Section 4.3.

As established earlier, there exists a sequence of right-hand sides for which solving the corresponding MILPs with warm-starting yields a BB tree encoding the RVF. The key challenge is how to identify such a sequence efficiently. To address this, we propose two algorithms, referred to as the *sample-based* and *batch-based* approaches.

4.4.1 General Framework

The overall algorithm iteratively refines a single BB tree, yielding a sequence of functions upper and lower bounding the RVF until convergence. Though the method does explicitly construct a dual bounding function, convergence is ultimately based only on the gap between an upper bounding function, similarly arising from a collection of feasible solutions as in (3.1), and the true RVF, just as in the RVF Algorithm. When this gap is (near) zero, then the algorithm terminates. Otherwise, there is some specific right-hand side $\hat{\zeta}$ at which there is a non-zero gap. Warm-starting the problem of evaluating $z(\hat{\zeta})$ results in an improved dual bounding function which is strong at $\hat{\zeta}$. Iterating in this fashion, the algorithm is guaranteed to converge under the assumptions described below.

We begin with an initial right-hand side ζ^1 and solve (MILP). At iteration k, we maintain a BB tree constructed from previous iterations, a set of discovered integer parts $\mathcal{S}^k$, and corresponding upper and lower approximations of the RVF. Given a candidate $\hat{\zeta}$, we evaluate the quality of the current approximation by computing both a lower bound and an upper bound. Convergence is assessed by comparing the primal upper approximation with the dual lower approximation, so both sides of the bounding framework are essential to the algorithm.

Lower Bound. The lower bound is obtained from the dual function induced by the current BB tree. Specifically, evaluating ($\text{RVF}^{\text{LB}}_{\text{Strong}}$) at $\hat{\zeta}$ yields the strongest available lower bound based on all previously generated dual solutions.

$$\underline{z}^k(\zeta) = \min_{t \in \widetilde{\mathcal{T}}^k} \max_{d \in \mathcal{D}^k} \left(\zeta^\top u^d + b^\top v^d + {lb^t}^\top \underline{\pi}^d + {ub^t}^\top \overline{\pi}^d \right) \qquad \forall \zeta \in \mathcal{C}, \qquad (\text{RVF}^{\text{LB}}_{\text{Strong}})$$

where k denotes the current iteration, $\widetilde{T}^k$ is the set of leaf nodes whose associated LP relaxations are feasible for the given ζ, and

$$\{(u^t, v^t, \underline{\pi}^t, \overline{\pi}^t) : d \in \mathcal{D}^k\}$$

is the set of all distinct dual feasible solutions collected up to iteration k.

This expression is a specialization of the dual bounding function $\underline{z}_{\mathcal{D}}^{\mathcal{V}}$ defined in Section 4.2, with $\mathcal{V}$ and $\mathcal{D}$ corresponding to the current tree and the collection of dual solutions accumulated up to iteration k.

For a fixed candidate $\hat{\zeta}$, evaluating ($\text{RVF}^{\text{LB}}_{\text{Strong}}$) at $\zeta = \hat{\zeta}$ yields the strongest valid lower bound available from the current BB tree and the dual information accumulated up to iteration k.

Upper Bound. The upper bound is obtained from the set of discovered integer parts $\mathcal{S}^k$. Using the continuous restriction defined in (CR), the upper bound at $\hat{\zeta}$ is given by

$$\min_{x_I \in \mathcal{S}^k} \bar{z}(\hat{\zeta}; x_I). \tag{RVF^{UB}}$$

For pure integer linear program (PILP) instances, this computation can be simplified using the set of nondominated points (NDP), which is equivalent to the EF, since each integer part corresponds to an NDP.

Gap and Refinement. The difference between the upper and lower bounds at $\hat{\zeta}$ defines a gap. If the gap is sufficiently small, the current tree is already accurate at $\hat{\zeta}$. Otherwise, solving the MILP at $\hat{\zeta}$ using warm-starting refines the tree and improves the approximation.

To illustrate how the lower and upper bounds are constructed and how the gap is reduced through successive refinements of the BB tree, we consider the following example.

Example 7. *Here, we illustrate how to construct the RVF using the SYMPHONY framework with a single BB approach employing the lower approximation method considering* ($\text{RVF}^{\text{LB}}_{\text{Strong}}$). *Consider*

an RVF defined by

$$
\begin{aligned}
z(\zeta) = \quad & \min \ 2x_1 + 5x_2 + 7x_4 + 10x_5 + 2x_6 + 10x_7 \\
& \text{s.t.} \ -x_1 - 10x_2 + 10x_3 - 8x_4 + x_5 - 7x_6 + 6x_7 \leq \zeta \\
& \quad -x_1 + 4x_2 + 9x_3 + 3x_4 + 2x_5 + 6x_6 - 10x_7 = 4 \\
& \quad 5x_2 + x_4 \leq 5 \\
& \quad 5x_2 + x_7 \leq 5 \\
& \quad x_j \in \{0, 1\}, \quad \forall j \in \{1, 2\} \\
& \quad x_j \in \mathbb{R}_+, \quad \forall j \in \{3, 4, \ldots, 7\}.
\end{aligned}
$$

For the purpose of illustration, suppose that the set $\mathcal{S}_{\min}$ and the corresponding continuous variables have been identified. We then calculated the set of corresponding ζ for which there is a one-to-one correspondence between ζs and the stability regions. The corresponding set of right-hand sides is given by $\zeta = \{0, -55.5, -11, -9\}$. Starting with $\zeta = 0$, the MILP $z(0)$ is solved at the root node, and the dual solution corresponding to that node is

- $dual_1 = \{0 : -0.146,\ 1 : 0.163,\ 2 : 0,\ 3 : 0,\ 4 : 2.016,\ 5 : 2.886,\ 6 : 0,\ 7 : 5.341,\ 8 : 9.821,\ 9 : 0,\ 10 : 12.504\}$,

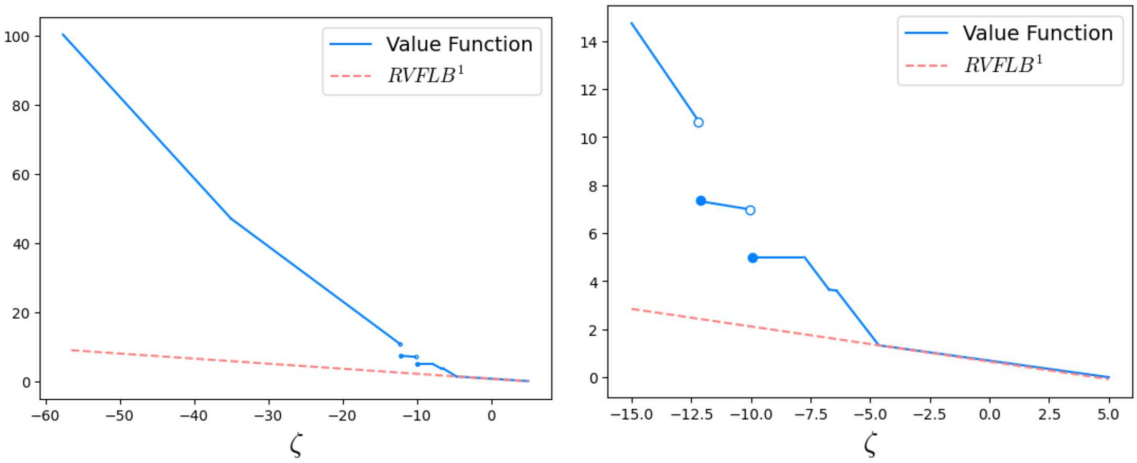
where index 0 corresponds to the dual multipliers corresponding to the parametric constraints, i.e., u; indices 1 to 3 correspond to the dual multipliers related to the constraints with fixed RHSs ($Ax = b$), and indices 4 to 10 correspond to the dual multipliers corresponding to the disjunctions at each node.

To construct the lower bound, we use the formula ($\text{RVF}^{\text{LB}}_{\text{Strong}}$)*, giving us*

- $\zeta \times -0.146 \ + \ 4 \times 0.163 \ + \ 5 \times 0 \ + \ 5 \times 0 \ + \ 0 \times 2.016 \ + \ 0 \times 2.886 \ + \ 0 \times 0 \ + \ 0 \times 5.341 \ + \ 0 \times 9.821 \ + \ 0 \times 0 \ + \ 0 \times 12.504 \ = \ -0.146 \times \zeta + 0.652$,

as the strongest valid lower bound for the RVF so far. This is depicted in Figure 4.1. $RVFLB^1$ refers to the strongest valid lower bound in the first evaluation of the RVF, which is $z(0)$. Note that for $\zeta = 0$, the approximation is exact, but for all other values of ζ, it may or may not be exact. For these other values of ζ, we need to evaluate the dual function using the formulation of Equation ($\text{RVF}^{\text{LB}}_{\text{Strong}}$)*, considering the corresponding potential feasible nodes, which can be determined using Farkas's Lemma.*

Please note that within SYMPHONY's LP solver, a dual multiplier linked to the disjunction in node "t" is provided. If the value is zero, it doesn't contribute to the dual evaluation. When the value is strictly greater than zero, it signifies the dual value connected to the lower bound constraint, denoted as $\underline{\pi}$. Conversely, if the value is strictly less than zero, it indicates the dual value associated with the upper bound constraint, represented as $\overline{\pi}$. This behavior originates from complementary slackness principles; simultaneously, both duals, $\underline{\pi}$ and $\overline{\pi}$, cannot possess non-zero values.



(a) The whole region of ζ *(b) Truncated region of ζ*

Figure 4.1: $z(\zeta)$ and $RVFLB^1$.

Then we set the RHS to -55.5 and re-evaluate the leaf nodes from the preceding BB iteration. Subsequently, for this adjusted RHS, the problem is resolved at its root node, which previously acted as the sole leaf node in the prior BB computation.

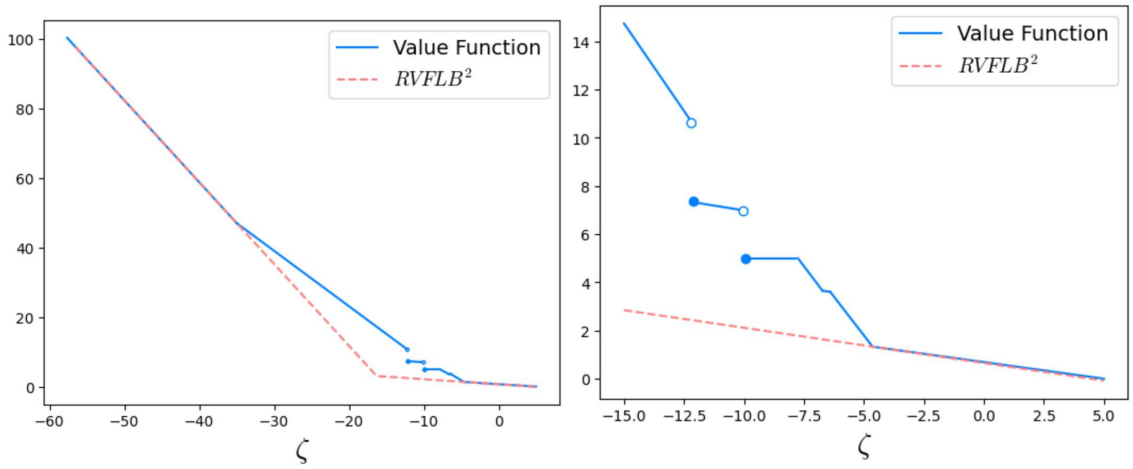
We obtain a new set of duals represented as

- $dual_2 = \{0 : -2.353,\ 1 : -2.412,\ 2 : -4.588,\ 3 : 0,\ 4 : -2.765,\ 5 : 14.059,\ 6 : 45.235,\ 7 : 0,\ 8 : 17.176,\ 9 : 0,\ 10 : 0\}$.

Similarly, to determine the strongest lower bound, we need to compute the dual evaluation corresponding to $dual_2$ as:

- $\zeta \times -2.353 + 4 \times -2.412 + 5 \times -4.588 + 5 \times 0 + 1 \times -2.765 + 0 \times 14.059 + 0 \times 45.235 + 0 \times 0 + 0 \times 17.176 + 0 \times 0 + 0 \times 0 = -2.353\zeta - 35.353$.

Subsequently, the expression represented by Equation ($\text{RVF}^{\text{LB}}_{\text{Strong}}$) *is assessed using the two distinct dual solutions that have been previously derived. This evaluation yields the tightest lower bound obtained so far. The graphical representation of this assessment is depicted in Figure 4.2. RVFLB*^{2} *denotes the strongest valid lower bound obtained during the second evaluation of the RVF, specifically concerning* $z(-55.5)$*. Similarly, for* $\zeta = -55.5$*, the approximation is exact, and for all other values of* ζ*, we need to evaluate the dual function of Equation* ($\text{RVF}^{\text{LB}}_{\text{Strong}}$) *with the corresponding feasible nodes.*



(a) The whole region of ζ *(b) Truncated region of* ζ

Figure 4.2: $z(\zeta)$ *and RVFLB*^{2}*.*

Next, we set $\zeta = -11$ *and re-evaluate the leaf node from the previous BB iteration, which comprises only one node. After solving the new MILP with the BB method for this adjusted RHS, we end up with four separate disjunctions as follows:*

- *Feasible-pruned:* $LBs = \{0 : 0.0,\ 1 : 0.0\},\ UBs = \{0 : 0.0,\ 1 : 0.0\}$,
- *Infeasible-pruned:* $LBs = \{0 : 0.0,\ 1 : 1.0\},\ UBs = \{0 : 0.0,\ 1 : 1.0\}$,
- *Feasible-pruned:* $LBs = \{0 : 1.0,\ 1 : 0.0\},\ UBs = \{0 : 1,\ 1 : 0.0\}$,
- *Feasible-pruned:* $LBs = \{0 : 1.0,\ 1 : 1.0\},\ UBs = \{0 : 1.0,\ 1 : 1.0\}$.

This new iteration yields 5 additional dual solutions:

- $dual_3 = \{0 : -1.077,\ 1 : -0.923,\ 2 : 0,\ 3 : -0.415,\ 4 : 0,\ 5 : 0,\ 6 : 19.077,\ 7 : 1.154,\ 8 : 12.923,\ 9 : 0,\ 10 : 7.646\}$,

- $dual_4 = \{0 : -1.752,\ 1 : -2.051,\ 2 : -0.863,\ 3 : 0,\ 4 : -1.803,\ 5 : 0,\ 6 : 35.983,\ 7 : 0,\ 8 : 15.855,\ 9 : 2.043,\ 10 : 0\}$,

- $dual_5 = \{0 : -1.613,\ 1 : -1.968,\ 2 : 0,\ 3 : 0,\ 4 : -1.581,\ 5 : -3.258,\ 6 : 33.839,\ 7 : 0,\ 8 : 15.548,\ 9 : 2.516,\ 10 : 0\}$,

- $dual_6 = \{0 : -0.687,\ 1 : -0.469,\ 2 : 0,\ 3 : 0,\ 4 : 0.844,\ 5 : 0,\ 6 : 11.094,\ 7 : 2.906,\ 8 : 11.625,\ 9 : 0,\ 10 : 9.437\}$,

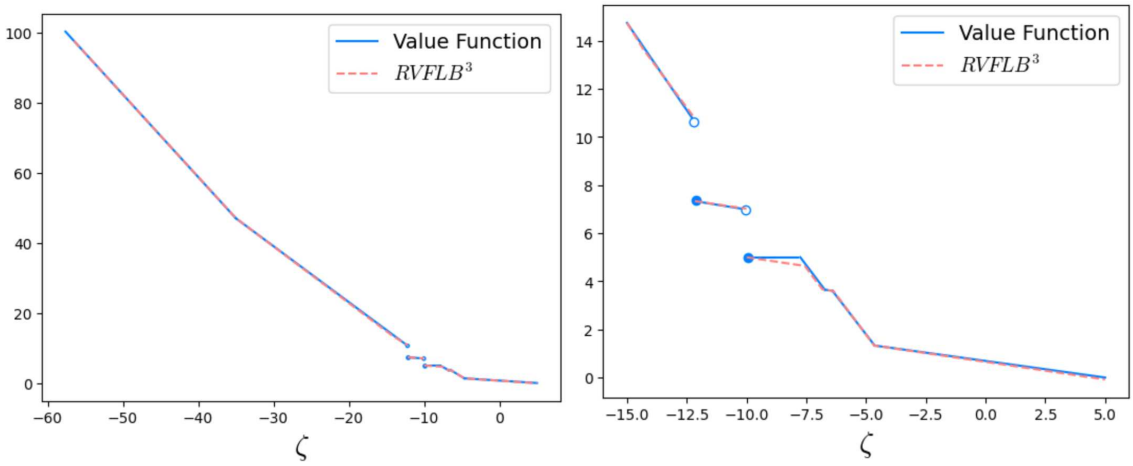
- $dual_7 = \{0 : -1.333,\ 1 : -1.222,\ 2 : 0,\ 3 : 0,\ 4 : -0.555,\ 5 : -3.444,\ 6 : 24.333,\ 7 : 0, 8 : 13.778,\ 9 : 0,\ 10 : 5.778\}$.

Subsequently, we proceed to assess all the dual solutions found so far for all leaf nodes except those that are pruned due to infeasibility. Specifically, we then assess the expression in ($\text{RVF}^{\text{LB}}_{\text{Strong}}$) *using the seven unique dual solutions discovered thus far, considering three leaf nodes or disjunctions that are feasible-pruned. We perform the same calculations as in the first two iterations. Due to space constraints, we omit the detailed mathematical derivations for this iteration and the subsequent one. This evaluation serves as the strongest valid lower bound attained at this iteration. It is illustrated in Figure 4.3. RVFLB*^{3} *denotes the strongest valid lower bound obtained during the third evaluation of the RVF, particularly concerning* $z\,(-11)$*. Likewise, when* $\zeta = -11$*, the approximation holds exact, while for any other* ζ *values, it is necessary to assess the dual function defined by Equation* ($\text{RVF}^{\text{LB}}_{\text{Strong}}$) *alongside the appropriate feasible nodes.*

Subsequently, we set $\zeta = -9$ *and re-evaluate the leaf nodes for this RHS. After solving the new MILP with the BB method for this adjusted RHS, we end up with four separate disjunctions as follows:*

- *Feasible-pruned:* $LBs = \{0 : 0.0,\ 1 : 0.0\},\ UBs = \{0 : 0.0,\ 1 : 0.0\}$,
- *Feasible-pruned:* $LBs = \{0 : 0.0,\ 1 : 1.0\},\ UBs = \{0 : 0.0,\ 1 : 1.0\}$,
- *Over-UB-pruned* $LBs = \{0 : 1.0,\ 1 : 0.0\},\ UBs = \{0 : 1,\ 1 : 0.0\}$,
- *Over-UB-pruned:* $LBs = \{0 : 1.0,\ 1 : 1.0\},\ UBs = \{0 : 1.0,\ 1 : 1.0\}$.

This new iteration yields 2 additional dual solutions:



(a) The whole region of ζ *(b) Truncated region of ζ*

Figure 4.3: $z(\zeta)$ and $RVFLB^3$.

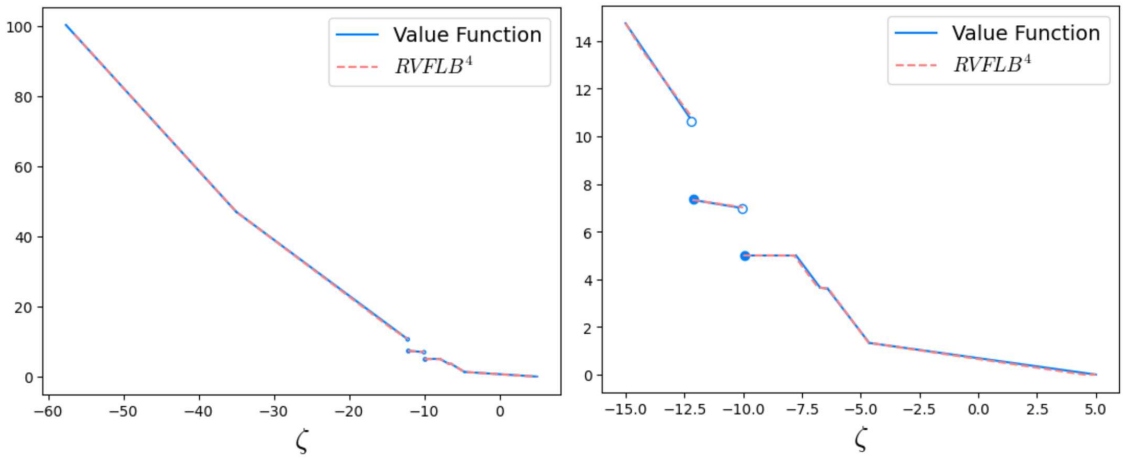
- $dual_8 = \{0 : 0.0,\ 1 : -1.0,\ 2 : 0.0,\ 3 : 0.0,\ 4 : 1.0,\ 5 : 9.0,\ 6 : 9.0,\ 7 : 10.0,\ 8 : 12.0,\ 9 : 8.0,\ 10 : 0.0\}$,

- $dual_9 = \{0 : 0.0,\ 1 : 0.0,\ 2 : 0.0,\ 3 : 0.0,\ 4 : 2.0,\ 5 : 5.0,\ 6 : 0.0,\ 7 : 7.0,\ 8 : 10.0,\ 9 : 2.0,\ 10 : 10.0\}$.

Then, we determine the strongest lower bound by evaluating ($\text{RVF}^{\text{LB}}_{\text{Strong}}$) *using the nine unique dual solutions found so far, and considering four leaf nodes or disjunctions that are feasible-pruned, as depicted in Figure 4.4. $RVFLB^4$ represents the strongest valid lower bound achieved during the fourth evaluation of the RVF, specifically related to $z(-9)$. In the same manner, when $\zeta = -9$, the approximation remains exact, while for any alternative ζ values, it is essential to analyze the dual function described in Equation* ($\text{RVF}^{\text{LB}}_{\text{Strong}}$) *along with the relevant feasible nodes. In this iteration, similar to iteration 3, we have chosen to omit the presentation of the calculation for the expression* ($\text{RVF}^{\text{LB}}_{\text{Strong}}$) *due to space constraints.*

As illustrated in Figure 4.4, the lower approximation aligns with the exact value function in the final iteration, demonstrating that the approximation becomes exact. This alignment is observed after considering four RHSs, corresponding to the four stability regions of this instance. Following four iterations, the approximation achieves exactness.

Figures 4.1 to 4.4 illustrate SYMPHONY's approach in constructing the RVF through a lower



(a) The whole region of ζ *(b) Truncated region of ζ*

Figure 4.4: $z(\zeta)$ and $RVFLB^4$.

approximation technique. Throughout the iterations, this lower approximation consistently approaches the true RVF, and as depicted in the figures, the lower approximation precisely aligns with the RVF for the evaluated RHS values of ζ. For all other RHS values, the lower bound approximation remains a valid lower bound on the RVF, which may or may not be exact.

4.4.2 Methods for Generating Sample Points

To identify candidate right-hand sides, we compute bounds for each component of ζ by solving:

$$\min_{(x_I,x_C)\in X_f} (C_I^{1:l}x_I + C_C^{1:l}x_C)^j, \tag{ζ_{LB}^j}$$

and

$$\max_{(x_I,x_C)\in X_f} (C_I^{1:l}x_I + C_C^{1:l}x_C)^j. \tag{ζ_{UB}^j}$$

We then generate a finite set of candidate values by sampling within these ranges. For multiobjective cases, the final set of ζ values is obtained via a Cartesian product of the sampled values.

Sample-Based Approach. In this approach, candidate ζ values are processed sequentially. At each step, we compute the gap and either skip the sample if the gap is sufficiently small or refine the tree at that ζ using warm-starting.

Batch-Based Approach. In this approach, a batch of candidate ζ values is evaluated simultaneously, and the value with the largest gap is selected for refinement.

4.4.3 Algorithm Description

Both approaches follow the same general structure:

1. Initialize $\mathcal{S}^0 = \emptyset$ and generate an initial set of candidate ζ values.
2. For each selected $\hat{\zeta}$:
   - Compute upper and lower bounds.
   - If the gap is significant, solve (MILP) at $\hat{\zeta}$ using warm-starting.
   - Update $\mathcal{S}$ and the BB tree.
3. Periodically solve an RVF subproblem to check for convergence.
4. Terminate when no further improvement is possible.

The detailed pseudocode for these procedures is provided below.

Sample-based Approach:

1. Initialize $\mathcal{S}^0 = \emptyset$.
2. Call genSamples for each objective (except the objective $c^0 x$) and then take the Cartesian product to generate the sample of ζs, called *finalZetaList*.
3. For $\hat{\zeta}$ in *finalZetaList*:
   - Call the function calcUBLB to obtain the UB and LB of the RVF at $\hat{\zeta}$. If the LB is *None*, it actually equates to 1e+20, indicating that the RVF at $\hat{\zeta}$ is infeasible. In such cases, we proceed to the next sample; otherwise, we continue with the current one.
   - We calculate the gap for the RVF at $\hat{\zeta}$ based on the UB and LB, which are the outputs of calcUBLB.
   - If the gap is less than 0.1, we proceed to the next sample; otherwise, we continue with the current one.

- Call runSymphonySubproblem, indicating that we invoke the SYMPHONY solver with $\zeta = \hat{\zeta}$ and update the set $\mathcal{S}$ if we discover a new stability region. In PILP, we can verify whether the new NDP is in the list of EF so far, or in MILP, we can check if the integer part is in the list of $\mathcal{S}$ so far. However, this is not a sufficient condition for MILP, as there may be more than one integer part for a stability region.

4. We set the variable *flag* to *False* and assign *considered_samples* $\leftarrow$ *finalZetaList*. The *flag* variable helps us determine the optimality condition. When set to *True*, it indicates that the last call of the RVF subproblem has ensured that we have reached optimality, and thus, we are finished and should not proceed with the next sample.

5. While True:

   - If the *flag* variable is True, then break from the loop and return $\mathcal{S}$, i.e., the list of integer efficient solutions required to construct the RVF or EF; otherwise, proceed.
   - We multiply the number of samples by 2.
   - Call genSamples with the new *sample_size* parameter to obtain the new *finalZetaList*. Then, remove the samples that have already been considered from this list.
   - Call genListIdx to obtain the desired list of indices to be considered, then sort the *finalZetaList* using that list of indices.
   - Set *extraIter* to 0.
   - While *finalZetaList*:
     - We call RVF Subproblem - MILP or RVF Subproblem - IP based on the *RVFCallFrequency*. If *extraIter* mod *RVFCallFrequency* equals zero, it indicates that we call the RVF Subproblem. Depending on the call, if we reach optimality, we set the *flag* to True, return $\mathcal{S}$, and break from the loop.
     - If *extraIter* mod *RVFCallFrequency* is not zero, we call calcUBLB repeatedly until a ζ with a potential gap is obtained. Then, we call runSymphonySubproblem for that ζ.
     - *extraIter* += 1
   - If the elapsed time exceeds the time limit, we set the *timeSurpassed* parameter to *True* and then return $\mathcal{S}$.

Batch-based Approach:

1. Initialize $\mathcal{S}^0 = \emptyset$.
2. Call genSamples for each objective (except the objective c^0x) and then take the Cartesian product to generate the sample of ζs, called *finalZetaList*.
3. While *finalZetaList*:
    - Call getZeta to retrieve a sample on batch size. If the return from the function is *None*, we continue and then call getZeta again with the updated finalZetaList. We continue this process until the return is not *None*. When the return is *None*, it indicates one of two cases: the RVF is infeasible for the considered samples, or the corresponding gaps are less than 0.1.
    - Call runSymphonySubproblem, which involves invoking the SYMPHONY solver with the output of getZeta and updating the set $\mathcal{S}$ if a new stability region is discovered. In the case of PILP, we can verify whether the new NDP is already in the list of EF so far, or in MILP, we can check if the integer part is present in the list of $\mathcal{S}$ so far. However, this condition is not sufficient for MILP, as there may be more than one integer part for a stability region.
4. We set the *flag* to *False* and assign *considered_samples* $\leftarrow$ *finalZetaList*.
5. While *True*:
    - If the *flag* variable is *True*, then break from the loop and return $\mathcal{S}$, i.e., the list of integer efficient solutions required to construct the RVF or EF; otherwise, proceed.
    - We multiply the number of samples by 2.
    - Call genSamples with the new *sample_size* parameter.
    - Update the list *finalZetaList* by retaining the samples that are in *finalZetaList* but not in *considered_samples*, effectively removing duplicates.
    - Call genListIdx to obtain the desired list of indices to be considered, then sort the *finalZetaList* using that desired list of indices.
    - Set *extraIter* to 0.

- While *finalZetaList*:
  - We call RVF Subproblem - MILP or RVF Subproblem - IP based on the *RVFCallFrequency*. If *extraIter* mod *RVFCallFrequency* is zero, it means that we call the RVF Subproblem. Depending on the call, if we reach optimality, we set the *flag* to *True*, return $\mathcal{S}$, and break from the loop.
  - If *extraIter* mod *RVFCallFrequency* is not zero, we call getZeta repeatedly until a ζ with a potential gap is obtained. Then, we call SYMPHONY for that ζ.
  - $extraIter += 1$
- If the elapsed time exceeds the time limit, we set the *timeSurpassed* parameter to *True* and then return $\mathcal{S}$.

We also provide detailed explanations of the proposed sample-based and batch-based approaches in pseudocodes BB (Sample-based) - P1 and BB (Sample-based) - P2 for the sample-based approach, and BB (Batch-based) for the batch-based approach, respectively. Note that in the RVF Algorithm, in each iteration, we need to convert the feasible solution to an efficient integer solution. We can employ either one-stage or two-stage conversion, as explained in Section 3.1 of Chapter 3, specifically discussed in Equation 3.36 and Equations 3.37–3.40.

Algorithm calcRVFUBAtZetaBiobjPILP : Calculate the UB of RVF at $\hat{\zeta}$ for Biobjective PILP

Input: $\hat{\zeta}$.
Output: UB of RVF at $\hat{\zeta}$.

```
filteredEF = [].
filteredEF = [NDP for NDP in EF if EF[1] ≤ ζ̂].
if not filteredEF then
    return U.
else
    return min([NDP[0] for NDP in filteredEF]).
end
```

Algorithm calcRVFUBAtZetaMultiobjPILP : Calculate the UB of RVF at $\hat{\zeta}$ for Multiobjective PILP

Input: $\hat{\zeta}$.
Output: UB of RVF at $\hat{\zeta}$.

```
filteredEF = [].
for NDP in EF do
    added ← True.
    for i ← 1 to numObj − 1 do
        if NDP[i] ≥ ζ̂[i-1] then
            added ← False.
            break.
        end
    end
    if added then
        filteredEF.append(NDP).
    end
end
if not filteredEF then
    return U.
else
    return min([NDP[0] for NDP in filteredEF]).
end
```

Algorithm calcUBLB : Calculate UB and LB of RVF at a given $\hat{\zeta}$

Input: $\hat{\zeta}$.
Output: UB, LB.

UB: For PILP, set UB = calcRVFUBAtZetaBiobjPILP($\hat{\zeta}$) for RVF with one parametric constraint or UB = calcRVFUBAtZetaMultiobjPILP($\hat{\zeta}$) for RVF with two or more than two parametric constraints. For MILP, solve $\min \bar{z}(\hat{\zeta};\ x_I), \forall x_I \in \mathcal{S}$ and set UB to the corresponding optimal objective value.

LB: Formulate ($\text{RVF}^{\text{LB}}_{\text{Strong}}$) with $\zeta = \hat{\zeta}$.
Set LB in SYMPHONY as `sym.evaluate_dual_function`(*RHS_list*), where *RHS_list* comprises all RHSs of both parametric and fixed constraints, i.e., *RHS_list* = $(\hat{\zeta}, b)$.

```
if LB == 1e + 20 then
    return UB, None.
end
return UB, LB.
```

Algorithm runSymphonySubproblem : Run SYMPHONY Subproblem

Input: $\hat{\zeta}$.

```
for each i in CONSVARRHS do
    sym.set_row_upper(i, ζ̂).
end
if (sym.warm_solve() == FUNCTION_TERMINATED_ABNORMALLY) then
    print("Something went wrong!").
end
sym.build_dual_function().
opt_sol ← sym.get_col_solution().
if opt_sol then
    Convert the feasible solution to an efficient feasible solution, then save the corresponding NDP
      in temp_ndp and the corresponding integer part and continuous part in int_part and cont_part,
      respectively.
    if temp_ndp not in EF (for PILP) or int_part not in intPartList (for MILP) then
        EF.append(temp_ndp).
        intPartList.append(int_part).
        contPartList.append(cont_part) (for MILP).
    end
end
```

Algorithm genSamples : Generate Samples

Input: *minZeta*, *maxZeta*, *sample_size*.

Output: *samples*.

```
samples ← [minZeta].
interval ← (maxZeta - minZeta) / (sample_size - 1).
for i ← 1 to sample_size − 1 do
    sample ← minZeta + i × interval.
    samples.append(round(sample, 1)).
end
samples.append(maxZeta).
return samples.
```

Algorithm genListIdx : Generate List of Indices

Input: *input_list* ← range(0, length(*tempListZeta*) - 1).
Output: *list_of_indices*.

list_of_indices ← [*input_list*[0], *input_list*[-1], *input_list*[len(*input_list*)//2]].
remaining ← set(*input_list*) - set(*list_of_indices*).
while *remaining* **do**
    sublist_1 ← [x **for** x in *remaining* **if** x ≤ *list_of_indices*[-1]].
    sublist_2 ← [x **for** x in *remaining* **if** x > *list_of_indices*[-1]].
    median_1 ← *sublist_1*[len(*sublist_1*)//2] **if** *sublist_1* **else** *None*.
    median_2 ← *sublist_2*[len(*sublist_2*)//2] **if** *sublist_2* **else** *None*.
    if *median_1* ≠ *None* **then**
        list_of_indices.append(*median_1*).
        remaining.remove(*median_1*).
    end
    if *median_2* ≠ *None* **then**
        list_of_indices.append(*median_2*).
        remaining.remove(*median_2*).
    end
end
return *list_of_indices*.

Algorithm RVF Subproblem - IP

Input: $\mathcal{S}$.
Output: *feas_status*: -1 means infeasibility, *None* means feasibility.

Solve the optimization problem PILP-LP-Feas.
if *PILP-LP-Feas is infeasible* **then**
    return -1.
else
    Determine (x_I, x_C) as the optimal solution to PILP-LP-Feas.
end
Convert (x_I, x_C) to an efficient solution by using one-stage or two-stage conversions.
Set $\mathcal{S} \leftarrow \mathcal{S} \cup \{x_I\}$.
EF.append($C_I^{1:l} x_I + C_C^{1:l} x_C$).
return *None*.

Algorithm RVF Subproblem - MILP

Input: $\mathcal{S}$.

Output: *feas_status*: -1 means infeasibility, *None* means feasibility.

Solve the optimization problem (MINLP^k-Feas).
if (MINLP^k-Feas) *is infeasible* **then**
  return -1.
else
  Determine (x_I, x_C) as the optimal solution to (MINLP^k-Feas).
end
Convert (x_I, x_C) to an efficient solution by using one-stage or two-stage conversions.
if $x_I \notin \mathcal{S}$ **then**
  Set $\mathcal{S} \leftarrow \mathcal{S} \cup \{x_I\}$.
  EF.append($C_I^{1:l} x_I + C_C^{1:l} x_C$).
end
return *None*.

4.5 Implementation

In this section, we describe the practical implementation of the proposed framework using the SYMPHONY MILP solver [Ralphs and Güzelsoy, 2005]. While the theoretical development in the previous sections is solver-independent, the implementation relies on specific capabilities of SYMPHONY for handling BB trees and warm-starting.

A key component of the implementation is the ability to reuse and refine an existing BB tree when solving the (MILP) at different right-hand sides. SYMPHONY provides built-in support for warm-starting, which allows previously explored nodes to be re-evaluated efficiently when the right-hand side changes.

To construct the lower bound, we extract dual information from the LP relaxations at the leaf nodes of the BB tree. These dual solutions are used to define affine lower-bounding functions, as described in Section 4.2. In practice, we maintain a collection of these dual solutions across iterations, which enables efficient evaluation of the lower bound at new right-hand sides.

The evaluation of the dual function is performed using a dedicated routine, which computes the minimum value of the affine functions corresponding to all stored dual solutions. This provides the strongest available lower bound at a given right-hand side.

For the upper bound, we maintain a set of discovered integer solutions, represented by their integer components. For a given right-hand side, the upper bound is computed by evaluating the continuous

Algorithm getZeta : Choose the ζ with the most gap within a batch

Input: *finalZetaList*.
Output: $\hat{\zeta}$.

```
index_to_consider ← empty list.
for i from 0 to batchSize −1 do
    append int((i · (length of finalZetaList - 1)) / (batchSize - 1)) to index_to_consider.
end
zeta_to_consider ← empty list.
for each index in index_to_consider do
    append (index, finalZetaList[index]) to zeta_to_consider.
end
gap_for_zeta ← empty list.
index_to_remove ← empty list.
for each (ind_zeta, _zeta) in zeta_to_consider do
    UB, LB ← findGapOfZeta(_zeta).
    if _LB is None or (UB - LB ≤ 0.1) then
        append ind_zeta to index_to_remove.
        continue.
    end
    append (UB - LB, ind_zeta) to gap_for_zeta.
end
if length of gap_for_zeta is 0 then
    finalZetaList ← [i for j, i in enumerate(finalZetaList) if j not in index_to_remove].
    return None.
end
else
    max_gap ← max(gap_for_zeta, key=lambda  x: x[0]).
    target_index ← max_gap[1].
    gap_zeta ← max_gap[0].
end
append target_index to index_to_remove.
ζ̂ ← finalZetaList[target_index].
finalZetaList ← [i for j, i in enumerate(finalZetaList) if j not in index_to_remove].
return ζ̂.
```

Algorithm BB (Sample-based) - P1 : Algorithm for constructing the RVF or EF for IP and MILP (Part 1)

Output: $\mathcal{S}$: A set of integer parts required to construct the RVF or EF.

Initialize `sym = Symphony()`, `sym.enable_warm_start()`, intPartList or $\mathcal{S}$ = [], contPartList=[] (for MILP), EF=[], *feas_status*=0, for MILP set *defaultDelta* = 0.1 and δ = 0.1, set *timeLimit*, *total_time* = 0, *timeSurpassed* ← *False*, *extra_iter* = 0, *finalZetaList*, set *init_sample_size*, CONSVARRHS: indices of the parametric constraints, *minZeta* and *maxZeta* are vectors where the value of each element j represents the minimum and maximum of the corresponding objective ζ^j, respectively.

st ← *current_time*.

for $\hat{\zeta}$ *in finalZetaList* **do**

    UB, LB = calcUBLB($\hat{\zeta}$).

    if *LB* == *None* **then**

        continue.

    end

    gap_zeta = UB - LB.

    if *gap_zeta* $\leq$ *0.1* **then**

        continue.

    end

    runSymphonySubproblem($\hat{\zeta}$).

end

flag ← *False*.

considered_samples ← *finalZetaList*.

extra_iter = 0.

restriction for each stored integer solution and selecting the minimum value.

When solving the MILP at a new right-hand side, the existing BB tree is reused. Each node in the tree is re-evaluated by updating the right-hand side of its LP relaxation. Nodes that become infeasible are pruned, while feasible nodes are retained and may be further branched. This process refines the tree and potentially generates new dual solutions and integer solutions.

Infeasibility at a node is detected through the LP solver. In such cases, dual rays may be used to identify infeasible regions, although in our implementation, we primarily rely on the solver's built-in pruning mechanisms.

The selection of candidate right-hand sides and the evaluation of upper and lower bounds are integrated with the solver through custom routines. These routines manage the interaction between the algorithmic framework and the solver, including passing updated right-hand sides, retrieving dual solutions, and updating the set of candidate points.

Overall, the implementation leverages SYMPHONY's capabilities for BB and warm-starting, while maintaining additional data structures to track dual solutions and integer solutions across iterations. This enables efficient construction of the RVF through iterative refinement of a single BB tree.

The discussion above outlines a procedure for refining the BB tree through successive evaluations at

Algorithm BB (Sample-based) - P2 : Algorithm for constructing the RVF or EF for IP and MILP (Part 2)

Output: $\mathcal{S}$: A set of integer parts required to construct the RVF or EF.

while *True* **do**
  if *flag* **then**
    return $\mathcal{S}$.
  end
  init_sample_size $*= 2$.
  tempZetaList $= []$.
  for *each i in CONSVARRHS* **do**
    tempZetaList.append(genSamples(*minZeta[i]*, *maxZeta[i]*, *init_sample_size*)).
  end
  Take the Cartesian product of the lists in *tempZetaList* and update *tempZetaList* with the output. If there is only one list, there is no need to take the Cartesian product.
  Remove the samples from *tempZetaList* that are also in *considered_samples*, then sort and update the list *tempZetaList* with this output.
  desired_index_list = genListIdx(range(0, length(*tempListZeta*) - 1)).
  Construct the list *finalZetaList* based on the *desired_index_list*.
  while *finalZetaList* **do**
    if *extra_iter mod RVFCallFrequency* == 0 **then**
      feas_status = RVF Subproblem - IP or RVF Subproblem - MILP.
      if *feas_status* == -1 **then**
        print("Reached to optimality!")
        flag $\leftarrow$ *True*.
        break.
      end
    else
      Extract the first item in *finalZetaList* and call it $\hat{\zeta}$.
      Remove $\hat{\zeta}$ from *finalZetaList*.
      UB, LB = calcUBLB($\hat{\zeta}$).
      if *LB* == *None* **then**
        continue.
      end
      gap_zeta = UB - LB.
      if *gap_zeta* $\leq$ *0.1* **then**
        continue.
      end
      Add $\hat{\zeta}$ to the list of *considered_samples*.
      runSymphonySubproblem($\hat{\zeta}$).
    end
    extra_iter += 1.
  end
  et $\leftarrow$ *current_time*.
  total_time = *et* - *st*.
  if *total_time* > *timeLimit* **then**
    timeSurpassed $\leftarrow$ *True*.
    return $\mathcal{S}$.
  end
end
if *timeSurpassed* **then**
  print("Finished due to time limit!")
else
  print("Finished!")
end

Algorithm BB (Batch-based) : Algorithm for constructing the RVF or EF for IP and MILP

Output: $\mathcal{S}$: A set of integer parts required to construct the RVF or EF.

Initialize `sym = Symphony()`, `sym.enable_warm_start()`, intPartList or $\mathcal{S}$ = [], contPartList=[] (for MILP), EF=[], *feas_status*=0, for MILP set *defaultDelta* = 0.1 and δ = 0.1, set *timeLimit*, *total_time* = 0, *timeSurpassed* ← *False*, *extra_iter* = 0, *finalZetaList*, set *init_sample_size*, CONSVARRHS: indices of the parametric constraints, *minZeta* and *maxZeta* are vectors where the value of each element j represents the minimum and maximum of the corresponding objective ζ^j, respectively.
st ← *current_time*.
while *finalZetaList* **do**
    $\hat{\zeta}$ = getZeta().
    if $\hat{\zeta}$ == *None* **then**
        continue.
    end
    runSymphonySubproblem($\hat{\zeta}$).
end
flag ← *False*.
considered_samples ← *finalZetaList*.
extra_iter = 0.
while *True* **do**
    if *flag* **then**
        return $\mathcal{S}$.
    end
    init_sample_size ∗ = 2.
    tempZetaList = [].
    for *each i in CONSVARRHS* **do**
        tempZetaList.append(genSamples(*minZeta[i]*, *maxZeta[i]*, *init_sample_size*)).
    end
    Take the Cartesian product of the lists in *tempZetaList* and update *tempZetaList* with the output. If there is only one list, there is no need to take the Cartesian product.
    Remove the samples from *tempZetaList* that are also in *considered_samples*, then sort and update the list *tempZetaList* with this output.
    desired_index_list = genListIdx(range(0, length(*tempListZeta*) - 1)).
    Construct the list *finalZetaList* based on the *desired_index_list*.
    while *finalZetaList* **do**
        if *extra_iter mod RVFCallFrequency* == 0 **then**
            feas_status = RVF Subproblem - IP or RVF Subproblem - MILP.
            if *feas_status* == *-1* **then**
                print("Reached to optimality!")
                flag ← *True*.
                break.
            end
        else
            $\hat{\zeta}$ = getZeta().
            if $\hat{\zeta}$ == *None* **then**
                continue.
            end
            runSymphonySubproblem($\hat{\zeta}$).
            Remove $\hat{\zeta}$ from *finalZetaList*.
        end
        extra_iter += 1.
    end
    et ← *current_time*.
    total_time = *et* - *st*.
    if *total_time* > *timeLimit* **then**
        timeSurpassed ← *True*.
        return $\mathcal{S}$
    end
end
if *timeSurpassed* **then**
    print("Finished due to time limit!")
else
    print("Finished!")
end

selected right-hand sides. The following result formalizes the convergence of this framework and shows that the RVF can be constructed exactly in a finite number of iterations.

Theorem 4.5.1 (Finite Convergence of the BB Framework)**.** *Assume that the set $\mathcal{S}^{\min}$ is finite. Then there exists a finite sequence of right-hand sides $\{\zeta^k\}_{k=1}^K$ such that, if the MILP is solved at each ζ^k using tree-based warm-starting, the resulting BB tree induces a lower-bounding function that is valid for all $\zeta \in \mathcal{C}$ and exact at each of the evaluated points $\{\zeta^1, \ldots, \zeta^K\}$.*

Proof. By Theorem 4.2.5, there exists a BB tree whose leaf-node structure contains components associated with all integer vectors in $\mathcal{S}^{\min}$.

Each such leaf node can be obtained by solving the MILP for some right-hand side $\zeta \in \mathcal{C}$ using BB. Moreover, by Theorem 4.2.2, solving the problem at a given ζ^k refines the current tree and produces a lower-bounding function that is exact at ζ^k. In particular, after k iterations, the induced lower-bounding function is exact at each of the evaluated points $\{\zeta^1, \ldots, \zeta^k\}$.

Since the set $\mathcal{S}^{\min}$ is finite, there are finitely many distinct components $\bar{z}(\zeta; x_I)$ associated with the RVF. Each iteration identifies and incorporates at least one such component that is not yet represented in the current tree. Therefore, after a finite number of iterations, all components corresponding to $\mathcal{S}^{\min}$ are included.

At that point, the induced lower-bounding function is valid for all $\zeta \in \mathcal{C}$ and exact at the evaluated points $\{\zeta^1, \ldots, \zeta^K\}$, completing the finite construction. ■

This result establishes that the proposed framework is not only theoretically well-founded but also guarantees finite termination under the stated assumptions. In the next section, we turn to the practical implementation of this approach within the SYMPHONY framework.

4.6 Computational Experiment

In this section, we evaluate the performance of the proposed BB algorithms—both sample-based and batch-based—for constructing the RVF. We compare these approaches against the RVF Algorithm introduced in Chapters 2 and 3, focusing primarily on computational efficiency and scalability.

The experiment utilizes datasets outlined in Subsection 4.6.1, with a comparative analysis of these datasets detailed in Subsection 4.6.2. The experiments were carried out on a machine equipped with dual AMD Opteron 6128 processors and 32 GB of RAM, operating on the Linux (Debian 9) platform. All algorithms were implemented in Python 3. The BB procedures were executed using SYMPHONY, which internally relies on the CLP solver [Ted et al., 2022] for LP relaxations. For computing upper bounds, we used the Gurobi 10 optimizer, while lower bounds were obtained from the dual functions derived from SYMPHONY.

We first describe the datasets used in the experiments, followed by a comparative study of the proposed BB algorithms and the RVF Algorithm. We then present detailed results for both the PILP and MILP settings.

4.6.1 Data Sets

In this experiment, we evaluate the proposed algorithm using benchmark instances from the SPP and the Knapsack problem, as well as randomly generated PILP and MILP instances. The benchmark problems help assess the algorithm's performance on well-established test cases, while the randomly generated instances provide insights into its robustness and effectiveness across a wider variety of problem types.

The datasets employed in this chapter are identical to those used in Chapter 3. Specifically, we use SPP and Knapsack benchmark instances, along with randomly generated PILP and MILP instances. For further details on all the instances mentioned above, please refer to their respective subsections in Section 3.4. For MILP, we use exactly the same instances generated in Chapter 3, whereas for the random PILP instances, we created a new instance structure for this chapter.

Each instance is identified by an ID of the format "IP_obj_{a}_var_{b}_nzr_{c}_u_{d}," where "a" represents the number of objectives, "b" represents the number of variables, "c" represents the non-zero ratio of the coefficient matrix, and "d" represents the upper bound on the integer variable. For the instances provided, we set the number of objectives (i.e., "a") to the list $[2, 3, 4, 5, 6]$, the number of variables (i.e., "b") to the list $[10, 15, 20, 25]$, the non-zero ratio (i.e., "c") to the list $[40\%, 50\%, 60\%]$, and the upper bound on the integer variable (i.e., "d") to the list $[1, 2]$.

4.6.2 A Comparative Study

In this section, we evaluate the effectiveness of our proposed BB algorithms (sample-based and batch-based) for solving multiobjective optimization problems by comparing their performance with that of the RVF Algorithm introduced in Chapters 2 and 3. To achieve this objective, we examine various instances, including biobjective SPP, multiobjective Knapsack (with up to 5 objectives), as well as randomly generated PILP and MILP instances. For further details, all computational experiment results are documented in tables provided within Appendix M.

Before presenting the computational findings, we undertake a comparative analysis between the proposed BB Algorithm and the RVF Algorithm, focusing on their respective execution times for both PILPs and MILPs. It is essential to note that both algorithms are exact methods; however, if terminated prematurely, the outcome represents an approximation of the RVF or EF. A key factor influencing the computational performance of the proposed BB algorithms is the use of tree-based warm-starting.

This mechanism allows successive MILP instances to be solved efficiently by reusing previously constructed BB trees. To illustrate its impact on computational performance, we present the following performance profile comparing execution times of the algorithm with and without warm-starting.

The reported times reflect only the duration required by SYMPHONY to evaluate the function at a new right-hand side, and do not include the time spent solving the RVF subproblem. The benchmark instances are the same as those used in Chapter 3, Section 3.4.

The results demonstrate that warm-starting significantly reduces computational time across a wide range of instances.

This improvement highlights the importance of reusing information across iterations in the proposed BB framework. We now turn to a more detailed comparison between the proposed BB algorithms and the RVF Algorithm, focusing on their computational behavior and structural differences.

In our proposed BB algorithms, nonlinear problems are not addressed unless the RVF Algorithm is invoked. It is important to emphasize that the RVF Algorithm is invoked either at the conclusion to ensure a comprehensive exploration of stability regions necessary for constructing the RVF or through the main algorithm. Specifically for PILPs, the linearized version of the RVF Algorithm

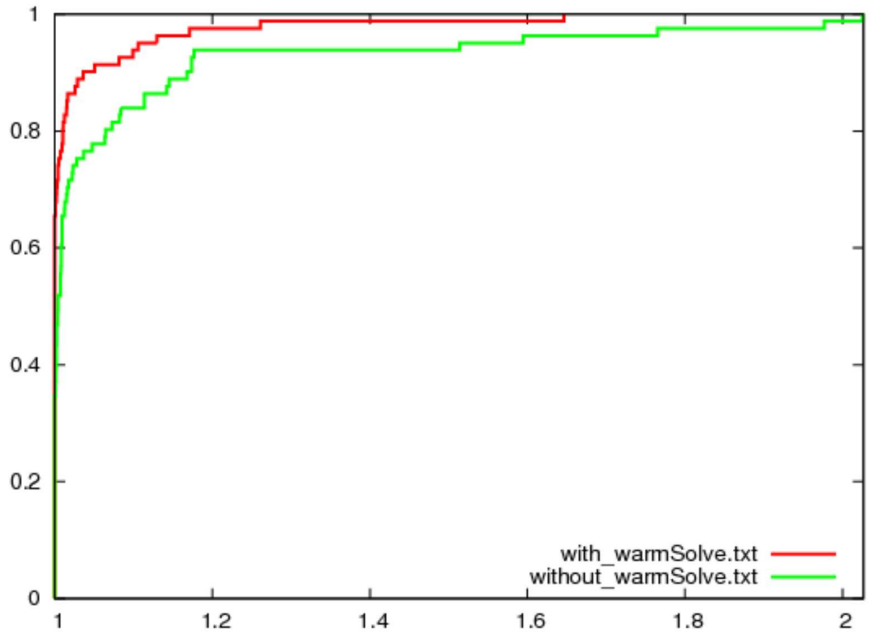


Figure 4.5: Performance profile comparing execution time of the algorithm with and without warm-starting for SPP and KP instances.

is employed. During iterations where the RVF Algorithm is not utilized, a single BB approach is executed, with each iteration initialized by the preceding one. Nonlinear problems are circumvented as we contrast upper and lower approximations using information from the dual solutions found thus far. The overall execution duration of the algorithm relies on the quantity and quality of samples to be evaluated, necessitating adjustment in accordance with the scale and type of instances.

Conversely, the RVF Algorithm tackles an MINLP in each iteration to ascertain the integer component corresponding to a newly identified stability region (linearization is performed for PILP cases). This optimization task determines the maximum discrepancy between the upper approximation and the actual value function (θ in the formulation). With each iteration, a new bounding function (cut) is formulated and incorporated, augmenting the problem's size progressively (the number of variables grows linearly with iterations), and potentially heightening computational complexity as iterations advance. Assuming the existence of N stability regions, the algorithm is obligated to solve $2N$ optimization problems (comprising N MINLPs and N MILPs) to construct the entire RVF.

Both the BB algorithms and the RVF Algorithm exhibit distinct advantages and drawbacks, necessitating computational scrutiny for comparison. We analyzed the same instances used in

Chapter 3, subsection 3.4. Additionally, it is important to note that the solver used within the SYMPHONY framework is CLP, whereas the results for the RVF Algorithm in Chapter 3 were obtained using the Gurobi solver. Therefore, a direct comparison may be unfair, given the differences in performance and robustness between the solvers.

In the following two subsections, we present the computational experiment involving the proposed BB algorithms, along with a comparative analysis against the RVF Algorithm for both PILP (covered in Subsection 4.6.2.1) and MILP (discussed in Subsection 4.6.2.2).

4.6.2.1 Pure Integer Case

Experimental Setup. In this study, we set a time limit of 24 hours for running each algorithm to produce the complete RVF or EF. If the algorithm exceeds this time limit, it will be halted.

In this study, we investigate the biobjective SPP, the multiobjective Knapsack problem, and the randomly generated PILP instances. For all instances, we employ the proposed BB algorithms alongside the linear version of the RVF Algorithm, incorporating Implementation 2. We select the RVF Algorithm with Implementation 2 because our results show that this version outperforms the alternative RVF variants discussed in Chapter 3.

First, we compare the performance of different versions of the BB algorithms, both sample-based and batch-based, across all PILP instances. Finally, we provide some insightful observations from the winning version of the BB algorithms. For both the sample-based and batch-based versions, we conducted a pilot study to tune the RVF call frequency, ultimately setting it to 10.

Figure 4.6 represents the performance profile comparing the execution times of the BB/sample-based Algorithm for different initial sample size parameters across all PILP instances. We provided the performance profiles for each category of the PILP instances as well as for all the PILP instances combined. This approach was taken to determine if the structure of the problem affects the performance of the algorithm. Note that the complexity or size of the instances is within the same range. As shown in the figure, the structure does not significantly affect performance, and with an initial sample size n equal to 10, we achieve the best version of the BB/sample-based Algorithm for the instances selected for our computational experiments. The BB/sample-based Algorithm with n=10 outperformed the one with n=20 by 17.7%, the one with n=30 by 52.77%,

and the one with n=40 by 109.46% in terms of running time. The formula for calculating the gap is $(TT_{n=x} - TT_{n=10})/TT_{n=10}$ multiplied by 100, where $TT_{n=x}$ represents the total running time of the BB/sample-based Algorithm with n=x, and x can take values from the set $\{20, 30, 40\}$.

Figures 4.7, 4.8, and 4.9 present the performance profiles comparing the execution times of the BB/batch-based algorithm with batch sizes of 3, 4, and 5, respectively, across various initial sample size parameters for all PILP instances. Figure 4.10 provides the performance profile comparing the execution times of the BB/batch-based algorithm across all PILP instances and batch sizes for each category. Specifically, for each category, the best-performing version among the different batches is identified and then compared.

From Figure 4.10, for the SPP category, the best-performing version is n_20_b_3, while for the KP and random PILP instances, the best-performing version is n_10_b_3. Here, n_{x}_b_{y} denotes that the initial sample size is x and the batch size is y. When considering all PILP instances collectively, n_20_b_3 outperforms the other versions. Specifically, the batch algorithm with n_20_b_3 surpasses n_20_b_4 and n_20_b_5 by 33.59% and 68.7%, respectively.

Figure 4.11 compares the best-performing versions of the BB/sample-based and BB/batch-based algorithms. Specifically, it presents a performance profile comparing the execution times of these algorithms across all PILP instances. The best version of the sample-based algorithm is the one with an initial sample size of 10 (i.e., n=10), while the best version of the batch-based algorithm is the one with an initial sample size of 20 (i.e., n=20) and a batch size of 3 (i.e., b=3). The sample-based version outperformed the batch-based version by 73% in terms of running time, as calculated by the formula $(TT_{batch} - TT_{sample})/TT_{sample}$ multiplied by 100.

Figure 4.12 presents a performance profile comparing the execution times of the BB Algorithm and the RVF Algorithm (with Implementation 2) across all PILP instances, including SPP, KP, and randomly generated PILP instances. The RVF Algorithm outperforms the BB Algorithm by approximately 76% in terms of average running time. This performance difference is expected, since the RVF Algorithm for PILP cases is already fully linearized and therefore performs efficiently in practice. Moreover, the primary objective of this chapter is not to improve upon RVF for PILP instances, but rather to develop a BB-based algorithm that can outperform RVF in the MILP setting, where a nonlinear subproblem must be solved at each iteration.

We also present the baseline performance profile, using RVF as the reference algorithm, in Figure N.1

in Appendix N. The results are consistent with those obtained from the standard performance profile. Moreover, the area under the curve (AUC) for RVF is higher than that of BB, indicating superior performance of RVF in terms of average running time.

Detailed Analysis: In this section, we provide a comprehensive analysis of the most effective version of the sample-based approach. Figures 4.13-4.20 illustrate the insights derived from our computational experiments. These analyses focus on the best-performing BB/sample-based Algorithm. In the figures, the labels RVF and BB are used, where RVF refers to the RVF call within the BB/sample-based algorithm, while BB refers to the SYMPHONY calls in the BB/sample-based Algorithm. Hence, we examine both the SYMPHONY call and the RVF call as components of the BB/sample-based Algorithm.

Figures 4.13, 4.14, and 4.15 present performance profiles comparing the RVF subproblem and SYMPHONY within the BB Algorithm for all PILP instances in terms of the number of iterations, total time, and average time, respectively. In terms of the total iterations, RVF is less compared to BB by 92.65%, calculated as $(\text{TI}_{\text{BB}} - \text{TI}_{\text{RVF}})/\text{TI}_{\text{BB}}$, where TI refers to the total number of iterations. In terms of the total time, RVF is less compared to BB by 14.06%, calculated as $(\text{TT}_{\text{BB}} - \text{TT}_{\text{RVF}})/\text{TT}_{\text{BB}}$, where TT refers to the total time. In terms of the average running time, RVF is worse compared to BB by 62.81%, calculated as $(\text{AT}_{\text{RVF}} - \text{AT}_{\text{BB}})/\text{AT}_{\text{BB}}$, where AT refers to the average running time.

Figure 4.16 represents the performance profile comparing the total time of solving PILP instances with SYMPHONY, the total time of building the dual function, and the total time of evaluating the dual function within the BB Algorithm. From the figure, we observe that `sym_time` takes more time compared to `build_time` (building the dual function) and `eval_time` (evaluating the dual function). On average, building the dual function takes less time than evaluating the dual function. `sym_time` takes more time compared to `build_time` by 96.24% and compared to `eval_time` by 49.7%, calculated as (sym_time – build_time)/sym_time and (sym_time – eval_time)/sym_time, respectively. Reducing each portion of the total time, including `sym_time`, `build_time`, and `eval_time`, can reduce the overall time of the BB Algorithm. However, since `sym_time` constitutes the largest portion of the total time, improving the SYMPHONY time for solving PILPs in each iteration can significantly reduce the total running time of the algorithm, although the other two factors are also important to consider.

Figure 4.17 presents the performance profile comparing the average time per node of the RVF subproblem and SYMPHONY within the BB Algorithm for all PILP instances. It shows that the time per node of BB is more than RVF by 8.65%, calculated as $(\text{TPN}_{\text{BB}} - \text{TPN}_{\text{RVF}})/\text{TPN}_{\text{BB}}$, where TPN refers to the time per node. The time per node of BB and RVF is not significantly different from each other.

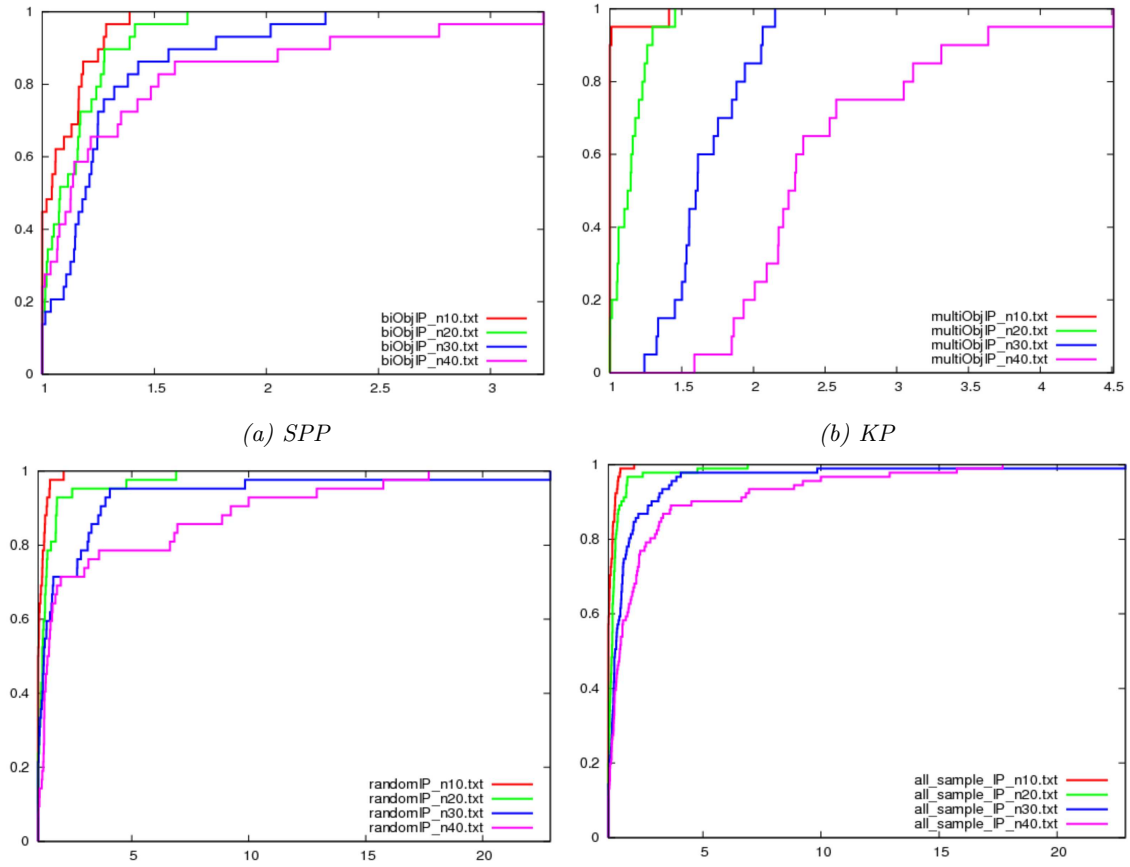
Figures 4.18, 4.19, and 4.20 illustrate the average number of new nodes generated by the RVF subproblem and SYMPHONY within the BB Algorithm for SPP, KP, and random PILP instances. From the figures, it is evident that the new nodes generated by RVF are higher compared to BB. Each iteration of RVF starts from scratch, while each iteration of SYMPHONY performs a warm-start from the previous iteration. This contributes to the difference between the BB and RVF approaches in terms of the number of new nodes generated. Another difference is that the solvers for BB and RVF are different: CLP is the solver for SYMPHONY, and Gurobi is the solver for RVF.

We also conducted another analysis, which provided us with additional insights. We observed that the RVF Algorithm outperformed the BB Algorithm, so we performed an analysis to understand what changes could make BB comparable to RVF. To do this, we considered the total time of RVF, the time of the last iteration of RVF within BB, the average time per iteration of SYMPHONY, and the number of NDPs for each PILP instance. For each instance, we calculated $(\text{TT}_{\text{RVF}} - \text{T}_{\text{last_iter_RVF}})/(\text{AT}_{\text{sym}} \times \text{num_NDP})$, where TT_{RVF} is the total time of the RVF Algorithm, $\text{T}_{\text{last_iter_RVF}}$ refers to the time of the last iteration of the RVF within the BB Algorithm, AT_{sym} is the average time per iteration of SYMPHONY, and num_NDP refers to the corresponding number of NDPs.

The reason we performed this calculation is that, in an ideal BB Algorithm, only one iteration of the RVF call would be necessary to find all efficient integer solutions. Therefore, we take the difference between the RVF Algorithm's total time and the last iteration of RVF within the BB Algorithm to determine the gap our BB Algorithm needs to close to be comparable. We then divide this value by the average time per iteration of SYMPHONY and the number of NDPs. If the final result is x, it means that for the corresponding instance, every x sample should yield a new NDP or efficient integer solution corresponding to a new stability region.

For PILP, if we average this final result across all instances, we get a value of 2.14. This means that for PILP instances, if we have a highly efficient BB Algorithm that finds a new NDP or efficient solution every 2.14 samples or iterations, the BB and RVF Algorithms would be comparable.

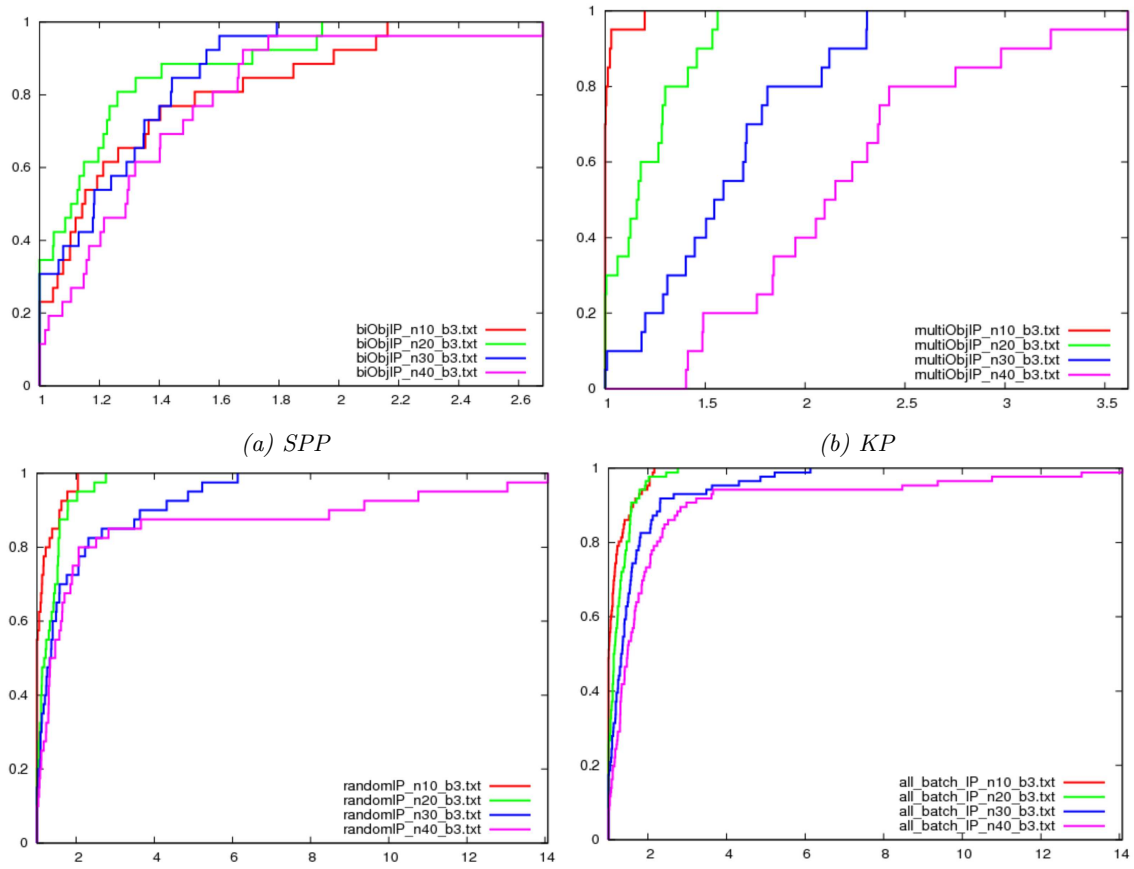
Detailed information on the running times for all PILP instances discussed in the analyses above is provided in Tables M.1–M.8.



(a) SPP *(b) KP*

(c) Random PILP instances *(d) All PILP instances*

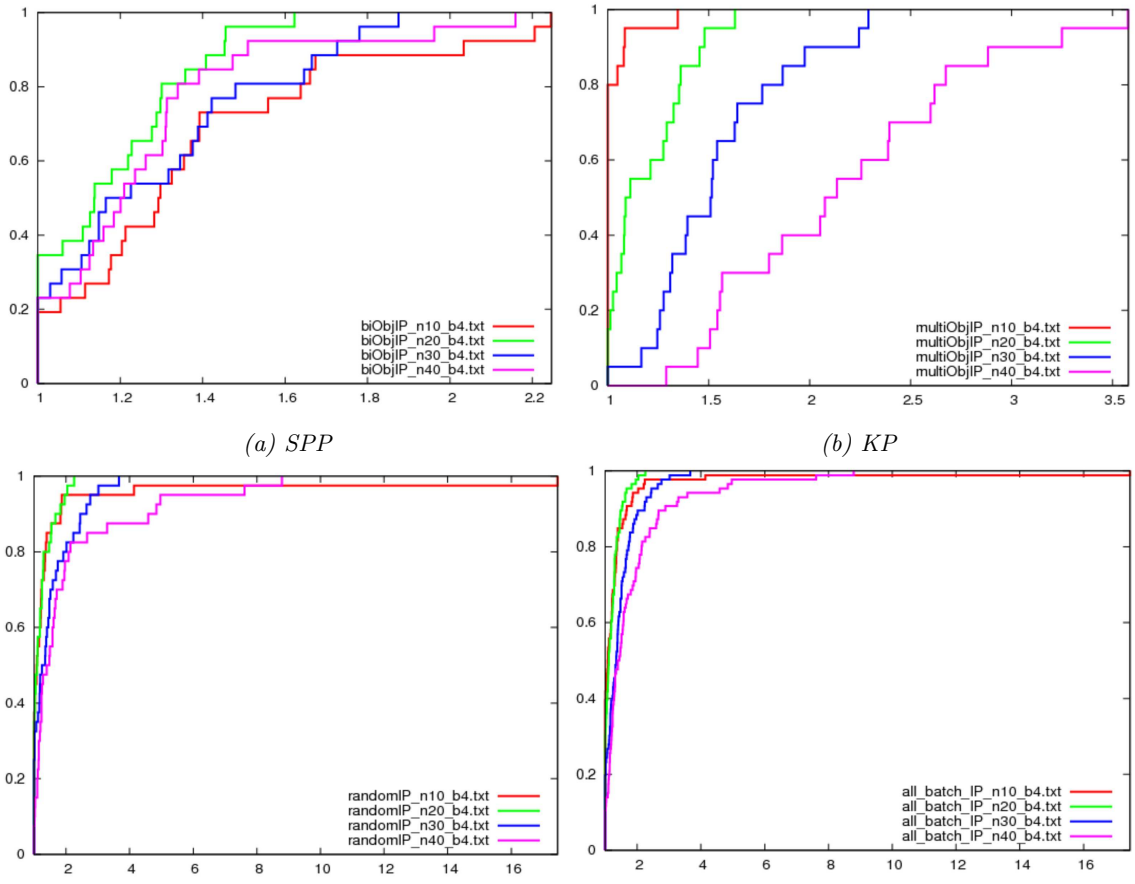
Figure 4.6: Performance profile comparing the execution times of the BB/sample-based Algorithm for different initial sample size parameters across all PILP instances.



(a) SPP *(b) KP*

(c) Random PILP instances *(d) All PILP instances*

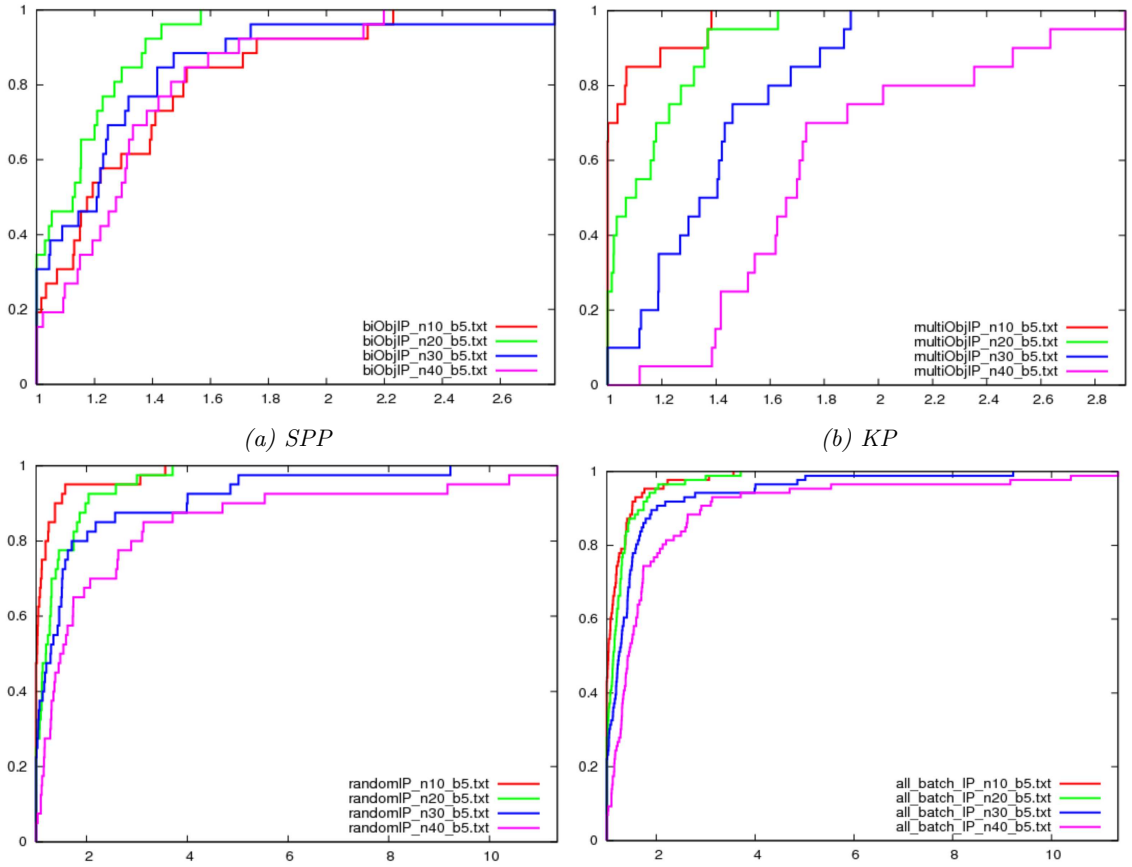
Figure 4.7: Performance profile comparing the execution times of the BB/batch-based algorithm, with a batch size of 3, for different initial sample size parameters across all PILP instances.



(a) SPP *(b) KP*

(c) Random PILP instances *(d) All PILP instances*

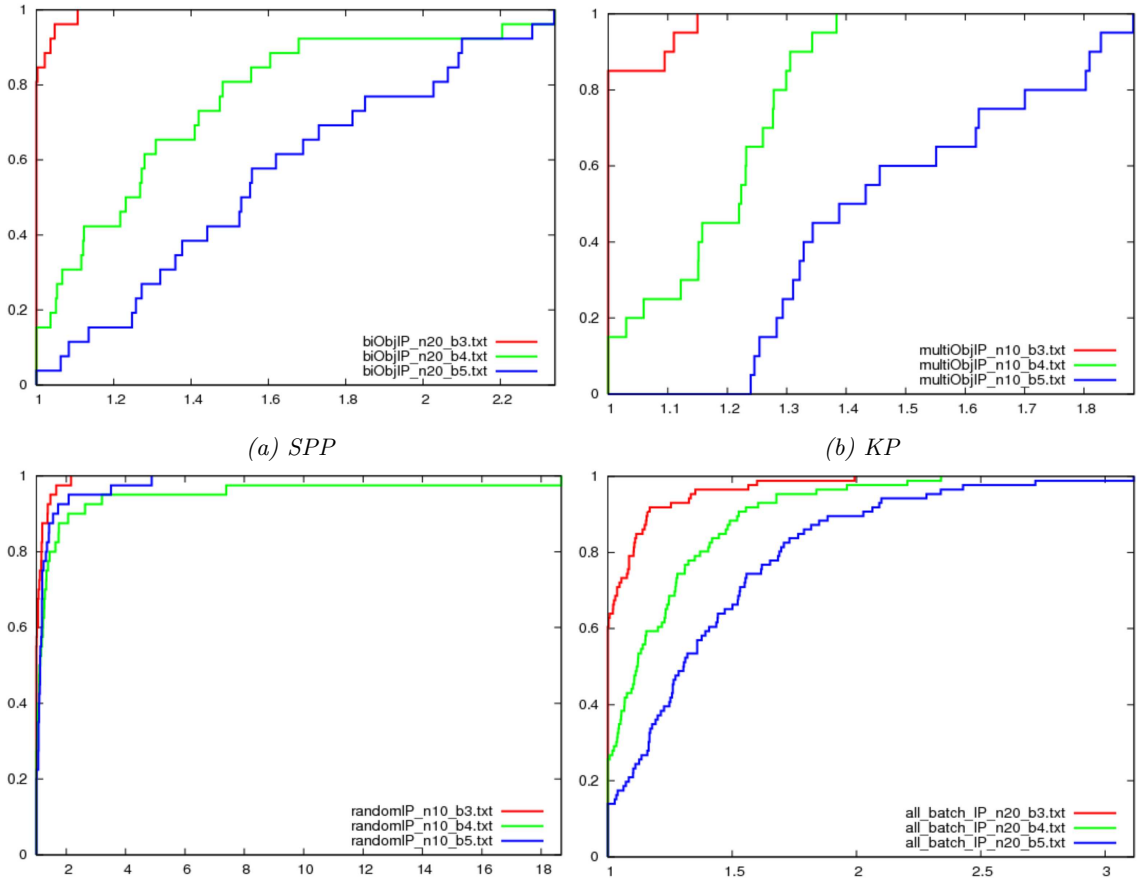
Figure 4.8: Performance profile comparing the execution times of the BB/batch-based algorithm, with a batch size of 4, for different initial sample size parameters across all PILP instances.



(a) SPP *(b) KP*

(c) Random PILP instances *(d) All PILP instances*

Figure 4.9: Performance profile comparing the execution times of the BB/batch-based algorithm, with a batch size of 5, for different initial sample size parameters across all PILP instances.



(a) SPP *(b) KP*

(c) Random PILP instances *(d) All PILP instances*

Figure 4.10: Performance profile comparing the execution times of the BB/batch-based algorithm for each category across all PILP instances.

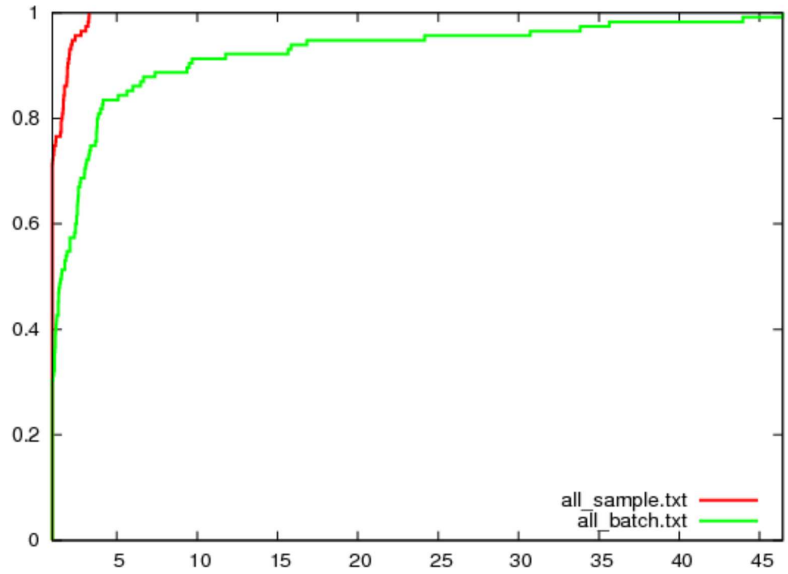


Figure 4.11: Performance profile comparing the execution times of the BB/sample-based and BB/batch-based algorithms across all PILP instances.

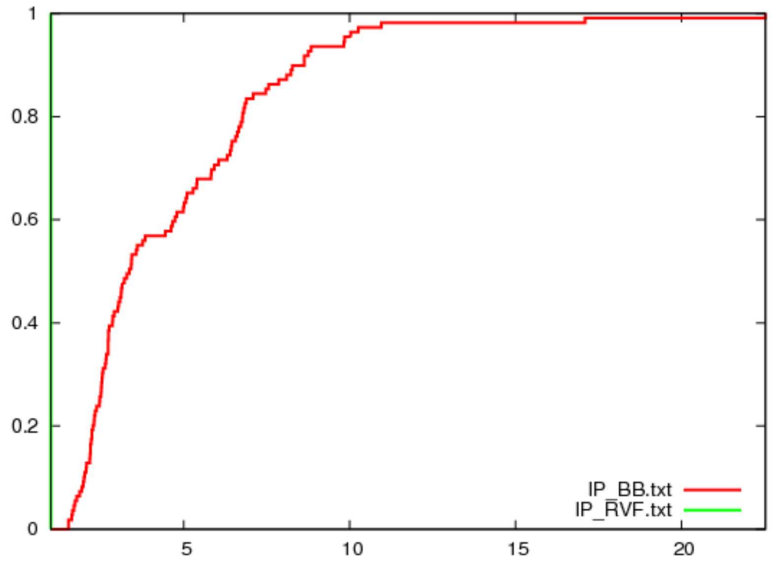


Figure 4.12: Performance profile comparing the execution times of the BB Algorithm and the RVF Algorithm (with Implementation 2) for all PILP instances, including SPP, KP, and randomly generated PILP instances.

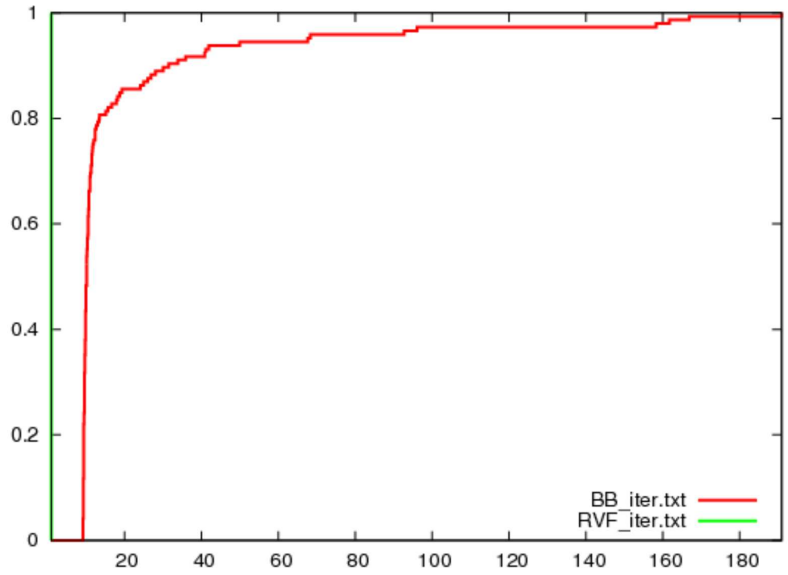


Figure 4.13: Performance profile comparing the number of iterations of the RVF subproblem and SYMPHONY within the BB Algorithm for all PILP instances.

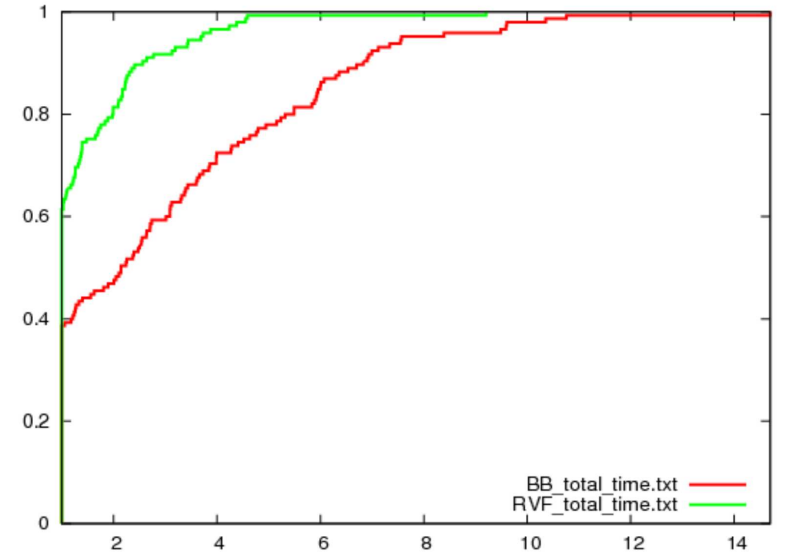


Figure 4.14: Performance profile comparing the total time of the RVF subproblem and SYMPHONY within the BB Algorithm for all PILP instances.

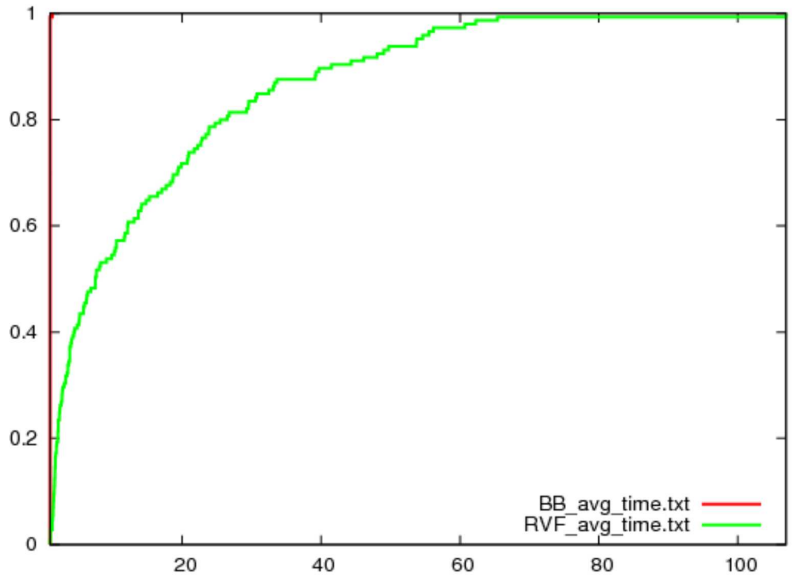


Figure 4.15: Performance profile comparing the average time (total time divided by total iterations) of the RVF subproblem and SYMPHONY within the BB Algorithm for all PILP instances.

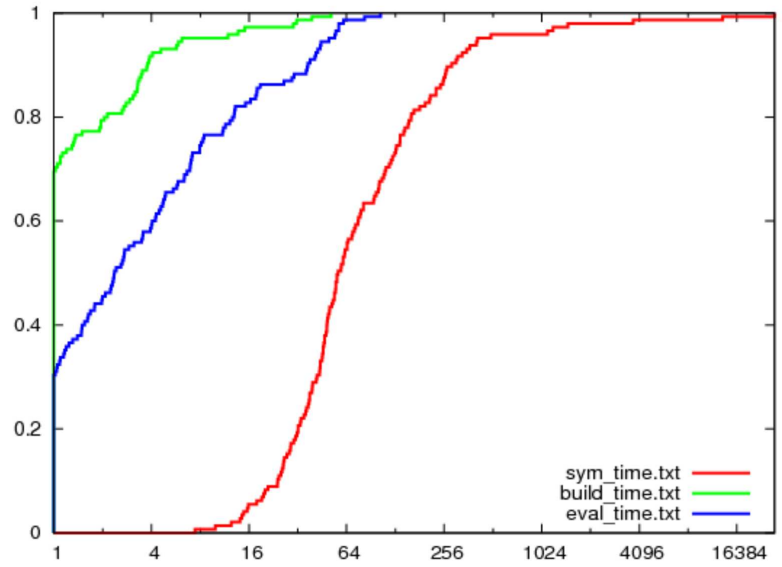


Figure 4.16: Performance profile comparing the total time of solving MILPs with SYMPHONY, the total time of building the dual function, and the total time of evaluating the dual function within the BB Algorithm for all PILP instances.

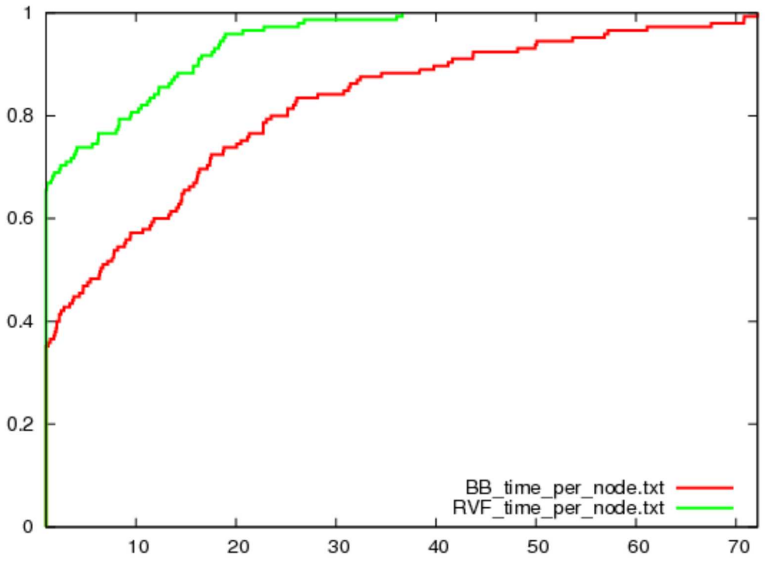


Figure 4.17: Performance profile comparing the average time (total time divided by total nodes) of the RVF subproblem and SYMPHONY within the BB Algorithm for all PILP instances.

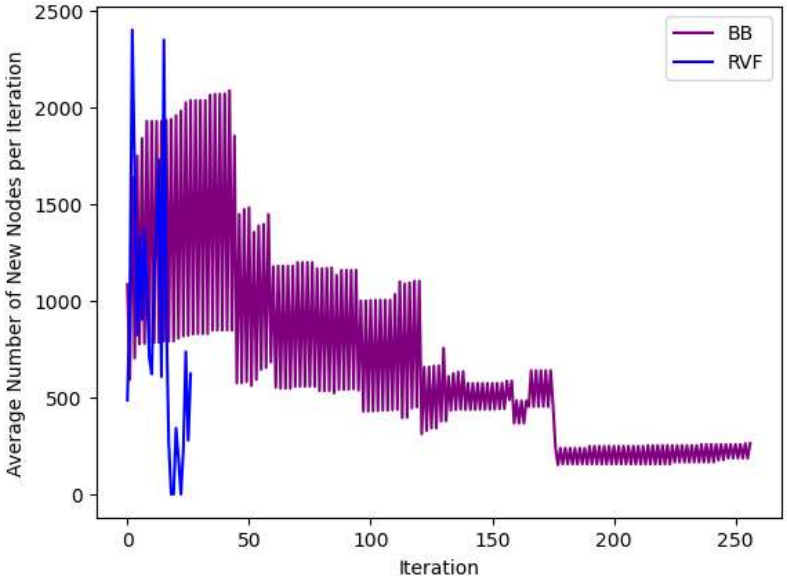


Figure 4.18: A graph illustrating the average number of new nodes generated by the RVF subproblem and SYMPHONY within the BB Algorithm for SPP.

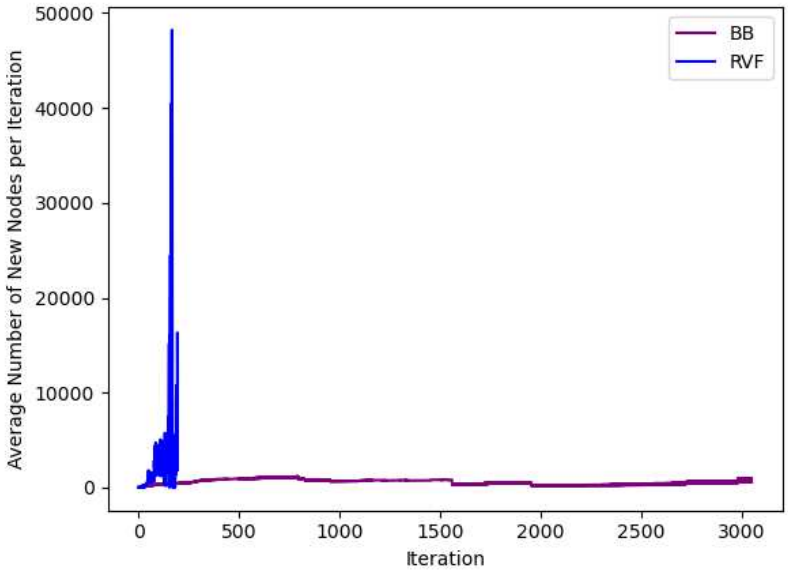


Figure 4.19: A graph illustrating the average number of new nodes generated by the RVF subproblem and SYMPHONY within the BB Algorithm for KP.

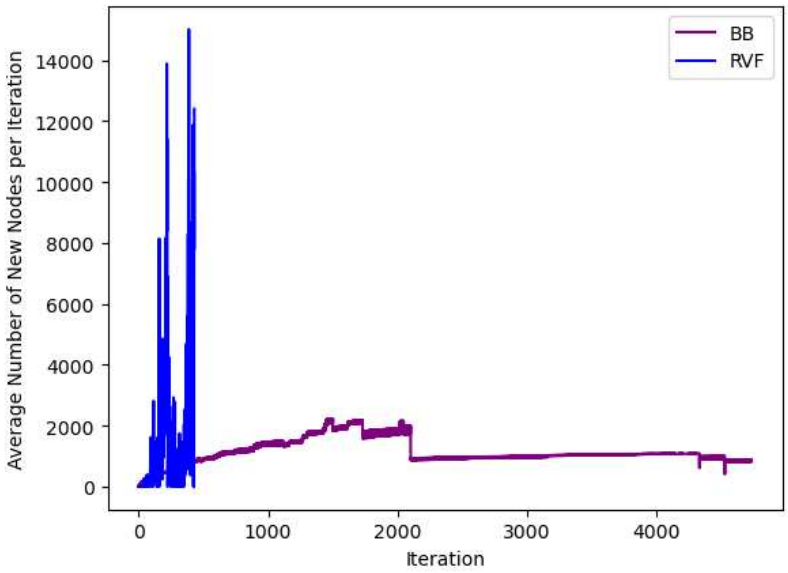


Figure 4.20: A graph illustrating the average number of new nodes generated by the RVF subproblem and SYMPHONY within the BB Algorithm for randomly generated PILP instances.

4.6.2.2 General (Mixed Integer) Case

Experimental Setup. In this study, we enforce a time constraint of 4 hours for the execution of each algorithm to generate the complete RVF or EF. If the time limit is exceeded, the algorithm is terminated.

In this study, we investigated a set of randomly generated MILP instances. We employed the proposed BB algorithms alongside the best-performing implementation of the RVF Algorithm from Chapter 3, namely Implementation 3. First, we compare the performance of the different versions of the BB algorithm across all MILP instances. Next, we compare the tuned version of the BB algorithm with the RVF Algorithm. Finally, we present several key insights derived from the best-performing version of the BB algorithm.

It is important to note that within the design of the BB algorithm, the RVF procedure is invoked at certain points to determine whether all necessary stability regions have been identified. The RVF subproblem used within the BB framework corresponds to Implementation 3, which either indicates that optimality has been reached—signaling that the algorithm should terminate—or identifies a new NDP corresponding to a new stability region that can be added to the set of efficient solutions found thus far. It is also worth noting that in Implementation 3 of the RVF subproblem, a callback mechanism with lazy constraints was incorporated. Further details regarding this implementation and its computational results can be found in Subsection 3.5.2.

Below, we discuss the results for the sample-based version and compare it with the RVF Algorithm. A pilot study was conducted for the batch-based algorithm, which showed that the sample-based version outperformed the batch-based version. Therefore, we decided to present the results for the sample-based version here.

The sample-based algorithm includes two key parameters:

1. **Initial Sample Size**: This determines the initial ζ list, required for the algorithm's first iteration, specifically the steps outlined in the `for` loop in pseudocode BB (Sample-based) - P1.
2. **RVF Call Frequency**: This parameter is used in subsequent iterations of the algorithm, specifically in pseudocode BB (Sample-based) - P2, where additional samples are generated

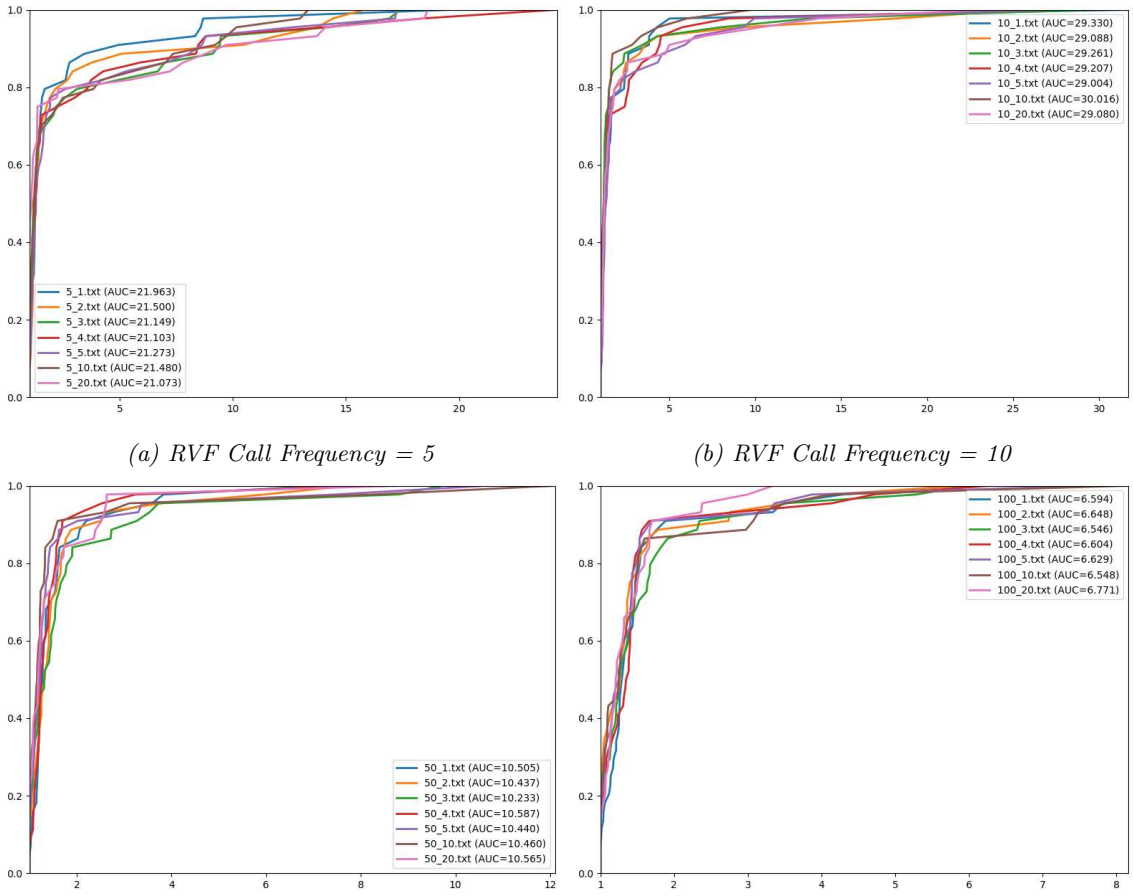
because the algorithm does not terminate in the first part. The *RVFCallFrequency* is a parameter that determines how often the RVF is called, specifying the number of iterations between each call.

Figure 4.21 presents the performance profiles comparing the execution times of the BB/sample-based algorithm under different RVF call frequencies and initial sample sizes across all MILP instances. The labels follow the format x_y, where x represents the RVF call frequency and y denotes the initial sample size. In terms of the AUC, the different versions exhibit similar performance; therefore, we focus on the average execution time for comparison. The corresponding average execution times are reported in Table 4.1 below for reference. Among all parameter combinations, the implementation with an RVF call frequency of 50 and an initial sample size of 20 demonstrates the best performance. Consequently, this configuration is selected as the representative version of the BB approach. In the following section, we compare this best-performing version with the pure RVF Algorithm. All detailed results, including execution time and the number of stability regions for call frequencies of 5, 10, 50, and 100, are presented in Tables M.9, M.10, M.11, and M.12.

Table 4.1: Average execution time (in seconds) for different RVF call frequencies and initial sample sizes

RVF Call Frequency	Initial Sample Size						
	1	2	3	4	5	10	20
5	611.68	589.20	**563.80**	611.68	620.04	619.70	574.00
10	575.59	589.20	598.13	574.99	575.32	**522.65**	607.15
50	558.79	542.25	568.75	498.40	534.47	502.15	**492.29**
100	599.25	563.35	610.56	**559.62**	569.84	625.49	560.62

The objective of this research was to develop a unified BB algorithm for constructing RVF for MILPs and to compare its performance with the RVF Algorithm, as discussed in detail in Chapters 2 and 3. We hypothesized that the sample-based version would outperform the RVF Algorithm in terms of runtime. The rationale behind this hypothesis lies in the structural differences between the two algorithms. The RVF Algorithm involves iterative nonlinear solves, whereas the sample-based version divides the process into two stages: (1) solving linear problems using `SYMPHONY` to determine the integer parts, and (2) calling the RVF subproblem periodically after a specified number of iterations, determined by the `RVFCallFrequency` parameter. Since the sample-based algorithm relies on `SYMPHONY` for identifying integer parts and reduces dependence on RVF calls, it was expected to



(a) RVF Call Frequency = 5

(b) RVF Call Frequency = 10

(c) RVF Call Frequency = 50

(d) RVF Call Frequency = 100

Figure 4.21: Performance profiles comparing the execution times of the BB/sample-based algorithm under different RVF call frequencies and initial sample sizes across all MILP instances.

achieve shorter runtimes.

Figure 4.22 presents the performance profiles comparing the execution times of the RVF and BB algorithms across all MILP instances. The RVF corresponds to the best-performing implementation identified in Chapter 3, namely Implementation 3. The BB algorithm refers to the tuned sample-based version that employs different RVF call frequencies and initial sample sizes, with the best-performing configuration identified as 50_20—corresponding to an RVF call frequency of 50 and an initial sample size of 20. In terms of the AUC, both algorithms exhibit comparable performance; therefore, we focus on their average execution times for further analysis. The average execution time for RVF is 508 seconds, whereas the corresponding time for BB is 489 seconds. This represents a 3.7% reduction in running time, calculated as (average time of RVF − average time of BB)/average time of RVF. Consequently, the BB algorithm outperforms RVF in terms of average execution time. This improvement is consistent with the design objective of BB, which aims to reduce the computational burden of the subproblems solved at each iteration while efficiently identifying new stability regions or NDPs. For completeness, we also report the corresponding baseline performance profile comparing RVF and BB for the MILP instances in Figure N.2 in Appendix N, which supports the conclusions drawn from the standard performance profile.

Detailed results of the RVF Algorithm referenced here, corresponding to Implementation 3, are provided in Table K.16.

Detailed Analysis: This section provides a comprehensive analysis of the most effective version of the sample-based approach, specifically 50_20. Figures 4.23-4.29 offer deeper insights into the winning configuration of the sample-based algorithm. Note that, similar to the pure integer case, in these figures, RVF refers to the RVF call within the BB algorithm, while BB refers to the SYMPHONY calls in the BB/sample-based Algorithm.

Figures 4.23, 4.24, and 4.25 present performance profiles comparing the RVF subproblem and SYMPHONY within the BB algorithm for all MILPs in terms of the number of iterations, total time, and average time, respectively. These figures confirm the same insight as the comparison of the overall BB/sample-based Algorithm with the RVF Algorithm: the BB algorithm outperforms the RVF Algorithm in terms of time. Additionally, the comparison between SYMPHONY and the RVF subproblem in the sample-based version further confirms that SYMPHONY achieves better

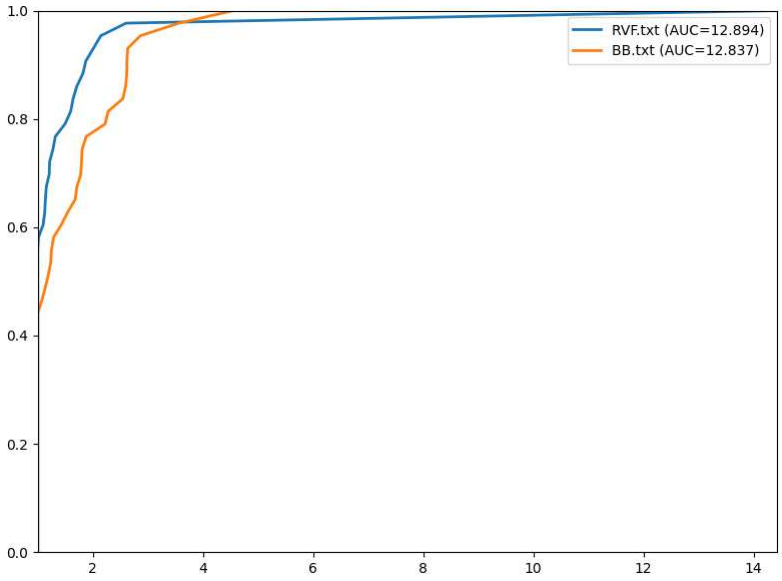


Figure 4.22: Performance profile comparing the execution times of RVF and BB for all MILPs.

performance in terms of both total time and average time (total time per iteration) compared to the RVF subproblem. The BB average time (time per iteration) is 0.06 seconds, while the RVF subproblem average time is 31.54 seconds. This is an expected behavior and one of the main motivations for this chapter, since in RVF, we solve a nonlinear problem, whereas in BB with SYMPHONY, a linear problem is solved.

Figure 4.26 presents the performance profile comparing the total time for solving MILPs with SYMPHONY (`sym_time`), the total time for building the dual function (`build_time`), and the total time for evaluating the dual function (`eval_time`) within the BB algorithm. As shown in the figure, `eval_time` requires more time than `sym_time`, and `sym_time` in turn requires more time than `build_time`. Reducing each component of the total time, including `sym_time`, `build_time`, and `eval_time`, can decrease the overall computational time of the BB algorithm. However, since `eval_time` and `sym_time` constitute the largest portions of the total time, improving these two components can significantly reduce the overall running time of the algorithm, although the other component also remains important to consider. Specifically, the average `eval_time`, `sym_time`, and `build_time` are 7.54 seconds, 2.78 seconds, and 0.05 seconds, respectively.

Figure 4.27 presents the performance profile comparing the average time per node of the RVF subproblem and SYMPHONY within the BB algorithm for all MILP instances. The figure shows

that BB exhibits better performance than RVF in terms of both the average time per node and the AUC; a larger AUC for BB indicates superior runtime performance. Specifically, the average time per node in RVF is 0.007 seconds, while in BB it is 0.004 seconds, implying that, on average, BB's time per node is approximately 42% lower than that of RVF (computed as (RVF − BB)/RVF). In the pure integer case, it was observed that RVF's time per node was lower than that of BB, which is reasonable since, for the PILP, the model is fully linearized, and the comparison involves two linear solvers: a commercial solver (the linear version of Gurobi) and an open-source solver (CLP, used in the SYMPHONY framework). In contrast, for MILPs, the comparison is between a nonlinear solver (the nonlinear version of Gurobi) and a linear solver (CLP). Therefore, it is expected that the time per node would be lower for SYMPHONY, which solves linear problems, than for RVF, which requires solving nonlinear subproblems.

Figures 4.28 and 4.29 illustrate the average number of new nodes generated by the RVF subproblem and SYMPHONY within the BB algorithm for the MILP instances. The first figure presents the results on a normal scale, while the second uses a logarithmic scale for better visualization. The figures clearly show that RVF generates more new nodes compared to BB. Each RVF iteration starts from scratch, whereas SYMPHONY employs a warm-start using the solution information from the previous iteration. Additionally, the number of new nodes in each SYMPHONY iteration remains relatively consistent, indicating that the problem's complexity does not increase exponentially. In contrast, since RVF solves the problem from scratch in each iteration, the number of new nodes can vary significantly, and as more integer parts are identified, the problem complexity increases substantially with each iteration.

All the data used in the above analysis for a deeper examination of MILP instances are outlined in Table M.13.

We also conducted another analysis, which provided us with additional insights. We observed that the BB Algorithm outperformed the RVF Algorithm, so we performed an analysis to understand what changes could make RVF comparable to BB. To do this, we considered the total time of RVF, the time of the last iteration of RVF within BB, the average time per iteration of SYMPHONY, and the number of stability regions for each MILP instance. For each instance, we calculated $(\mathrm{TT_{RVF}} - \mathrm{T_{last_iter_RVF}})/(\mathrm{AT_{sym}} \times \mathrm{num_stab})$, where $\mathrm{TT_{RVF}}$ is the total time of the RVF Algorithm, $\mathrm{T_{last_iter_RVF}}$ refers to the time of the last iteration of the RVF within the BB Algorithm, $\mathrm{AT_{sym}}$ is

the average time per iteration of SYMPHONY, and num_stab refers to the corresponding number of stability regions.

The reason we performed this calculation is that, in an ideal BB Algorithm, only one iteration of the RVF call would be necessary to find all efficient integer solutions or stability regions. Therefore, we take the difference between the RVF Algorithm's total time and the last iteration of RVF within the BB Algorithm to determine the ideal time of our BB Algorithm. We then divide this value by the average time per iteration of SYMPHONY and the number of stability regions. If the final result is x, it means that for the corresponding instance, every x sample should yield a new NDP or efficient integer solution corresponding to a new stability region.

For MILP, if we average this final result across all instances, we obtain a value of 170. This means that for MILP instances, if we have a highly efficient BB Algorithm that finds a new stability region every 170 samples or iterations, the BB and RVF Algorithms would be comparable. In our computational experiments, we observed that the BB algorithm outperformed the RVF Algorithm, indicating that, on average, BB finds a new stability region in fewer than 170 iterations.

Here, we mention some points worth noticing. As outlined in the pseudocode for the sample-based version, the algorithm begins with an initial sample of ζ values and, after iterating through all of them, calls the RVF subproblem. If the RVF subproblem confirms all integer parts are found, the algorithm terminates. Otherwise, the second round begins, generating more samples and invoking the RVF subproblem every few iterations, based on the RVFCallFrequency. The trade-off lies in sample generation: while samples aim to identify new integer parts, in MILPs, finding a new integer part does not always guarantee a new stability region. Regardless, these integer parts are added to the list, potentially increasing the complexity of subsequent RVF calls. This additional complexity does not arise in the pure RVF Algorithm, where each iteration finds an integer part corresponding to a new stability region.

Overall, the computational experiments demonstrate that the proposed BB algorithms provide a competitive alternative to the RVF Algorithm, particularly in the MILP setting where nonlinear subproblems become computationally expensive. The results highlight the effectiveness of warm-starting and iterative refinement in reducing computational effort while maintaining exactness. These findings support the practical applicability of the proposed framework for constructing the RVF in large-scale problems.

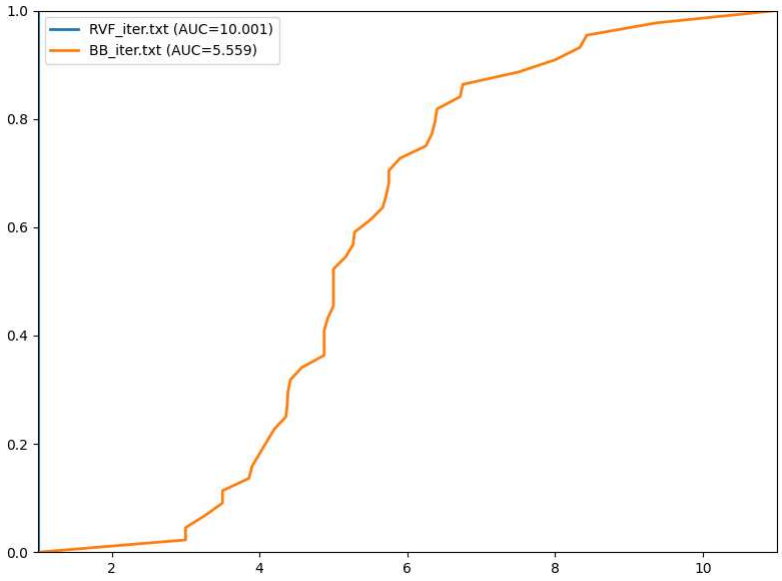


Figure 4.23: Performance profile comparing the number of iterations of the RVF subproblem and SYMPHONY within the BB Algorithm for all MILPs.

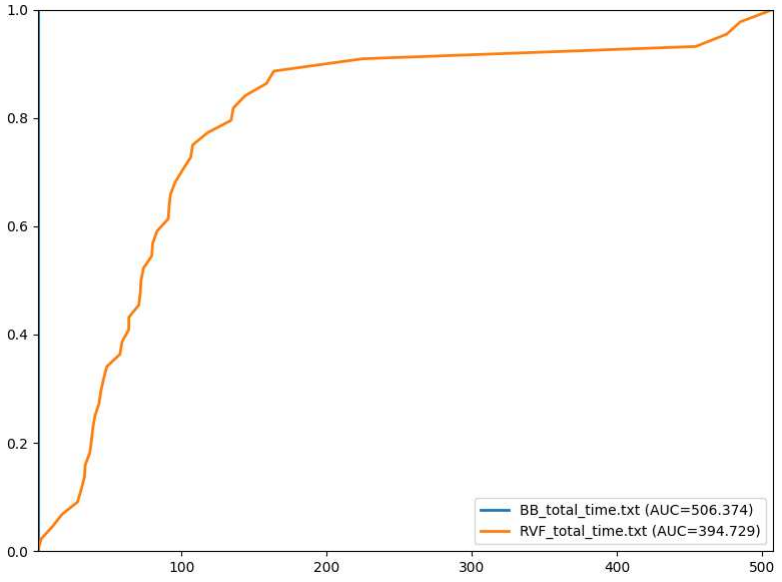


Figure 4.24: Performance profile comparing the total time of the RVF subproblem and SYMPHONY within the BB Algorithm for all MILPs.

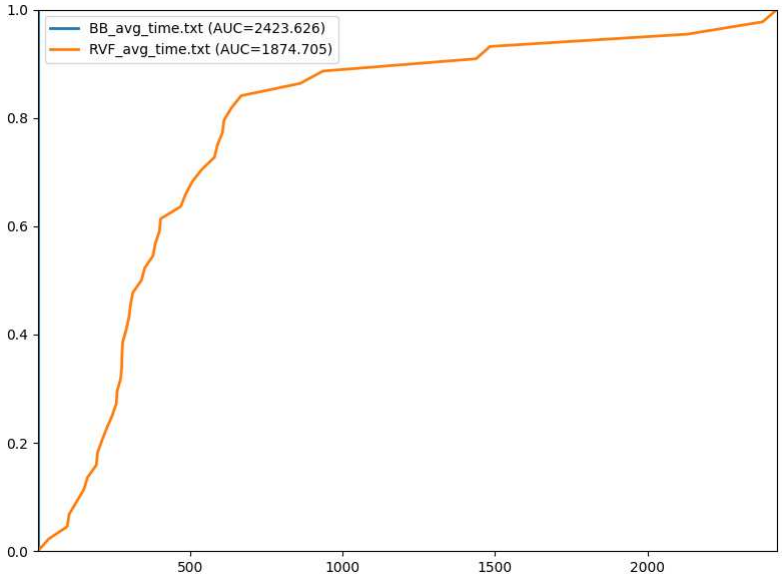


Figure 4.25: Performance profile comparing the average time (total time divided by total iterations) of the RVF subproblem and SYMPHONY within the BB Algorithm for all MILPs.

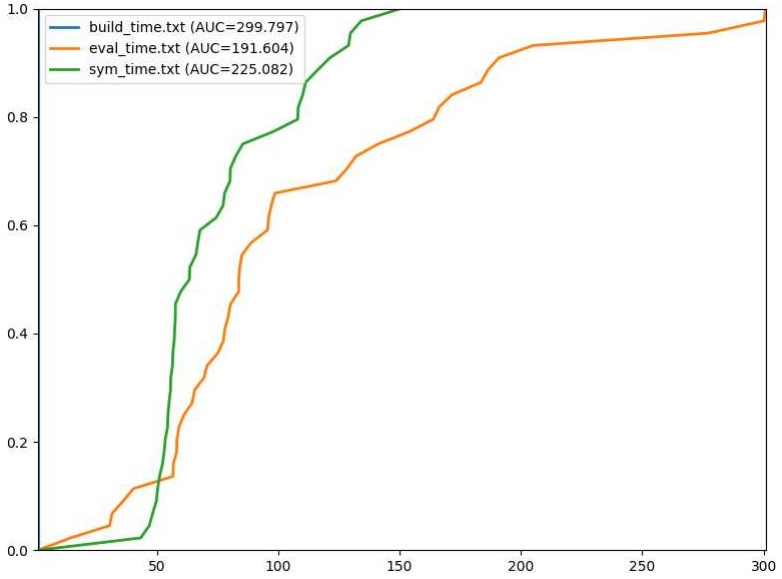


Figure 4.26: Performance profile comparing the total time of solving MILPs with SYMPHONY, the total time of building the dual function, and the total time of evaluating the dual function within the BB Algorithm for all MILPs.

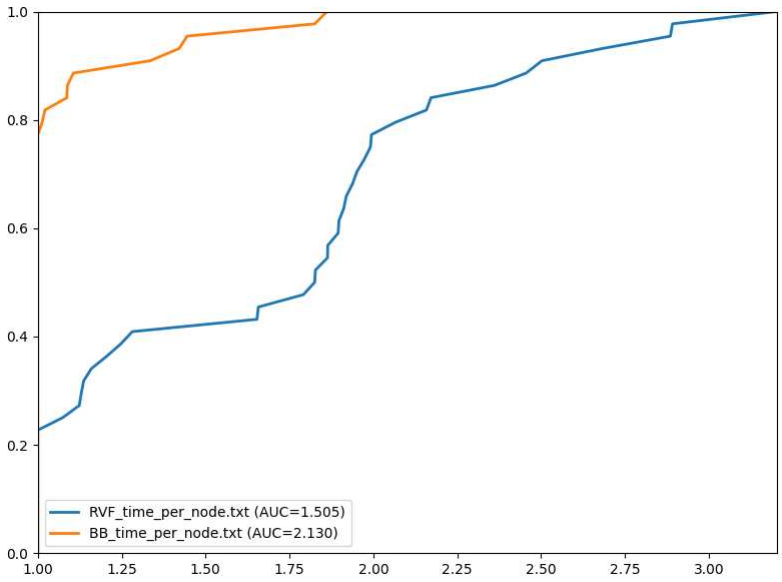


Figure 4.27: Performance profile comparing the average time (total time divided by total nodes) of the RVF subproblem and SYMPHONY within the BB Algorithm for all MILPs.

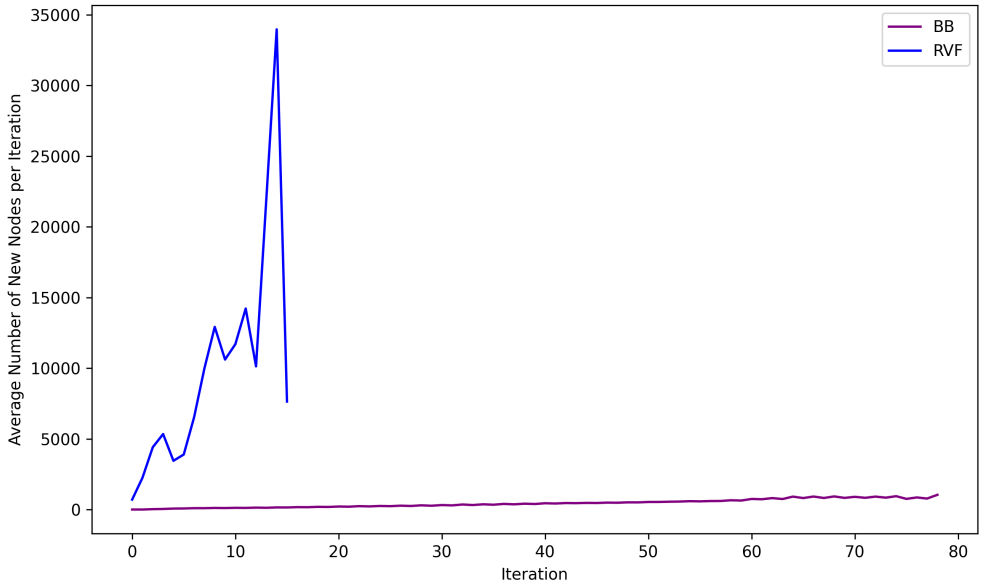


Figure 4.28: A graph illustrating the average number of new nodes generated by the RVF subproblem and SYMPHONY within the BB Algorithm for MILPs.

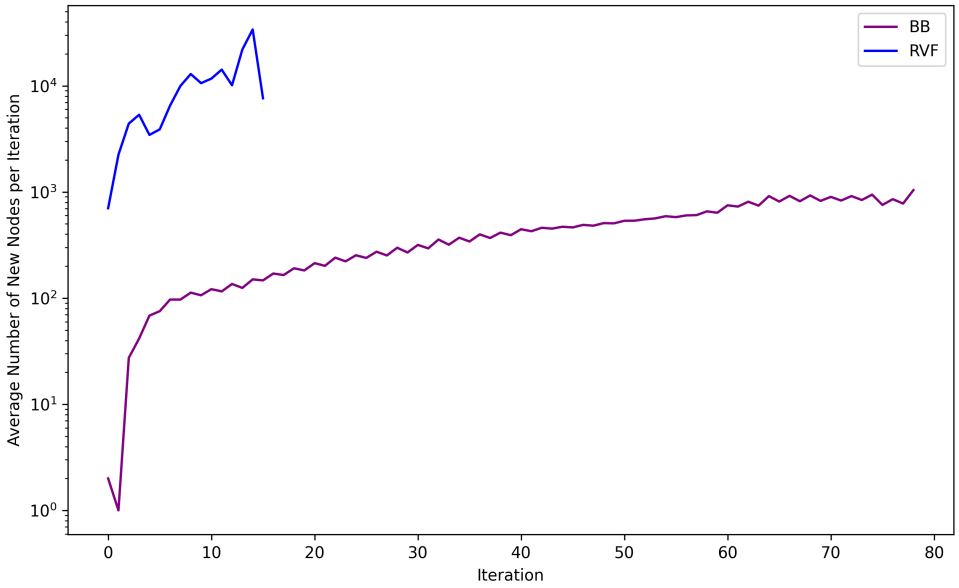


Figure 4.29: A graph illustrating the average number of new nodes generated by the RVF subproblem and SYMPHONY within the BB Algorithm for MILPs (log scale).

4.7 Software

We have developed a Python package that implements the proposed BB algorithms. The source code and computational instances used in this chapter are publicly available on GitHub [Fallah, 2025e], enabling full reproducibility of the reported results.

4.7.1 Dependencies required to run the program

The package supports the enumeration of nondominated points for multiobjective integer programs and the construction of efficient solutions for mixed-integer cases. The required dependencies include Python 3, Gurobi, SYMPHONY, and the CLP solver.

In this section, we describe how to install SYMPHONY and integrate it as a library within a Python package. To use SYMPHONY as an external library in Python, the following steps should be followed:

1. Clone the "rvf" branch of SYMPHONY from the provided link using the command:

   git clone -b rvf `https://github.com/febattista/SYMPHONY.git`

2. Obtain "symphony.py" to create a SYMPHONY environment in Python by cloning the repository:

   git clone `https://github.com/febattista/pySYMPHONY.git`

3. Obtain "coinbrew", a tool for building Coin-OR software:

   wget `https://raw.githubusercontent.com/coin-or/coinbrew/master/coinbrew`

4. Make "coinbrew" executable:

   chmod u+x coinbrew

5. Build the SYMPHONY "rvf" branch using "coinbrew" with the following command:

   ./coinbrew build SYMPHONY -b build-SYMPHONY-rvf -p –enable-debug
   –enable-shared –enable-sensitivity-analysis -j 10 –tests none

   Note that -enable-debug is optional and intended for debugging purposes. Therefore, feel free to omit it if you do not require debugging.

6. Ensure that the "lib_path" variable in your Python code points to the "libSym.so" file of the SYMPHONY build you just created:

   lib_path = <path-to-build-SYMPHONY-rvf>/lib/libSym.so

7. Include the following lines in your Python code to use SYMPHONY as an external library:

   - from symphony import *
   - Symphony.dlopen(lib_path)

8. Create a SYMPHONY environment by calling "Symphony()" and then set additional parameters as needed:

- `sym = Symphony()`
- `sym.set_param`("verbosity", -2) # Set the verbosity level to desired value
- `sym.read_mps`(model_path) or `sym.read_lp`(model_path) # Load the instance (model) from an MPS file or LP file
- `sym.enable_warm_start`() # Enable warm-start if required

9. Ensure that the instances (models) are in the format of MPS files or LP files, and specify the path to the instance using "model_path":

   model_path = <path-to-instance>

   Additionally, note that the data can also be loaded directly using sym_load_problem().

After completing these steps, you should be able to run or debug your program using SYMPHONY as an external library within your Python environment.

Here are the important functionalities of SYMPHONY used in the BB Algorithm:

- `sym.solve()`: This function is used to solve the problem.

  If `sym.solve()` == FUNCTION_TERMINATED_ABNORMALLY, it indicates that an issue occurred, and the program should be exited. Otherwise, the optimal solution can be retrieved using `sym.get_col_solution()`.

- To obtain the dual function inherent to SYMPHONY, the following steps are necessary:
  - Build the dual function using `sym.build_dual_function()`.
  - Evaluate the dual function for a specific RHS using `sym.evaluate_dual_function(RHS_list)`, where RHS_list is a list containing the RHS values of all constraints, including parametric and fixed RHS constraints.

- After the first iteration, to warm-start the program, use `sym.warm_solve()` instead of `sym.solve()`. Therefore, for the initial iteration, the problem is solved using `sym.solve()`, while for subsequent iterations, `sym.warm_solve()` is used to exploit previously constructed BB information.

- To modify the RHS values and warm-start the program, use `sym.set_row_upper(i, j)`, where i represents the index of the constraint, and j denotes the new value for the RHS.

This summary outlines the key functionalities of SYMPHONY within the context of the BB Algorithm, ensuring clarity and correctness in usage.

Certain SYMPHONY parameters must be configured to ensure correct warm-start behavior. These include enabling warm-start retention, dual function tracking, and sensitivity analysis, while disabling preprocessing and cut generation that may invalidate reuse across different right-hand sides. The required parameter settings are summarized in Table 4.2 and are already configured in the provided implementation.

Table 4.2: Parameters and their corresponding values that need to be fixed within the warm solve functionality in SYMPHONY

Parameter	Value
keep_warm_start	TRUE
keep_dual_function_description	TRUE
should_warmstart_node	TRUE
sensitivity_analysis	TRUE
sensitivity_bounds	TRUE
sensitivity_rhs	TRUE
do_primal_heuristic	FALSE
prep_level	-1
generate_cgl_cuts	FALSE

4.7.2 Data Files

We describe how to provide input instances to the package. We assume that the BB Algorithm solves the multiobjective problem formulated as (MO-MILP). It is presumed that each objective function should be minimized. For maximization problems, all coefficients of the objective functions need to be multiplied by -1.

Input preparation is as follows: "OBJ" represents a list of coefficients for the first objective (the one in the objective function, i.e., c^0 in (MILP)). "Mat" is a dictionary for the coefficients of the objectives considered in the constraint set (i.e., $C^{1:l}$). "MatFixed" is a dictionary for the coefficients of the left-hand side of the constraints with fixed right-hand sides (i.e., A). "RHS" is a dictionary containing the right-hand side values of the constraints, i.e., b in the equation $Ax = b$.

For mixed integer cases, if your problem structure includes $A_I x_I + A_C x_C \leq b$, it should be converted into equality constraints by introducing slack variables. Similarly, for the pure integer case, if you have $A_I x_I \leq b$, slack should be added after ensuring that the coefficients of matrix A and the RHS vector b are integers, as the slack variable in the pure integer case is considered an integer in the code.

4.7.3 Running and Results

To execute the project, navigate to the directory where it is stored and run the Python files by entering the command "python3 BBRVF_<version>.py" in the terminal or command prompt. Upon completion, the algorithm outputs the set of nondominated points along with the total running time. A time limit can be specified via the `timeLimit` parameter. If the time limit is reached, the algorithm returns the best available approximation of the EF.

4.8 Conclusions

This chapter presents a unified BB framework for constructing the RVF and the EF of a multiobjective mixed integer linear program. The proposed approach differs from the RVF Algorithm of Fallah et al. [2024] in that it maintains both an upper approximation and a lower approximation of the RVF, with the lower approximation induced by the BB tree. In contrast, the RVF Algorithm refines its approximation by comparing the current upper approximation with the true value function, which requires solving a nonlinear subproblem. By working instead with the gap between upper and lower approximations, the proposed framework enables tree-based warm-starting and can improve computational efficiency.

The framework is built on three main components. First, in Section 4.2, we established that any BB tree induces a valid lower-bounding function and that there exists a tree whose associated lower-bounding function coincides with the RVF. Second, in Section 4.3, we introduced a tree-based warm-starting strategy that allows the tree to be iteratively refined across different right-hand sides. Third, in Section 4.4, we developed algorithmic strategies for selecting right-hand sides based on the gap between upper and lower approximations. We further showed that, under mild assumptions, the proposed framework converges to the exact RVF in a finite number of iterations.

For a fixed ζ in each iteration, the upper approximation is constructed from the feasible integer solutions identified so far, given by the minimum over the corresponding continuous restrictions. The lower approximation is obtained from dual solutions associated with the leaf nodes of the current BB tree. The gap between these approximations determines whether further refinement is needed. When a non-zero gap is identified, the corresponding right-hand side is selected, and the tree is warm-started to incorporate additional information. This process continues until the gap is eliminated.

The implementation of the proposed framework within SYMPHONY is discussed in Section 4.5, where we describe how warm-starting is leveraged in practice. The computational experiments presented in Section 4.6 demonstrate that the proposed approach provides a competitive alternative to the RVF Algorithm.

For PILP instances, the RVF Algorithm performs better, largely due to the use of Gurobi and the linear structure of the subproblems. In contrast, for MILP instances, the proposed BB framework outperforms the RVF Algorithm, as it avoids solving nonlinear subproblems and benefits from warm-starting within SYMPHONY.

The results highlight the effectiveness of combining lower and upper approximations with iterative refinement. In particular, the ability to reuse information across iterations through warm-starting plays a key role in reducing computational effort while maintaining exactness.

Overall, this work provides a structured and theoretically grounded framework for constructing the RVF and EF, offering both practical computational benefits and a foundation for future developments in this area.

Chapter 5

Future Research Directions

This dissertation established a unified framework for understanding and constructing the restricted value function (RVF) and the efficient frontier (EF) in multiobjective mixed integer optimization.

These contributions suggest several directions for future research.

Adaptive sampling and RHS selection. A central component of the proposed framework is the selection of right-hand sides at which the RVF is evaluated. A promising direction for future work is to develop more effective and adaptive strategies for selecting these points during the iterative construction process. In particular, such strategies could leverage information from both primal and dual approximations to identify the most informative regions of the domain, where the approximation gap is largest. This could reduce the number of required evaluations and iterations, thereby improving overall computational efficiency.

Dynamic integration of primal and dual frameworks. This dissertation highlights the complementary roles of primal (upper) and dual (lower) approximations within the proposed framework. While the current branch-and-bound approach incorporates primal information during the iterative process, a promising direction for future work is the development of more principled strategies for integrating these two components. In particular, one could design adaptive schemes that determine when additional primal refinement is necessary based on the observed gap between upper and lower approximations. Such strategies could reduce the frequency of these refinements, ideally

reserving them for targeted improvement or final certification, while preserving the convergence properties of the branch-and-bound framework and further improving computational efficiency.

Refinement strategies for branch-and-bound trees. The theoretical results in Chapter 4 show that refining a branch-and-bound tree strengthens the associated dual bounding function. However, the current framework does not fully exploit this property algorithmically. Future work could focus on designing principled refinement strategies that prioritize nodes or regions of the tree that contribute most to improving the global approximation. This includes developing criteria for node selection, branching strategies, and methods for pruning or compressing trees without sacrificing approximation quality.

Scalability and high-dimensional objective spaces. While the proposed methods handle multiple objectives, the computational complexity grows with the dimension of the criterion space. Future work could investigate techniques for handling higher-dimensional objective spaces more efficiently.

Solver integration. The proposed framework relies on advanced features of modern optimization solvers, such as callbacks and warm-start capabilities. A natural direction for future research is to integrate the methodology more deeply into solver architectures. Embedding the RVF construction framework directly within branch-and-bound solvers could enable more efficient reuse of information across solves, improve scalability, and reduce overhead associated with repeated solver calls. This direction also includes exploring parallelization strategies and distributed implementations for large-scale instances.

Overall, this dissertation provides both a theoretical foundation and practical algorithms for constructing the RVF and EF in multiobjective optimization. The proposed framework opens multiple avenues for future research, combining ideas from value functions, branch-and-bound, and multiobjective optimization, and has the potential to influence both theory and practice in this area.

Bibliography

Shabbir Ahmed, Mohit Tawarmalani, and Nikolaos V Sahinidis. A finite branch-and-bound algorithm for two-stage stochastic integer programs. *Mathematical Programming*, 100(2):355–377, 2004.

Bernd Bank, Jürgen Guddat, Diethard Klatte, Bernd Kummer, and Klaus Tammer. *Non-linear parametric optimization*. Springer, 1982.

Pietro Belotti. Couenne: a user's manual. Technical report, Technical report, Lehigh University, 2009.

Pietro Belotti, Banu Soylu, and Margaret M Wiecek. A branch-and-bound algorithm for biobjective mixed-integer programs. *Optimization Online*, 2013.

Harold P Benson. Existence of efficient solutions for vector maximization problems. *Journal of Optimization Theory and Applications*, 26(4):569–580, 1978.

Harold P Benson. An outer approximation algorithm for generating all efficient extreme points in the outcome set of a multiple objective linear programming problem. *Journal of Global Optimization*, 13(1):1–24, 1998.

Ksenia Bestuzheva, Mathieu Besançon, Wei-Kun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros Gleixner, et al. The scip optimization suite 8.0. *arXiv preprint arXiv:2112.08872*, 2021.

Charles Blair. A closed-form representation of mixed-integer program value functions. *Mathematical Programming*, 71(2):127–136, 1995.

Charles E Blair and Robert G Jeroslow. The value function of a mixed integer program: I. *Discrete Mathematics*, 19(2):121–138, 1977.

Charles E Blair and Robert G Jeroslow. The value function of an integer program. *Mathematical Programming*, 23(1):237–273, 1982.

Charles E Blair and Robert G Jeroslow. Constructive characterizations of the value-function of a mixed-integer program i. *Discrete Applied Mathematics*, 9(3):217–233, 1984.

Charles Eugene Blair and Robert G Jeroslow. The value function of a mixed integer program: Ii. *Discrete Mathematics*, 25(1):7–19, 1979.

Merve Bodur, Shabbir Ahmed, Natashia Boland, and George L Nemhauser. Decomposition of loosely coupled integer programs: a multiobjective perspective. *Mathematical Programming*, 196 (1):427–477, 2022.

Natashia Boland, Hadi Charkhgard, and Martin Savelsbergh. A criterion space search algorithm for biobjective integer programming: The balanced box method. *INFORMS Journal on Computing*, 27(4):735–754, 2015a.

Natashia Boland, Hadi Charkhgard, and Martin Savelsbergh. A criterion space search algorithm for biobjective mixed integer programming: The triangle splitting method. *INFORMS Journal on Computing*, 27(4):597–618, 2015b.

Natashia Boland, Hadi Charkhgard, and Martin Savelsbergh. A new method for optimizing a linear function over the efficient set of a multiobjective integer program. *European journal of operational research*, 260(3):904–919, 2017.

V Joseph Bowman. On the relationship of the tchebycheff norm and the efficient frontier of multiple-criteria objectives. In *Multiple criteria decision making*, pages 76–86. Springer, 1976.

Seth Brown, Wenxin Zhang, Temitayo Ajayi, and Andrew J Schaefer. A gilmore-gomory construction of integer programming value functions. *Operations Research Letters*, 2021.

LG Chalmet, L Lemonidis, and DJ Elzinga. An algorithm for the bi-criterion integer programming problem. *European Journal of Operational Research*, 25(2):292–300, 1986.

Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. *Integer programming*, volume 271. Springer, 2014.

Pasqualina Conti and Carlo Traverso. Buchberger algorithm and integer programming. In *International Symposium on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, pages 130–139. Springer, 1991.

László Csirmaz. Using multiobjective optimization to map the entropy region. *Computational Optimization and Applications*, 63(1):45–67, 2016.

Kerstin Dächert and Kathrin Klamroth. A linear bound on the number of scalarizations needed to solve discrete tricriteria optimization problems. *Journal of Global Optimization*, 61(4):643–676, 2015.

George Dantzig. Linear programming and extensions. In *Linear programming and extensions*. Princeton university press, 2016.

George B Dantzig and Philip Wolfe. Decomposition principle for linear programs. *Operations research*, 8(1):101–111, 1960.

Marianna De Santis, Gabriele Eichfelder, Julia Niebling, and Stefan Rocktäschel. Solving multiobjective mixed integer convex optimization problems. *SIAM Journal on Optimization*, 30(4): 3122–3145, 2020.

Xavier Delorme, Xavier Gandibleux, and Fabien Degoutin. Evolutionary, constructive and hybrid procedures for the bi-objective set packing problem. *European Journal of Operational Research*, 204(2):206–217, 2010.

Miguel Ángel Domínguez-Ríos, Francisco Chicano, and Enrique Alba. Effective anytime algorithm for multiobjective combinatorial optimization problems. *Information Sciences*, 565:210–228, 2021.

Oscar Dowson, Xavier Gandibleux, and Gökhan Kof. Multiobjectivealgorithms. jl: a julia package for solving multi-objective optimization problems. *arXiv preprint arXiv:2507.05501*, 2025.

Alex Dunbar, Saumya Sinha, and Andrew J Schaefer. Relaxations and duality for multiobjective integer programming. *arXiv preprint arXiv:2309.08801*, 2023.

Matthias Ehrgott. *Multicriteria optimization*, volume 491. Springer Science & Business Media, 2005.

Matthias Ehrgott. A discussion of scalarization techniques for multiple objective integer programming. *Annals of Operations Research*, 147(1):343–360, 2006.

Matthias Ehrgott and Xavier Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR-spektrum*, 22(4):425–460, 2000.

Matthias Ehrgott and Margaret M. Wiecek. *Mutiobjective Programming*, pages 667–708. Springer New York, New York, NY, 2005.

Matthias Ehrgott, Xavier Gandibleux, and Anthony Przybylski. Exact methods for multi-objective combinatorial optimisation. In *Multiple criteria decision analysis*, pages 817–850. Springer, 2016.

Gabriele Eichfelder and Leo Warnow. On implementation details and numerical experiments for the hypad algorithm to solve multi-objective mixed-integer convex optimization problems. *Preprint*, pages 08–8538, 2021.

Gabriele Eichfelder and Leo Warnow. A hybrid patch decomposition approach to compute an enclosure for multi-objective mixed-integer convex optimization problems. *Mathematical Methods of Operations Research*, pages 1–30, 2023.

Gabriele Eichfelder, Peter Kirst, Laura Meng, and Oliver Stein. A general branch-and-bound framework for continuous global multiobjective optimization. *Journal of Global Optimization*, 80: 195–227, 2021.

Gabriele Eichfelder, Oliver Stein, and Leo Warnow. A solver for multiobjective mixed-integer convex and nonconvex optimization. *Journal of Optimization Theory and Applications*, pages 1–31, 2023.

Samira Fallah. RVF algorithms - IP versions. `https://github.com/samira7293/RVF_Implementations/tree/master/IP_codes`, 2025a.

Samira Fallah. Random IP instances. `https://github.com/samira7293/RVF_Implementations/tree/master/random_IP_instances`, 2025b.

Samira Fallah. RVF algorithms - MILP versions. `https://github.com/samira7293/RVF_Implementations/tree/master/MILP_codes`, 2025c.

Samira Fallah. Random MILP instances. `https://github.com/samira7293/RVF_Implementations/tree/master/random_MILP_instances`, 2025d.

Samira Fallah. BBRVF algorithms. `https://github.com/samira7293/BBRVF`, 2025e.

Samira Fallah, Ted K Ralphs, and Natashia L Boland. On the relationship between the value function and the efficient frontier of a mixed integer linear optimization problem. *Mathematical Methods of Operations Research*, 100(1):175–220, 2024.

Julius Farkas. Theorie der einfachen ungleichungen. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, 1902(124):1–27, 1902.

Nicolas Forget, Sune Lauth Gadegaard, Kathrin Klamroth, Lars Relund Nielsen, and Anthony Przybylski. Branch-and-bound and objective branching with three or more objectives. *Computers & Operations Research*, 148:106012, 2022a.

Nicolas Forget, Sune Lauth Gadegaard, and Lars Relund Nielsen. Warm-starting lower bound set computations for branch-and-bound algorithms for multi objective integer linear programs. *European Journal of Operational Research*, 2022b.

M.V. Galati. *Decomposition in Integer Programming*. PhD, Lehigh University, 2009. URL `http://coral.ie.lehigh.edu/~ted/files/papers/MatthewGalatiDissertation09.pdf`.

Xavier Gandibleux, Gauthier Soleilhac, Anthony Przybylski, and Stefan Ruzika. voptsolver: An open source software environment for multiobjective mathematical optimization. In *Proc. 21st Conf. Int. Fed. Oper. Res. Societies (IFORS)*, pages 17–21, 2017.

Juergen Guddat, F Guerra Vasquez, Klaus Tammer, and Klaus Wendler. Multiobjective and stochastic optimization based on parametric optimization. *Mathematical Research*, 26, 1985.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022. URL `https://www.gurobi.com`.

M. Güzelsoy. *Dual Methods in Mixed Integer Linear Programming*. PhD Dissertation, Lehigh University, 2009. URL `http://coral.ie.lehigh.edu/~ted/files/papers/MenalGuzelsoyDissertation09.pdf`.

M. Güzelsoy and T.K. Ralphs. Duality for Mixed-Integer Linear Programs. *International Journal of Operations Research*, 4:118–137, 2007a. URL `http://coral.ie.lehigh.edu/~ted/files/papers/MILPD06.pdf`.

MENAL Güzelsoy and T Ralphs. The value function of a mixed-integer linear program with a single constraint. *To be submitted*, 2006.

Menal Güzelsoy and Theodore K Ralphs. Duality for mixed-integer linear programs. *International Journal of Operations Research*, 4(3):118–137, 2007b.

Yacov Haimes. On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE transactions on systems, man, and cybernetics*, (3):296–297, 1971.

Pascal Halffmann, Luca E Schäfer, Kerstin Dächert, Kathrin Klamroth, and Stefan Ruzika. Exact algorithms for multiobjective linear optimization problems with integer variables: A state of the art survey. *Journal of Multi-Criteria Decision Analysis*, 2022.

Andreas H Hamel, Andreas Löhne, and Birgit Rudloff. Benson type algorithms for linear vector optimization and applications. *Journal of Global Optimization*, 59(4):811–836, 2014.

A. Hassanzadeh. *Two-Stage Stochastic Mixed Integer Optimization*. PhD, Lehigh, 2015.

Anahita Hassanzadeh and Ted K Ralphs. A generalization of benders' algorithm for twostage stochastic optimization problems with mixed integer recourse. In *Technical Report*. 2014.

John Hooker. *Logic-based methods for optimization: combining optimization and constraint satisfaction*. John Wiley & Sons, 2011.

John N Hooker. Logic-based methods for optimization. In *International Workshop on Principles and Practice of Constraint Programming*, pages 336–349. Springer, 1994.

Gokhan Kirlik and Serpil Sayın. A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems. *European Journal of Operational Research*, 232(3): 479–488, 2014.

Nan Kong, Andrew J Schaefer, and Brady Hunsaker. Two-stage integer programs with stochastic right-hand sides: a superadditive dual approach. *Mathematical Programming*, 108(2):275–296, 2006.

Jon Lee. All-different polytopes. *J Combin Optim*, 2002.

Moritz Link and Stefan Volkwein. Computing an enclosure for multiobjective mixed-integer nonconvex optimization problems using piecewise linear relaxations, 2022.

A Löhne and B Weißing. Bensolve-vlp solver, version 2.0. 1. *URL http://bensolve. org*, 2015.

Banu Lokman and Murat Köksalan. Finding all nondominated points of multi-objective integer programs. *Journal of Global Optimization*, 57(2):347–365, 2013.

George Mavrotas and Danae Diakoulaki. A branch and bound algorithm for mixed zero-one multiple objective linear programming. *European Journal of Operational Research*, 107(3):530–541, 1998.

George Mavrotas and Danae Diakoulaki. Multi-criteria branch and bound: A vector maximization algorithm for mixed 0-1 multiple objective linear programming. *Applied mathematics and computation*, 171(1):53–71, 2005.

George Nemhauser and Laurence Wolsey. The scope of integer and combinatorial optimization. *Integer and combinatorial optimization*, pages 1–26, 1988.

Melih Özlen and Meral Azizoğlu. Multi-objective integer programming: A general approach for generating all non-dominated solutions. *European Journal of Operational Research*, 199(1):25–35, 2009.

Adriano Pascoletti and Paolo Serafini. Scalarizing vector optimization problems. *Journal of Optimization Theory and Applications*, 42(4):499–524, 1984.

Tyler Perini, Natashia Boland, Diego Pecin, and Martin Savelsbergh. A criterion space method for biobjective mixed integer programming: The boxed line method. *INFORMS Journal on Computing*, 32(1):16–39, 2020.

William Pettersson and Melih Ozlen. Multi-objective mixed integer programming: An objective space algorithm. In *AIP Conference Proceedings*, volume 2070, page 020039. AIP Publishing LLC, 2019.

Anthony Przybylski and Xavier Gandibleux. Multi-objective branch and bound. *European Journal of Operational Research*, 260(3):856–872, 2017.

Ted K Ralphs and Menal Güzelsoy. The symphony callable library for mixed integer programming. In *The next wave in computing, optimization, and decision technologies*, pages 61–76. Springer, 2005.

Ted K Ralphs and Anahita Hassanzadeh. On the value function of a mixed integer linear optimization problem and an algorithm for its construction. *COR@ L Technical Report 14T–004*, 2014.

Ted K Ralphs, Matthew J Saltzman, and Margaret M Wiecek. An improved algorithm for solving biobjective integer programs. *Annals of Operations Research*, 147(1):43–70, 2006.

Seyyed Amir Babak Rasmi and Metin Türkay. Gondef: an exact method to generate all non-dominated points of multi-objective mixed-integer linear programs. *Optimization and Engineering*, 20(1):89–117, 2019.

R Tyrrell Rockafellar. *Convex analysis*, volume 11. Princeton university press, 1997.

R Tyrrell Rockafellar and Roger J-B Wets. *Variational analysis*, volume 317. Springer Science & Business Media, 2009.

Stefan Ruzika and Margaret M Wiecek. Approximation methods in multiobjective programming. *Journal of optimization theory and applications*, 126(3):473–501, 2005.

Serpil Sayın. Measuring the quality of discrete representations of efficient sets in multiple objective mathematical programming. *Mathematical Programming*, 87(3):543–560, 2000.

Rüdiger Schultz, Leen Stougie, and Maarten H Van Der Vlerk. Solving stochastic programs with integer recourse by enumeration: A framework using gröbner basis. *Mathematical Programming*, 83(1):229–252, 1998.

John Sylva and Alejandro Crema. A method for finding the set of non-dominated vectors for multiple objective integer linear programs. *European Journal of Operational Research*, 158(1):46–55, 2004.

S. Tahernejad. *Two-stage Mixed Integer Stochastic Bilevel Linear Optimization*. PhD, Lehigh University, 2019.

Satya Tamby and Daniel Vanderpooten. Enumeration of the nondominated set of multiobjective discrete optimization problems. *INFORMS Journal on Computing*, 33(1):72–85, 2021.

Ralphs Ted, Gambini Santos Haroldo, Saltzman Matthew, Kristjansson Bjarni, Vigerske Stefan, Brito Samuel, King Alan, Forrest John, Hafer Lou, et al. coin-or/clp: Release releases/1.17. 7. 2022.

Andrew C Trapp and Oleg A Prokopyev. A note on constraint aggregation and value functions for two-stage stochastic integer programs. *Discrete Optimization*, 15:37–45, 2015.

Andrew C Trapp, Oleg A Prokopyev, and Andrew J Schaefer. On a level-set characterization of the value function of an integer program and its application to stochastic programming. *Operations Research*, 61(2):498–511, 2013.

Laurence A Wolsey. Integer programming duality: Price functions and sensitivity analysis. *Mathematical Programming*, 20(1):173–195, 1981.

Po-Lung Yu. A class of solutions for group decision problems. *Management science*, 19(8):936–946, 1973.

Lofti Zadeh. Optimality and non-scalar-valued performance criteria. *IEEE transactions on Automatic Control*, 8(1):59–60, 1963.

Milan Zeleny. Compromise programming. *Multiple criteria decision making*, 1973.

Junlong Zhang and Osman Y Özaltın. Bilevel integer programs with stochastic right-hand sides. *INFORMS Journal on Computing*, 2021.

Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M Fonseca, and Viviane Grunert Da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on evolutionary computation*, 7(2):117–132, 2003.

Appendices

Appendix A

An Instance of RLPVF

Example 8. *In Figure A.1, we plot an instance of* (RLPVF) *obtained from Example 1 by setting* $(x_1, x_2) = (0, 0)$. *In the plot, the affine functions associated with each extreme point of* $\mathcal{P}_D$ *are shown for a part of the finite domain. For this example, the dual problem* (D-RLPVF) *can be expressed as*

$$\begin{aligned} z_{\text{LP}}(\zeta) = \max\ & \zeta u + 4v_1 + 5v_2 + 5v_3 \\ \text{s.t.}\ & 10u + 9v_1 \leq 0,\ -8u + 3v_1 + v_2 \leq 7,\ u + 2v_1 \leq 10,\ -7u + 6v_1 \leq 2 \\ & 6u - 10v_1 + v_3 \leq 10,\ u \leq 0,\ v_2 \leq 0,\ v_3 \leq 0, \end{aligned} \tag{Ex1-D}$$

and we have that the set of extreme points, $\mathcal{E}$, *and the set of extreme rays,* $\mathcal{R}$, *of* $\mathcal{P}_D$ *are*

$$\begin{aligned} \mathcal{E} = \{&(-0.15, 0.16, 0, 0), (0, 0, 0, 0), (-2.35, -2.41, -4.59, 0), \\ &(-1.33, -1.22, 0, 0), (-1.61, -1.97, 0, 0), (0, -1, 0, 0)\}, \\ \mathcal{R} = \{&(0, 0, 0, -1), (0, -1, 0, -10), (-1, -2.66, 0, -20.66), \\ &(-1, -1.166, -4.5, -5.66), (0, 0, -1, 0)\}. \end{aligned}$$

As expected, the function defined by (RLPVF) *is a polyhedral function with the nondifferentiable points being those at which the optimal basis changes. Not all affine pieces are needed to describe the function and a minimal description is as follows. The finite domain is* $\mathcal{C}_{\text{LP}} = [-55.464, +\infty]$

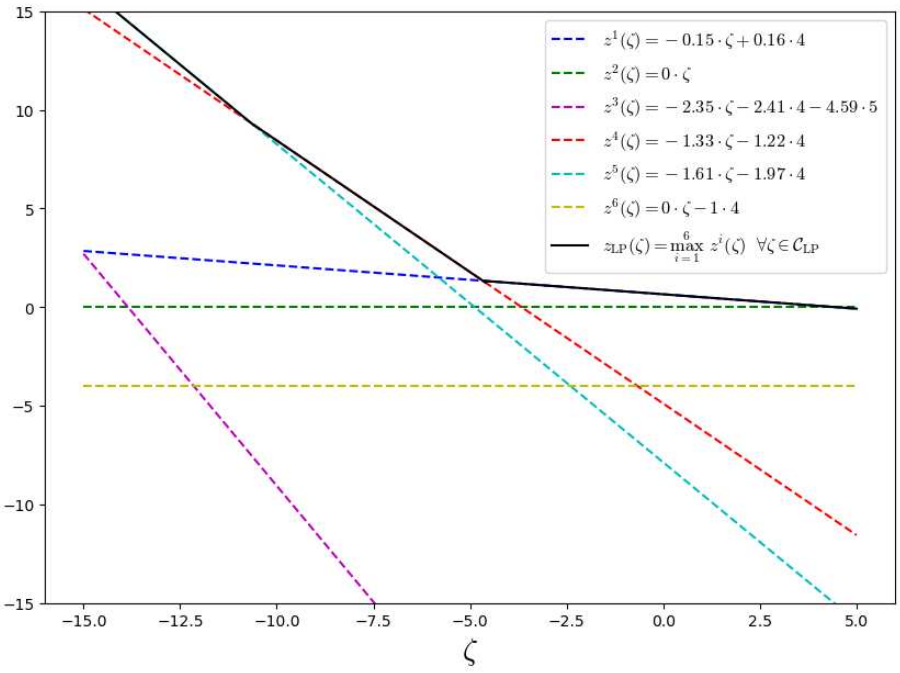


Figure A.1: The RLPVF associated with Example 1 when $(x_1, x_2) = (0, 0)$.

and for $\zeta \in \mathcal{C}_{\mathrm{LP}}$, *we have*

$$z_{\mathrm{LP}}(\zeta) = \max(0, -2.35\zeta - 2.41 \cdot 4 - 4.59 \cdot 5, -1.61\zeta - 1.97 \cdot 4,$$
$$-1.33\zeta - 1.22 \cdot 4, -0.15\zeta + 0.16 \cdot 4).$$

Observe that the gradients of the RLPVF are precisely the optimal dual solutions corresponding to the parametric constraints. At points of nondifferentiability (the breakpoints), the directional derivatives in direction d *(in this case, we have* $d \in \{1, -1\}$*) of the RLPVF are given by* $d^\top u$, *where* u *is one of the alternative optimal dual solutions corresponding to the parametric constraint.*

Appendix B

The LP Frontiers of Example 1

The entire EF for Example 1 is shown below in Figure B.1. The bounding functions associated with each integer vector of Example 1 are illustrated in Figure B.2. To highlight the detail in the bottom right parts of the frontiers more clearly, an enlarged view is presented in Figure B.3.

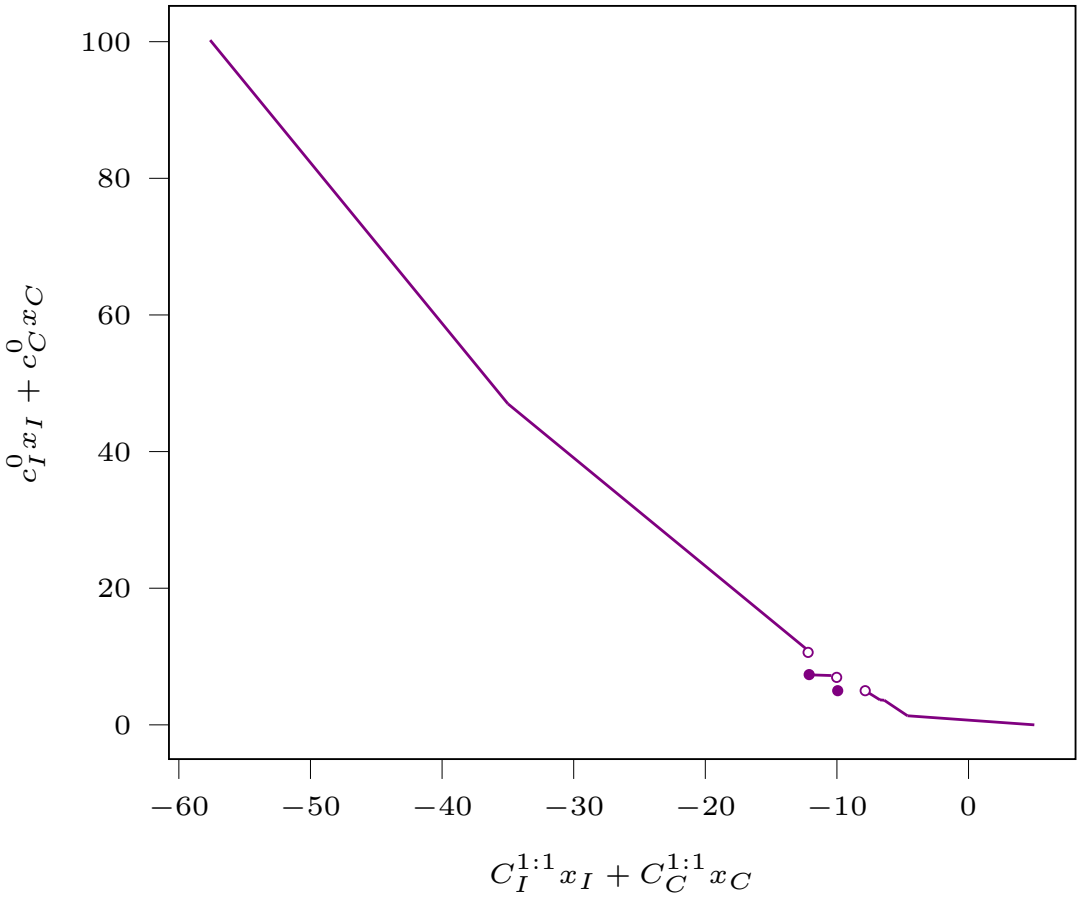


Figure B.1: The EF associated with Example 1.

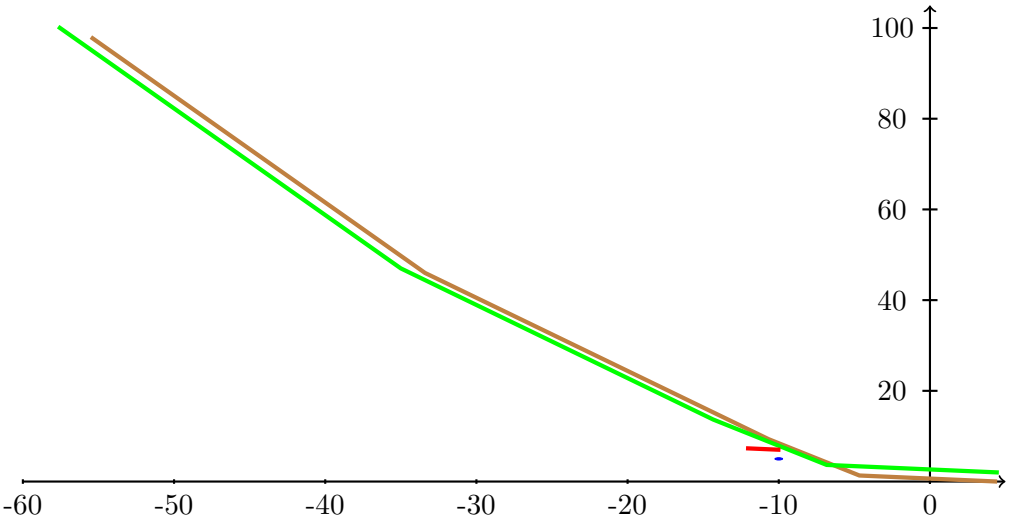


Figure B.2: The LP frontiers. Brown is for $(x_1, x_2) = (0, 0)$. Green is for $(x_1, x_2) = (1, 0)$. Red is for $(x_1, x_2) = (1, 1)$. Blue is for $(x_1, x_2) = (0, 1)$. Note that the axes are unequally scaled.

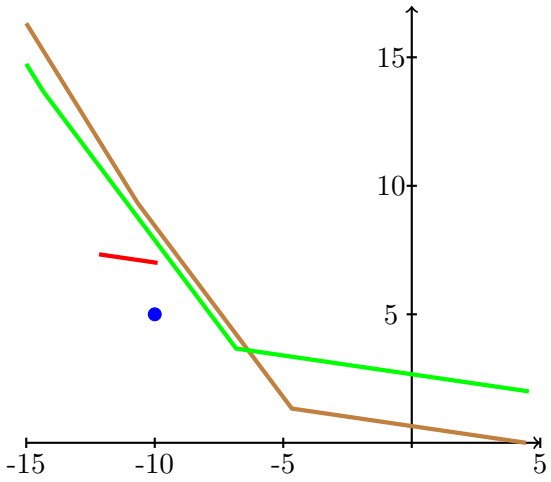


Figure B.3: The (truncated at the left) LP frontiers. Brown is for $(x_1, x_2) = (0, 0)$. *Green is for* $(x_1, x_2) = (1, 0)$. *Red is for* $(x_1, x_2) = (1, 1)$. *Blue is for* $(x_1, x_2) = (0, 1)$.

Appendix C

Proofs Regarding the Structure of RVF

Here, we present the detailed proofs of results from Section 2.1, along with the statements of and proofs for supporting lemmas and propositions.

Proposition C.0.1. *For $\zeta \in \mathcal{C}_{\mathrm{LP}}$, we have that*

$$\nabla_d z_{\mathrm{LP}}(\zeta) = \begin{cases} \max_{u \in \mathrm{proj}_u(\mathrm{OPT}(\zeta))} u^\top d, & \text{if } d \notin \delta^-(\zeta), \\ +\infty, & \text{otherwise,} \end{cases}$$

for all $d \in \mathbb{R}^\ell$.

Proof. We let $\zeta \in \mathcal{C}_{\mathrm{LP}}$ and $d \in \mathbb{R}^\ell$ be given and consider $\nabla_d z_{\mathrm{LP}}(\zeta)$. There are two cases.

(i) If $d \in \delta^-(\zeta)$, then we have already seen that $\nabla_d z_{\mathrm{LP}}(\zeta) = +\infty$, since $z_{\mathrm{LP}}(\zeta + \epsilon d) = +\infty$ for all $\epsilon > 0$. Equivalently, we have that $\mathrm{proj}_u(\mathrm{OPT}(\zeta + \epsilon d)) = \emptyset$ and thus $\max_{u \in \mathrm{proj}_u(\mathrm{OPT}(\zeta))} u^\top d = +\infty$, proving the result in this first case.

(ii) If $d \notin \delta^-(\zeta)$, by (D-RLPVF), there exists $\epsilon > 0$ such that $z_{\mathrm{LP}}(\zeta + \epsilon d) < +\infty$. As such, there

exists $(\hat{u}, \hat{v}) \in \mathcal{E}$ such that

$$z_{\text{LP}}(\zeta + td) = (\zeta + td)^\top \hat{u} + \beta^\top \hat{v}, \qquad \forall t \in [0, \epsilon]. \tag{C.1}$$

Thus $\nabla_d z_{\text{LP}}(\zeta) = \hat{u}^\top d$. Note that by taking $t = 0$ we have that $\hat{u} \in \text{proj}_u(\text{OPT}(\zeta))$. Now suppose, for the sake of contradiction, that $\overline{u}^\top d > \hat{u}^\top d$ for some $\overline{u} \in \text{proj}_u(\text{OPT}(\zeta))$. Then by (2.2), there must exist $(u, v) \in \mathcal{E}$ with $u \in \text{proj}_u(\text{OPT}(\zeta))$ and $u^\top d > \hat{u}^\top d$. But then for any $t \in (0, \epsilon]$, we have

$$\begin{aligned}(\zeta + td)^\top u + \beta^\top v = z_{\text{LP}}(\zeta) + td^\top u > z_{\text{LP}}(\zeta) + td^\top \hat{u} = \zeta^\top \hat{u} + \beta^\top \hat{v} + td^\top \hat{u} \\ = z_{\text{LP}}(\zeta + td),\end{aligned}$$

where the final equality follows from (C.1), which contradicts (D-RLPVF) at $\zeta + td$. The result follows.

■

Proof of Prosition 2.1.2. The result follows from lemmas C.0.2, C.0.3, and C.0.4, which together show containment and reverse containment of $\text{proj}_u(\text{OPT}(\zeta))$ and $\partial z_{\text{LP}}(\zeta)$. Lemma C.0.3 is a known result needed to prove Lemma C.0.4, which we prove here again because part of the proof is referred to later.

■

Lemma C.0.2. *For all $\zeta \in \mathcal{C}_{\text{LP}}$, we have that*

$$\text{proj}_u(\text{OPT}(\zeta)) \subseteq \partial z_{\text{LP}}(\zeta).$$

Proof. Let $\hat{\zeta} \in \mathcal{C}_{\text{LP}}$ and $\hat{u} \in \text{proj}_u(\text{OPT}(\hat{\zeta}))$ be given. We show that $\hat{u} \in \partial z_{\text{LP}}(\hat{\zeta})$. Let $\hat{v}$ be such that $(\hat{u}, \hat{v}) \in \mathcal{P}_D$ and $\hat{\zeta}^\top \hat{u} + \beta^\top \hat{v} = z_{\text{LP}}(\hat{\zeta})$. Such $\hat{v}$ must exist by the definitions of $\text{proj}_u(\text{OPT}(\hat{\zeta}))$. Then for any $\zeta \in \mathcal{C}_{\text{LP}}$, we have

$$\begin{aligned}(\zeta - \hat{\zeta})^\top \hat{u} + z_{\text{LP}}(\hat{\zeta}) &= \zeta^\top \hat{u} - \hat{\zeta}^\top \hat{u} + \hat{\zeta}^\top \hat{u} + \beta \hat{v} \\ &= \zeta^\top \hat{u} + \beta \hat{v} \\ &\leq z_{\text{LP}}(\zeta).\end{aligned} \tag{C.2}$$

It follows directly that $\hat{u} \in \partial z_{\mathrm{LP}}(\hat{\zeta})$. Since $\hat{\zeta}$ was chosen arbitrarily, the result follows. ■

Lemma C.0.3 ([Ralphs and Hassanzadeh, 2014])**.**

$$\begin{aligned} \operatorname{epi} z_{\mathrm{LP}} = \big\{(\zeta, w) \ : \zeta^\top e + \beta^\top h \leq 0, \ \forall (e,h) \in \mathcal{R} \ \textit{and} \\ w \geq \zeta^\top u + \beta^\top v, \ \forall (u,v) \in \mathcal{E}\big\}. \end{aligned} \tag{C.3}$$

Proof. By Proposition 2.1.1, z_{LP} is a polyhedral function, so its epigraph, given by

$$\operatorname{epi} z_{\mathrm{LP}} = \left\{(\zeta, w) \in \mathbb{R}^\ell \times \mathbb{R} \ : \ \zeta \in \mathcal{C}_{\mathrm{LP}}, \ w \geq z_{\mathrm{LP}}(\zeta)\right\},$$

is a polyhedron. By the definition of the subdifferential, for any $\hat{\zeta} \in \mathcal{C}_{\mathrm{LP}}$ and subgradient $g \in \partial z_{\mathrm{LP}}(\hat{\zeta})$, the hyperplane

$$\left\{(\zeta, w) \in \mathbb{R}^\ell \times \mathbb{R} : (-g, 1)^\top \begin{pmatrix} \zeta \\ w \end{pmatrix} = z_{\mathrm{LP}}(\hat{\zeta}) - g^\top \hat{\zeta} \right\},$$

is a hyperplane that supports epi z_{LP} at $\zeta = \hat{\zeta}$ and the inequality

$$(-g, 1)^\top \begin{pmatrix} \zeta \\ w \end{pmatrix} \geq z_{\mathrm{LP}}(\hat{\zeta}) - g^\top \hat{\zeta}, \tag{C.4}$$

is satisfied by all $(\zeta, w) \in \operatorname{epi} z_{\mathrm{LP}}$, with equality attained at $(\hat{\zeta}, z_{\mathrm{LP}}(\hat{\zeta}))$.

The proof also makes use of the extreme points and extreme rays of the dual LP feasible set. Recall that $\mathcal{E}$ and $\mathcal{R}$ are the (finite) sets of extreme points and extreme rays of $\mathcal{P}_D$, respectively. Recall that for $\zeta \in \mathcal{C}_{\mathrm{LP}}$, we have that

$$\zeta^\top e + \beta^\top h \leq 0, \qquad \forall (e,h) \in \mathcal{R},$$

and that by LP duality, we have

$$z_{\mathrm{LP}}(\zeta) = \max\left\{\zeta^\top u + \beta^\top v \ : \ \ (u,v) \in \mathcal{E}\right\}.$$

■

Lemma C.0.4. *For all $\zeta \in \mathcal{C}_{\text{LP}}$,*

$$\partial z_{\text{LP}}(\zeta) \subseteq \text{proj}_u(\text{OPT}(\zeta)).$$

Proof. Let $\hat{\zeta} \in \mathcal{C}_{\text{LP}}$ and $g \in \partial z_{\text{LP}}(\hat{\zeta})$ be given. Then by Lemma C.0.3 (and (C.4)), it must be that $(\hat{\zeta}, z_{\text{LP}}(\hat{\zeta}))$ is an optimal solution of the LP

$$\left\{ \begin{array}{ll} \min & (-g,1)^\top \begin{pmatrix} \zeta \\ w \end{pmatrix} \\ \text{s.t.} & (\zeta, w) \in \text{epi}\, z_{\text{LP}} \end{array} \right. = \left\{ \begin{array}{lll} \min_{\zeta,w} & -g^\top \zeta + w & \\ \text{s.t.} & \zeta^\top e + \beta^\top h \leq 0, & \forall (e,h) \in \mathcal{R}, \\ & w \geq \zeta^\top u + \beta^\top v, & \forall (u,v) \in \mathcal{E}, \end{array} \right. \tag{C.5}$$

which has LP dual

$$\begin{array}{ll} \max_{\lambda,\gamma} & \displaystyle\sum_{(u,v)\in\mathcal{E}} \lambda_{uv} \beta^\top v + \sum_{(e,h)\in\mathcal{R}} \gamma_{eh} \beta^\top h \\ \text{s.t.} & \displaystyle\sum_{(u,v)\in\mathcal{E}} \lambda_{uv} u + \sum_{(e,h)\in\mathcal{R}} \gamma_{eh} e = g \\ & \displaystyle\sum_{(u,v)\in\mathcal{E}} \lambda_{uv} = 1 \\ & \lambda \in \mathbb{R}^{\mathcal{E}}_+, \quad \gamma \in \mathbb{R}^{\mathcal{R}}_+, \end{array} \tag{C.6}$$

where the γ is the vector of dual variables associated with the first set of constraints, and λ is the vector of dual variables associated with the second set of constraints. Recall that $(\hat{\zeta}, z_{\text{LP}}(\hat{\zeta}))$ is an optimal solution of (C.5) and let (λ^*, γ^*) be an optimal solution of (C.6). Then taking

$$\begin{aligned} (u^*, v^*) &= \sum_{(u,v)\in\mathcal{E}} \lambda^*_{uv}(u,v) \\ (e^*, h^*) &= \sum_{(e,h)\in\mathcal{R}} \lambda^*_{eh}(e,h), \end{aligned} \tag{C.7}$$

we have that $(u^* + e^*, v^* + h^*) \in \mathcal{P}_D$ and also that $\hat{\zeta}^\top (u^* + e^*) = z_{\text{LP}}(\hat{\zeta}) - \beta^\top (v^* + h^*)$. And finally, since the constraints state that $g = u^* + e^*$, we have that $g \in \text{proj}_u(\text{OPT}(\hat{\zeta}))$. Since $\hat{\zeta}$ was arbitrary, the result follows. ■

Corollary C.0.5. *$z_{\text{LP}}(\zeta)$ is differentiable at ζ if and only if $\zeta \in \text{int}\, \mathcal{C}_{\text{LP}}$ and the dual problem (D-RLP) has a unique optimal solution.*

Proof. For $\zeta \in \text{int}\, \mathcal{C}_{\text{LP}}$, $\text{proj}_u(\text{OPT}(\zeta))$ consists of a singleton if and only if (D-RLP) has a unique

optimum. By Proposition 2.1.2, the subdifferential $\partial z_{\mathrm{LP}}(\zeta)$ is also a singleton if and only if the dual problem given in (D-RLP) has a unique optimum. The result follows. ■

Proposition C.0.6. *The restricted value function* (RVF) z *is a lower semi-continuous, nonincreasing function that is composed of a minimum of a finite number of polyhedral functions.*

Proof. The RVF is the minimum of a finite number of functions, all of which are polyhedral by Proposition 2.1.1. The lower semi-continuity property can be established using a proof similar to that presented in [Nemhauser and Wolsey, 1988]. The nonincreasing property of the VF follows from the fact that it is the minimum of a set of nonincreasing functions and must therefore be nonincreasing itself. ■

Proof of Proposition 2.1.6. Let $d \in \mathbb{R}^{\ell}$, and a minimal description $\mathcal{S}_{\min}$ of z be given. By Theorem 2.1.5, z is the minimum of a finite set of functions, $\overline{z}(\cdot;\ x_I)$, for each $x_I \in \mathcal{S}_{\min}$, which is a finite set. Furthermore, by Proposition 2.1.1, each of these functions is polyhedral. Thus, Proposition G.0.1 (in Appendix G), which is a general result about functions that are the minimum of a finite number of polyhedral functions, applies and yields the first part.

The second part follows by substituting the formula for the directional derivative of the bounding function from Proposition C.0.1.

■

Lemma C.0.7. *For any* $\zeta \in \mathcal{C}$ *and any minimal description* $\mathcal{S}_{\min}$*, there exists* $\epsilon > 0$ *such that*

$$z(\zeta') \geq \min_{x_I \in S_I^*(\zeta) \cap \mathcal{S}_{\min}} \overline{z}(\zeta';\ x_I), \qquad \forall \zeta' \in B(\zeta;\ \epsilon), \tag{C.8}$$

where $B(\zeta;\ \epsilon)$ *denotes the open ball of radius* ϵ *centered on* ζ*.*

Proof. Let $\mathcal{S}_{\min}$ be given as per Theorem 2.1.5 and let $\zeta \in \mathcal{C}$. Recall that by Theorem 2.1.5,

$$z(\zeta') = \min_{x_I \in \mathcal{S}_{\min}} \overline{z}(\zeta';\ x_I), \qquad \forall \zeta' \in \mathbb{R}^{\ell}. \tag{C.9}$$

Define

$$\alpha(\zeta') = \min_{x_I \in S_I^*(\zeta) \cap \mathcal{S}_{\min}} \overline{z}(\zeta';\ x_I), \qquad \forall \zeta' \in \mathbb{R}^{\ell}.$$

Here, the bounding functions $\overline{z}(\cdot;\ x_I)$ for $x_I \in S_I^*(\zeta) \cap \mathcal{S}_{\min}$ are those that are both active at ζ and included in the given minimal description of the RVF provided by $\mathcal{S}_{\min}$. Using the notation just introduced, the inequality (C.8) simply says $z(\zeta') \geq \alpha(\zeta')$. Now suppose that (C.8) does *not* hold. Since the inequality trivially holds for $\zeta' \notin \mathcal{C}$ ($z(\zeta') = +\infty$ in that case), then this means that there is some $\zeta' \in B(\zeta;\ \epsilon) \cap \mathcal{C}$ such that for every $\epsilon > 0$, there exists $x_I \in \mathcal{S}_{\min}$ with $\overline{z}(\zeta';\ x_I) < \alpha(\zeta')$. This is because $z(\zeta') = \overline{z}(\zeta';\ x_I)$ for some $x_I \in \mathcal{S}_{\min}$, by (C.9). Since there are only a finite number of $x_I \in \mathcal{S}_{\min}$, and each bounding function $\overline{z}(\cdot;\ x_I)$ is polyhedral, actually there must exist one $\hat{x}_I \in \mathcal{S}_{\min}$ so that for all $\epsilon > 0$, some $\zeta' \in B(\zeta;\ \epsilon)$ has $\overline{z}(\zeta';\ \hat{x}_I) < \alpha(\zeta')$. Now $\overline{z}(\cdot;\ \hat{x}_I)$ is continuous on its finite domain, which is a closed set, so it must be that $\overline{z}(\zeta;\ \hat{x}_I) \leq \alpha(\zeta)$. But $\alpha(\zeta) = z(\zeta) \leq \overline{z}(\zeta;\ \hat{x}_I)$. So it must be that $\overline{z}(\zeta;\ \hat{x}_I) = z(\zeta)$, and $\hat{x}_I \in \mathcal{S}_I^*(\zeta)$. But then $\hat{x}_I \in \mathcal{S}_I^*(\zeta) \cap \mathcal{S}_{\min}$, so in fact $\overline{z}(\zeta';\ \hat{x}_I) \geq \alpha(\zeta')$ for all $\zeta' \in \mathbb{R}^\ell$, and we obtain a contradiction. The result follows. ■

Proof of Proposition 2.1.8. Let $\mathcal{S}_{\min}$ be a minimal description of the RVF, and consider $\zeta \in \mathcal{C}$. Let $g \in \partial_L z(\zeta)$. Then there exists $\epsilon > 0$ such that

$$z(\zeta') \geq z(\zeta) + g^\top(\zeta' - \zeta), \quad \forall \zeta' \in B(\zeta;\ \epsilon).$$

Let $x_I \in S_I^*(\zeta)$ be chosen arbitrarily. Now for any $\zeta' \in B(\zeta;\ \epsilon)$,

$$\overline{z}(\zeta';\ x_I) \geq z(\zeta') \geq z(\zeta) + g^\top(\zeta' - \zeta) = \overline{z}(\zeta;\ x_I) + g^\top(\zeta' - \zeta),$$

where the first inequality follows from the definition of z, the second since g is a local subgradient of z at ζ, and the final equality follows since $x_I \in S_I^*(\zeta)$. Thus g is also a local subgradient of $\overline{z}(\cdot;\ x_I)$ at ζ. But $\overline{z}(\cdot;\ x_I)$ is a (RLPVF), and so is a convex function over a convex finite domain. Hence

$$g \in \partial \overline{z}(\zeta;\ x_I) = \partial z_{\mathrm{LP}}(\zeta - C_I^{1:\ell} x_I;\ b - A_I x_I).$$

Since x_I was chosen arbitrarily from $S_I^*(\zeta)$, we have proved that

$$\partial_L z(\zeta) \subseteq \bigcap_{x_I \in S_I^*(\zeta)} \partial z_{\mathrm{LP}}(\zeta - C_I^{1:\ell} x_I;\ b - A_I x_I) \subseteq \bigcap_{x_I \in S_I^*(\zeta) \cap \mathcal{S}_{\min}} \partial z_{\mathrm{LP}}(\zeta - C_I^{1:\ell} x_I;\ b - A_I x_I).$$

To prove containment in the opposite direction, let

$$g \in \bigcap_{x_I \in S_I^*(\zeta) \cap \mathcal{S}_{\min}} \partial z_{\mathrm{LP}}(\zeta - C_I^{1:\ell} x_I;\ b - A_I x_I) = \bigcap_{x_I \in S_I^*(\zeta) \cap \mathcal{S}_{\min}} \partial \overline{z}(\zeta;\ x_I).$$

By Lemma C.0.7, there must exist $\epsilon > 0$ so that (C.8) holds. Choose $\zeta' \in B(\zeta;\ \epsilon)$ arbitrarily. Then, by (C.8),

$$z(\zeta') \geq \min_{x_I \in S_I^*(\zeta) \cap \mathcal{S}_{\min}} \overline{z}(\zeta';\ x_I),$$

and so there exists some $x_I \in S_I^*(\zeta) \cap \mathcal{S}_{\min}$ with

$$z(\zeta') \geq \overline{z}(\zeta';\ x_I) \geq \overline{z}(\zeta;\ x_I) + g^\top(\zeta' - \zeta) = z(\zeta) + g^\top(\zeta' - \zeta),$$

where the second inequality follows since $g \in \partial \overline{z}(\zeta;\ x_I)$ and the final equality follows since $x_I \in S_I^*(\zeta)$. Thus $g \in \partial_L z(\zeta)$ and it must be that

$$\partial_L z(\zeta) \supseteq \bigcap_{x_I \in S_I^*(\zeta) \cap \mathcal{S}_{\min}} \partial z_{\mathrm{LP}}(\zeta - C_I^{1:\ell} x_I;\ b - A_I x_I).$$

Hence

$$\partial_L z(\zeta) = \bigcap_{x_I \in S_I^*(\zeta) \cap \mathcal{S}_{\min}} \partial z_{\mathrm{LP}}(\zeta - C_I^{1:\ell} x_I;\ b - A_I x_I).$$

The result follows from Proposition 2.1.2. ■

Appendix D

Theorem 2.2.1 with $C^{1:\ell}x = \zeta$

As mentioned in Section 2.2, we can have another representation of Theorem 2.2.1. Consider the RVF $z' : \mathbb{R}^\ell \to \mathbb{R} \cup \{\pm\infty\}$ as

$$z'(\zeta) = \inf_{(x_I, x_C) \in \mathcal{S}(\zeta)} c_I^0 x_I + c_C^0 x_C, \tag{RVF$'$}$$

where

$$\mathcal{S}(\zeta) = \left\{(x_I, x_C) \in \mathbb{Z}_+^r \times \mathbb{R}_+^{n-r} \ : \ C_I^{1:\ell} x_I + C_C^{1:\ell} x_C = \zeta, \ A_I x_I + A_C x_C = b\right\}.$$

It is worth noting that the only distinction here is that we have an equality sign for the constraints that serve as our objectives. We formalize the relationship between the RVF and the EF through the following theorem.

Theorem D.0.1. *The following statements hold for X_{MO} and the (RVF$'$) z'.*

(a) If $(x_I, x_C) \in X_{\mathrm{MO}}$ is an efficient solution (equivalently, $C_I x_I + C_C x_C$ is an NDP), then $(\zeta, c^0 x_I + c^0 x_C)$ is a point on the boundary of the epigraph of z' for $\zeta = C_I^{1:\ell} x_I + C_C^{1:\ell} x_C$.

(b) If $(\zeta, z'(\zeta))$ is a point on the boundary of the epigraph of z' and $\nabla_d z'(\zeta) > 0$ for all $d \in \mathbb{R}_-^\ell \setminus \{\mathbf{0}\}$ for which $\nabla_d z'(\zeta)$ exists, then there exists an efficient solution $(x_I, x_C) \in X_{\mathrm{MO}}$ that yields $z'(\zeta)$ and satisfies $C_I^{1:\ell} x_I + C_C^{1:\ell} x_C = \zeta$.

Proof. The proofs for Part (a) and Part (b) are in accordance with the proofs for Part 1 and Part 2

with the exception of the portion designated by $\Leftarrow$ in Theorem 2.2.1. ■

Here we have a very similar proposition compared to Proposition C.0.6 as follows.

Proposition D.0.2. *The RVF is a lower semi-continuous function and decreasing over $\mathcal{C}$ where the optimal dual value is negative. Furthermore, it is comprised of a minimum of a finite number of polyhedral functions.*

Proof. The proof for the property of semi-continuity and being a minimum of a finite number of polyhedral functions is the same as that in Proposition C.0.6. The decreasing property is straightforward; with negative optimal dual values, the VF is decreasing. ■

Therefore, in the steps of the algorithm, although we have equality for the constraints with the parametric RHS, i.e., the constraints that serve as our objectives, we must have negative dual variables $u \in \mathbb{R}^{\ell}_{-}$, which relates to the constraints with the parametric RHS. In this case, the algorithm generates that part of the VF, which is decreasing and is the same as the related EF. As a result, the VF algorithm remains unchanged in this scenario.

Appendix E

Proofs of Correctness for the RVF Algorithm

The following lemmas are used to prove the correctness of the RVF algorithm, given in Theorem 3.1.1. We start with the following simple observation.

Lemma E.0.1. *For all $(\hat{x}_I, \hat{x}_C) \in X_{\text{MO}}$, we have that*

$$\bar{z}(C_I^{1:\ell}\hat{x}_I + C_C^{1:\ell}\hat{x}_C;\ \hat{x}_I) \leq c_I^0\hat{x}_I + c_C^0\hat{x}_C.$$

Proof. The bounding function (CR) associated with $\hat{x}_I$ is equivalently expressed as

$$\bar{z}(\zeta;\ \hat{x}_I) = \min\left\{c_I^0 x_I + c_C^0 x_C\ :\ (x_I, x_C) \in X_{\text{MO}},\ x_I = \hat{x}_I,\ C_I^{1:\ell}x_I + C_C^{1:\ell}x_C \leq \zeta\right\}. \tag{E.1}$$

The result follows as $(\hat{x}_I, \hat{x}_C)$ is feasible for the problem above for $\zeta = C_I^{1:\ell}\hat{x}_I + C_C^{1:\ell}\hat{x}_C$. ■

The following lemma helps to establish that the integer parts added to $\mathcal{S}^k$ by the algorithm eventually suffice to describe the whole VF.

Lemma E.0.2. *At iteration k of the algorithm, $\theta^{k+1} = \max_{\zeta\in\mathcal{C}} \bar{z}^k(\zeta) - z(\zeta)$.*

Proof. Let $\zeta^* = \arg\max_{\zeta\in\mathcal{C}}(\bar{z}^k(\zeta) - z(\zeta))$ and denote $\zeta^{k+1} = C_I^{1:\ell}x_I^{k+1} + C_C^{1:\ell}x_C^{k+1}$. Since $\zeta^* \in \mathcal{C}$

there must exist $(\hat{x}_I, \hat{x}_C) \in X_{\mathrm{MO}}$ with $\hat{\zeta} := C_I^{1:\ell}\hat{x}_I + C_C^{1:\ell}\hat{x}_C \leq \zeta^*$ and $z(\zeta^*) = c_I^0\hat{x}_I + c_C^0\hat{x}_C = z(\hat{\zeta})$. Further, since $\bar{z}^k$ is nonincreasing, $\bar{z}^k(\hat{\zeta}) \geq \bar{z}^k(\zeta^*)$, and hence

$$\begin{aligned}\bar{z}^k(\hat{\zeta}) - z(\hat{\zeta}) &\geq \bar{z}^k(\zeta^*) - z(\hat{\zeta}) \\ &= \bar{z}^k(\zeta^*) - z(\zeta^*) \\ &\geq \bar{z}^k(\zeta^{k+1}) - z(\zeta^{k+1}).\end{aligned}$$

The last of these inequalities follows from the fact that $\zeta^{k+1} \in \mathcal{C}$. Since

$$(x_I^{k+1}, x_C^{k+1}) \in \underset{(x_I, x_C) \in X_{\mathrm{MO}}}{\arg\max} \left(\bar{z}^k(C_I^{1:\ell}x_I + C_C^{1:\ell}x_C) - (c_I^0 x_I + c_C^0 x_C)\right),$$

it follows that

$$\bar{z}^k(\zeta^{k+1}) - z(\zeta^{k+1}) \geq \bar{z}^k(\hat{\zeta}) - z(\hat{\zeta}) \Rightarrow \bar{z}^k(\zeta^{k+1}) - z(\zeta^{k+1}) = \bar{z}^k(\hat{\zeta}) - z(\hat{\zeta}).$$

Finally, by the definition of θ^{k+1}, we have

$$\theta^{k+1} = \bar{z}^k(\zeta^{k+1}) - z(\zeta^{k+1}) = \bar{z}^k(\zeta^*) - z(\zeta^*),$$

and the result follows. ■

The crucial property for establishing termination of the algorithm is that as long as $\bar{z}^k \neq z$, then in iteration k, we are guaranteed to produce a point not already contained in $\mathcal{S}^k$. This is established in the lemma below, which states that unless the optimal value of the optimization problem in (3.2) is zero, the integer part of the solution obtained is not contained in $\mathcal{S}^k$.

Lemma E.0.3. *Any optimal solution (x_I^{k+1}, x_C^{k+1}) to the optimization problem in (3.2) having optimal value $\theta^{k+1} > 0$ has the property that $x_I^{k+1} \notin \mathcal{S}^k$.*

Proof. We show the contrapositive of the lemma as follows. Let (x_I^{k+1}, x_C^{k+1}) solve the optimization problem in (3.2) having optimal value θ^{k+1} and suppose that $x_I^{k+1} \in \mathcal{S}^k$. Then

$$\bar{z}^k(C_I^{1:\ell}x_I^{k+1} + C_C^{1:\ell}x_C^{k+1}) \leq \bar{z}(C_I^{1:\ell}x_I^{k+1} + C_C^{1:\ell}x_C^{k+1};\ x_I^{k+1}) \leq c_I^0 x_I^{k+1} + c_C^0 x_C^{k+1},$$

where the first inequality follows from (3.1), since $x_I^{k+1} \in \mathcal{S}^k$, and the second inequality follows from Lemma E.0.1, since $(x_I^{k+1}, x_C^{k+1}) \in X_{\mathrm{MO}}$. It thus follows, by the definition of θ^{k+1} and x^{k+1}, that

$$\theta^{k+1} = \bar{z}^k(C_I^{1:\ell} x_I^{k+1} + C_C^{1:\ell} x_C^{k+1}) - (c_I^0 x_I^{k+1} + c_C^0 x_C^{k+1}) \leq 0,$$

as required to obtain the contrapositive. ■

Appendix F

Proof of the Nonempty Stability Region of the Integer Part in Each Iteration

In this appendix, we show the details of the proof that the integer part added to $\mathcal{S}^k$ in iteration k has a nonempty stability region *at the time it is added*. Note, however, that this stability region may end up being redundant by the end of the algorithm. First, we show that the true VF $z(\zeta^*) = c_I^0 x_I^{k+1} + c_C^0 x_C^{k+1}$ whenever (x_I^{k+1}, x_C^{k+1}) solves the optimization problem (3.2) and $\zeta^* = C_I^{1:\ell} x_I^{k+1} + C_C^{1:\ell} x_C^{k+1}$.

Lemma F.0.1. *If* (x_I^{k+1}, x_C^{k+1}) *is defined by* (3.2) *then* $z(\zeta^*) = c_I^0 x_I^{k+1} + c_C^0 x_C^{k+1}$ *where* $\zeta^* = C_I^{1:\ell} x_I^{k+1} + C_C^{1:\ell} x_C^{k+1}$.

Proof. Let (x_I^{k+1}, x_C^{k+1}) be defined by (3.2) and suppose, for the sake of contradiction, that $z(C_I^{1:\ell} x_I^{k+1} + C_C^{1:\ell} x_C^{k+1}) \neq c_I^0 x_I^{k+1} + c_C^0 x_C^{k+1}$. Then there must exist $(\hat{x}_I, \hat{x}_C)$ a solution of the optimization problem that defines the RVF, $z(C_I^{1:\ell} x_I^{k+1} + C_C^{1:\ell} x_C^{k+1})$, given by

$$z(C_I^{1:\ell} x_I^{k+1} + C_C^{1:\ell} x_C^{k+1}) = \min \big\{ c_I^0 x_I + c_C^0 x_C \ : (x_I, x_C) \in X_{\text{MO}}, \\ C_I^{1:\ell} x_I + C_C^{1:\ell} x_C \leq C_I^{1:\ell} x_I^{k+1} + C_C^{1:\ell} x_C^{k+1} \big\},$$

and it must be that

$$z(C_I^{1:\ell}x_I^{k+1} + C_C^{1:\ell}x_C^{k+1}) = c_I^0\hat{x}_I + c_C^0\hat{x}_C < c_I^0x_I^{k+1} + c_C^0x_C^{k+1}.$$

Now since $\bar{z}^k$ is nonincreasing, and, by definition, $(\hat{x}_I, \hat{x}_C)$ satisfies

$$C_I^{1:\ell}\hat{x}_I + C_C^{1:\ell}\hat{x}_C \leq C_I^{1:\ell}x_I^{k+1} + C_C^{1:\ell}x_C^{k+1},$$

it must be that

$$\bar{z}^k(C_I^{1:\ell}\hat{x}_I + C_C^{1:\ell}\hat{x}_C) \geq \bar{z}^k(C_I^{1:\ell}x_I^{k+1} + C_C^{1:\ell}x_C^{k+1}).$$

Thus

$$\bar{z}^k(C_I^{1:\ell}\hat{x}_I + C_C^{1:\ell}\hat{x}_C) - (c_I^0\hat{x}_I + c_C^0\hat{x}_C) > \bar{z}^k(C_I^{1:\ell}x_I^{k+1} + C_C^{1:\ell}x_C^{k+1}) - (c_I^0x_I^{k+1} + c_C^0x_C^{k+1}),$$

and $(\hat{x}_I, \hat{x}_C)$ must be a feasible solution for the optimization problem in (3.2) having better objective value than (x_I^{k+1}, x_C^{k+1}), which is a contradiction. ■

Next, we justify our claim that converting to a (strong) efficient solution via (3.3) again yields a solution to (3.2). We omit formal proof that the resulting solution must be efficient for (MO-MILP) since this is a straightforward and well-known result from multiobjective optimization.

Lemma F.0.2. *If (x_I^{k+1}, x_C^{k+1}) is defined by (3.2) and $(\hat{x}_I, \hat{x}_C)$ solves (3.3) then $(\hat{x}_I, \hat{x}_C)$ is also an optimal solution for the optimization problem in (3.2).*

Proof. Let (x_I^{k+1}, x_C^{k+1}) be defined by (3.2) and suppose $(\hat{x}_I, \hat{x}_C)$ solves (3.3). Then $(\hat{x}_I, \hat{x}_C) \in X_{\text{MO}}$, so is feasible for the optimization problem in (3.2). Furthermore,

$$C_I\hat{x}_I + C_C\hat{x}_C \leq C_Ix_I^{k+1} + C_Cx_C^{k+1},$$

so

$$c_I^0\hat{x}_I + c_C^0\hat{x}_C \leq c_I^0x_I^{k+1} + c_C^0x_C^{k+1},$$

and

$$C_I^{1:\ell}\hat{x}_I + C_C^{1:\ell}\hat{x}_C \leq C_I^{1:\ell}x_I^{k+1} + C_C^{1:\ell}x_C^{k+1}.$$

Since $\bar{z}^k$ is nonincreasing it must thus be that

$$\bar{z}^k(C_I^{1:\ell}\hat{x}_I + C_C^{1:\ell}\hat{x}_C) \geq \bar{z}^k(C_I^{1:\ell}x_I^{k+1} + C_C^{1:\ell}x_C^{k+1}),$$

and so

$$\bar{z}^k(C_I^{1:\ell}\hat{x}_I + C_C^{1:\ell}\hat{x}_C) - (c_I^0\hat{x}_I + c_C^0\hat{x}_C) \geq \bar{z}^k(C_I^{1:\ell}x_I^{k+1} + C_C^{1:\ell}x_C^{k+1}) - (c_I^0x_I^{k+1} + c_C^0x_C^{k+1}).$$

Since $(\hat{x}_I, \hat{x}_C)$ is feasible for the optimization problem in (3.2) and has at least as good an objective value as that of (x_I^{k+1}, x_C^{k+1}), it must also solve the optimization problem in (3.2). ■

These two lemmas combine to prove that the integer part added to $\mathcal{S}^k$ at each iteration k of the algorithm has a nonempty stability region.

Proposition F.0.3. *If x_I^{k+1} is added to $\mathcal{S}^k$ in iteration k of the RVF Algorithm then its stability region $\mathcal{C}(x_I^{k+1})$ is nonempty.*

Proof. By Lemmas F.0.1 and F.0.2, if x_I^{k+1} is added to $\mathcal{S}^k$ in iteration k of the RVF Algorithm, then x_C^{k+1} found in the process has $c_I^0x_I^{k+1} + c_C^0x_C^{k+1} = z(\zeta)$ where $\zeta = C_I^{1:\ell}x_I^{k+1} + C_C^{1:\ell}x_C^{k+1}$. But by the characterization of the VF as the minimum of a finite set of bounding functions (as per Theorem 2.1.5), we have that

$$z(\zeta) \leq \bar{z}(\zeta;\ x_I^{k+1}) \leq c_I^0x_I^{k+1} + c_C^0x_C^{k+1},$$

where the latter inequality follows by Lemma E.0.1, since $(x_I^{k+1}, x_C^{k+1}) \in X_{\text{MO}}$. Thus it must be that $z(\zeta) = \bar{z}(\zeta;\ x_I^{k+1})$, so $\zeta \in \mathcal{C}(x_I^{k+1})$ and the stability region $\mathcal{C}(x_I^{k+1})$ is nonempty, as required. ■

Appendix G

Directional Derivatives of the Minimum of Polyhedral Functions

Here, we describe the directional derivatives of a function formed by taking the minimum of a finite set of polyhedral functions. We are particularly interested in functions defined over the extended reals.

The properties we identify here are for any such function; they are not specific to VFs. We think it is very likely that the material we provide here has already been published. However, we have been unable to locate a source for it, so we have included it for the sake of completeness.

First, we give definitions and conventions for directional derivatives of functions defined over the extended reals. Then we briefly review the properties of polyhedral functions, before giving our main result on the directional derivatives of the minimum of a finite set of polyhedral functions.

Consider $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$, a function defined over the extended reals that is bounded below, and so cannot have the value $-\infty$. We refer to the set of points $x \in \mathbb{R}^n$ for which $f(x) < +\infty$ as the *finite domain* of f. We are interested in functions with a closed finite domain. The directional derivative of f at the point $x \in \mathbb{R}^n$ in direction $d \in \mathbb{R}^n$ is denoted by $\nabla_d f(x)$ and is defined by

$$\nabla_d f(x) = \lim_{t \to 0^+} \frac{f(x + td) - f(x)}{t}.$$

We take $\nabla_d f(x) = 0$ if $f(x) = +\infty$ (meaning x is not in the finite domain of f). If $f(x) < +\infty$ and there exists $\epsilon > 0$ such that $f(x + td) = +\infty$ for all $t \in (0, \epsilon)$, then we say x is on the boundary of the finite domain of f and d points out of it. In this case, $\nabla_d f(x) = +\infty$.

We now turn our attention to the case of a function defined as the minimum of a finite set of polyhedral functions.

Say $z^k : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ is a given polyhedral function for each $k = 1, \ldots, K$, which means that the epigraph over its finite domain is a polyhedron. A property of a polyhedral function is that for any point x and direction d, one of the following cases must hold:

1. x is not in the finite domain of z^k, in which case $\nabla_d z^k(x) = 0$,
2. x is on the boundary of the finite domain of z^k and d points out of it, in which case $\nabla_d z^k(x) = +\infty$, or
3. there exists $\epsilon > 0$ such that $x + td$ is in the finite domain of z^k for all $t \in [0, \epsilon]$, and, furthermore, ϵ may be taken to be sufficiently small that all points in the line segment defined as $\{x + td \ : \ 0 \leq t \leq \epsilon\}$ lie on the same facet of the polyhedron created by the epigraph of z^k over its finite domain. In this case, there exists $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$ such that $z^k(x + td) = a^\top (x + td) + b$ for all $t \in [0, \epsilon]$. Note that a and b depend on x and d, but not on ϵ, and it follows that $\nabla_d z^k(x) = a^\top d$.

It is also helpful to observe that the finite domain of a polyhedral function must itself be a polyhedron, which is closed.

Let $z : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ be defined to be

$$z(x) = \min_{k=1\ldots,K} z^k(x).$$

Since each function z^k bounds z from above, we call z^k a bounding function. Define

$$\kappa^*(x) = \{k \in \{1, \ldots, K\} : z^k(x) = z(x)\},$$

to be the index set of the bounding functions that are active at x. Clearly, the finite domain of z is the union of the finite domains of its bounding functions, so it is the union of a finite collection of

closed sets, and hence is closed.

It is useful to observe that in the case that x is in the finite domain of z and d does *not* point out of it, then it may be that there is a jump, or discontinuity, in z when moving away from x in the direction d. In other words, for such x and d, it may be that

$$\lim_{t \to 0^+} z(x + td) > z(x).$$

In such a case, there must exist $k \in \{1, \ldots, K\}$ and $\epsilon > 0$ so that $z(x + td) = z^k(x + td)$ for all $t \in (0, \epsilon)$, where, importantly, $t = 0$ is *not* included in this interval. So

$$\lim_{t \to 0^+} z(x + td) = \lim_{t \to 0^+} z^k(x + td) = z^k(x) > z(x),$$

and it follows that $\nabla_d z(x) = +\infty$.

Whether or not z has a discontinuity when moving away from x in direction d, the structure of z as a minimum of a finite set of polyhedral functions ensures that when x is in the finite domain of z and d does not point out of it, there must exist $k \in \{1, \ldots, K\}$, $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$ defining a facet of the epigraph of z^k over its finite domain, and $\epsilon > 0$ sufficiently small so that

$$z(x + td) = z^k(x + td) = a^\top (x + td) + b, \qquad \forall t \in (0, \epsilon).$$

We say that the facet of z^k defined by (a, b) *yields* z *when moving away from* x *in direction* d.

We now give the main result.

Proposition G.0.1. *For any point $x \in \mathbb{R}^n$ and any direction $d \in \mathbb{R}^n$,*

$$\nabla_d z(x) = \min_{k \in \kappa^*(x)} \nabla_d z^k(x).$$

Proof. Let the point $x \in \mathbb{R}^n$ and direction $d \in \mathbb{R}^n$ be given. There are three main cases.

Case 1: x is not in the finite domain of z.

In this case, $\nabla_d z(x) = 0$. Furthermore, since $z(x) = +\infty$, it follows from the definition of z that $z^k(x) = +\infty$, and hence $\nabla_d z^k(x) = 0$, for all $k = 1, \ldots, K$. The result follows.

Case 2: x is in the boundary of the finite domain of z and d points out of it.

This means that there exists $\epsilon > 0$ so that $z(x + td) = +\infty$ for all $t \in (0, \epsilon)$. By the definition of z, it must be that for all $k = 1, \ldots, K$, $z^k(x + td) = +\infty$ for all $t \in (0, \epsilon)$. Now for all $k \in \kappa^*(x)$, x is in the finite domain of the bounding function z^k, since $z^k(x) = z(x) < +\infty$. Thus $\nabla_d z^k(x) = +\infty$ and the result follows.

Case 3. there exists $\epsilon > 0$ such that $z(x + td) < +\infty$ for all $t \in [0, \epsilon)$.

Note that $t = 0$ is included in the definition of this case, since x is not in the finite domain of z was covered in Case 1. Assume, without loss of generality, that $k = 1$ is the index of the bounding function having a facet that yields z when moving away from x in direction d. Thus there must exist $\epsilon' > 0$, $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$ such that

$$z(x + td) = z^1(x + td) = a^\top (x + td) + b, \qquad \forall t \in (0, \epsilon').$$

Let k^* denote the minimizer of $\nabla_d z^k(x)$ over $k \in \kappa^*(x)$. Note that $\nabla_d z^1(x) = a^\top d$, which is finite. So $1 \in \kappa^*(x)$ implies $\nabla_d z^{k^*}(x)$ is finite. Thus if $\nabla_d z^{k^*}(x) = +\infty$ it must be that $1 \notin \kappa^*(x)$. Hence

$$\lim_{t \to 0^+} z(x + td) = \lim_{t \to 0^+} z^1(x + td) = z^1(x) > z(x),$$

and $\nabla_d z(x) = +\infty = \nabla_d z^{k^*}(x)$, as required. Now consider the case that $\nabla_d z^{k^*}(x)$ is finite. By the properties of polyhedral functions discussed above, there must exist $\epsilon'' > 0$, $\hat{a} \in \mathbb{R}^n$ and $\hat{b} \in \mathbb{R}$ such that $z^{k^*}(x + td) = \hat{a}^\top (x + td) + \hat{b}$ for all $t \in [0, \epsilon'')$, so $\nabla_d z^{k^*}(x) = \hat{a}^\top d$. To summarize, we have

$$\min_{k \in \kappa^*(x)} \nabla_d z^k(x) = \hat{a}^\top d \qquad \text{and} \qquad \nabla_d z(x) = a^\top d.$$

Now suppose, for the sake of contradiction, that $\hat{a}^\top d \neq a^\top d$. Define ϵ''' by

$$\epsilon''' = \begin{cases} +\infty, & \text{if } \hat{a}^\top d < a^\top d \\ \frac{z^1(x) - z(x)}{\hat{a}^\top d - a^\top d}, & \text{if } \hat{a}^\top d > a^\top d. \end{cases}$$

Observe that $\epsilon''' > 0$ since $1 \in \kappa^*(x)$ implies $\hat{a}^\top d = \nabla_d z^{k^*}(x) \leq \nabla_d z^1(x) = a^\top d$, so the case $\hat{a}^\top d > a^\top d$ implies that $1 \notin \kappa^*(x)$, and so $z^1(x) > z(x)$. Then for $0 < t < \epsilon'''$ we have that

$$t(\hat{a}^\top d - a^\top d) < z^1(x) - z(x).$$

Now $z(x) = \hat{a}^\top x + \hat{b}$ since $k^* \in \kappa^*(x)$, and by continuity of z^1, we also have $z^1(x) = a^\top x + b$. Substituting these in, we obtain

$$t(\hat{a}^\top d - a^\top d) < a^\top x + b - (\hat{a}^\top x + \hat{b}),$$

which is equivalently written as

$$\hat{a}^\top (x + td) + \hat{b} < a^\top (x + td) + b.$$

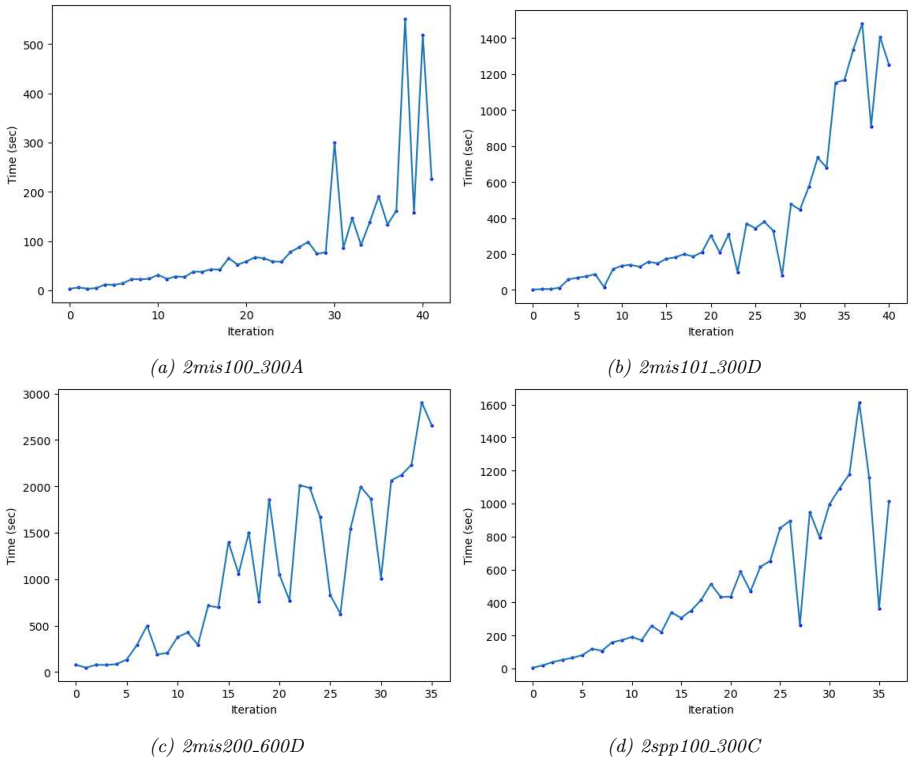
Since $t > 0$, $t < \epsilon'$ and $t < \epsilon''$, it must be that

$$z^{k^*}(x + td) = \hat{a}^\top (x + td) + \hat{b} < a^\top (x + td) + b = z(x + td),$$

which contradicts the definition of z. Thus it must be that $\hat{a}^\top d = a^\top d$, as required. ■

Appendix H

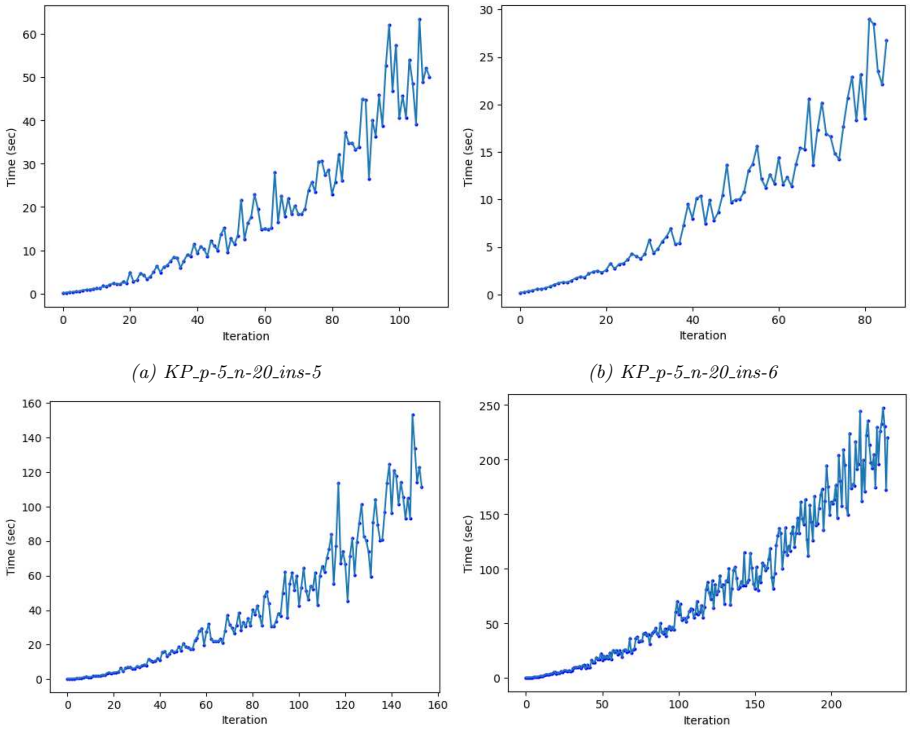
Polynomial Time Growth with Iterations



(a) 2mis100_300A *(b) 2mis101_300D*

(c) 2mis200_600D *(d) 2spp100_300C*

Figure H.1: Execution time vs. iteration for biobjective SPP.



(a) KP_p-5_n-20_ins-5

(b) KP_p-5_n-20_ins-6

(c) KP_p-4_n-20_ins-6

(d) KP_p-4_n-20_ins-10

Figure H.2: Execution time vs. iteration for multiobjective Knapsack.

Appendix I

Analysis of the Number of ILPs Solved in DCM

The Disjunctive Constraints Method (DCM), which is outlined as Algorithm 1 in [Boland et al., 2017], is a technique for solving PILP (MO-MILP)s with an arbitrary number of objectives. In [Boland et al., 2017], the performance of DCM is analyzed, and it is shown that the worst-case number of ILPs that must be solved is exponential in N, the number of NDPs. We demonstrate that the number of ILPs that need to be solved is actually upper bounded by $2N + 1$, regardless of the number of objectives.

As is noted in [Boland et al., 2017], the workings of the DCM correspond to a binary tree (actually a Fibonacci tree, which is a special case) in which each node of the tree represents an object generated by the algorithm and is searched using an ILP subproblem. If the ILP subproblem has a solution, then it is an NDP that is excluded from the search region for all child nodes created (via Proposition 11 in [Boland et al., 2017]), and at least one child node is created. Otherwise, no child nodes are created; the ILP at this node is infeasible, and the node is a leaf node.

Therefore, the DCM algorithm guarantees that each NDP is found exactly once at an internal node of the binary tree, and the total number of ILPs solved is equal to the number of NDPs plus the number of leaf nodes.

Before starting our formal proof, we recall some basic definitions related to binary trees. A binary tree is a finite tree in which each node has 0, 1, or 2 child nodes. An internal node is a node that has at least one child node, while a leaf node is a node that has no child nodes. A *full* binary tree is a binary tree in which every node has 0 or 2 child nodes.

We use the following lemma.

Lemma I.0.1. *The number of leaf nodes in a full binary tree is precisely the number of internal nodes plus 1.*

Proof. This lemma can be proved by induction on the number of internal nodes in the tree. The inductive step considers the removal of an internal node of maximum depth. ■

It is worth noting that a binary tree that is not full may have fewer leaf nodes per internal node. For instance, consider a tree consisting of a "chain" of nodes, each having only one child until the single leaf node is reached. The following proposition demonstrates that such a tree cannot have more leaf nodes per internal node.

Proposition I.0.2. *The number of leaf nodes in a binary tree is no greater than the number of internal nodes plus 1.*

Proof. Let L denote the number of leaf nodes and I the number of internal nodes in a binary tree. We prove the result by induction on I. Clearly, the case $I = 0$ is the tree consisting of a single node, which is a leaf, so $L = 1 = I + 1$ as required. Now suppose that the result holds for all binary trees with $I - 1$ or fewer internal nodes, and consider a binary tree with I internal nodes and L leaf nodes. If every internal node has 2 children, then the tree is a full binary tree and $L = I + 1$ by Lemma I.0.1, so $L \leq I + 1$, as required. Thus, suppose there exists an internal node with 1 child. We create a new binary tree by replacing this internal node with its descendant subtree. This operation does not change the number of child nodes of the parent node, as its descendant subtree starts from a single child. Thus, the new binary tree has $I - 1$ internal nodes but still L leaf nodes. By the inductive hypothesis, $L \leq I - 1 + 1 = I$, which implies $L \leq I + 1$, as required. ■

We can now present the worst-case performance result for the DCM algorithm.

Theorem I.0.3. *The total number of ILP subproblems solved by DCM is at most* $2N + 1$*, where* N *denotes the number of NDPs of the instance solved.*

Proof. From results in [Boland et al., 2017], each ILP subproblem solved by the DCM algorithm is either infeasible or yields a new NDP. Therefore, the total number of ILP subproblems solved by DCM is $N + M$, where M is the number of infeasible ILPs solved by DCM. Furthermore, the ILP subproblems are organized in a binary tree structure, with an internal node corresponding to each feasible ILP and a leaf node corresponding to each infeasible ILP. Thus, by Proposition I.0.2, we have $M \leq N + 1$. As a result, the total number of ILP subproblems solved by DCM is at most $N + (N + 1) = 2N + 1$. ■

Appendix J

Comparison of RVF Algorithm and BLM Variants with MILP Example

The (MO-MILP) illustrated in Figures 3.2 and 3.3, having four elements of $\mathcal{S}_{\min}$, has the LP EF for each formed as follows. The LP frontier shown in red in the figures, which has the right-most upper left corner point, is created by joining its extreme supported points given by

$$F_1 = \{(3\frac{1}{2}, 27), (4, 21), (7, 13), (11, 9), (19, 6), (30, 3\frac{4}{5}), (36, 3\frac{3}{4})\}.$$

The LP frontier shown in blue in the figures, which has the second right-most upper left corner point, is created by joining its extreme supported points given by

$$F_2 = \{(3, 27), (5, 17), (8\frac{1}{2}, 11), (15, 7), (23, 5), (33, 3\frac{1}{2}), (36, 3\frac{1}{4})\}.$$

The LP frontier shown in brown in the figures, which has the third right-most upper left corner point, is created by joining its extreme supported points given by

$$F_3 = \{(1\frac{3}{4}, 27), (16, 6\frac{2}{3}), (36, 3\frac{1}{8})\}.$$

The LP frontier shown in green in the figures, which has the left-most upper left corner point, is created by joining its extreme supported points given by

$$F_4 = \{(0, 27), (36, 2)\}.$$

The four individual LP frontiers are shown in Figures J.1–J.3, with the fourth LP frontier, shown in green, duplicated in each, to provide a common reference in addition to the axes.

There are various methods for constructing a multiobjective optimization model of the form given in (MO-MILP) that corresponds to these LP EFs. One straightforward approach involves introducing four binary variables, one for each of the LP frontiers, to switch the corresponding LP constraints on or off, as well as two continuous variables to represent the two objectives.

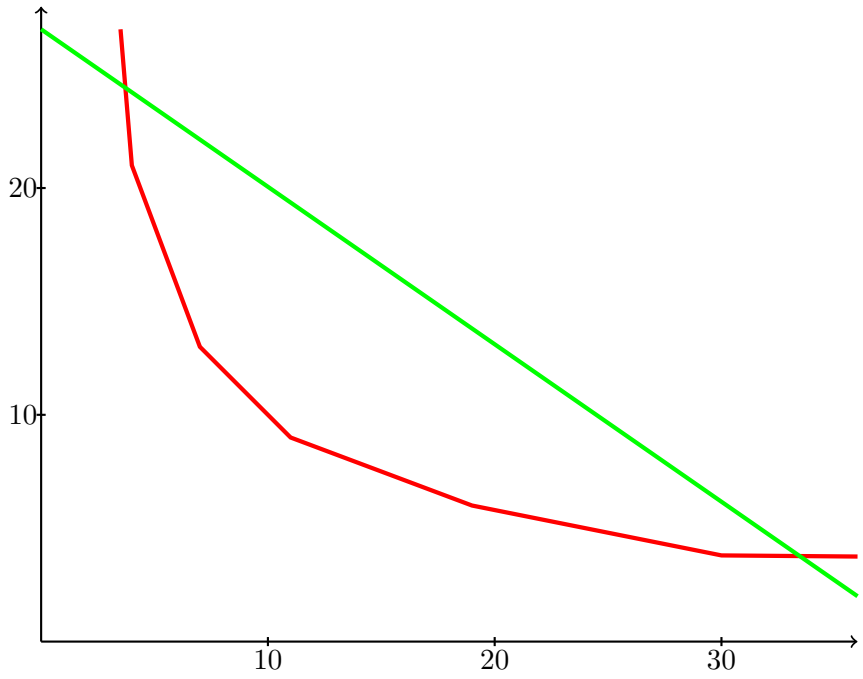


Figure J.1: The complete LP frontier for the parts shown in red in Figure 3.2, together with the complete LP frontier for the part shown in green.

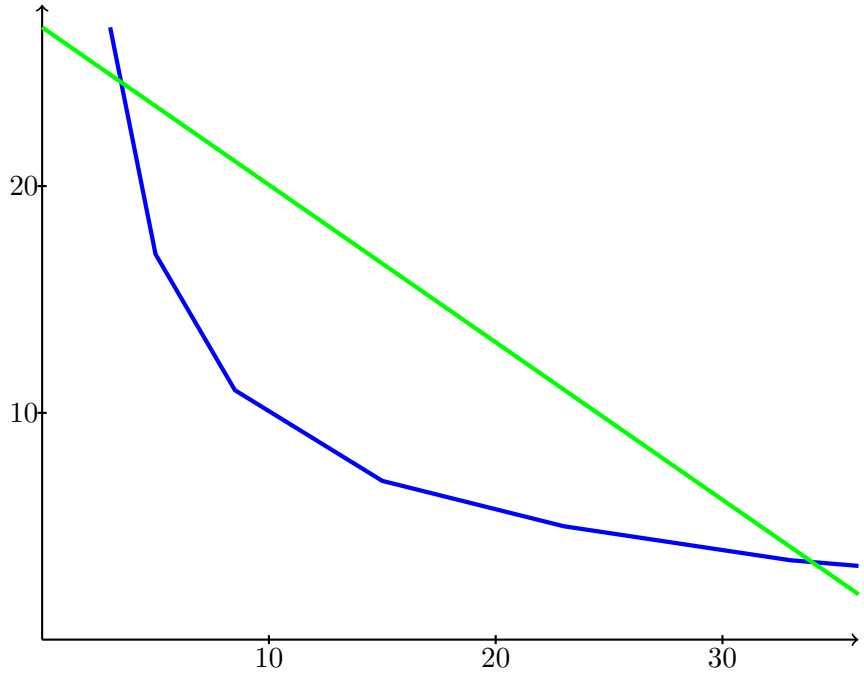


Figure J.2: The complete LP frontier for the parts shown in blue in Figure 3.2, together with the complete LP frontier for the part shown in green.

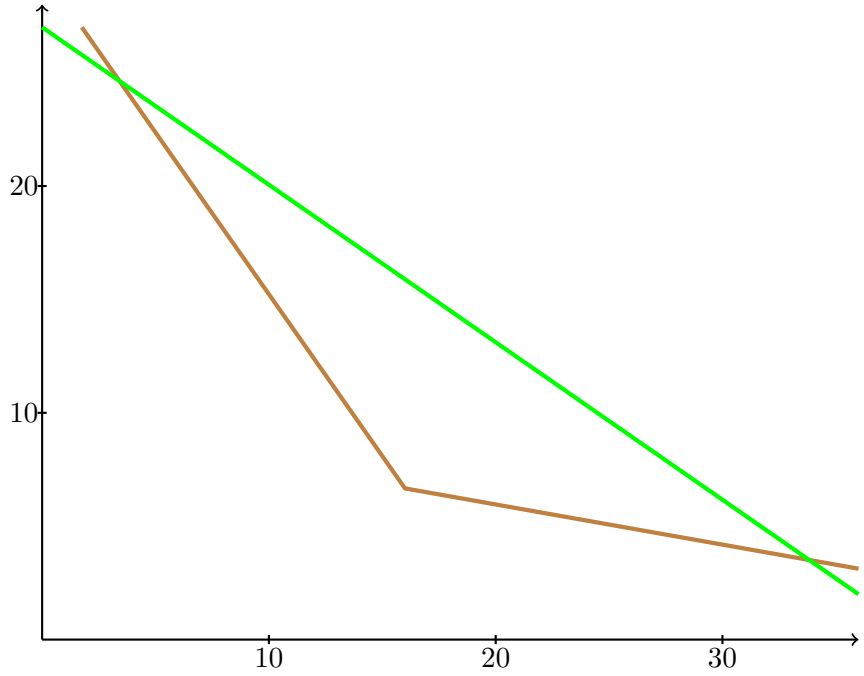


Figure J.3: The complete LP frontier for the parts shown in brown in Figure 3.2, together with the complete LP frontier for the part shown in green.

Appendix K

Tables from the Computational Experiment in Chapter 3

In this appendix, we present the results of our computational experiments conducted for Chapter 3. The results include the running time and number of NDP for PIP instances, as well as the stability region for MILP instances.

Table K.1: Running time and iterations of the nonlinear and linear versions of the RVF Algorithm for SPP

Instance ID	# NDPs	Imp 0 (L)	Imp 0 (NL)	Imp 1		Imp 6	
		Time (sec)	Time (sec)	Time (sec)	Iter.	Time (sec)	Iter.
2mis100_300A	41	77.02	4948.98	85.46	24	234.64	76
2mis100_300B	31	70.32	4377.44	68.49	16	179.5	56
2mis100_300C	22	47.66	1030	44.3	12	128.81	40
2mis100_300D	16	30.05	271.46	30.21	11	84.27	28
2mis101_300A	22	39.77	559.99	41.36	14	116.82	40
2mis101_300B	30	57.01	1799.08	55.82	17	168.61	56
2mis101_300C	26	50.98	3075.32	49.34	14	150.98	48
2mis100_500A	17	82.17	1195.65	77.91	12	215.78	30
2mis100_500B	17	80.49	712.04	79.31	10	217.82	30
2mis100_500C	13	65.88	542.45	65.11	8	163.74	22
2mis100_500D	19	93.96	935.31	91.74	12	255.22	34
2mis101_500A	21	109.82	1259.81	102.23	10	287.21	38
2mis101_500B	20	104.17	1630.26	99.56	13	268.39	36
2mis101_500C	20	96.41	888.96	92.22	11	258.72	35
2mis101_500D	15	69.57	488.29	68.92	9	187.23	26
2spp100_300A	10	26.84	481.86	26.95	8	49.97	16
2spp100_300B	10	27.33	198.71	27.27	6	51.13	16
2spp100_300C	35	89.19	4541.88	85.02	17	222.84	65
2spp100_300D	20	43.38	1236.85	40.41	12	111.63	35
2spp101_300C	20	53.76	4467.65	49.32	8	122.3	36
2spp101_300D	11	26.08	158.64	24.97	6	56.57	18
2spp100_500A	14	89.37	701.63	86.36	9	186.12	24
2spp100_500B	14	87.83	403.28	85.39	9	187.76	24
2spp100_500C	13	76.05	455.56	71.96	7	171.39	22
2spp101_500A	12	73.72	459.42	73.12	8	156.58	20
2spp101_500B	20	125.12	711.84	118.54	9	277.7	35
2spp101_500C	9	54.59	266.76	53.37	5	106.98	14
2spp202_600A	13	261.87	44991.15	239.02	6	540.7	22
2spp202_600B	10	172.3	18800.84	176.14	8	300.64	16
2spp202_600D	10	178.77	15176.92	172.77	6	281	16
2spp203_600A	14	515.83	28273.65	572.59	10	838.56	24
2spp203_600B	11	260.51	37868.86	234.99	7	361.51	18
2spp203_600D	16	538.76	44465.94	468.07	9	883.89	28
2spp202_1000A	8	335.15	1453.47	323.29	6	452.44	12
2spp202_1000D	19	897.19	7203.49	756.08	10	1235.97	30
2spp203_1000A	16	803.71	4877.05	677.19	7	1136.89	28
2spp203_1000B	9	468.55	4261.46	476.16	6	496.82	13
2spp203_1000C	24	905.38	7984.11	901.46	11	1686.1	44
2spp203_1000D	19	622.06	6215.06	669.41	9	1199.97	34

Table K.2: Running time and iterations of the nonlinear and linear versions of the RVF Algorithm for KP

Instance ID	# NDPs	Imp 0 (L)	Imp 0 (NL)	Imp 1		Imp 6	
		Time (sec)	Time (sec)	Time (sec)	Iter.	Time (sec)	Iter.
KP_p-3_n-10_ins-5	25	1.74	6.24	1	10	2.04	46
KP_p-3_n-20_ins-1	35	16.44	37.26	12.03	13	11.18	66
KP_p-3_n-20_ins-2	35	13.65	43.5	7.58	14	7.85	65
KP_p-3_n-20_ins-3	43	13.24	50.01	8.05	13	12.43	82
KP_p-3_n-20_ins-4	28	9.5	24.21	7.49	13	8.36	52
KP_p-3_n-20_ins-5	62	39.48	130.53	38.26	23	47.03	120
KP_p-3_n-20_ins-6	63	34.72	125.66	26.86	19	33.51	122
KP_p-3_n-20_ins-7	31	9.06	25.78	5.23	14	8.29	58
KP_p-3_n-20_ins-8	30	9.59	24.41	6.78	13	7.54	55
KP_p-3_n-20_ins-9	31	7.83	21.12	6.22	12	6.75	56
KP_p-3_n-20_ins-10	22	3.76	11.89	2.6	10	3.3	40
KP_p-3_n-30_ins-1	88	133.72	947.53	117.83	28	138.35	171
KP_p-3_n-30_ins-2	196	746.21	21909.92	1223.26	52	1399.15	387
KP_p-3_n-30_ins-3	118	303.46	4985.57	306.68	34	390.79	231
KP_p-3_n-30_ins-4	63	75.47	829.66	44.19	18	64.27	122
KP_p-3_n-30_ins-5	69	73.45	768.19	69.44	22	74.21	133
KP_p-3_n-30_ins-6	103	184.57	4201.91	159.14	28	245.13	202
KP_p-3_n-30_ins-8	98	158.85	4280.27	147.14	32	161.2	189
KP_p-3_n-30_ins-9	62	56.95	1357.59	39.25	22	47.27	120
KP_p-3_n-30_ins-10	95	130.25	577.41	204.56	32	206.61	184
KP_p-4_n-10_ins-6	19	1.07	4.07	0.7	9	1.35	33
KP_p-4_n-10_ins-7	15	0.78	2.33	0.49	7	0.84	26
KP_p-4_n-10_ins-9	22	1.86	5.63	1.35	9	1.89	40
KP_p-4_n-20_ins-1	60	53.11	155.67	15.64	19	32.47	116
KP_p-4_n-20_ins-2	143	453.67	2152.56	178.27	36	392.93	282
KP_p-4_n-20_ins-5	74	99.27	227.95	51.11	22	59.02	144
KP_p-4_n-20_ins-6	152	386.64	2684.7	315.63	38	533.43	299
KP_p-4_n-20_ins-7	175	932.45	6875.41	412.85	43	443.23	343
KP_p-4_n-20_ins-8	116	169.4	665.42	53.46	31	142.77	226
KP_p-4_n-20_ins-9	77	110.84	327.59	75.56	24	74.44	150
KP_p-4_n-20_ins-10	229	1201.98	6889.73	705.04	53	890.13	453
KP_p-5_n-10_ins-1	20	1.3	4.83	0.62	9	1.5	36
KP_p-5_n-10_ins-2	19	1.29	4.5	0.71	9	1.71	34
KP_p-5_n-10_ins-3	23	2.26	7.1	1.12	10	2.37	41
KP_p-5_n-10_ins-4	28	2.92	9.33	1.34	11	2.62	50
KP_p-5_n-10_ins-8	25	2.31	7.43	1.25	11	2.44	44
KP_p-5_n-20_ins-2	95	177.72	579.83	65.93	29	119.53	184
KP_p-5_n-20_ins-3	76	120.74	307.19	37.34	20	77.75	145
KP_p-5_n-20_ins-4	89	131.81	409.17	55.05	30	90.53	167
KP_p-5_n-20_ins-5	110	301.46	1058.1	133.85	30	289.03	215
KP_p-5_n-20_ins-6	87	148.4	497.95	51.27	24	140.38	169
KP_p-5_n-20_ins-7	61	62.73	210.64	24.54	17	48.82	115

Table K.3: Running time in seconds for all the SPP using the RVF Algorithm with Implementation 2

Instance ID	# NDPs	Time (sec)	Instance ID	# NDPs	Time (sec)	Instance ID	# NDPs	Time (sec)
2mis100_300A	41	79.36	2mis201_600E	5	67.43	2spp200_1000C	12	825.48
2mis100_300B	31	65.57	2mis201_600F	6	58.87	2spp200_1000D	31	5102.11
2mis100_300C	22	48.37	2mis201_1000A	45	11122.64	2spp200_1000E	5	586.37
2mis100_300D	16	31.68	2mis201_1000B	64	11618.36	2spp200_1000F	5	1387.5
2mis100_300E	5	11.92	2mis201_1000C	38	2677.34	2spp201_600A	54	3021.31
2mis100_300F	5	12.14	2mis201_1000D	97	5270.91	2spp201_600B	52	6290.23
2mis100_500A	17	74.86	2mis201_1000E	7	272.1	2spp201_600C	26	511.31
2mis100_500B	17	77.02	2mis201_1000F	13	616.93	2spp201_600D	31	2063.83
2mis100_500C	13	64.76	2spp100_300A	10	23.23	2spp201_600E	10	258.66
2mis100_500D	19	89.19	2spp100_300B	10	22.73	2spp201_600F	6	258.65
2mis100_500E	2	17.68	2spp100_300C	35	79.76	2spp201_1000A	30	7648.44
2mis100_500F	4	25.46	2spp100_300D	20	42	2spp201_1000B	26	3777.4
2mis101_300A	22	38.38	2spp100_300E	3	9.39	2spp201_1000C	25	5484.29
2mis101_300B	30	55.11	2spp100_300F	6	14.69	2spp201_1000D	21	3187.28
2mis101_300C	26	54.69	2spp100_500A	14	73.57	2spp201_1000E	6	954.29
2mis101_300D	41	83.28	2spp100_500B	14	74.23	2spp201_1000F	7	876.39
2mis101_300E	3	9.05	2spp100_500C	13	68.41	2spp202_600A	13	231.61
2mis101_300F	4	10.32	2spp100_500D	2	18.89	2spp202_600B	10	170.88
2mis101_500A	21	99.3	2spp100_500E	4	27.08	2spp202_600C	17	679.6
2mis101_500B	20	93.84	2spp100_500F	2	19.01	2spp202_600D	10	238.15
2mis101_500C	20	93.79	2spp101_300A	9	19.85	2spp202_600E	4	73.81
2mis101_500D	15	68.66	2spp101_300B	5	13.09	2spp202_600F	4	140.15
2mis101_500E	8	40.24	2spp101_300C	20	52.17	2spp202_1000A	8	251.15
2mis101_500F	5	28.64	2spp101_300D	11	24.74	2spp202_1000B	7	250.84
2mis200_600A	77	4765.21	2spp101_300E	1	5.93	2spp202_1000C	17	730.27
2mis200_600B	55	1075.55	2spp101_300F	4	11.56	2spp202_1000D	19	664.57
2mis200_600C	126	2560.58	2spp101_500A	12	64.58	2spp202_1000E	3	175.02
2mis200_600D	30	235.81	2spp101_500B	20	105.72	2spp202_1000F	3	164.33
2mis200_600E	7	69.66	2spp101_500C	9	49.26	2spp203_600A	14	442.53
2mis200_600F	3	37.01	2spp101_500D	9	48.38	2spp203_600B	11	182.71
2mis200_1000A	43	6350.92	2spp101_500E	3	24.77	2spp203_600C	29	1695.01
2mis200_1000B	63	5592.98	2spp101_500F	3	24.43	2spp203_600D	16	363.41
2mis200_1000C	69	4221.94	2spp200_600A	38	3520.39	2spp203_600E	3	86.54
2mis200_1000D	41	801.43	2spp200_600B	37	4122.95	2spp203_600F	5	171.54
2mis200_1000E	1	128.44	2spp200_600C	55	1179.91	2spp203_1000A	16	504.93
2mis200_1000F	3	160.45	2spp200_600D	31	252.1	2spp203_1000B	9	288.5
2mis201_600A	96	3574.88	2spp200_600E	6	145.35	2spp203_1000C	24	688.48
2mis201_600B	58	1759.57	2spp200_600F	9	126.82	2spp203_1000D	19	485.22
2mis201_600C	73	3158.88	2spp200_1000A	25	3955.55	2spp203_1000E	3	158.21
2mis201_600D	44	477.67	2spp200_1000B	20	5353.44	2spp203_1000F	4	133.71

Table K.4: Running time in seconds for all the KP using the RVF Algorithm with Implementation 2

Instance ID	# NDPs	Time (sec)	Instance ID	# NDPs	Time (sec)	Instance ID	# NDPs	Time (sec)	Instance ID	# NDPs	Time (sec)
p-3_n-10_ins-1	7	0.1	p-3_n-50_ins-1	507	36108.38	p-3_n-90_ins-1	1751	-	p-4_n-30_ins-1	563	13815.89
p-3_n-10_ins-2	19	0.34	p-3_n-50_ins-2	202	1033.76	p-3_n-90_ins-2	3262	-	p-4_n-30_ins-2	535	11716.02
p-3_n-10_ins-3	8	0.19	p-3_n-50_ins-3	530	21458.31	p-3_n-90_ins-3	1195	-	p-4_n-30_ins-3	226	1049.73
p-3_n-10_ins-4	5	0.09	p-3_n-50_ins-4	402	15429.7	p-3_n-90_ins-4	3354	-	p-4_n-30_ins-4	517	6932.83
p-3_n-10_ins-5	25	0.65	p-3_n-50_ins-5	557	37368.52	p-3_n-90_ins-5	2228	-	p-4_n-30_ins-5	281	1907.07
p-3_n-10_ins-6	7	0.14	p-3_n-50_ins-6	803	-	p-3_n-90_ins-6	3144	-	p-4_n-30_ins-6	191	827.88
p-3_n-10_ins-7	5	0.11	p-3_n-50_ins-7	519	35262.69	p-3_n-90_ins-7	4651	-	p-4_n-30_ins-7	480	7564.79
p-3_n-10_ins-8	7	0.19	p-3_n-50_ins-8	178	1254.66	p-3_n-90_ins-8	2778	-	p-4_n-30_ins-8	262	2474.99
p-3_n-10_ins-9	9	0.24	p-3_n-50_ins-9	361	7422.81	p-3_n-90_ins-9	5568	-	p-4_n-30_ins-9	186	744.7
p-3_n-10_ins-10	6	0.16	p-3_n-50_ins-10	383	10523.43	p-3_n-90_ins-10	3147	-	p-4_n-30_ins-10	735	33833.59
p-3_n-20_ins-1	35	6.99	p-3_n-60_ins-1	743	-	p-3_n-100_ins-1	6779	-	p-4_n-40_ins-1	901	-
p-3_n-20_ins-2	35	6.05	p-3_n-60_ins-2	837	-	p-3_n-100_ins-2	3696	-	p-4_n-40_ins-2	5018	-
p-3_n-20_ins-3	43	7.79	p-3_n-60_ins-3	550	89909.82	p-3_n-100_ins-3	10179	-	p-4_n-40_ins-3	508	41908.22
p-3_n-20_ins-4	28	6.13	p-3_n-60_ins-4	645	84833.58	p-3_n-90_ins-4	9882	-	p-4_n-40_ins-4	1248	-
p-3_n-20_ins-5	62	23.73	p-3_n-60_ins-5	741	-	p-3_n-100_ins-5	6679	-	p-4_n-40_ins-5	2351	-
p-3_n-20_ins-6	63	19.63	p-3_n-60_ins-6	1142	-	p-3_n-100_ins-6	3875	-	p-4_n-40_ins-6	1920	-
p-3_n-20_ins-7	31	4.05	p-3_n-60_ins-7	897	-	p-3_n-100_ins-7	2193	-	p-4_n-40_ins-7	1435	-
p-3_n-20_ins-8	30	6.15	p-3_n-60_ins-8	1116	-	p-3_n-100_ins-8	6444	-	p-4_n-40_ins-8	741	67928.97
p-3_n-20_ins-9	31	5.53	p-3_n-60_ins-9	1283	-	p-3_n-100_ins-9	5573	-	p-4_n-40_ins-9	1903	-
p-3_n-20_ins-10	22	2.5	p-3_n-60_ins-10	1217	-	p-3_n-100_ins-10	3190	-	p-4_n-40_ins-10	555	38427.22
p-3_n-30_ins-1	88	67.36	p-3_n-70_ins-1	1019	-	p-4_n-10_ins-1	10	0.19	p-5_n-10_ins-1	20	0.53
p-3_n-30_ins-2	196	768.69	p-3_n-70_ins-2	1236	-	p-4_n-10_ins-2	13	0.27	p-5_n-10_ins-2	19	0.53
p-3_n-30_ins-3	118	243.5	p-3_n-70_ins-3	2392	-	p-4_n-10_ins-3	7	0.11	p-5_n-10_ins-3	23	0.7
p-3_n-30_ins-4	63	43.73	p-3_n-70_ins-4	1113	-	p-4_n-10_ins-4	6	0.12	p-5_n-10_ins-4	28	1
p-3_n-30_ins-5	69	49.46	p-3_n-70_ins-5	2632	-	p-4_n-10_ins-5	9	0.16	p-5_n-10_ins-5	13	0.33
p-3_n-30_ins-6	103	157	p-3_n-70_ins-6	1443	-	p-4_n-10_ins-6	19	0.45	p-5_n-10_ins-6	9	0.2
p-3_n-30_ins-7	266	1149.11	p-3_n-70_ins-7	1142	-	p-4_n-10_ins-7	15	0.34	p-5_n-10_ins-7	9	0.16
p-3_n-30_ins-8	98	89.34	p-3_n-70_ins-8	1935	-	p-4_n-10_ins-8	5	0.12	p-5_n-10_ins-8	25	0.89
p-3_n-30_ins-9	62	31.03	p-3_n-70_ins-9	2161	-	p-4_n-10_ins-9	22	0.9	p-5_n-10_ins-9	6	0.11
p-3_n-30_ins-10	95	108.67	p-3_n-70_ins-10	1361	-	p-4_n-10_ins-10	10	0.21	p-5_n-10_ins-10	10	0.2
p-3_n-40_ins-1	486	14262.07	p-3_n-80_ins-1	1383	-	p-4_n-20_ins-1	60	19.25	p-5_n-20_ins-1	220	631.7
p-3_n-40_ins-2	252	1362.77	p-3_n-80_ins-2	2357	-	p-4_n-20_ins-2	143	224.06	p-5_n-20_ins-2	95	67.98
p-3_n-40_ins-3	541	16067.87	p-3_n-80_ins-3	2515	-	p-4_n-20_ins-3	325	1119.79	p-5_n-20_ins-3	76	43.73
p-3_n-40_ins-4	256	3156.48	p-3_n-80_ins-4	1697	-	p-4_n-20_ins-4	17	2.24	p-5_n-20_ins-4	89	46.32
p-3_n-40_ins-5	385	8424.43	p-3_n-80_ins-5	1790	-	p-4_n-20_ins-5	74	37.27	p-5_n-20_ins-5	110	200.92
p-3_n-40_ins-6	222	1748.12	p-3_n-80_ins-6	3282	-	p-4_n-20_ins-6	152	301.15	p-5_n-20_ins-6	87	75.08
p-3_n-40_ins-7	238	1524.88	p-3_n-80_ins-7	3250	-	p-4_n-20_ins-7	175	277.48	p-5_n-20_ins-7	61	32.7
p-3_n-40_ins-8	136	294.78	p-3_n-80_ins-8	3315	-	p-4_n-20_ins-8	116	85.42	p-5_n-20_ins-8	211	388.02
p-3_n-40_ins-9	337	4511.99	p-3_n-80_ins-9	1738	-	p-4_n-20_ins-9	77	36.77	p-5_n-20_ins-9	237	408.52
p-3_n-40_ins-10	259	2266.23	p-3_n-80_ins-10	1631	-	p-4_n-20_ins-10	229	433.52	p-5_n-20_ins-10	426	1902.99

Table K.5: Running time in seconds for all the randomly generated IPs using the RVF Algorithm with Implementation 2

Instance ID	Time (sec)	# NDPs
IP_obj_10_var_10_nzr_3_u_2	392.98	221
IP_obj_10_var_10_nzr_4_u_2	588.12	252
IP_obj_10_var_10_nzr_5_u_2	569.54	269
IP_obj_10_var_10_nzr_6_u_2	732.44	269
IP_obj_10_var_10_nzr_7_u_2	315.47	203
IP_obj_15_var_10_nzr_3_u_2	963.25	260
IP_obj_15_var_10_nzr_4_u_2	1167.28	267
IP_obj_15_var_10_nzr_5_u_2	1388.69	269
IP_obj_15_var_10_nzr_6_u_2	1221.39	269
IP_obj_15_var_10_nzr_7_u_2	1265.05	267
IP_obj_20_var_10_nzr_3_u_2	1690.99	265
IP_obj_20_var_10_nzr_5_u_2	1605.53	266
IP_obj_20_var_10_nzr_6_u_2	1870.2	274
IP_obj_20_var_10_nzr_7_u_2	1232.5	253
IP_obj_25_var_10_nzr_3_u_2	2605	268
IP_obj_25_var_10_nzr_4_u_1	510.33	176
IP_obj_25_var_10_nzr_4_u_2	2525.97	264
IP_obj_25_var_10_nzr_5_u_1	567.7	176
IP_obj_25_var_10_nzr_5_u_2	2331.68	250
IP_obj_25_var_10_nzr_6_u_1	578.82	176
IP_obj_25_var_10_nzr_6_u_2	2534.48	266
IP_obj_25_var_10_nzr_7_u_2	2205.97	264
IP_obj_30_var_10_nzr_3_u_2	2613.55	252
IP_obj_30_var_10_nzr_4_u_2	2879.96	270
IP_obj_30_var_10_nzr_5_u_1	580.98	176
IP_obj_30_var_10_nzr_5_u_2	3190.05	264
IP_obj_30_var_10_nzr_6_u_1	749.24	176
IP_obj_30_var_10_nzr_6_u_2	3315.76	267
IP_obj_30_var_10_nzr_7_u_1	668.63	176
IP_obj_30_var_10_nzr_7_u_2	2436.82	249

Table K.6: Running times (in seconds) for all KP instances solved with the RVF Algorithm under Implementation 3, using different numbers of feasible solutions

Instance ID	5		4		3		2		1
	Time (sec)	Iter.	Time (sec)	Iter.	Time (sec)	Iter.	Time (sec)	Iter.	Time (sec)
KP_p-3_n-10_ins-5	0.78	9	1.02	11	0.98	12	0.93	14	0.78
KP_p-3_n-20_ins-1	7.41	13	8.87	15	8.93	17	9.61	20	8.44
KP_p-3_n-20_ins-2	7.65	15	6.31	14	5.49	16	5.25	20	5.37
KP_p-3_n-20_ins-3	9.51	17	8	16	7.49	18	7.47	26	6.86
KP_p-3_n-20_ins-4	6.8	12	4.49	11	4.13	11	4.82	17	5.46
KP_p-3_n-20_ins-5	44.32	23	42.99	23	35.4	25	27.68	34	23.49
KP_p-3_n-20_ins-6	30.33	21	28.15	23	28.96	26	19.97	34	16.09
KP_p-3_n-20_ins-7	4.4	11	5.09	13	3.76	13	5.23	19	4.57
KP_p-3_n-20_ins-8	6.13	14	5.95	14	5.32	16	5.92	18	6.21
KP_p-3_n-20_ins-9	8.35	13	6.03	13	10.54	17	4.41	19	4.8
KP_p-3_n-20_ins-10	2.09	11	2.65	11	2.85	11	1.86	14	2.32
KP_p-3_n-30_ins-1	76.05	27	99.76	30	116.61	37	101.65	46	113.36
KP_p-3_n-30_ins-2	887.92	54	691.22	58	871.12	74	861.11	106	962.98
KP_p-3_n-30_ins-3	211.03	31	186.21	35	180.13	45	197.07	63	253.96
KP_p-3_n-30_ins-4	36.93	20	41.02	24	34.76	27	39.58	35	49.16
KP_p-3_n-30_ins-5	49.15	21	64.97	29	43.94	30	41.75	37	66.61
KP_p-3_n-30_ins-6	121	31	181.14	35	139.97	42	169.15	58	171.47
KP_p-3_n-30_ins-8	123.85	28	94.48	32	105.67	39	110.94	52	141.69
KP_p-3_n-30_ins-9	32.12	22	37.6	22	27.96	27	38.11	34	41.05
KP_p-3_n-30_ins-10	97.39	26	139.02	31	117.26	37	105.05	51	91.25
KP_p-4_n-10_ins-6	0.9	10	1.1	9	0.61	8	1.76	11	0.6
KP_p-4_n-10_ins-7	0.54	8	0.54	8	0.7	7	0.41	9	0.4
KP_p-4_n-10_ins-9	0.79	7	0.83	8	2.06	10	1.76	13	0.89
KP_p-4_n-20_ins-1	14.26	18	15.55	20	14.96	23	17.13	35	24.17
KP_p-4_n-20_ins-2	156.17	38	187.98	48	162.87	54	181.61	82	228.69
KP_p-4_n-20_ins-5	39.25	23	41.64	24	37.94	31	30.61	43	42.32
KP_p-4_n-20_ins-6	266.13	41	226.26	47	188.29	57	174.24	79	311.44
KP_p-4_n-20_ins-7	293.77	45	288.94	53	281.41	70	271.53	98	424.05
KP_p-4_n-20_ins-8	63.1	34	59.21	39	58.52	43	70.76	64	83.41
KP_p-4_n-20_ins-9	28.54	21	38.3	28	32.18	30	41.9	45	44.4
KP_p-4_n-20_ins-10	560.75	59	450.84	72	447.12	89	512.04	135	467.77
KP_p-5_n-10_ins-1	0.61	8	0.67	9	0.95	12	0.76	13	0.62
KP_p-5_n-10_ins-2	0.69	8	0.67	8	0.72	9	0.63	10	0.65
KP_p-5_n-10_ins-3	1.09	10	1.53	14	1.47	12	1.6	14	0.82
KP_p-5_n-10_ins-4	1.19	11	1.12	11	1.22	13	1.7	17	1.12
KP_p-5_n-10_ins-8	1.12	11	0.94	10	1.17	12	1.02	15	1.12
KP_p-5_n-20_ins-2	53.87	28	46.38	31	49.44	37	57.75	47	89.96
KP_p-5_n-20_ins-3	29.79	22	32.24	25	36.58	32	40.34	42	58.45
KP_p-5_n-20_ins-4	30.84	25	36.79	31	36.13	37	41.29	47	63.95
KP_p-5_n-20_ins-5	100.71	30	84.25	33	110.65	41	96.69	55	162.32
KP_p-5_n-20_ins-6	41.93	26	54.05	30	43.12	33	57.87	45	84.7
KP_p-5_n-20_ins-7	25.27	21	21.04	22	24.43	27	23.05	32	37.74

Table K.7: Running times (in seconds) for all SPP instances solved with the RVF Algorithm under Implementation 3, using different numbers of feasible solutions

Instance ID	5		4		3		2		1
	Time (sec)	Iter.	Time (sec)	Iter.	Time (sec)	Iter.	Time (sec)	Iter.	Time (sec)
2mis100_300A	91.39	16	86.93	17	93.62	20	95.3	24	114.44
2mis100_300B	75.1	15	73.41	16	76.23	18	80.3	19	88.78
2mis100_300C	48.06	13	45.55	12	46.93	14	48.09	15	52.62
2mis100_300D	32.44	12	31.39	12	30.63	10	31.75	12	30.98
2mis101_300A	43.24	15	43.56	15	44.52	17	44.4	17	46.33
2mis101_300B	67.69	17	67.56	17	65.75	17	63.81	18	74.85
2mis101_300C	51.13	18	52.17	18	53.92	19	55.09	17	63.75
2mis100_500A	78.97	11	78.45	11	79.09	12	80.74	14	82.64
2mis100_500B	81.39	10	79.09	9	80.27	11	81.53	14	79.04
2mis100_500C	62.83	6	66.51	8	66.07	9	67.08	10	67.33
2mis100_500D	92.82	10	97.5	13	93.79	12	93.5	13	98.51
2mis101_500A	105.26	9	106.92	10	108.17	11	110.25	14	114.65
2mis101_500B	99.99	11	103.64	15	104.52	14	104	15	111.12
2mis101_500C	94.15	12	95.45	12	95.3	13	95.18	14	95.57
2mis101_500D	72.76	10	69.74	9	69.98	9	70.81	11	74.4
2spp100_300A	27.25	8	27.05	7	27.06	8	26.38	7	27.86
2spp100_300B	28.11	7	29.1	8	28.7	9	29.11	9	25.01
2spp100_300C	82.75	16	91.36	18	83.49	19	82.22	23	90.76
2spp100_300D	41.53	11	43.39	13	42.99	13	41.31	13	42.56
2spp101_300C	52.72	10	57.77	12	56.24	13	52.17	14	60.69
2spp101_300D	26.25	8	26.01	8	26.36	8	24.74	8	26.67
2spp100_500A	85.16	8	86.43	8	89.19	9	82.97	10	87.2
2spp100_500B	87.44	10	87.03	10	86.34	10	81.58	10	87.07
2spp100_500C	76.29	8	74.44	7	76.85	9	79.38	11	80.54
2spp101_500A	74.77	8	74.11	8	72.85	8	72.6	9	70.43
2spp101_500B	118.06	9	122.56	11	116.69	12	116.76	15	117.59
2spp101_500C	55.53	6	54.13	6	54.52	6	54.69	7	56.71
2spp202_600A	409.87	8	372.3	9	443.44	9	345.16	10	451.92
2spp202_600B	243.15	8	235.86	8	232.33	7	216.04	8	252.6
2spp202_600D	169.81	5	182.74	6	216.94	7	175.45	7	259.22
2spp203_600A	721.32	8	886.52	9	1076.27	11	847.53	11	1148.35
2spp203_600B	318.86	7	318.35	7	385.72	7	435.51	8	390.29
2spp203_600D	602.29	10	575.37	10	689.13	11	555.56	12	620.32
2spp202_1000A	384.33	5	395.75	6	398.28	6	363.54	6	319.32
2spp202_1000D	822.42	10	782.24	10	862.11	14	795.44	13	844.18
2spp203_1000A	745.83	8	831.98	8	809.62	9	938.87	13	1021.62
2spp203_1000B	473.13	6	476.18	6	449.97	6	422.35	8	435.87
2spp203_1000C	875.8	10	886.43	12	925.97	14	957.12	17	984.16
2spp203_1000D	697.91	10	634.63	10	613.81	11	583.31	11	691.08

Table K.8: Running times (in seconds) for all KP instances solved with the RVF Algorithm under Implementation 4, using different cut ratios - Part I

Instance ID	0.01		0.05		0.1		0.2		0.3		0.4	
	Time (sec)	Iter.	Time (sec)	Iter.	Time (sec)	Iter.	Time (sec)	Iter.	Time (sec)	Iter.	Time (sec)	Iter.
KP_p-3_n-10_ins-5	1.11	19	1.12	19	1.23	20	1.3	20	1.24	19	1.5	21
KP_p-3_n-20_ins-1	7.65	21	7.44	21	7.8	21	10.57	27	9.14	25	11.63	25
KP_p-3_n-20_ins-2	7.79	27	8.18	27	7.9	27	8.73	28	9.37	26	6.54	21
KP_p-3_n-20_ins-3	7.4	22	7.37	22	9.95	24	8.16	22	7.3	28	9.25	25
KP_p-3_n-20_ins-4	6.23	23	6.03	23	6.63	23	6.57	24	5.37	20	5.33	20
KP_p-3_n-20_ins-5	51.48	47	35.65	39	35.27	42	35.99	41	62.49	45	50.2	39
KP_p-3_n-20_ins-6	30.38	35	32.26	40	42.32	42	43.53	43	23.22	34	29.82	34
KP_p-3_n-20_ins-7	6.98	27	6.9	27	7.42	27	6.96	29	7.57	26	7.16	24
KP_p-3_n-20_ins-8	5.68	21	5.55	21	6.21	21	5.79	21	6.46	22	4.29	18
KP_p-3_n-20_ins-9	7.12	26	6.19	26	6.97	26	6.25	22	6.73	21	6.93	21
KP_p-3_n-20_ins-10	3.69	23	3.69	23	3.71	23	2.68	19	2.61	17	3.42	20
KP_p-3_n-30_ins-1	111.5	51	100.86	51	103.42	54	111.67	49	108.81	51	141.37	52
KP_p-3_n-30_ins-2	946.39	94	759.35	94	804.21	87	777.63	84	961.69	94	984.91	96
KP_p-3_n-30_ins-3	262.08	66	216.05	64	322.6	61	259	61	296.92	64	293.93	53
KP_p-3_n-30_ins-4	36.78	36	70.74	41	65.04	41	50.54	37	51.27	34	47.07	40
KP_p-3_n-30_ins-5	55.54	45	70.8	48	56.71	42	54.84	42	85.78	48	57.74	40
KP_p-3_n-30_ins-6	126.09	51	129.92	52	147.35	52	124.7	54	138.74	63	210.73	53
KP_p-3_n-30_ins-8	148.79	54	125.53	56	100.17	56	129.04	52	139.36	57	131.96	50
KP_p-3_n-30_ins-9	35.65	40	35.84	40	37.19	41	43.39	43	33.95	38	33.11	42
KP_p-3_n-30_ins-10	159.53	54	127.86	53	183.97	51	146.41	57	169.86	58	163.2	46
KP_p-4_n-10_ins-6	1.07	20	1.06	20	1.06	20	1.01	19	1.03	19	1.13	17
KP_p-4_n-10_ins-7	0.53	11	0.54	11	0.55	11	0.56	11	0.59	11	0.94	18
KP_p-4_n-10_ins-9	1.55	19	1.58	19	1.59	19	2.21	23	2.46	24	1.59	20
KP_p-4_n-20_ins-1	18.95	35	19.18	35	23.07	40	23.1	37	22.65	37	23.54	35
KP_p-4_n-20_ins-2	188.39	68	189	68	208.28	76	219.62	69	204.44	68	240.52	69
KP_p-4_n-20_ins-5	44.09	41	44	41	45.14	43	50.98	46	49.31	41	62.39	41
KP_p-4_n-20_ins-6	242.98	76	259.08	82	187.25	65	330.63	71	340.91	75	361.09	75
KP_p-4_n-20_ins-7	324.2	75	360.22	76	336.28	77	385.3	78	482.43	80	486.07	85
KP_p-4_n-20_ins-8	62.54	62	62.28	63	69.64	64	68.9	56	84.66	58	105.17	64
KP_p-4_n-20_ins-9	47.9	38	48.26	38	48.61	38	50.02	38	50.44	37	58.04	37
KP_p-4_n-20_ins-10	517.96	88	660.59	97	509.85	93	844.6	99	805.66	93	974.22	102
KP_p-5_n-10_ins-1	0.87	15	0.86	15	0.86	15	0.91	15	0.96	15	1	15
KP_p-5_n-10_ins-2	1.06	18	1.06	18	1.07	18	1.11	18	1.16	18	1.2	18
KP_p-5_n-10_ins-3	1.62	19	1.65	19	1.65	19	1.43	17	1.66	18	2.06	21
KP_p-5_n-10_ins-4	1.85	18	1.82	18	1.88	18	2.24	21	2.64	22	2.94	21
KP_p-5_n-10_ins-8	1.45	17	1.46	17	1.48	17	1.54	17	1.65	17	1.8	19
KP_p-5_n-20_ins-2	69.35	51	54.52	45	61.27	45	70.99	53	74.27	54	96.35	58
KP_p-5_n-20_ins-3	36.37	34	36.72	34	43.97	41	40.97	40	52.76	41	62.4	53
KP_p-5_n-20_ins-4	52.2	51	52.39	51	59.49	52	61.38	56	67.88	55	78.87	53
KP_p-5_n-20_ins-5	101.7	51	102.76	48	104.33	49	110.26	55	110.28	52	122.77	47
KP_p-5_n-20_ins-6	50.01	51	61.47	50	49.37	49	67.74	54	67.63	46	64.8	46
KP_p-5_n-20_ins-7	27.51	41	27.67	41	31.09	45	20.8	31	36.9	37	46.3	43

Table K.9: Running times (in seconds) for all KP instances solved with the RVF Algorithm under Implementation 4, using different cut ratios - Part II

Instance ID	0.5		0.6		0.7		0.8		0.9	
	Time (sec)	Iter.	Time (sec)	Iter.	Time (sec)	Iter.	Time (sec)	Iter.	Time (sec)	Iter.
KP_p-3_n-10_ins-5	1.61	21	1.5	18	1.55	18	1.6	19	1.88	21
KP_p-3_n-20_ins-1	15.41	31	10.54	25	11.78	24	12.52	25	11.38	25
KP_p-3_n-20_ins-2	9.01	25	10.24	29	10.39	29	8.45	25	12.64	33
KP_p-3_n-20_ins-3	7.26	20	8.2	23	9.74	23	13.26	27	10.91	24
KP_p-3_n-20_ins-4	5.97	20	6.4	20	8.22	26	4.59	16	6.5	21
KP_p-3_n-20_ins-5	50.77	39	48.75	36	34.8	32	45.44	34	53.39	38
KP_p-3_n-20_ins-6	39.14	38	34.07	31	41.07	35	44.32	35	50.18	43
KP_p-3_n-20_ins-7	5.07	19	5.4	19	6.95	22	6.97	22	7.46	22
KP_p-3_n-20_ins-8	6.32	19	7.44	22	9.21	24	7.43	25	6.08	19
KP_p-3_n-20_ins-9	7.56	21	7.5	21	6.52	19	8.31	22	7.37	19
KP_p-3_n-20_ins-10	3.47	20	3.16	18	3.05	18	3.12	18	4.78	23
KP_p-3_n-30_ins-1	147.69	57	104.77	44	118.27	49	155.62	47	170.97	54
KP_p-3_n-30_ins-2	1039.66	85	1316.43	89	1475.47	97	1284.18	93	1496.25	100
KP_p-3_n-30_ins-3	392.89	57	306.99	54	354.4	60	283.41	54	335.94	55
KP_p-3_n-30_ins-4	63.45	38	43.75	32	43.86	34	47.07	36	52.19	35
KP_p-3_n-30_ins-5	83.41	47	73.32	43	60.65	40	84.33	46	94.58	45
KP_p-3_n-30_ins-6	252.7	57	188.63	52	201.29	57	212.58	57	239.89	53
KP_p-3_n-30_ins-8	159.42	49	109.52	44	143.56	53	159.25	56	196.6	59
KP_p-3_n-30_ins-9	32.38	33	42.61	42	52.68	43	51.02	38	43	35
KP_p-3_n-30_ins-10	159.51	53	201.7	56	200.97	57	162.38	52	162.95	52
KP_p-4_n-10_ins-6	1.11	16	1.14	16	1.29	17	1.34	17	1.31	16
KP_p-4_n-10_ins-7	0.74	13	0.76	13	0.99	18	0.7	12	0.73	12
KP_p-4_n-10_ins-9	1.72	17	1.75	17	1.88	17	1.74	16	2.63	17
KP_p-4_n-20_ins-1	24.48	34	26.61	35	32.49	40	34.97	40	36.46	40
KP_p-4_n-20_ins-2	343.98	73	334.96	72	316.32	70	410.55	73	370.58	69
KP_p-4_n-20_ins-5	61.89	43	50.6	37	51.66	40	75.86	43	53.07	37
KP_p-4_n-20_ins-6	464.14	67	358.37	70	578.05	80	461.05	72	647.54	77
KP_p-4_n-20_ins-7	479.01	78	527.85	81	710.65	82	613.9	81	643.59	76
KP_p-4_n-20_ins-8	137.89	70	117.95	65	142.54	64	170.39	67	155.37	62
KP_p-4_n-20_ins-9	66.51	41	61.23	42	64.28	42	67.12	37	71.8	39
KP_p-4_n-20_ins-10	1106.57	96	1161.94	88	1381.53	108	1438.84	92	1791.85	103
KP_p-5_n-10_ins-1	1.04	15	1.07	15	1.11	15	1.15	15	1.19	15
KP_p-5_n-10_ins-2	1.28	18	1.32	18	1.36	18	1.43	18	1.48	18
KP_p-5_n-10_ins-3	1.92	20	2.16	22	1.74	18	2.55	22	1.98	17
KP_p-5_n-10_ins-4	3.01	21	3.14	21	3.3	21	3.45	21	3.63	21
KP_p-5_n-10_ins-8	2.14	20	1.83	17	2.32	20	2.42	18	2.62	20
KP_p-5_n-20_ins-2	84.23	50	72.32	40	117.47	54	96.9	50	108.35	51
KP_p-5_n-20_ins-3	68.22	43	61.84	45	61.51	43	61.76	37	65.18	39
KP_p-5_n-20_ins-4	87.4	59	83.51	53	90.03	54	101.63	60	94.49	55
KP_p-5_n-20_ins-5	113.17	48	198.12	60	178.56	54	195.71	53	164.45	52
KP_p-5_n-20_ins-6	77.2	47	84.65	48	101.26	49	107.33	48	103.48	57
KP_p-5_n-20_ins-7	31.5	38	34.59	38	34.4	35	41.48	39	45.48	39

Table K.10: Running times (in seconds) for all SPP instances solved with the RVF Algorithm under Implementation 4, using different cut ratios - Part I

Instance ID	0.01		0.05		0.1		0.2		0.3		0.4	
	Time (sec)	Iter.	Time (sec)	Iter.	Time (sec)	Iter.	Time (sec)	Iter.	Time (sec)	Iter.	Time (sec)	Iter.
2mis100_300A	121.89	34	122.95	35	121.05	33	116.23	30	119.42	31	118.09	28
2mis100_300B	106.77	27	106.19	27	102.41	26	109.47	26	115.71	30	104.86	24
2mis100_300C	66.88	17	66.83	17	66.23	17	65.53	16	70.25	19	71.34	19
2mis100_300D	49.79	16	50.58	16	49.57	16	49.12	15	49.37	16	50.59	16
2mis101_300A	58.8	16	58.67	16	57.84	16	58.29	16	65.47	20	64.87	19
2mis101_300B	78.62	20	78.81	20	78.87	20	86.16	24	85.15	24	87.64	24
2mis101_300C	77.5	21	76.16	21	77.3	21	76.4	20	78.49	20	77.34	20
2mis100_500A	124.09	15	124.79	15	124.85	15	129.67	16	125.07	15	118.58	14
2mis100_500B	116.38	13	116.16	13	117.15	13	127.03	15	125.97	15	123.16	15
2mis100_500C	97.17	11	96.51	11	97.33	11	103.57	12	98.97	11	98.15	11
2mis100_500D	145.44	17	144.49	17	145.12	17	148.61	17	146.48	17	141.95	16
2mis101_500A	165.98	18	162.48	18	163.96	18	170.67	19	164.03	17	169.45	18
2mis101_500B	148.21	17	150.7	17	146.83	17	143.27	16	139.7	15	136.22	14
2mis101_500C	140.43	16	141.98	16	142.46	16	142.8	16	140.23	16	143.05	16
2mis101_500D	105.65	12	105.27	12	105.68	12	96.49	10	109.8	13	117.89	15
2spp100_300A	31.99	8	31.66	8	31.92	8	31.69	8	31.68	8	31.95	8
2spp100_300B	42.84	12	43.27	12	43.6	12	42.82	12	43.5	12	43.88	12
2spp100_300C	115.57	27	115.54	27	120.96	28	113.72	25	109.37	23	106.56	21
2spp100_300D	61.7	16	60.87	16	59.83	16	57.83	14	67.59	19	65.72	17
2spp101_300C	71.08	15	71.08	15	71.87	15	69.59	14	69.33	13	73.02	14
2spp101_300D	36.74	10	36.41	10	36.58	10	36.85	10	36.36	10	38.82	11
2spp100_500A	118.42	11	117.74	11	118.45	11	125.24	12	118.03	11	112.37	10
2spp100_500B	119.16	12	119	12	119.46	12	124.11	13	118.55	12	123.22	12
2spp100_500C	115.13	12	115.63	12	113.06	12	113.82	12	117	12	110.22	11
2spp101_500A	102.22	11	103.3	11	103	11	104.57	11	107.39	11	100.84	10
2spp101_500B	165.38	16	168.38	16	164.76	16	174.05	18	175.06	18	174.26	17
2spp101_500C	87.42	10	86.39	10	86.39	10	86.54	10	87.17	10	86.7	10
2spp202_600A	344.68	10	344.72	10	371.8	10	361.68	11	357.31	12	378.9	12
2spp202_600B	251.62	11	267.9	11	267.47	11	255.69	11	308.9	12	265.77	11
2spp202_600D	275.59	11	276.58	11	274.61	11	315.98	11	292.16	11	304.08	11
2spp203_600A	655.94	11	749.98	11	654.07	11	712.48	12	668.85	11	653.25	11
2spp203_600B	396.53	11	396.77	11	359.21	11	389.05	11	324.91	10	323.97	10
2spp203_600D	519.73	12	588.37	12	588.65	12	520.25	12	512.02	12	567.88	13
2spp202_1000A	413.13	8	417.17	8	413.65	8	416.29	8	427.13	8	421.74	8
2spp202_1000D	1059.17	14	995.91	14	1065.48	14	1081.72	15	1117.56	15	1260.24	19
2spp203_1000A	813.53	11	808.14	11	808.59	11	783.34	10	970.01	13	878.78	11
2spp203_1000B	652.35	11	620.67	11	623.95	11	654.53	11	629.2	10	612.21	9
2spp203_1000C	1158.21	19	1155.33	19	1162.86	18	1120.45	17	1121.08	16	1102.55	16
2spp203_1000D	798.49	15	798.9	15	799.1	15	841.2	16	767.89	12	764.16	12

Table K.11: Running times (in seconds) for all SPP instances solved with the RVF Algorithm under Implementation 4, using different cut ratios - Part II

Instance ID	0.5		0.6		0.7		0.8		0.9	
	Time (sec)	Iter.	Time (sec)	Iter.	Time (sec)	Iter.	Time (sec)	Iter.	Time (sec)	Iter.
2mis100_300A	119.14	28	122.09	28	122.03	30	117.55	27	124.12	28
2mis100_300B	91.76	21	108.92	27	99.59	23	88.08	20	107.37	26
2mis100_300C	77.31	21	72.73	20	69.06	17	73.08	19	70.03	18
2mis100_300D	49.26	15	50.61	16	51.25	16	47.44	14	45.24	13
2mis101_300A	66.15	20	60.94	17	64.22	19	60.06	17	61.4	17
2mis101_300B	79.54	20	91.62	26	85.48	22	86.43	23	83.3	22
2mis101_300C	81.51	22	70.77	17	74.37	18	76.1	19	80.8	21
2mis100_500A	120.51	14	124.02	15	115.44	13	134.08	17	129.7	16
2mis100_500B	124.11	15	123.66	15	126.67	15	125.76	15	110.32	11
2mis100_500C	94.78	10	88.83	9	103.76	12	103.95	12	94.37	10
2mis100_500D	142.4	16	137.18	15	147.4	17	162.82	20	153.42	18
2mis101_500A	161.17	17	161.87	17	162.45	17	162.76	18	169.06	18
2mis101_500B	148.22	16	156.4	18	152.78	17	155.01	18	150.09	16
2mis101_500C	133.21	14	134.71	14	141.65	16	151.79	18	148.33	17
2mis101_500D	114.02	14	102.67	11	106.33	12	102.47	11	97.7	10
2spp100_300A	32.17	8	31.97	8	32.07	8	32.15	8	34.54	9
2spp100_300B	41.74	11	41.58	11	41.62	11	41.17	11	41.41	11
2spp100_300C	111.53	22	108.8	21	118.03	24	123.02	25	110.05	21
2spp100_300D	63.86	17	65.83	18	65.07	17	65.45	17	65.4	18
2spp101_300C	69.16	13	70.63	13	73.03	14	71.21	13	76.6	15
2spp101_300D	38.07	10	37.65	10	38.13	10	38.14	10	38.16	10
2spp100_500A	108.03	9	119.77	11	119.17	11	108.51	9	120.56	11
2spp100_500B	118.22	12	124.26	13	124.03	13	121.88	12	118.93	12
2spp100_500C	110.67	11	110.79	11	113.99	12	115.53	12	111.81	11
2spp101_500A	100.5	10	99.84	10	99.96	10	105.19	11	105.22	11
2spp101_500B	175.5	17	189.09	19	186.62	19	176.2	17	170.97	16
2spp101_500C	89.04	10	87.72	10	88.17	10	88.13	10	88.87	10
2spp202_600A	405.32	12	404.09	11	403.5	11	461.36	13	451.75	12
2spp202_600B	270.89	11	281.11	11	287.24	12	293.04	12	292.62	12
2spp202_600D	296.71	11	286.65	11	293.03	10	273.54	10	275.84	10
2spp203_600A	759.71	11	661.17	11	762.6	11	799.9	12	765.07	11
2spp203_600B	328.88	10	311.92	10	312.54	10	382.91	11	341.62	10
2spp203_600D	634.68	14	628.18	14	731.25	15	592.37	14	594.74	12
2spp202_1000A	415.79	8	440.69	8	402.19	7	400.83	7	422.97	8
2spp202_1000D	1203.77	18	1134.47	17	1152.04	17	1191.1	18	1118.42	14
2spp203_1000A	973.09	13	817.66	10	948.52	13	963.72	12	1170.05	16
2spp203_1000B	606	9	579.17	9	686.33	10	631.65	10	667.18	10
2spp203_1000C	1185.36	16	1151.64	17	1139.13	15	1250.95	18	1077.27	16
2spp203_1000D	957.25	16	883.81	15	895.89	15	872.38	16	914.69	16

Table K.12: Running time in seconds for multiobjective KP using DCM and MOMIX

Instance	DCM		MOMIX	
	Time	NDPs	Time	NDPs
KP_p-3_n-10_ins-5	0.96	25	84.71	25
KP_p-3_n-20_ins-1	2.38	35	419.51	35
KP_p-3_n-20_ins-2	2.17	35	396.02	35
KP_p-3_n-20_ins-3	3.15	43	559.62	43
KP_p-3_n-20_ins-4	1.58	28	441.46	28
KP_p-3_n-20_ins-5	4.53	62	936.97	62
KP_p-3_n-20_ins-6	4.91	63	581.51	63
KP_p-3_n-20_ins-7	1.48	31	412.75	31
KP_p-3_n-20_ins-8	1.97	30	307.28	30
KP_p-3_n-20_ins-9	1.55	31	298.78	31
KP_p-3_n-20_ins-10	1.06	22	452.11	22
KP_p-3_n-30_ins-1	8.32	88	2756.27	88
KP_p-3_n-30_ins-2	25.68	196	9542.12	196
KP_p-3_n-30_ins-3	13.24	118	3288.93	118
KP_p-3_n-30_ins-4	5.72	63	2028.00	63
KP_p-3_n-30_ins-5	5.15	69	1893.77	69
KP_p-3_n-30_ins-6	11.16	103	4481.15	103
KP_p-3_n-30_ins-8	11.12	98	5118.02	98
KP_p-3_n-30_ins-9	5.88	62	1757.47	62
KP_p-3_n-30_ins-10	9.74	95	2029.44	95
KP_p-4_n-10_ins-6	1.34	19	68.44	19
KP_p-4_n-10_ins-7	0.55	15	29.98	15
KP_p-4_n-10_ins-9	1.34	22	51.91	22
KP_p-4_n-20_ins-1	6.12	60	682.57	60
KP_p-4_n-20_ins-2	29.39	143	2290.28	143
KP_p-4_n-20_ins-5	10.99	74	1428.98	74
KP_p-4_n-20_ins-6	33.65	152	2476.15	152
KP_p-4_n-20_ins-7	42.27	175	5152.27	175
KP_p-4_n-20_ins-8	18.11	116	1649.04	116
KP_p-4_n-20_ins-9	10.76	77	1188.91	77
KP_p-4_n-20_ins-10	56.65	229	4364.69	229
KP_p-5_n-10_ins-1	1.25	20	93.71	20
KP_p-5_n-10_ins-2	1.10	19	50.75	19
KP_p-5_n-10_ins-3	1.36	23	90.16	23
KP_p-5_n-10_ins-4	1.92	28	109.42	28
KP_p-5_n-10_ins-8	1.73	25	83.19	25
KP_p-5_n-20_ins-2	26.64	95	1800.64	95
KP_p-5_n-20_ins-3	13.40	76	1594.91	76
KP_p-5_n-20_ins-4	32.08	89	1389.77	89
KP_p-5_n-20_ins-5	49.60	110	1894.26	110
KP_p-5_n-20_ins-6	16.64	87	1738.56	87
KP_p-5_n-20_ins-7	11.66	61	1189.03	61

Table K.13: Running time in seconds for biobjective SPP using DCM

Instance	DCM	
	Time	NDPs
2mis100_300A	21.20	41
2mis100_300B	58.10	31
2mis100_300C	32.86	22
2mis100_300D	12.00	16
2mis101_300A	7.00	22
2mis101_300B	26.33	30
2mis101_300C	30.69	26
2mis100_500A	29.69	17
2mis100_500B	32.25	17
2mis100_500C	30.48	13
2mis100_500D	41.80	19
2mis101_500A	61.15	21
2mis101_500B	44.73	20
2mis101_500C	48.74	20
2mis101_500D	26.95	15
2spp100_300A	21.53	10
2spp100_300B	24.44	10
2spp100_300C	67.73	35
2spp100_300D	31.34	20
2spp101_300C	45.73	20
2spp101_300D	19.24	11
2spp100_500A	51.39	14
2spp100_500B	47.83	14
2spp100_500C	36.25	13
2spp101_500A	39.56	12
2spp101_500B	79.96	20
2spp101_500C	31.20	9
2spp202_600A	656.02	13
2spp202_600B	386.55	10
2spp202_600D	433.97	10
2spp203_600A	2437.27	14
2spp203_600B	951.93	11
2spp203_600D	2473.46	16
2spp202_1000A	685.74	8
2spp202_1000D	2346.32	19
2spp203_1000A	2071.44	16
2spp203_1000B	1195.47	9
2spp203_1000C	2574.71	24
2spp203_1000D	821.41	19

Table K.14: Running time in seconds for multiobjective Knapsack using MOA packages

Instance	TambyVanderpooten		DominguezRios		KirlikSayin	
	Time	NDPs	Time	NDPs	Time	NDPs
KP_p-3_n-10_ins-5	33.14	25	36.93	25	30.57	25
KP_p-3_n-20_ins-1	41.67	35	46.71	35	37.78	35
KP_p-3_n-20_ins-2	36.10	35	42.49	35	33.68	35
KP_p-3_n-20_ins-3	45.87	43	55.01	43	42.94	43
KP_p-3_n-20_ins-4	35.99	28	41.56	28	33.11	28
KP_p-3_n-20_ins-5	52.34	62	60.83	62	50.45	62
KP_p-3_n-20_ins-6	52.79	64	60.67	63	51.35	63
KP_p-3_n-20_ins-7	33.26	31	37.98	31	30.60	31
KP_p-3_n-20_ins-8	36.37	30	39.72	30	32.61	30
KP_p-3_n-20_ins-9	35.11	31	39.25	31	32.42	31
KP_p-3_n-20_ins-10	33.39	22	38.04	22	30.19	22
KP_p-3_n-30_ins-1	66.37	88	78.15	88	65.85	88
KP_p-3_n-30_ins-2	190.38	196	194.50	195	185.04	196
KP_p-3_n-30_ins-3	105.04	118	116.46	118	111.11	118
KP_p-3_n-30_ins-4	54.14	63	62.38	62	50.70	63
KP_p-3_n-30_ins-5	46.60	69	56.11	69	46.34	69
KP_p-3_n-30_ins-6	88.54	103	101.86	103	92.43	103
KP_p-3_n-30_ins-8	93.76	98	105.90	97	93.89	98
KP_p-3_n-30_ins-9	57.35	62	60.78	60	58.95	62
KP_p-3_n-30_ins-10	84.09	95	98.05	95	83.44	95
KP_p-4_n-10_ins-6	32.06	19	37.76	19	29.67	19
KP_p-4_n-10_ins-7	31.38	15	36.31	15	28.35	15
KP_p-4_n-10_ins-9	35.34	22	39.15	22	32.47	22
KP_p-4_n-20_ins-1	56.20	60	69.43	60	56.30	60
KP_p-4_n-20_ins-2	192.98	143	175.28	143	272.23	143
KP_p-4_n-20_ins-5	87.12	74	99.36	74	100.67	74
KP_p-4_n-20_ins-6	230.33	152	210.97	152	409.14	152
KP_p-4_n-20_ins-7	369.68	175	282.64	175	584.57	175
KP_p-4_n-20_ins-8	128.92	116	137.15	116	150.87	116
KP_p-4_n-20_ins-9	87.12	77	89.69	75	84.95	77
KP_p-4_n-20_ins-10	488.23	229	307.31	229	2581.20	229
KP_p-5_n-10_ins-1	35.01	20	42.37	20	32.31	20
KP_p-5_n-10_ins-2	36.69	19	42.44	19	36.18	19
KP_p-5_n-10_ins-3	39.06	23	43.12	23	41.79	23
KP_p-5_n-10_ins-4	43.89	28	53.05	28	51.49	28
KP_p-5_n-10_ins-8	39.21	25	41.70	22	43.19	25
KP_p-5_n-20_ins-2	226.88	95	175.26	95	4328.06	95
KP_p-5_n-20_ins-3	139.88	76	128.54	76	1170.35	76
KP_p-5_n-20_ins-4	408.75	89	241.26	89	5424.41	89
KP_p-5_n-20_ins-5	382.60	110	246.39	110	10.48	1
KP_p-5_n-20_ins-6	201.25	87	156.82	87	2572.96	87
KP_p-5_n-20_ins-7	111.55	61	106.26	61	478.73	61

Table K.15: Running time in seconds for biobjective SPP using MOA packages

Instance	Chalmet		EpsilonConstraint		DominguezRios		KirlikSayin		TambyVanderpooten	
	Time	NDPs	Time	NDPs	Time	NDPs	Time	NDPs	Time	NDPs
2mis100_300A	145.01	41	87.44	41	121.73	40	124.04	41	126.27	41
2mis100_300B	178.06	31	113.24	31	184.15	31	212.04	31	206.04	31
2mis100_300C	137.65	22	96.32	22	149.42	22	154.68	22	158.67	22
2mis100_300D	84.18	16	57.81	16	83.11	16	84.20	16	85.16	16
2mis101_300A	96.62	22	59.43	22	113.18	22	91.78	22	89.62	22
2mis101_300B	126.82	30	78.54	30	120.62	30	119.86	30	127.23	30
2mis101_300C	129.64	26	97.98	26	137.03	26	150.44	26	147.35	26
2mis100_500A	123.70	17	76.61	17	121.66	17	126.20	17	130.09	17
2mis100_500B	164.07	17	91.90	17	140.60	17	151.16	17	157.10	17
2mis100_500C	185.45	13	144.04	13	176.88	13	172.95	13	173.86	13
2mis100_500D	257.82	19	176.47	19	320.16	19	271.89	19	285.38	19
2mis101_500A	301.69	21	196.66	21	326.03	21	320.03	21	327.39	21
2mis101_500B	247.04	20	151.34	20	273.84	20	265.00	20	248.14	20
2mis101_500C	217.20	20	153.36	20	232.91	20	217.15	20	230.83	20
2mis101_500D	139.67	15	92.07	15	141.71	15	136.91	15	150.04	15
2spp100_300A	89.06	10	76.81	10	81.59	10	91.02	10	100.16	10
2spp100_300B	95.11	10	72.55	10	101.80	10	93.20	10	97.78	10
2spp100_300C	296.23	35	179.90	35	310.39	35	321.53	35	323.75	35
2spp100_300D	173.15	20	109.04	20	150.20	20	171.10	20	178.14	20
2spp101_300C	226.02	20	145.15	20	256.23	20	248.25	20	257.79	20
2spp101_300D	98.02	11	77.30	11	101.06	11	100.57	11	110.39	11
2spp100_500A	299.77	14	176.37	14	358.08	14	285.48	14	299.94	14
2spp100_500B	234.55	14	162.39	14	284.90	14	263.03	14	260.64	14
2spp100_500C	181.22	13	134.30	13	262.96	13	194.74	13	204.59	13
2spp101_500A	201.23	12	149.98	12	244.89	12	226.70	12	227.21	12
2spp101_500B	408.02	20	251.11	20	498.91	20	414.59	20	416.48	20
2spp101_500C	149.06	9	127.51	9	160.62	9	165.40	9	168.91	9
2spp202_600A	1931.14	13	1317.88	13	2053.42	13	1784.91	13	1904.82	13
2spp202_600B	1238.92	10	909.54	10	1316.38	11	1288.16	10	1376.38	10
2spp202_600D	1567.52	10	1074.44	10	1786.92	10	1623.66	10	1616.94	10
2spp203_600A	5872.85	14	3199.26	14	5929.87	14	5690.45	14	6044.00	14
2spp203_600B	2567.46	11	1545.38	11	2629.63	11	2523.62	11	2557.91	11
2spp203_600D	5258.12	16	3268.93	16	5886.46	16	5109.27	16	5280.63	16
2spp202_1000A	2587.18	8	1562.48	8	2113.75	8	2101.81	8	2205.76	8
2spp202_1000D	9792.31	19	4030.27	19	8556.76	19	6020.38	19	7055.99	19
2spp203_1000A	6533.35	16	2867.45	16	5893.24	16	4720.38	16	5271.92	16
2spp203_1000B	3354.44	9	1756.03	9	3313.95	9	2335.43	9	2888.96	9
2spp203_1000C	7817.58	24	4356.01	24	7196.76	24	6472.89	24	6636.80	24
2spp203_1000D	4183.57	19	2032.09	19	3976.37	19	3337.23	19	3550.32	19

Table K.16: Running times (in seconds) for all MILP instances solved with the RVF Algorithm under Implementation 3, using different numbers of feasible solutions

Instance ID	1		2		3		4		5	
	Time (sec)	#Stability Regions	Time (sec)	#Stability Regions	Time (sec)	#Stability Regions	Time (sec)	#Stability Regions	Time (sec)	#Stability Regions
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_0	531.42	42	461.71	42	645.8	42	547.35	41	804.67	41
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_10	255.54	42	482.76	42	1249.65	42	730.2	42	537.73	42
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_2	112.03	29	103.54	32	103.24	32	89.17	32	89.85	32
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_4	93.72	26	120.41	29	192.64	29	152.34	29	86.84	29
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_5	149.85	25	91.18	25	326.1	24	193.84	25	119.62	25
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_0	19.13	18	12.95	16	39.48	16	12.47	16	12.71	16
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_10	332.13	18	532.26	18	453.56	18	490.04	15	278.16	14
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_3	324.96	18	526.64	18	254.14	18	495.31	15	349.42	14
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_6	195.65	31	150.74	31	129.99	27	645.53	30	222.91	28
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_9	494.12	45	262.82	42	349.05	43	419.07	39	410.82	41
MILP_obj_2_var_30_nzr_6_intR_9_u_1_ins_10	46.47	21	142.06	20	114.07	19	58.21	18	57.7	18
MILP_obj_2_var_30_nzr_6_intR_9_u_1_ins_6	52.57	16	27.29	16	40.31	16	22.97	16	30.18	16
MILP_obj_2_var_30_nzr_7_intR_9_u_1_ins_11	397.77	39	250.03	36	239.44	35	315.93	36	336.03	35
MILP_obj_2_var_30_nzr_7_intR_9_u_1_ins_8	316.31	34	116.1	34	153.66	34	70.78	34	82.17	34
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_1	226.91	32	94.55	32	126.07	32	153.3	32	149.91	32
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_5	314.11	29	245.54	28	200.37	25	200.54	29	895.41	26
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_6	8.81	18	213.75	33	151.67	34	172.99	34	278.04	34
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_7	184.71	27	75.46	27	74.77	26	71.75	26	60.96	26
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_9	142.6	34	57.93	34	69.79	34	76.36	34	55.27	34
MILP_obj_2_var_30_nzr_8_intR_9_u_1_ins_11	52.97	16	27.19	16	40.38	16	22.81	16	29.84	16
MILP_obj_2_var_30_nzr_8_intR_9_u_1_ins_4	305.23	48	251.84	48	317.44	48	205.24	48	189.57	48
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_10	27.54	15	44.91	15	47.81	16	122.54	14	128.2	14
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_2	14.52	16	114.94	37	86.59	37	100.01	37	89.59	37
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_5	255.91	48	591.86	58	1201.71	55	669.18	53	657.09	53
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_7	129.34	27	74.59	27	80.78	27	65.18	27	65.32	27
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_9	57.1	29	78.4	36	70.38	36	98.45	36	73.59	36
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_10	741.85	42	476.34	39	2753.54	40	1891.18	38	1923.56	38
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_5	596.95	47	407.55	47	568.22	47	679.35	47	580.63	47
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_6	1238.86	44	809.76	42	669.23	44	960.55	41	862.39	41
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_8	43.93	22	28.1	22	55.91	22	42.59	22	42.87	22
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_0	2440.25	50	2316.61	50	9578.04	50	11999.62	50	2590.42	50
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_2	405.97	29	232.52	29	244.79	29	198.89	30	-	-
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_7	1272.15	61	767.11	61	464.9	61	3607.6	61	542.87	61
MILP_obj_2_var_40_nzr_9_intR_95_u_1_ins_10	3998.12	45	1437.46	43	1356.57	44	940.82	42	962.19	43
MILP_obj_2_var_40_nzr_9_intR_95_u_1_ins_2	4059.19	38	1917.81	37	1248.72	38	1162.47	38	1080.13	37
MILP_obj_2_var_30_nzr_7_intR_9_u_1_ins_2	181.83	23	120.31	22	263.19	21	152.08	22	151.3	22
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_9	1720.13	49	912.4	48	945.43	48	3436.41	45	682.85	47
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_3	771.02	45	-	-	500.49	49	-	-	-	-
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_11	212.91	32	96.02	30	-	-	591.2	32	164.25	32
MILP_obj_2_var_30_nzr_8_intR_9_u_1_ins_2	685	33	423.85	30	1698.86	29	535.51	29	393.28	29
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_6	1087.9	50	628.56	49	637.13	44	698.85	44	697.93	43
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_8	1193.19	71	701.38	71	662.47	71	1032.9	73	401.38	68
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_9	3683.66	72	3910.18	57	-	-	3749.08	61	5911.44	57
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_10	2956.72	56	1542.26	49	-	-	1208.88	48	2537.59	51

Table K.17: Running times (in seconds) for all MILP instances solved with the RVF Algorithm under Implementation 4, using different cut ratios

Instance ID	0.1		0.5		0.9	
	Time (sec)	#Stability Regions	Time (sec)	#Stability Regions	Time (sec)	#Stability Regions
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_0	1550.44	42	2427.19	42	2041.05	42
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_10	141.6	42	167.05	42	193.5	42
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_2	53.14	32	72.58	32	108.43	32
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_4	630.29	29	286.46	29	626.26	29
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_5	29.08	25	45.64	25	63.41	25
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_0	88.08	18	64.57	18	89.86	18
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_10	346.42	18	357.43	17	375.21	17
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_3	348.11	18	359.8	17	367.67	17
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_6	111.79	31	150.79	31	162.98	31
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_9	897.28	52	2999.07	54	1179.98	51
MILP_obj_2_var_30_nzr_6_intR_9_u_1_ins_10	899.78	22	673.88	23	1973.02	21
MILP_obj_2_var_30_nzr_6_intR_9_u_1_ins_6	40.32	15	45.49	16	65.23	16
MILP_obj_2_var_30_nzr_7_intR_9_u_1_ins_11	265.93	37	260.56	38	209.02	38
MILP_obj_2_var_30_nzr_7_intR_9_u_1_ins_8	152.09	34	179.43	34	233.32	34
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_1	62.94	32	60.97	32	84.77	32
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_5	159.73	29	116.47	28	257.95	28
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_6	364.12	35	190.41	37	483.67	35
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_7	84.55	26	118.72	26	140.8	25
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_9	76.9	34	90.4	34	104.27	34
MILP_obj_2_var_30_nzr_8_intR_9_u_1_ins_11	40.26	15	45.57	16	65.34	16
MILP_obj_2_var_30_nzr_8_intR_9_u_1_ins_4	201.91	48	261.3	48	302.7	48
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_10	37.09	17	51.46	17	68.64	17
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_2	120.6	36	107.88	37	159.62	36
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_5	1386.47	58	1718.93	58	1272.49	58
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_7	28.3	27	45.1	27	42.13	27
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_9	79.78	36	99.58	36	109.22	36
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_10	2182.24	41	1276.14	41	1758.67	41
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_5	383.97	47	531.03	47	473.15	47
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_6	673.88	48	442.78	47	769.4	48
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_8	39.06	22	43.82	22	57.64	22
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_0	3021.36	50	-	-	5201.67	50
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_2	51.18	28	67.24	28	81.95	28
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_7	491.55	61	688.77	61	692.87	61
MILP_obj_2_var_40_nzr_9_intR_95_u_1_ins_10	2772.63	44	3417.43	44	2769.32	43
MILP_obj_2_var_40_nzr_9_intR_95_u_1_ins_2	4661.18	38	2201.6	38	3720.42	38
MILP_obj_2_var_30_nzr_7_intR_9_u_1_ins_2	246.35	22	204.55	23	196.03	22
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_9	1960.41	48	3494.13	49	4078.55	49
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_3	790.49	46	406.62	48	562.31	47
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_11	211.8	32	275.21	31	136.83	29
MILP_obj_2_var_30_nzr_8_intR_9_u_1_ins_2	2152.14	33	3804.14	30	12786.56	30
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_6	2909.6	47	4257.1	51	5458.69	50
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_8	-	-	1687.35	78	2895.97	78
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_9	-	-	-	-	-	-
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_10	6773.63	54	4531.6	55	7146.12	55

Table K.18: Running time (in seconds) for all MILP instances using the RVF Algorithm with Implementations 0, 1, 2, and 6

Instance ID	Imp 0		Imp 1		Imp 2		Imp 6	
	Time (sec)	#Stability Regions	Time (sec)	#Stability Regions	Time (sec)	#Stability Regions	Time (sec)	#Stability Regions
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_0	826.56	41	439.98	41	2064.05	42	2107.3	39
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_10	785.95	42	330.91	42	1566.82	42	1437.34	42
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_2	183.47	32	105.85	32	625.61	32	767.72	32
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_4	131.64	28	115.5	29	163.95	29	275.48	29
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_5	203.77	25	107.46	25	226.59	24	229.22	25
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_0	15.63	18	11.82	18	16.17	18	22.29	18
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_10	357.7	18	317.18	17	2066.17	18	1314.11	18
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_3	361.96	18	310.54	17	2064.71	18	1324.27	18
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_6	265.04	29	126.74	31	317.07	29	481.27	30
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_9	1626.99	58	515.45	50	833.37	52	1061.28	51
MILP_obj_2_var_30_nzr_6_intR_9_u_1_ins_10	62.16	22	41.17	20	72.53	21	81.7	21
MILP_obj_2_var_30_nzr_6_intR_9_u_1_ins_6	55	16	48.57	16	142.62	15	151.75	16
MILP_obj_2_var_30_nzr_7_intR_9_u_1_ins_11	384.48	36	316.35	39	588	37	614.71	37
MILP_obj_2_var_30_nzr_7_intR_9_u_1_ins_8	202.82	34	178.96	34	646.88	34	606.73	34
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_1	241.08	32	120	32	683.32	32	563.32	32
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_5	7558.48	29	193.81	27	855.38	29	825.16	28
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_6	93.52	37	82.26	35	153.62	36	184.35	36
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_7	174.23	27	76.7	27	166.54	27	234.2	27
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_9	120.37	34	102.61	34	84.76	34	85	34
MILP_obj_2_var_30_nzr_8_intR_9_u_1_ins_11	55.36	16	48.26	16	143.19	15	151.88	16
MILP_obj_2_var_30_nzr_8_intR_9_u_1_ins_4	530.73	48	326.74	48	822.67	48	661.06	48
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_10	27.89	15	22.69	15	49.61	15	66.05	15
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_2	219.4	37	186.37	37	269.77	36	303.88	34
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_5	1401.69	57	1014.29	58	2652.72	56	2405.42	57
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_7	165.03	27	97.41	27	397.99	27	358.68	27
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_9	199.14	36	89.22	36	315.36	36	303.9	36
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_10	855.65	42	598.59	41	4885.8	42	6215.34	42
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_5	793.96	47	482.33	47	3405.27	47	3283.64	47
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_6	1898.96	46	1214.48	48	4715.27	46	4895.37	46
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_8	38.64	22	34.52	22	44.87	21	46.18	22
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_0	4914.96	50	2936.45	50	12229.19	49	-	-
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_2	1531.11	30	328.98	30	834.98	29	905.98	29
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_7	2658.14	61	937.33	61	6197.25	60	6379.03	58
MILP_obj_2_var_40_nzr_9_intR_95_u_1_ins_10	6961.48	44	3019.68	45	11884.72	45	11055.72	45
MILP_obj_2_var_40_nzr_9_intR_95_u_1_ins_2	3195.26	38	2202.93	38	10799.53	37	11284.1	38
MILP_obj_2_var_30_nzr_7_intR_9_u_1_ins_2	-	-	587.65	23	929.72	24	659.79	24
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_9	-	-	4870.42	47	7618.7	48	5115.12	46
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_3	-	-	-	-	1421.74	47	1734.32	46
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_11	-	-	-	-	627.27	30	-	-
MILP_obj_2_var_30_nzr_8_intR_9_u_1_ins_2	-	-	-	-	1162.18	33	1618.29	33
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_6	-	-	-	-	4870.96	51	4092.54	50
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_8	-	-	-	-	6864.81	76	-	-
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_9	-	-	-	-	9162.69	68	10425.69	69
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_10	-	-	-	-	11902.72	56	12348.37	57

Appendix L

Base Performance Profile for the Computational Experiments in Chapter 3

In this appendix, we present the base performance profile for the computational experiments of Chapter 3.

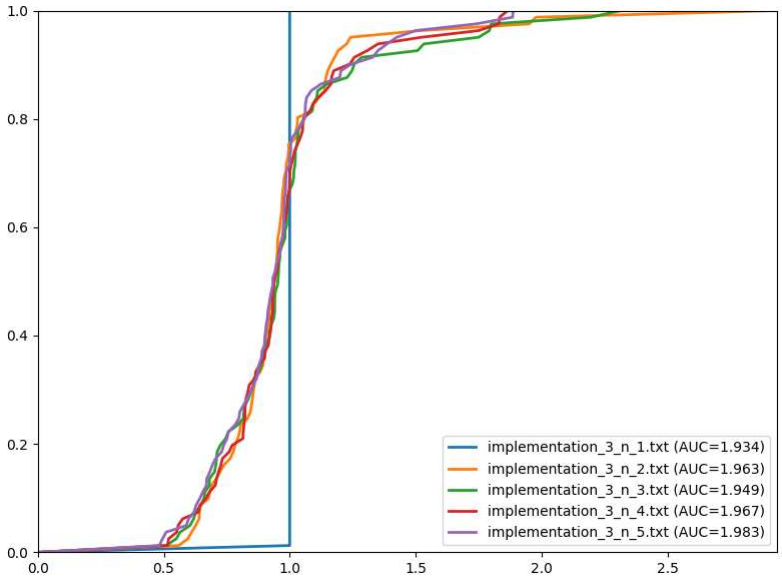


Figure L.1: Base performance profile comparing execution time of Implementation 3 for PIPs.

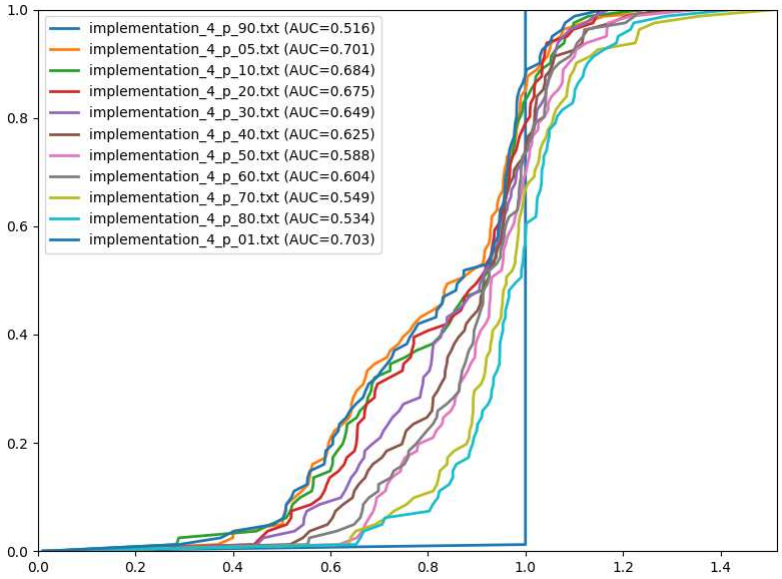


Figure L.2: Base performance profile comparing execution time of Implementation 4 using different percentages of cuts for PIPs.

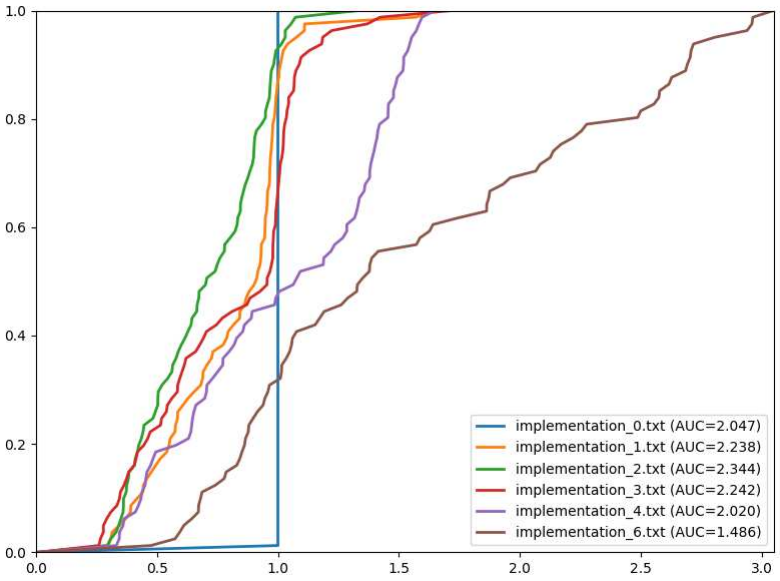


Figure L.3: Base performance profile comparing the execution time of the naive linear version and different implementations of the RVF algorithm for PIPs.

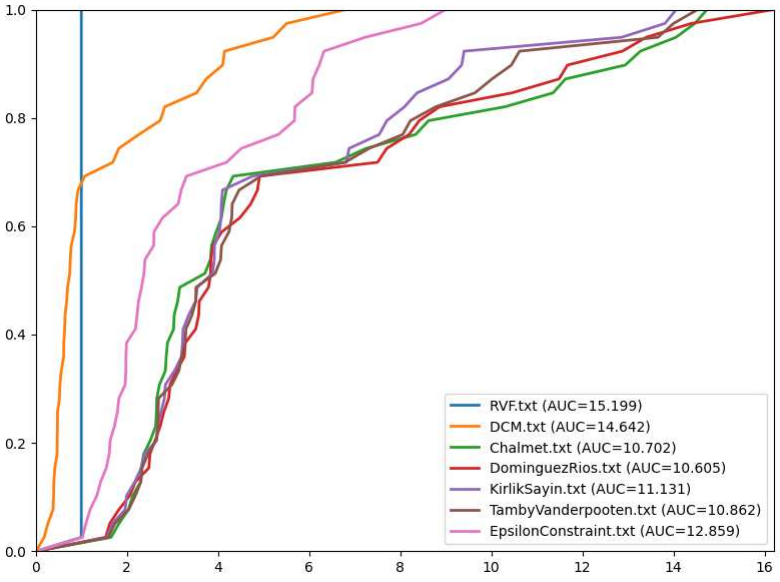


Figure L.4: Base performance profile comparing the execution time of the RVF Algorithm (Implementation 2) with all existing algorithms for the biobjective SPP.

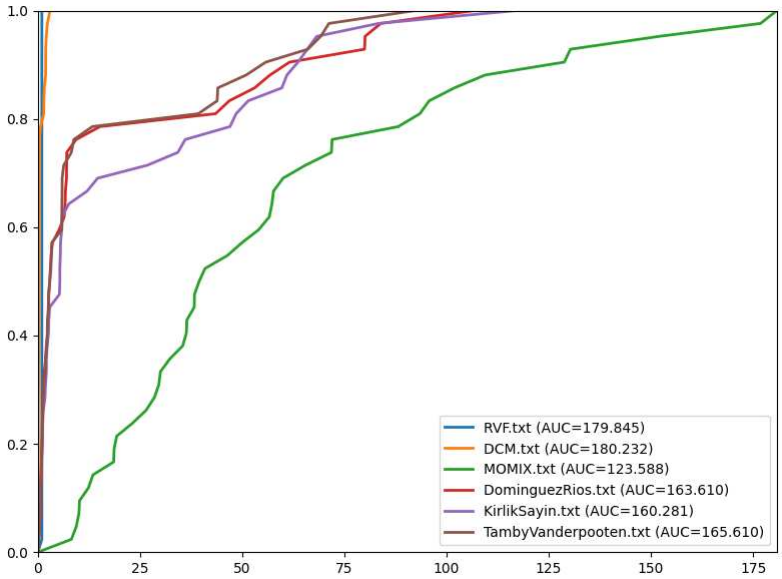


Figure L.5: Base performance profile comparing the execution time of the RVF Algorithm (Implementation 2) with all existing algorithms for the multiobjective KP.

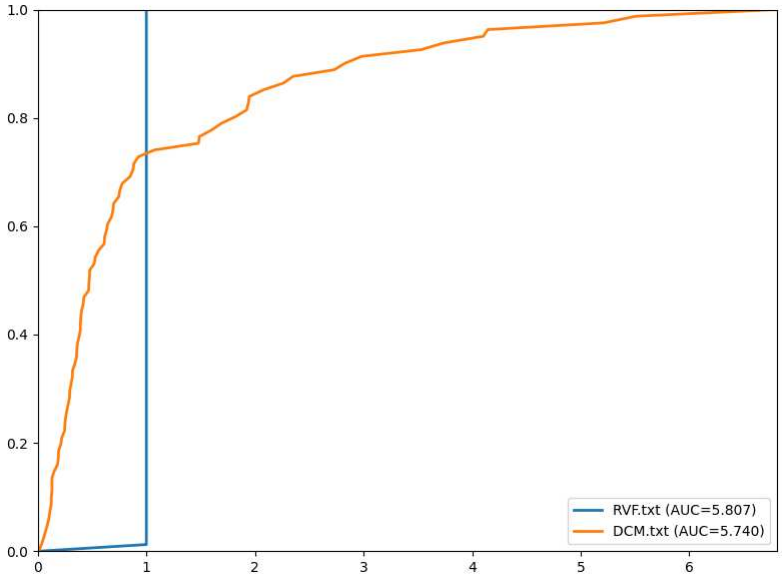


Figure L.6: Base performance profile comparing execution time of RVF and DCM for all PIPs.

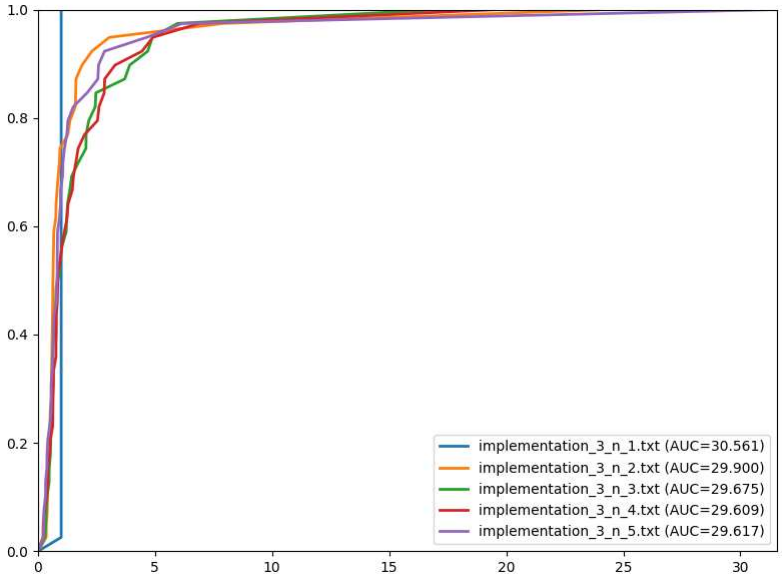


Figure L.7: Base performance profile comparing execution time of Implementation 3 for MILPs.

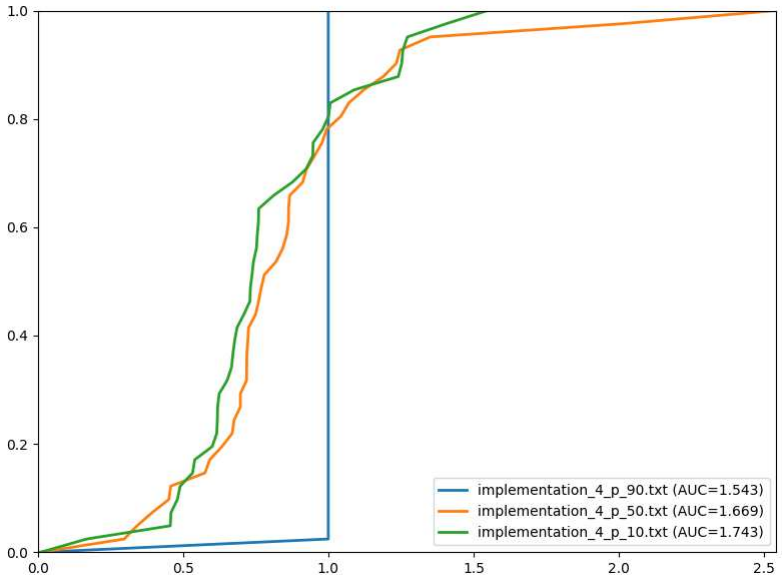


Figure L.8: Base performance profile comparing execution time of Implementation 4 for MILPs.

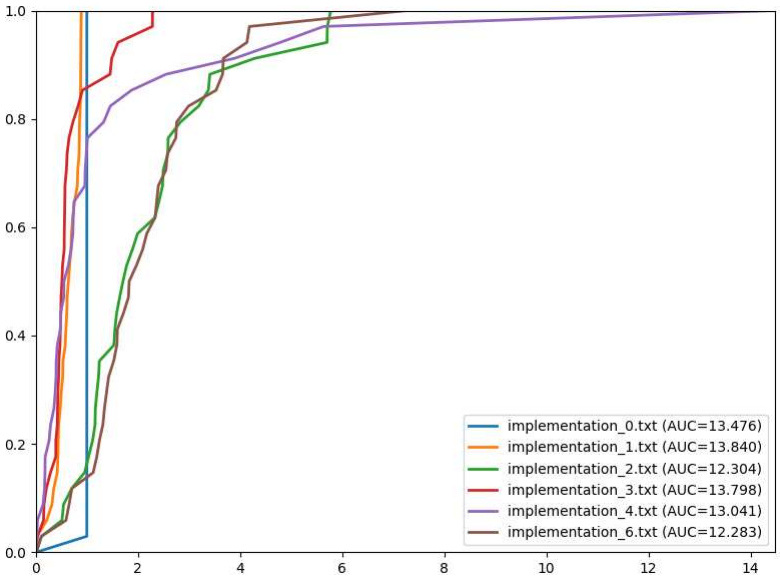


Figure L.9: Base performance profile comparing the execution time of the naive version and different implementations of the RVF algorithm for MILPs.

Appendix M

Tables from the Computational Experiment in Chapter 4

In this appendix, we present the results of our computational experiments for the pure integer instances in Tables M.1-M.8, and for the randomly generated MILP instances in Tables M.9-M.13. The results include the running time and the number of NDPs for PIP instances, as well as the number of stability regions for MILP instances.

Table M.1: Running time in seconds for the SPP using the BB/sample-based Algorithm for different initial sample sizes (denoted as n)

Instance ID	n=10	n=20	n=30	n=40	#NDP
2mis100_300A	528.68	529.58	514.01	507.27	41
2mis100_300B	451.82	393.46	479.5	384.2	31
2mis100_300C	328.08	358.84	343	309.92	22
2mis100_300D	214.35	180.34	179.11	154.21	16
2mis101_300A	227.21	252.3	256.44	217.57	22
2mis101_300B	321.33	315.31	327.08	318.65	30
2mis101_300C	318.41	323.72	346.37	290.28	26
2mis100_500A	606.35	522.89	653.18	588.34	17
2mis100_500B	665.38	730.37	588.88	786.95	17
2mis100_500C	572.45	795.22	1296.04	1586.02	13
2mis100_500D	915.47	1116.39	1122.26	1456.77	19
2mis101_500A	750.44	946.35	1332.6	1713.73	21
2mis101_500B	776.89	834.54	891.69	883.6	20
2mis101_500C	737.82	776.22	808.04	788.25	20
2mis101_500D	487.19	418.8	480	509.29	15
2spp100_300A	153.83	151.74	130.14	134.98	10
2spp100_300B	101.2	109.27	113.92	107.91	10
2spp100_300C	503.76	500.8	392	529.99	35
2spp100_300D	194.18	236.26	190.93	167.18	20
2spp101_300C	333.78	425.67	441.05	368.41	20
2spp101_300D	133.53	106.94	130.39	122.16	11
2spp100_500A	635.48	683.61	791.54	943.21	14
2spp100_500B	479.07	488.51	579.88	515.5	14
2spp100_500C	460.15	528.78	586.69	518.67	13
2spp101_500A	482.31	460.37	455.44	548.31	12
2spp101_500B	723.12	567.02	783.39	861.04	20
2spp101_500C	265.42	269.04	379.1	378.23	9
2spp202_600A	3609.97	8332.62	10540.25	-	13
2spp202_600B	1679.67	1710.75	2626.86	3443.2	10
2spp202_600D	2085.38	3437.11	4211.52	6749.58	10
2spp203_600A	-	-	-	-	14
2spp203_600B	-	9968.14	-	-	11
2spp203_600D	-	-	-	-	16
2spp202_1000A	8042.73	-	-	-	8
2spp202_1000D	-	-	-	-	19
2spp203_1000A	-	-	-	-	16
2spp203_1000B	6276.44	-	-	-	9
2spp203_1000C	-	-	-	-	24
2spp203_1000D	-	-	-	-	19

Table M.2: Running time in seconds for the KP using the BB/sample-based Algorithm for different initial sample sizes (denoted as n)

Instance ID	n=10	n=20	n=30	n=40	#NDP
KP_p-3_n-10_ins-5	3.08	3.39	4.97	7.23	25
KP_p-3_n-20_ins-1	18.42	21.33	37.83	42.32	35
KP_p-3_n-20_ins-2	11.95	17.36	25.7	43.46	35
KP_p-3_n-20_ins-3	19.86	22.73	26.3	37.01	43
KP_p-3_n-20_ins-4	15.53	19.55	30.14	40.04	28
KP_p-3_n-20_ins-5	64.92	68.76	103.71	145.81	62
KP_p-3_n-20_ins-6	52.46	64.3	78.75	101.36	63
KP_p-3_n-20_ins-7	11.55	11.56	16.78	25.1	31
KP_p-3_n-20_ins-8	14.3	16.84	26.93	43.62	30
KP_p-3_n-20_ins-9	14.66	16.47	22.49	33.59	31
KP_p-3_n-20_ins-10	5.04	5.79	7.82	15.68	22
KP_p-3_n-30_ins-1	221.37	234.32	409.58	732.9	88
KP_p-3_n-30_ins-2	1687.51	2095.13	2906.7	3532.48	196
KP_p-3_n-30_ins-3	544.73	571.63	955.49	1094.88	118
KP_p-3_n-30_ins-4	150.29	148.87	239.91	328.77	63
KP_p-3_n-30_ins-5	117.21	151.97	145.3	296.88	69
KP_p-3_n-30_ins-6	431.52	453.76	576.21	685.33	103
KP_p-3_n-30_ins-8	471.79	334.2	509.78	617.91	98
KP_p-3_n-30_ins-9	76.79	92.17	158.48	346.4	62
KP_p-3_n-30_ins-10	236.83	239.99	367.87	515.32	95
KP_p-4_n-10_ins-6	7.69	348.04	3814.77	-	19
KP_p-4_n-10_ins-7	7.66	328.86	4246.11	-	15
KP_p-4_n-10_ins-9	9.86	330.38	4091.4	-	22
KP_p-4_n-20_ins-1	69.38	512.64	4478.83	-	60
KP_p-4_n-20_ins-2	700.08	1915.74	9016.68	-	143
KP_p-4_n-20_ins-5	95.69	571.46	4696.68	-	74
KP_p-4_n-20_ins-6	671.46	2012.04	-	-	152
KP_p-4_n-20_ins-7	872.87	1815.42	8954.86	-	175
KP_p-4_n-20_ins-8	186.64	713.63	5204.78	-	116
KP_p-4_n-20_ins-9	118.06	733.15	5601.79	-	77
KP_p-4_n-20_ins-10	1181.33	2592.91	10858.54	-	229
KP_p-5_n-10_ins-1	2522.28	-	-	-	20
KP_p-5_n-10_ins-2	2396.23	-	-	-	19
KP_p-5_n-10_ins-3	2655.16	-	-	-	23
KP_p-5_n-10_ins-4	2448.82	-	-	-	28
KP_p-5_n-10_ins-8	2960.23	-	-	-	25
KP_p-5_n-20_ins-2	2814.4	-	-	-	95
KP_p-5_n-20_ins-3	2791.18	-	-	-	76
KP_p-5_n-20_ins-4	2420.96	-	-	-	89
KP_p-5_n-20_ins-5	2783.15	-	-	-	110
KP_p-5_n-20_ins-6	2849.97	-	-	-	87
KP_p-5_n-20_ins-7	2739.52	-	-	-	61

Table M.3: Running time in seconds for the random PIP instances using the BB/sample-based Algorithm for different initial sample sizes (denoted as n) - part I

Instance ID	n=10	n=20	n=30	n=40	#NDP
IP_obj_2_var_10_nzr_4_u_1_ins_0	0.16	0.19	0.14	0.15	6
IP_obj_2_var_10_nzr_4_u_2_ins_0	0.28	0.22	0.24	0.28	10
IP_obj_2_var_10_nzr_5_u_1_ins_0	0.08	0.08	0.09	0.1	4
IP_obj_2_var_10_nzr_5_u_2_ins_0	0.21	0.14	0.17	0.17	6
IP_obj_2_var_10_nzr_6_u_1_ins_0	0.15	0.15	0.16	0.17	4
IP_obj_2_var_10_nzr_6_u_2_ins_0	0.61	0.65	0.57	0.55	12
IP_obj_2_var_15_nzr_4_u_1_ins_0	0.53	0.45	0.25	0.27	9
IP_obj_2_var_15_nzr_4_u_2_ins_0	0.91	0.61	0.63	0.81	12
IP_obj_2_var_15_nzr_5_u_1_ins_0	0.54	0.61	0.52	0.64	9
IP_obj_2_var_15_nzr_5_u_2_ins_0	1.33	1.49	1.49	1.34	15
IP_obj_2_var_15_nzr_6_u_1_ins_0	1.43	1.44	1.2	1.67	22
IP_obj_2_var_15_nzr_6_u_2_ins_0	1.07	0.93	0.78	1.14	11
IP_obj_2_var_20_nzr_4_u_1_ins_0	7.65	7.16	6.44	8.07	37
IP_obj_2_var_20_nzr_4_u_2_ins_0	14.07	21.76	19.14	22.34	48
IP_obj_2_var_20_nzr_5_u_1_ins_0	1.88	2.55	1.83	2.74	15
IP_obj_2_var_20_nzr_5_u_2_ins_0	1.43	1.15	1.42	1.11	12
IP_obj_2_var_20_nzr_6_u_1_ins_0	1.06	1.24	1.3	1.34	14
IP_obj_2_var_20_nzr_6_u_2_ins_0	14.41	14.83	11.27	13.23	42
IP_obj_2_var_25_nzr_4_u_1_ins_0	19.18	20.71	23.82	29.14	43
IP_obj_2_var_25_nzr_4_u_2_ins_0	1.37	1.52	1.59	1.18	8
IP_obj_2_var_25_nzr_5_u_1_ins_0	3.11	3.02	4.97	4.15	21
IP_obj_2_var_25_nzr_5_u_2_ins_0	8.97	9.42	14.05	15.53	12
IP_obj_2_var_25_nzr_6_u_1_ins_0	6.6	7.25	6.19	7.7	17
IP_obj_2_var_25_nzr_6_u_2_ins_0	12.92	15.88	19.44	20.77	22
IP_obj_3_var_10_nzr_4_u_1_ins_0	0.63	0.86	1.68	4.2	11
IP_obj_3_var_10_nzr_4_u_2_ins_0	12.51	9.98	12.06	12.66	53
IP_obj_3_var_10_nzr_5_u_1_ins_0	0.68	1.19	2.22	4.62	14
IP_obj_3_var_10_nzr_5_u_2_ins_0	19.13	14.09	16.23	20.63	68
IP_obj_3_var_10_nzr_6_u_1_ins_0	2.06	2.06	5.5	6.48	22
IP_obj_3_var_10_nzr_6_u_2_ins_0	0.44	0.8	1.63	4.11	11
IP_obj_3_var_15_nzr_4_u_1_ins_0	27.63	23.63	84.31	46.55	69
IP_obj_3_var_15_nzr_4_u_2_ins_0	55.72	75.73	175.56	200.38	97
IP_obj_3_var_15_nzr_5_u_1_ins_0	204.25	224.05	829.35	367.71	100
IP_obj_3_var_15_nzr_5_u_2_ins_0	47.14	53.39	58.8	61.89	91
IP_obj_3_var_15_nzr_6_u_1_ins_0	57.17	394.28	562.45	737.59	79
IP_obj_3_var_15_nzr_6_u_2_ins_0	18.27	18.98	18.48	22.81	56

Table M.4: Running time in seconds for the random PIP instances using the BB/sample-based Algorithm for different initial sample sizes (denoted as n) - part II

Instance ID	n=10	n=20	n=30	n=40	#NDP
IP_obj_3_var_20_nzr_4_u_1_ins_0	84.04	109.02	136.77	249.78	105
IP_obj_3_var_20_nzr_4_u_2_ins_0	311.62	762.69	497.05	3116.49	162
IP_obj_3_var_20_nzr_5_u_1_ins_0	2913.19	7928.94	11439.19	-	323
IP_obj_3_var_20_nzr_5_u_2_ins_0	126.15	221.63	392.31	876.39	105
IP_obj_3_var_20_nzr_6_u_1_ins_0	28.14	134.21	644.02	497.83	42
IP_obj_3_var_20_nzr_6_u_2_ins_0	4208.27	4116.47	8594.25	-	365
IP_obj_3_var_25_nzr_4_u_1_ins_0	86.33	115.3	244.34	1359.31	87
IP_obj_3_var_25_nzr_4_u_2_ins_0	3196.27	6059.69	7144.03	-	319
IP_obj_3_var_25_nzr_5_u_1_ins_0	917.68	5329.93	-	-	193
IP_obj_3_var_25_nzr_6_u_1_ins_0	2564.2	5348.19	-	-	249
IP_obj_3_var_25_nzr_6_u_2_ins_0	400.06	721.68	1551.07	3545.71	136
IP_obj_4_var_10_nzr_4_u_1_ins_0	9.47	382.63	3975.01	-	36
IP_obj_4_var_10_nzr_4_u_2_ins_0	29.62	362.19	4475.99	-	76
IP_obj_4_var_10_nzr_5_u_1_ins_0	10.27	349.64	3852.51	-	38
IP_obj_4_var_10_nzr_5_u_2_ins_0	25.16	420.15	4178.66	-	64
IP_obj_4_var_10_nzr_6_u_1_ins_0	14.63	376.59	4287.43	-	54
IP_obj_4_var_10_nzr_6_u_2_ins_0	20.47	388.92	3846.96	-	51
IP_obj_4_var_15_nzr_4_u_1_ins_0	287.71	832.54	6436.39	-	201
IP_obj_4_var_15_nzr_4_u_2_ins_0	327.15	827.22	6798.85	-	196
IP_obj_4_var_15_nzr_5_u_1_ins_0	522.09	2282.18	10255.41	-	238
IP_obj_4_var_15_nzr_5_u_2_ins_0	1525.55	6469.38	-	-	327
IP_obj_4_var_15_nzr_6_u_1_ins_0	533.06	3596.66	13130.18	-	124
IP_obj_4_var_15_nzr_6_u_2_ins_0	545.66	3472.69	13384.13	-	193
IP_obj_4_var_20_nzr_5_u_1_ins_0	504.07	2300.11	-	-	194
IP_obj_5_var_10_nzr_4_u_1_ins_0	2167.3	-	-	-	43
IP_obj_5_var_10_nzr_4_u_2_ins_0	2259.93	-	-	-	100
IP_obj_5_var_10_nzr_5_u_1_ins_0	2144.12	-	-	-	41
IP_obj_5_var_10_nzr_5_u_2_ins_0	2268.26	-	-	-	99
IP_obj_5_var_10_nzr_6_u_1_ins_0	2420.25	-	-	-	96
IP_obj_5_var_10_nzr_6_u_2_ins_0	2342.76	-	-	-	27
IP_obj_5_var_15_nzr_4_u_1_ins_0	7461.85	-	-	-	572
IP_obj_5_var_15_nzr_4_u_2_ins_0	10120.74	-	-	-	496
IP_obj_5_var_15_nzr_5_u_1_ins_0	2322.25	-	-	-	107
IP_obj_5_var_15_nzr_6_u_2_ins_0	5826.36	-	-	-	400
IP_obj_5_var_20_nzr_4_u_1_ins_0	9415.95	-	-	-	535

Table M.5: Running time in seconds for the SPP using the BB/batch-based Algorithm for different initial sample sizes (denoted as n) and batch sizes

Instance ID	Batch=3				Batch=4				Batch=5			
	n=10	n=20	n=30	n=40	n=10	n=20	n=30	n=40	n=10	n=20	n=30	n=40
2mis100_300A	307.65	274.74	427.75	384.91	367.79	389.9	422.2	405.95	473.11	420.2	457.04	635.07
2mis100_300B	1890.47	1347.49	1728.92	1126.02	3802.07	1723.79	2113.69	1998.46	4394.96	2818.08	1969.78	2624.34
2mis100_300C	725.07	800.9	935.64	938.2	1514.02	1765.68	1433.02	1088.23	1752.95	1623.57	1564.05	1255.71
2mis100_300D	79.27	65.38	69.53	70.34	67.54	82.9	78.68	81.69	82.93	69.47	72.44	89.87
2mis101_300A	67.97	69.75	61.59	93.11	78.62	72.3	61.31	72.63	71.46	86.99	63.25	82.64
2mis101_300B	178.6	164.95	130.93	152.51	215.27	200.72	138.1	155.39	249.24	224.19	176.77	193.89
2mis101_300C	419.75	437.37	327.6	227.12	1008.25	644.83	524.12	495.78	703.08	902.83	751.93	576.21
2mis100_500A	787.67	824.39	927.3	910.83	1298.85	1078.8	1533.88	1361.18	1651.09	1088.02	1891.36	1592.67
2mis100_500B	2054.17	2491.72	2772.45	2472.09	3078.08	3067.51	2381.57	2701.55	3650.27	4209.53	4174.52	4165.33
2mis100_500C	5662.52	5367.82	5139.8	13777.16	8963.81	8349.74	6437.37	8431.73	12047.38	11274.81	9329.1	11377.03
2mis100_500D	9300.52	8699.64	9143.02	-	14027.37	11882.39	12254.74	10741.73	-	-	14222.77	12529.28
2mis101_500A	6436.13	7273.8	7926.18	9430.49	11530.88	11706.19	-	11569.28	-	9860.79	-	-
2mis101_500B	1808.78	2920.78	1707.16	2073.64	2889.38	2790.03	2816.11	2453.79	3145.35	3022.95	3549.54	2939.87
2mis101_500C	3455.69	3026.18	3419.09	3343.97	4210.88	3178.41	5959.68	4257.6	3879.03	4362.17	4807.73	4627.74
2mis101_500D	1326.38	1213.57	624.49	1037.37	2089.86	1188.43	1656.96	930.82	3237.91	1511.46	2139.68	2566.44
2spp100_300A	131.48	93.3	78.4	66.31	141.09	84.3	94.82	109.77	122.25	106	295.48	225.59
2spp100_300B	49.84	65.82	67.12	64.7	57.7	65.66	79.41	69.29	64.77	74.52	91.7	103.11
2spp100_300C	2336.09	1664.47	2984.3	1909.03	3875.12	3894.8	5259.47	3195.41	6572.75	3893.1	3735.59	4078.56
2spp100_300D	125.64	114.42	105.41	175.52	166.57	128.42	172.75	158.68	162.63	157.54	164.88	200.73
2spp101_300C	1391.85	1265.89	1102.93	1123.36	2212.72	1875.79	1849.91	1332.49	1854.08	2190.57	3017.14	1825.99
2spp101_300D	55.1	64.65	52.75	74.03	62.04	69.91	63.9	66.86	62.21	62.37	54.11	67.53
2spp100_500A	1905.18	1653.13	1951.82	2127.88	2277.72	2774.56	2043.42	2679.46	3114.31	3004.59	2654.43	2711.87
2spp100_500B	712.63	878.93	768.44	1054.49	1034.22	980.57	1084.55	1480.24	1092.1	1340.33	1331.66	1551.08
2spp100_500C	496.54	558.96	714.11	654.97	728.48	596.27	537.82	791.93	923.91	905.22	664.19	916.33
2spp101_500A	1172.14	542.2	758.59	956.56	783.46	571.1	806.3	1120.57	1511.27	1003.1	1477.72	1315.17
2spp101_500B	2539.18	2354.44	3101.03	2426.45	2595.1	3777.45	4320.8	3407.99	5105.88	5373.56	6195.2	5869.24
2spp101_500C	323.14	238.88	382.33	401.09	394.7	336.66	498.07	726.89	545.42	370.95	488.61	815.32
2spp202_600A	-	-	-	-	-	-	-	-	-	-	-	-
2spp202_600B	6490.76	4274.57	5296.38	6745.85	8906.99	5437.2	9386.83	7563.39	11392.02	6657.6	8186.36	8777.7
2spp202_600D	13389.29	13896.57	-	-	-	-	-	-	-	-	-	-

Table M.6: Running time in seconds for the KP using the BB/batch-based Algorithm for different initial sample sizes (denoted as n) and batch sizes

Instance ID	Batch=3				Batch=4				Batch=5			
	n=10	n=20	n=30	n=40	n=10	n=20	n=30	n=40	n=10	n=20	n=30	n=40
KP_p-3_n-10_ins-5	1.97	3.02	4.17	6.35	1.77	2.62	4.06	6.33	2.21	2.71	4.13	6.42
KP_p-3_n-20_ins-1	17.96	21.15	32.04	42.47	22.93	23.51	34.86	42.71	24.14	23.3	34.05	40.38
KP_p-3_n-20_ins-2	15.92	16.9	27.09	38.52	14.54	17.62	27.13	34.73	20.83	24.37	27.88	39.27
KP_p-3_n-20_ins-3	17.28	22.46	26.7	36.25	20.01	21.67	25.17	31.33	22.84	25.22	27.13	36.99
KP_p-3_n-20_ins-4	14.51	21.15	33.47	39.94	14.95	24.35	33.53	38.84	18.2	24.69	32.46	45.42
KP_p-3_n-20_ins-5	77.37	76.4	110.62	140.36	86.79	87.89	121.01	130.68	107.41	109.73	127.69	150.15
KP_p-3_n-20_ins-6	56.59	63.2	72.93	99.55	65.17	68.04	75.98	94.17	72.61	77.47	81.51	100.59
KP_p-3_n-20_ins-7	12.07	12.01	15.75	25.84	10.5	11.66	17.22	27.48	13.76	13.74	17.41	27.71
KP_p-3_n-20_ins-8	12.97	16.63	23.5	38.66	15.98	21.19	24.63	38.27	17.23	19.96	24.49	40.56
KP_p-3_n-20_ins-9	12.62	14.73	20.08	29.17	15.4	22.39	21.33	34.73	16.32	16.57	21.2	28.08
KP_p-3_n-20_ins-10	4.09	6.38	9.44	14.77	5.49	5.09	8.28	13.61	5.06	6.43	8.49	13.35
KP_p-3_n-30_ins-1	456.24	577.29	640.2	936.88	483.5	616.48	853.17	1570.6	707.71	933.1	1127.53	1209.29
KP_p-3_n-30_ins-2	4100.72	5278.32	4919.92	6111.76	5016.63	6478.05	6613.97	7801.03	5972.18	8179.39	8428.39	8466.37
KP_p-3_n-30_ins-3	1166.34	1140.05	1347.62	1600.18	1523.1	1454.42	1904.31	1875.11	2110.05	2039.43	1973.47	2206.71
KP_p-3_n-30_ins-4	170.16	171.09	354.29	332.09	195.93	213.14	387.02	401.87	311.08	260.4	494.22	442.35
KP_p-3_n-30_ins-5	182.59	176.5	152.51	340.95	230.09	231.27	170.87	307.35	310.51	226.72	253.3	368.72
KP_p-3_n-30_ins-6	797.3	1127.23	1361.34	1127.15	1035.87	1119.06	1321.46	2149.48	1294.05	2107.72	1852.99	1997.52
KP_p-3_n-30_ins-8	763.8	741.76	748.1	1101.82	976.22	1041.4	1215.85	2814.96	1438.17	1225.49	1039.54	1473.54
KP_p-3_n-30_ins-9	160.24	159.97	240.69	379.5	197.25	268.44	297.49	420.97	288.89	271.46	322.02	450.33
KP_p-3_n-30_ins-10	371.81	418.62	628.68	685.53	514.46	474.43	718.61	731.58	601.66	615.36	845.65	913.29
KP_p-4_n-10_ins-6	7.37	338.23	4651.12	-	7.32	348.91	4534.39	-	7.77	326.91	4237.26	-
KP_p-4_n-10_ins-7	7.36	355.62	4239.45	-	7.63	336.95	4271.38	-	7.49	383.62	4470.08	-
KP_p-4_n-10_ins-9	8.71	333.66	4264.95	-	8.9	338.71	4259.35	-	9.26	349.91	4178.61	-
KP_p-4_n-20_ins-1	65	512.05	4712.87	-	74.29	482.47	4783.03	-	80.2	487.51	4578.44	-
KP_p-4_n-20_ins-2	757.61	1899.91	8129.98	-	759.03	1856.03	8457.48	-	876.43	2199.93	7366.51	-
KP_p-4_n-20_ins-5	112.59	620.02	5004.04	-	119.71	587.82	4843.44	-	128.78	602.87	4582.87	-
KP_p-4_n-20_ins-6	1022.7	2535.22	10721.89	-	1216.67	2187.36	10627.87	-	1381.58	3046.04	10245.05	-
KP_p-4_n-20_ins-7	1081.86	2404.96	10376.8	-	1212.5	2579.83	9696.6	-	1276.74	2710.32	9300.62	-
KP_p-4_n-20_ins-8	215.93	746.28	4968.65	-	228.07	736.75	5530.28	-	266.63	811.11	5348.31	-
KP_p-4_n-20_ins-9	144.11	709.42	5412.38	-	155.66	664.98	4994.79	-	174.99	673.87	5411.36	-
KP_p-4_n-20_ins-10	1455.83	2464.8	8203.92	-	1499.38	2549.96	8566.54	-	2221.56	2959.02	9144.23	-
KP_p-5_n-10_ins-1	2330.75	-	-	-	2546.63	-	-	-	2530.32	-	-	-
KP_p-5_n-10_ins-2	2599.38	-	-	-	2377.02	-	-	-	2535.93	-	-	-
KP_p-5_n-10_ins-3	2618.45	-	-	-	2319.64	-	-	-	2474.6	-	-	-
KP_p-5_n-10_ins-4	2529.71	-	-	-	2435.59	-	-	-	2614.51	-	-	-
KP_p-5_n-10_ins-8	2207.16	-	-	-	2662.07	-	-	-	2466.25	-	-	-
KP_p-5_n-20_ins-2	2533.41	-	-	-	2757.03	-	-	-	2991.25	-	-	-
KP_p-5_n-20_ins-3	2514.98	-	-	-	2560.84	-	-	-	2935.71	-	-	-
KP_p-5_n-20_ins-4	2840.59	-	-	-	2604.71	-	-	-	2937.23	-	-	-
KP_p-5_n-20_ins-5	2861.31	-	-	-	3159.38	-	-	-	3184.06	-	-	-
KP_p-5_n-20_ins-6	2640.17	-	-	-	2620.83	-	-	-	2894.73	-	-	-
KP_p-5_n-20_ins-7	2649.86	-	-	-	2490.38	-	-	-	2846.51	-	-	-

Table M.7: Running time in seconds for the random PIP instances using the BB/batch-based Algorithm for different initial sample sizes (denoted as n) and batch sizes - part I

Instance ID	Batch=3				Batch=4				Batch=5			
	n=10	n=20	n=30	n=40	n=10	n=20	n=30	n=40	n=10	n=20	n=30	n=40
IP_obj_2_var_10_nzr_4_u_1_ins_0	0.1	0.09	0.06	0.07	0.09	0.08	0.09	0.07	0.07	0.07	0.08	0.09
IP_obj_2_var_10_nzr_4_u_2_ins_0	0.21	0.13	0.18	0.12	0.15	0.11	0.16	0.14	0.17	0.16	0.14	0.16
IP_obj_2_var_10_nzr_5_u_1_ins_0	0.04	0.05	0.06	0.06	0.04	0.04	0.06	0.06	0.04	0.04	0.06	0.08
IP_obj_2_var_10_nzr_5_u_2_ins_0	0.09	0.1	0.1	0.09	0.08	0.09	0.09	0.09	0.08	0.1	0.08	0.09
IP_obj_2_var_10_nzr_6_u_1_ins_0	0.07	0.07	0.08	0.12	0.07	0.08	0.08	0.09	0.07	0.08	0.09	0.1
IP_obj_2_var_10_nzr_6_u_2_ins_0	0.25	0.35	0.26	0.32	0.28	0.32	0.21	0.33	0.26	0.33	0.24	0.27
IP_obj_2_var_15_nzr_4_u_1_ins_0	0.19	0.17	0.21	0.18	0.2	0.16	0.19	0.2	0.21	0.16	0.19	0.24
IP_obj_2_var_15_nzr_4_u_2_ins_0	0.36	0.47	0.48	0.47	0.42	0.43	0.35	0.49	0.36	0.35	0.45	0.41
IP_obj_2_var_15_nzr_5_u_1_ins_0	0.29	0.27	0.37	0.35	0.31	0.34	0.37	0.35	0.34	0.33	0.34	0.36
IP_obj_2_var_15_nzr_5_u_2_ins_0	0.73	1.11	0.76	0.97	0.97	0.84	0.8	0.88	0.82	0.98	0.92	1.3
IP_obj_2_var_15_nzr_6_u_1_ins_0	0.69	0.59	0.52	1.08	0.89	0.88	0.48	0.94	0.73	0.69	0.53	0.62
IP_obj_2_var_15_nzr_6_u_2_ins_0	0.44	0.61	0.4	0.66	0.43	0.46	0.35	0.59	0.43	0.53	0.66	0.7
IP_obj_2_var_20_nzr_4_u_1_ins_0	6.88	7.63	7.82	4.96	6.54	4.77	9.61	5.3	9.25	7.16	7.08	6.73
IP_obj_2_var_20_nzr_4_u_2_ins_0	13.97	24.95	24.61	15.51	17.52	23.97	15.69	26.87	21.75	21.9	19.8	19.63
IP_obj_2_var_20_nzr_5_u_1_ins_0	1.08	1.12	1.34	1.4	1.04	1.12	1.02	1.67	1.16	1.4	0.98	1.25
IP_obj_2_var_20_nzr_5_u_2_ins_0	0.52	0.44	0.62	0.46	0.45	0.53	0.51	0.55	0.47	0.56	0.47	0.48
IP_obj_2_var_20_nzr_6_u_1_ins_0	0.57	0.67	0.6	0.71	0.6	0.61	0.59	0.61	0.63	0.8	0.67	0.81
IP_obj_2_var_20_nzr_6_u_2_ins_0	23.16	17.85	11.32	23.4	13.93	19.65	13.39	15.29	13.9	24.25	22.66	20.12
IP_obj_2_var_25_nzr_4_u_1_ins_0	28.35	57.46	98.8	48.18	49.63	87.71	95.89	42.91	30.23	78.1	278.86	112.17
IP_obj_2_var_25_nzr_4_u_2_ins_0	0.86	0.85	0.95	0.94	0.72	0.92	1.22	0.89	0.86	0.81	1.18	0.88
IP_obj_2_var_25_nzr_5_u_1_ins_0	3.19	4.27	6.12	2.75	5.57	3.65	5.36	3.62	4.54	4.6	11.64	5.22
IP_obj_2_var_25_nzr_5_u_2_ins_0	22.77	21.75	27.05	26.74	39.89	21.34	33.84	33.66	19.32	24.96	42.11	40.08
IP_obj_2_var_25_nzr_6_u_1_ins_0	16.23	12.52	7.94	12.79	17.58	14.46	14.68	16.49	21.93	21.38	22.13	27.92
IP_obj_2_var_25_nzr_6_u_2_ins_0	19.65	17.5	19.06	18.84	28.38	21.78	32.23	26.03	23.28	22.85	29.37	39.5
IP_obj_3_var_10_nzr_4_u_1_ins_0	0.48	0.71	1.76	4.1	0.53	0.66	1.61	4.06	0.44	0.64	1.76	4.04
IP_obj_3_var_10_nzr_4_u_2_ins_0	5.98	6.57	6.69	9.38	5.63	6.69	6.99	10.65	6.51	6.06	9.54	9.32
IP_obj_3_var_10_nzr_5_u_1_ins_0	0.32	0.8	1.98	4.55	0.85	1.07	2.07	4.14	0.38	0.78	1.85	4.33
IP_obj_3_var_10_nzr_5_u_2_ins_0	14.44	12.8	14.94	16.65	14.39	14.33	14.48	14.54	16.18	15.25	13.67	23.67
IP_obj_3_var_10_nzr_6_u_1_ins_0	1.24	1.93	2.88	4.55	2.04	1.73	4.79	7.96	1.54	1.68	2.31	4.45
IP_obj_3_var_10_nzr_6_u_2_ins_0	0.4	0.71	1.71	4.26	0.47	0.78	1.72	4.12	0.42	0.78	1.68	4.38
IP_obj_3_var_15_nzr_4_u_1_ins_0	16.93	17.79	34.8	32.39	316.07	18.11	34.73	38.84	82.63	23.2	31.09	60
IP_obj_3_var_15_nzr_4_u_2_ins_0	84.26	53.76	72.99	135.25	70.59	71.16	65.03	104.92	247.27	146.22	80.6	446.23
IP_obj_3_var_15_nzr_5_u_1_ins_0	96.38	73.21	59.29	167.75	104.73	134.65	256.12	278.86	115.2	228.11	75.99	198.61
IP_obj_3_var_15_nzr_5_u_2_ins_0	58.64	50.53	52.56	48.14	42.54	43.59	74.31	50.31	50.85	32.3	48.72	56.07

Table M.8: Running time in seconds for the random PIP instances using the BB/batch-based Algorithm for different initial sample sizes (denoted as n) and batch sizes - part II

Instance ID	Batch=3				Batch=4				Batch=5			
	n=10	n=20	n=30	n=40	n=10	n=20	n=30	n=40	n=10	n=20	n=30	n=40
IP_obj_3_var_15_nzr_6_u_1_ins_0	165.88	220.31	343.41	221.5	457.65	110.58	292.34	165.11	142.66	186.23	243.87	194.1
IP_obj_3_var_15_nzr_6_u_2_ins_0	13.4	14.81	19.89	22.1	16.88	15.78	18.55	17.26	15.95	20.84	23.26	21.38
IP_obj_3_var_20_nzr_4_u_1_ins_0	414.84	-	1275.94	4122.78	124.52	179.99	421.09	225.42	298.12	-	456.73	13321.71
IP_obj_3_var_20_nzr_4_u_2_ins_0	606.77	740.88	952.42	888.24	740.21	1453.32	751.05	1543.06	1263.35	1134.86	1163.81	2983.38
IP_obj_3_var_20_nzr_5_u_1_ins_0	10464.18	10891.21	-	-	-	9048.71	-	-	12880.21	-	11035.76	-
IP_obj_3_var_20_nzr_5_u_2_ins_0	580.92	642.07	1544.48	1100.84	268.81	605.88	377.4	885.03	299.09	591.84	602.74	935.75
IP_obj_3_var_20_nzr_6_u_1_ins_0	26.01	71.94	126.54	243.81	192.24	104.03	140.88	204.64	34.27	127.04	171.92	160.98
IP_obj_3_var_25_nzr_4_u_1_ins_0	155.55	222.51	812.3	2027.62	206.08	192.5	290.07	956.6	268.17	467.61	381.51	832.5
IP_obj_3_var_25_nzr_6_u_2_ins_0	4451.83	6337.97	9186.83	13048.8	5420.88	6932.79	7092.48	10990.12	-	8009.52	10802.74	11279.86
IP_obj_4_var_10_nzr_4_u_1_ins_0	8.39	337.68	-	-	8.95	333.89	4494.95	-	8.74	346.76	4310.16	-
IP_obj_4_var_10_nzr_4_u_2_ins_0	31.81	348.89	4252.56	-	31.95	373.33	4242.09	-	28.57	417.22	4433.01	-
IP_obj_4_var_10_nzr_5_u_1_ins_0	8.46	315.53	4413.26	-	10.76	336.25	4259.51	-	9.04	357.25	4272.15	-
IP_obj_4_var_10_nzr_5_u_2_ins_0	23.37	394.37	4318.58	-	23.22	344.61	3856.05	-	23.42	386.42	3855.94	-
IP_obj_4_var_10_nzr_6_u_1_ins_0	11.54	353.31	4487.88	-	12.28	341.16	4341.69	-	12.72	336.75	3979.26	-
IP_obj_4_var_10_nzr_6_u_2_ins_0	17.48	344.6	4315.9	-	20.75	340.06	4210.75	-	21.5	353.83	4305.74	-
IP_obj_4_var_15_nzr_4_u_1_ins_0	232.9	716.66	5145.2	-	251.96	684.74	5503.28	-	251.52	628.93	5248.07	-
IP_obj_4_var_15_nzr_4_u_2_ins_0	395.72	1003.77	5415.02	-	403.97	1422.79	6865.68	-	379.3	978.18	7021.48	-
IP_obj_4_var_15_nzr_5_u_1_ins_0	629.13	1761.12	7782.65	-	621.99	1193.29	7183.83	-	1079.64	1688.17	7820.81	-
IP_obj_4_var_15_nzr_5_u_2_ins_0	2618.39	6345.72	13442.28	-	4002.01	4090.41	12373.72	-	4790.53	8082.11	-	-
IP_obj_4_var_15_nzr_6_u_1_ins_0	212.9	771.14	7109.79	-	242.27	981.53	6853.79	-	5728.02	1405.14	-	-
IP_obj_4_var_15_nzr_6_u_2_ins_0	735.44	1999.15	8041.71	-	962.12	1758.88	7604.2	-	1058.5	2445.05	8321.22	-
IP_obj_4_var_20_nzr_5_u_1_ins_0	884.76	2840	14230.64	-	950.96	3156.92	13180.51	-	1071.1	3034.1	13503.54	-
IP_obj_4_var_20_nzr_5_u_2_ins_0	-	-	-	-	-	-	-	-	-	-	-	-
IP_obj_5_var_10_nzr_4_u_1_ins_0	2607.36	-	-	-	2288.56	-	-	-	2426.02	-	-	-
IP_obj_5_var_10_nzr_4_u_2_ins_0	2744.68	-	-	-	2276.81	-	-	-	2397.07	-	-	-
IP_obj_5_var_10_nzr_5_u_1_ins_0	2188.12	-	-	-	2196.37	-	-	-	2661.82	-	-	-
IP_obj_5_var_10_nzr_5_u_2_ins_0	2686.87	-	-	-	2470.15	-	-	-	2429.63	-	-	-
IP_obj_5_var_10_nzr_6_u_1_ins_0	2438.6	-	-	-	2550.89	-	-	-	2313.03	-	-	-
IP_obj_5_var_10_nzr_6_u_2_ins_0	2156.94	-	-	-	2455.08	-	-	-	2164.66	-	-	-
IP_obj_5_var_15_nzr_4_u_1_ins_0	7142.4	-	-	-	7762.53	-	-	-	9018.07	-	-	-
IP_obj_5_var_15_nzr_4_u_2_ins_0	10683.43	-	-	-	9200.71	-	-	-	13343.87	-	-	-
IP_obj_5_var_15_nzr_5_u_1_ins_0	2325.92	-	-	-	2513.41	-	-	-	2692.27	-	-	-
IP_obj_5_var_15_nzr_6_u_2_ins_0	5757.34	-	-	-	6660.51	-	-	-	7745.55	-	-	-
IP_obj_5_var_20_nzr_4_u_1_ins_0	11919.74	-	-	-	12562.7	-	-	-	-	-	-	-

Table M.9: Average running time (in seconds) for MILP instances solved by the BB/sample-based algorithm with an RVF call frequency of 5 and varying initial sample sizes

Instance	5_1		5_2		5_3		5_4		5_5		5_10		5_20	
	Time	# Stab	Time	# Stab	Time	# Stab	Time	# Stab	Time	# Stab	Time	# Stab	Time	# Stab
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_0	517.03	42	224.57	39	511.99	42	445.55	42	550.81	42	401.05	42	104.60	32
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_10	395.94	42	316.75	42	285.68	42	367.79	42	253.06	42	20.04	22	194.48	42
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_2	191.75	32	126.70	30	184.42	32	215.51	32	185.88	32	177.87	32	71.84	27
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_3	691.09	49	80.56	33	562.14	48	673.84	49	649.21	48	592.29	48	581.39	47
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_4	126.87	29	152.91	29	146.38	29	169.90	29	177.32	29	150.01	29	169.81	29
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_5	115.82	25	134.24	25	118.82	25	121.75	25	126.11	25	131.00	25	116.48	25
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_0	24.88	17	24.16	17	21.88	17	23.93	17	23.59	17	22.46	17	22.26	17
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_10	23.54	11	273.99	19	330.92	19	19.01	11	327.40	18	253.39	18	353.92	18
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_3	23.56	11	268.56	19	330.55	19	19.54	11	335.66	18	253.50	18	360.93	18
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_6	116.96	26	99.70	26	116.29	28	107.66	26	88.97	26	83.98	27	85.76	26
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_9	407.36	47	386.38	48	551.15	47	332.42	46	315.55	46	440.48	46	426.14	47
MILP_obj_2_var_30_nzr_6_intR_9_u_1_ins_10	46.48	21	45.85	21	38.31	20	45.26	21	47.51	22	46.79	22	37.05	21
MILP_obj_2_var_30_nzr_6_intR_9_u_1_ins_6	72.10	16	66.48	16	59.97	16	61.81	16	62.54	16	57.87	16	78.78	16
MILP_obj_2_var_30_nzr_7_intR_9_u_1_ins_11	279.24	37	297.59	34	314.67	35	335.48	37	314.11	36	315.28	35	303.47	37
MILP_obj_2_var_30_nzr_7_intR_9_u_1_ins_2	187.51	22	241.65	21	159.16	24	167.12	22	165.06	23	127.33	23	141.39	24
MILP_obj_2_var_30_nzr_7_intR_9_u_1_ins_8	309.97	34	319.24	34	260.07	34	267.42	34	228.29	34	119.14	30	262.97	34
MILP_obj_2_var_30_nzr_8_intR_9_u_1_ins_11	73.14	16	68.91	16	59.06	16	61.15	16	62.76	16	58.29	16	78.34	16
MILP_obj_2_var_30_nzr_8_intR_9_u_1_ins_2	574.30	30	460.58	29	541.35	31	444.92	30	574.17	31	523.09	31	429.96	31
MILP_obj_2_var_30_nzr_8_intR_9_u_1_ins_4	342.20	48	41.09	30	389.17	48	347.96	48	354.53	48	399.03	48	361.19	48
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_1	204.60	32	187.60	32	153.77	32	199.93	32	205.23	32	174.27	32	131.51	32
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_11	192.67	32	232.02	32	202.15	32	195.19	32	195.40	32	225.31	32	22.19	17
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_5	225.90	25	266.23	28	182.59	26	275.07	26	226.47	27	197.45	26	220.60	26
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_6	25.50	26	20.71	25	138.32	34	122.07	35	145.26	34	145.37	34	161.45	35
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_7	169.74	26	162.31	26	170.53	26	138.92	25	170.21	26	174.41	26	159.78	26
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_9	144.02	34	153.74	34	152.97	34	143.81	34	148.46	34	149.68	34	146.04	34
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_10	29.74	15	30.36	15	23.77	15	26.25	15	41.57	15	37.44	15	25.04	16
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_2	64.94	27	118.98	33	48.07	24	87.17	30	23.43	20	215.95	37	125.52	33
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_5	829.01	52	731.31	52	738.05	52	781.62	52	645.92	51	837.52	51	734.82	51
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_6	564.61	44	770.24	47	743.50	46	616.92	45	821.79	45	765.36	43	511.65	43
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_7	10.18	11	121.47	25	95.86	25	142.24	25	42.86	19	42.27	20	139.68	27
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_8	53.24	36	155.23	48	68.02	37	1296.14	76	89.39	40	296.45	56	750.03	69
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_9	181.75	36	201.41	36	143.17	36	178.98	36	184.23	36	122.33	35	162.57	36
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_10	688.56	40	575.10	39	580.66	40	604.73	41	666.19	40	533.28	40	410.57	40
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_5	540.43	47	616.46	47	499.83	47	507.94	47	503.03	47	537.77	47	496.51	47
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_6	1024.58	42	1120.17	43	925.18	42	1058.21	43	357.57	34	672.56	40	298.21	33
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_8	72.18	22	74.85	22	75.51	22	73.08	22	81.57	22	81.77	22	82.37	22
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_9	3407.41	67	3132.60	67	2234.27	67	2963.17	67	4296.62	67	4401.38	68	5113.28	68
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_0	2347.79	52	2592.04	52	2324.42	52	2308.11	51	2436.45	52	2062.52	51	2060.07	51
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_10	1921.69	49	1818.63	47	1579.60	47	1937.03	46	1837.47	45	2042.56	46	1582.69	48
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_2	393.71	30	399.53	30	354.51	30	380.15	30	448.06	30	369.45	30	347.47	30
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_7	1472.95	61	1232.53	61	1039.72	61	1253.94	62	1193.56	62	1146.72	62	1135.92	62
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_9	1356.90	49	1328.03	48	1319.38	48	1446.49	48	1365.82	48	1340.01	48	1202.04	48
MILP_obj_2_var_40_nzr_9_intR_95_u_1_ins_10	3692.86	44	3618.49	43	2891.95	43	3383.21	43	3689.80	45	3484.17	44	2734.05	43
MILP_obj_2_var_40_nzr_9_intR_95_u_1_ins_2	2779.01	34	2634.84	34	3139.31	34	2565.43	34	2622.92	34	3037.81	34	2321.19	34

Table M.10: Average running time (in seconds) for MILP instances solved by the BB/sample-based algorithm with an RVF call frequency of 10 and varying initial sample sizes

Instance	10_1		10_2		10_3		10_4		10_5		10_10		10_20	
	Time	# Stab	Time	# Stab	Time	# Stab	Time	# Stab	Time	# Stab	Time	# Stab	Time	# Stab
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_0	278.82	39	213.55	38	65.247	28	294.96	40	385.74	41	219.12	39	458.63	42
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_10	314.88	42	255.18	42	295.23	42	271.99	42	269.19	42	9.9363	18	229.47	42
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_2	223.37	32	181.44	32	196.98	32	194.5	32	200.93	32	171.87	32	195.35	32
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_3	641.81	49	555.61	49	552.02	48	541.59	49	554.05	49	127.97	40	639.88	48
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_4	139.77	29	137.94	29	98.89	27	120.67	29	149.73	29	130.22	29	138.61	29
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_5	122.28	25	110.16	25	98.369	25	126.52	25	118.76	25	110.41	25	143.14	25
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_0	28.776	17	27.901	17	29.464	17	28.953	17	25.149	17	23.342	17	26.619	17
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_10	397.16	18	297.5	18	165.01	18	413.47	18	246.87	18	273.15	18	264.22	18
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_3	394.63	18	296.79	18	166.51	18	402.78	18	248.38	18	269.57	18	264.1	18
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_6	78.619	26	87.625	26	96.983	28	80.194	26	96.267	26	101.41	27	106.2	27
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_9	415.94	46	398.73	47	465.49	47	457.5	47	375.19	45	392.07	46	444.96	47
MILP_obj_2_var_30_nzr_6_intR_9_u_1_ins_10	56.537	22	57.163	22	42.244	21	49.763	22	53.175	22	45.919	22	34.946	21
MILP_obj_2_var_30_nzr_6_intR_9_u_1_ins_6	79.308	16	72.911	16	48.474	16	67.934	16	77.419	16	64.864	16	85.259	16
MILP_obj_2_var_30_nzr_7_intR_9_u_1_ins_11	326.41	37	304.52	37	378.04	36	249.56	36	280.05	36	335.5	36	251.52	36
MILP_obj_2_var_30_nzr_7_intR_9_u_1_ins_2	236.04	24	177.76	24	180.77	23	153.25	23	199.88	24	257.19	24	150.91	24
MILP_obj_2_var_30_nzr_7_intR_9_u_1_ins_8	296.94	34	278.84	34	267.93	34	266.43	34	233.89	34	77.705	27	197.75	34
MILP_obj_2_var_30_nzr_8_intR_9_u_1_ins_11	79.989	16	72.846	16	48.923	16	67.727	16	77.63	16	65.172	16	84.728	16
MILP_obj_2_var_30_nzr_8_intR_9_u_1_ins_2	508.82	31	539.14	31	371.36	31	519.66	30	462.52	31	481.6	31	453.47	31
MILP_obj_2_var_30_nzr_8_intR_9_u_1_ins_4	324.5	48	362.7	48	314.46	48	278.84	48	318.67	48	416.82	48	351.47	48
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_1	174.66	32	167.97	32	209.58	32	151.67	32	187.97	32	133.09	32	154.97	32
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_11	179.6	32	210.35	32	223.16	32	205.14	32	206.2	32	198.06	32	201.01	32
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_5	235.64	27	159.75	26	192.8	25	147.85	26	212.86	26	220.35	27	175.11	28
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_6	15.666	25	142.5	35	124.66	34	131.95	34	145.9	36	153.94	36	161.03	35
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_7	141.44	26	146	26	136.5	26	144.02	26	155.19	26	161.05	26	169.89	26
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_9	143.72	34	116.49	34	144.17	34	144.22	34	160.33	34	170.72	34	166.22	34
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_10	29.157	15	30.773	15	29.075	15	27.972	15	34.244	16	24.849	15	28.885	16
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_2	85.493	31	86.351	32	34.54	21	198.5	37	225.84	36	208.52	37	39.447	23
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_5	792.17	52	799.8	53	725.19	53	985.99	53	660.97	51	712.38	51	660.41	51
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_6	489.52	47	516.43	47	518.21	45	710.6	48	741.77	45	599.36	45	578.11	45
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_7	109.54	26	28.59	18	41.936	22	126.84	26	131.33	26	124.12	26	136.79	27
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_8	85.719	43	1560.5	77	1126.6	78	262.53	56	851.33	72	243.29	56	1173.1	75
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_9	191.35	36	143.98	36	166.18	36	161.19	36	170.03	36	150.71	36	156.79	36
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_10	764.44	41	533.13	40	492.8	41	579.47	40	584.7	41	465.4	40	521.52	40
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_5	586.93	47	557.15	47	592.19	47	541.34	47	516.8	47	451.49	47	551.82	47
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_6	1129	44	1005.3	44	1028.3	44	1145.5	43	930.07	44	430.09	38	1108.3	42
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_8	82.319	22	74.899	22	81.353	22	72.077	22	82.662	22	83.025	22	92.099	22
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_9	2589.6	67	2330.7	69	4039.3	69	2704.4	70	2637.5	69	3778	68	4240.5	67
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_0	2545.3	52	2311.3	51	2283.3	52	2568.9	51	2168.4	51	979.28	48	2063.9	51
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_10	1552	45	1831.2	46	1437.3	49	1445.6	44	1270.4	48	1508.8	49	1322.1	47
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_2	398.95	30	336.94	30	471.31	30	364.43	30	367.12	30	320.84	30	346.56	30
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_7	1127.4	61	1170.2	62	1157.2	61	969.54	62	1184.8	62	1311.4	61	1026.6	61
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_9	1254	48	1281.2	48	979.52	48	1311	48	1512.8	48	1365	48	1342.5	48
MILP_obj_2_var_40_nzr_9_intR_95_u_1_ins_10	3219.2	45	3178.8	42	3784.4	43	3399.5	45	3533.5	44	3524.5	44	3295.3	44
MILP_obj_2_var_40_nzr_9_intR_95_u_1_ins_2	2458.3	34	2776.3	34	2415.8	34	2212.9	34	2267.9	34	2104.2	34	2480.5	34

Table M.11: Average running time (in seconds) for MILP instances solved by the BB/sample-based algorithm with an RVF call frequency of 50 and varying initial sample sizes

Instance	50_1		50_2		50_3		50_4		50_5		50_10		50_20	
	Time	# Stab	Time	# Stab	Time	# Stab	Time	# Stab	Time	# Stab	Time	# Stab	Time	# Stab
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_0	257.54	38	401.05	42	67.267	34	217.5	38	501.08	42	555.94	42	594.27	42
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_10	311.53	42	329.86	42	277.1	42	288.46	42	245.99	42	232.44	42	294.1	42
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_2	212.75	32	234.44	32	194.48	32	248.18	32	202.63	32	176.12	32	184.45	32
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_3	542.45	49	534.78	49	507.42	48	568.27	49	640.74	49	667.86	49	616.42	49
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_4	119.93	29	136.68	29	203.02	29	137.91	29	54.463	27	142.28	29	143.43	29
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_5	97.181	25	101.46	25	116.21	25	92.697	25	60.884	25	64.03	25	71.429	25
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_0	39.613	17	40.559	17	50.789	17	41.565	17	40.464	17	32.371	17	33.914	17
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_10	142.96	19	141.02	18	305.62	18	121.22	18	125.62	18	112.55	18	283.96	18
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_3	142.9	19	141	18	306.53	18	121.17	18	125.32	18	112.2	18	289.41	18
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_6	92.212	28	86.452	27	85.338	27	85.601	27	59.675	26	91.809	28	70.275	26
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_9	318.72	51	343.56	50	463.79	50	407.53	47	317.18	49	289.29	48	259.65	50
MILP_obj_2_var_30_nzr_6_intR_9_u_1_ins_10	59.882	21	45.898	21	70.964	22	47.13	21	64.815	22	54.806	22	54.551	21
MILP_obj_2_var_30_nzr_6_intR_9_u_1_ins_6	48.571	16	57.048	16	44.306	16	57.154	16	41.182	16	47.983	16	70.874	16
MILP_obj_2_var_30_nzr_7_intR_9_u_1_ins_11	258.33	37	379.16	38	354.83	36	313.02	38	201.62	36	321.75	37	221.95	36
MILP_obj_2_var_30_nzr_7_intR_9_u_1_ins_2	188.96	24	139.75	23	203.88	24	154.04	24	200.75	24	171.26	24	150.25	24
MILP_obj_2_var_30_nzr_7_intR_9_u_1_ins_8	320.84	34	352.18	34	306.26	34	310.87	34	242.08	34	227.11	34	198.5	34
MILP_obj_2_var_30_nzr_8_intR_9_u_1_ins_11	48.581	16	56.642	16	44.271	16	57.822	16	41.232	16	48.271	16	71.163	16
MILP_obj_2_var_30_nzr_8_intR_9_u_1_ins_2	468.79	32	366.83	32	457.84	32	395.33	32	434.42	31	463.45	33	348.89	31
MILP_obj_2_var_30_nzr_8_intR_9_u_1_ins_4	383.57	48	415.34	48	414.3	48	452.47	48	414.96	48	432.18	48	452.31	48
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_1	276.49	32	252.72	32	246.76	32	261.47	32	241.65	32	221.04	32	270.75	32
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_11	271.57	32	235.93	32	256.69	32	285.51	32	259.83	32	278.88	32	252.97	32
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_5	184.5	29	199.92	29	217.56	27	223.08	29	231.61	29	225.43	29	215.64	29
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_6	107.32	34	115.03	35	144.16	36	117.24	35	158.01	35	179.42	36	14.819	22
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_7	138.09	27	142.88	27	173.01	26	118.76	27	152.47	26	177.18	26	167.81	26
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_9	191.13	34	187.36	34	164.35	34	193.45	34	158.95	34	212.39	34	206.58	34
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_10	37.71	15	38.316	15	23.025	15	39.064	15	46.415	16	25.673	15	37.161	16
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_2	201.45	36	259.07	37	325.7	37	161.23	36	336.36	37	99.675	31	261.81	37
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_5	762.15	53	895.95	53	842.16	53	811.23	53	766.93	52	681.52	52	635.79	52
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_6	478.02	48	608.6	49	612.06	45	472.11	48	494.9	48	480.53	45	367.46	48
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_7	93.38	26	90.568	26	172.16	26	126.04	26	118.66	26	120.28	26	140.2	27
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_8	405.45	67	252.42	61	1282.5	79	304.9	64	233.71	65	145.6	52	348.83	66
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_9	213.87	36	239.17	36	166.68	36	208.62	36	188.63	36	182.52	36	199.56	36
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_10	721.04	42	548.53	42	412.86	40	590.46	42	569.85	41	349.88	40	541.06	41
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_5	687.31	47	701.16	47	694.64	47	597.85	47	834.8	47	729.7	47	736.15	47
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_6	975.45	44	979.3	43	998.19	43	916	42	852.63	43	642.42	45	614.05	44
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_8	143.7	22	147.53	22	120.4	22	149.7	22	124.47	22	126.05	22	128	22
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_9	3578.9	71	2532.8	68	2531.4	72	2704.8	70	2881.8	72	2451.1	70	2615.2	72
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_0	2201.8	52	2317.2	51	2171	51	613.11	43	2024.8	51	1907.5	50	1449	50
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_10	1684.9	50	1385.7	49	1076.5	51	1171.2	50	1115.6	52	1087.4	51	1335	52
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_2	388.38	30	485.8	30	470.65	30	404.5	30	379.86	30	377.75	30	332.07	30
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_7	1249.6	61	1193.9	61	1405.5	61	1468.7	61	1240.9	61	1385.5	61	1289.3	61
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_9	1154.8	48	1279.1	48	1127.4	48	1353.3	48	1467.5	48	1375.4	48	1127	48
MILP_obj_2_var_40_nzr_9_intR_95_u_1_ins_10	2980.4	45	3130.9	44	2909	44	2976.4	44	2978.1	44	2737.8	44	2218.8	44
MILP_obj_2_var_40_nzr_9_intR_95_u_1_ins_2	1404.1	35	1335.5	35	2006.6	35	1542.9	35	1643.1	35	1650.3	35	1745.9	35

Table M.12: Average running time (in seconds) for MILP instances solved by the BB/sample-based algorithm with an RVF call frequency of 100 and varying initial sample sizes

Instance	100_1		100_2		100_3		100_4		100_5		100_10		100_20	
	Time	# Stab	Time	# Stab	Time	# Stab	Time	# Stab	Time	# Stab	Time	# Stab	Time	# Stab
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_0	446.94	42	713.86	42	518.81	42	523.85	42	400.21	40	605.72	42	681.08	42
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_10	294.05	42	289.53	42	350.92	42	358.59	42	407.85	42	375.11	42	363.05	42
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_2	234.94	32	189.31	32	309.95	32	197.34	32	255.90	32	209.68	32	220.16	32
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_3	757.31	49	608.06	49	506.28	49	712.08	49	632.60	49	740.53	49	619.11	49
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_4	116.86	29	191.86	29	223.51	29	134.67	29	120.29	29	126.94	29	129.86	29
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_5	108.93	25	71.71	25	123.90	25	74.80	25	71.17	25	74.02	25	78.41	25
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_0	57.95	17	59.37	17	58.04	17	60.03	17	58.63	17	60.34	17	61.75	17
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_10	133.90	18	145.53	18	246.75	18	148.69	18	112.09	18	106.72	18	254.00	18
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_3	133.32	18	145.12	18	249.58	18	149.41	18	112.11	18	106.42	18	251.95	18
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_6	123.99	29	114.36	28	70.34	27	117.04	28	105.73	29	74.33	28	79.70	28
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_9	283.81	51	326.66	51	425.61	52	323.74	50	357.36	53	233.75	50	373.35	51
MILP_obj_2_var_30_nzr_6_intR_9_u_1_ins_10	83.27	23	53.58	21	60.80	22	54.85	21	86.50	23	58.63	21	63.55	21
MILP_obj_2_var_30_nzr_6_intR_9_u_1_ins_6	65.71	16	66.78	16	49.95	16	71.76	16	71.43	16	74.31	16	83.15	16
MILP_obj_2_var_30_nzr_7_intR_9_u_1_ins_11	309.08	38	336.35	38	440.93	36	297.71	38	337.00	38	433.65	39	318.49	39
MILP_obj_2_var_30_nzr_7_intR_9_u_1_ins_2	229.01	24	153.48	24	234.44	24	205.30	24	162.70	23	191.27	24	185.40	24
MILP_obj_2_var_30_nzr_7_intR_9_u_1_ins_8	411.47	34	374.10	34	246.88	34	363.44	34	263.70	34	259.92	34	276.98	34
MILP_obj_2_var_30_nzr_8_intR_9_u_1_ins_11	65.16	16	65.36	16	50.52	16	71.69	16	72.02	16	74.32	16	84.03	16
MILP_obj_2_var_30_nzr_8_intR_9_u_1_ins_2	400.49	31	352.73	32	493.31	32	494.63	34	353.94	32	433.10	31	407.44	32
MILP_obj_2_var_30_nzr_8_intR_9_u_1_ins_4	540.04	48	423.20	48	525.58	48	522.27	48	486.42	48	583.77	48	508.99	48
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_1	328.41	32	359.82	32	338.18	32	375.22	32	292.02	32	304.47	32	278.36	32
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_11	345.71	32	273.83	32	289.14	32	303.72	32	353.59	32	348.18	32	334.88	32
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_5	229.76	29	182.23	29	205.38	27	256.49	30	278.98	29	245.50	29	292.16	29
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_6	164.69	35	164.77	36	165.80	36	174.61	35	212.64	36	226.93	36	27.80	25
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_7	156.60	27	163.17	27	159.24	26	169.26	27	157.22	26	157.94	26	159.14	26
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_9	211.86	34	203.20	34	266.51	34	159.69	34	199.74	34	200.03	34	210.70	34
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_10	40.79	15	41.66	15	39.83	15	44.01	15	47.66	16	45.05	15	28.13	16
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_2	111.02	34	286.28	37	104.58	32	296.34	37	351.86	37	367.98	37	350.07	37
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_5	1006.86	53	965.00	53	862.76	54	980.40	53	888.99	52	695.98	52	790.40	52
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_6	585.48	49	553.13	49	664.74	45	594.73	49	441.18	49	408.22	49	534.34	49
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_7	189.18	27	132.63	26	168.16	26	143.06	26	143.08	26	100.47	26	150.67	27
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_8	1486.23	79	1419.14	79	1832.85	80	1631.63	79	1340.69	77	1428.61	77	346.35	66
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_9	279.46	36	277.79	36	238.66	36	205.78	36	291.67	36	278.37	36	220.69	36
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_10	670.72	42	435.13	41	440.05	40	620.77	42	511.87	41	675.65	41	725.13	42
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_5	731.05	47	702.37	47	808.26	47	701.59	47	899.04	47	1031.05	47	907.11	47
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_6	823.66	44	679.57	43	786.79	45	1019.88	44	246.33	43	775.36	45	355.08	44
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_8	234.15	22	237.81	22	198.07	22	238.27	22	155.45	22	157.60	22	162.16	22
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_9	2619.69	71	2799.33	71	3444.44	72	3101.08	71	3001.01	72	2635.24	71	2592.98	70
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_0	2448.68	51	2491.40	52	2466.69	50	705.50	43	2305.16	51	2099.35	51	2114.27	51
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_10	1054.03	50	1163.06	52	925.33	50	996.60	49	1370.13	53	1396.24	51	1376.83	54
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_2	502.91	30	501.29	30	537.13	30	593.96	30	601.53	30	429.11	30	516.53	30
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_7	1690.54	61	1224.29	61	1641.63	61	1788.19	61	1228.21	61	1281.75	61	1169.87	61
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_9	1334.56	48	1393.92	48	1099.68	48	1450.39	48	1326.73	48	3379.99	48	1578.14	48
MILP_obj_2_var_40_nzr_9_intR_95_u_1_ins_10	3081.12	44	2476.48	45	2687.82	44	2237.14	43	2616.49	44	2840.21	46	2955.49	44
MILP_obj_2_var_40_nzr_9_intR_95_u_1_ins_2	1243.45	35	979.14	35	1306.79	35	953.19	35	1344.11	36	1189.58	36	1449.68	35

Table M.13: Summary of all data used in the deep dive analysis of MILP cases

Instance	Iter.		Total Time (s)		Avg. Time per Iter. (s)		Component Times (s)			Time per Node (s)	
	BB	RVF	BB	RVF	BB	RVF	Build	Eval	Sym	BB	RVF
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_0	53	12	2.590	372.642	0.049	31.054	0.047	3.921	2.543	0.004	0.007
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_10	54	8	3.337	148.296	0.062	18.537	0.066	4.556	3.271	0.003	0.006
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_2	39	8	2.309	77.764	0.059	9.720	0.043	6.109	2.266	0.003	0.005
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_3	60	12	3.001	354.150	0.050	29.512	0.051	9.387	2.950	0.005	0.008
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_4	31	6	1.203	71.041	0.039	11.840	0.015	1.462	1.189	0.004	0.007
MILP_obj_2_var_30_nzr_4_intR_95_u_1_ins_5	30	4	0.978	36.062	0.033	9.015	0.011	0.457	0.967	0.004	0.004
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_0	22	4	0.482	8.550	0.022	2.138	0.004	0.109	0.479	0.004	0.003
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_10	20	4	0.535	254.263	0.027	63.566	0.005	0.315	0.530	0.004	0.002
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_3	20	4	0.536	260.081	0.027	65.020	0.005	0.317	0.531	0.004	0.002
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_6	32	4	1.164	33.053	0.036	8.263	0.015	0.900	1.148	0.003	0.004
MILP_obj_2_var_30_nzr_5_intR_95_u_1_ins_9	50	6	3.112	144.307	0.062	24.051	0.053	4.125	3.058	0.004	0.008
MILP_obj_2_var_30_nzr_6_intR_9_u_1_ins_10	25	4	0.636	19.632	0.025	4.908	0.006	0.184	0.630	0.004	0.004
MILP_obj_2_var_30_nzr_6_intR_9_u_1_ins_6	15	5	0.438	40.522	0.029	8.104	0.003	0.190	0.435	0.003	0.002
MILP_obj_2_var_30_nzr_7_intR_9_u_1_ins_11	47	7	2.841	110.547	0.060	15.792	0.049	2.919	2.791	0.003	0.007
MILP_obj_2_var_30_nzr_7_intR_9_u_1_ins_2	30	6	0.836	90.057	0.028	15.009	0.010	0.973	0.826	0.004	0.003
MILP_obj_2_var_30_nzr_7_intR_9_u_1_ins_8	40	7	1.422	100.458	0.036	14.351	0.021	1.219	1.401	0.004	0.004
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_11	41	10	1.400	116.369	0.034	11.637	0.021	1.992	1.379	0.004	0.007
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_1	39	10	1.584	126.972	0.041	12.697	0.027	3.403	1.557	0.003	0.005
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_5	37	7	1.780	127.513	0.048	18.216	0.032	4.943	1.748	0.003	0.006
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_6	22	2	0.696	2.333	0.032	1.167	0.006	0.090	0.690	0.004	0.007
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_7	35	8	0.896	81.957	0.026	10.245	0.009	0.728	0.887	0.005	0.009
MILP_obj_2_var_30_nzr_8_intR_95_u_1_ins_9	36	9	2.029	77.095	0.056	8.566	0.037	3.576	1.991	0.003	0.008
MILP_obj_2_var_30_nzr_8_intR_9_u_1_ins_11	15	5	0.447	40.591	0.030	8.118	0.003	0.195	0.443	0.003	0.002
MILP_obj_2_var_30_nzr_8_intR_9_u_1_ins_2	32	5	1.255	281.889	0.039	56.378	0.015	1.293	1.239	0.003	0.004
MILP_obj_2_var_30_nzr_8_intR_9_u_1_ins_4	57	13	2.874	183.189	0.050	14.091	0.051	3.850	2.823	0.004	0.008
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_10	17	3	0.425	18.384	0.025	6.128	0.003	0.102	0.423	0.004	0.004
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_2	35	10	1.429	105.547	0.041	10.555	0.018	1.611	1.411	0.004	0.007
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_5	57	9	4.452	426.370	0.078	47.374	0.085	6.666	4.366	0.004	0.010
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_6	59	7	4.007	230.922	0.068	32.989	0.074	5.243	3.933	0.004	0.008
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_7	32	7	1.014	64.541	0.032	9.220	0.013	1.029	1.001	0.003	0.004
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_8	75	8	10.703	118.503	0.143	14.813	0.241	20.516	10.462	0.005	0.016
MILP_obj_2_var_30_nzr_9_intR_95_u_1_ins_9	39	8	1.974	79.726	0.051	9.966	0.031	2.558	1.943	0.003	0.007
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_10	51	8	3.713	295.287	0.073	36.911	0.064	10.458	3.649	0.005	0.011
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_5	61	14	5.474	265.627	0.090	18.973	0.106	18.282	5.367	0.005	0.009
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_6	46	8	3.649	388.169	0.079	48.521	0.057	7.474	3.593	0.005	0.013
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_8	27	7	0.949	31.424	0.035	4.489	0.008	0.496	0.941	0.005	0.005
MILP_obj_2_var_40_nzr_4_intR_95_u_1_ins_9	79	15	10.959	1792.696	0.139	119.513	0.222	66.683	10.737	0.006	0.017
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_0	69	12	9.674	976.593	0.140	81.383	0.202	37.607	9.472	0.005	0.007
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_10	59	10	6.016	954.099	0.102	95.410	0.106	21.729	5.910	0.005	0.012
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_2	28	8	1.349	181.057	0.048	22.632	0.011	1.416	1.338	0.005	0.005
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_7	78	16	8.383	604.468	0.107	37.779	0.148	28.246	8.235	0.006	0.014
MILP_obj_2_var_40_nzr_8_intR_95_u_1_ins_9	64	13	5.092	690.869	0.080	53.144	0.087	14.430	5.005	0.005	0.010
MILP_obj_2_var_40_nzr_9_intR_95_u_1_ins_10	49	15	3.829	1739.436	0.078	115.962	0.056	16.735	3.774	0.006	0.007
MILP_obj_2_var_40_nzr_9_intR_95_u_1_ins_2	42	10	2.955	1499.220	0.070	149.922	0.049	13.464	2.906	0.004	0.004

Appendix N

Base Performance Profile for the Computational Experiments in Chapter 4

In this appendix, we present the base performance profile for the computational experiments of Chapter 4.

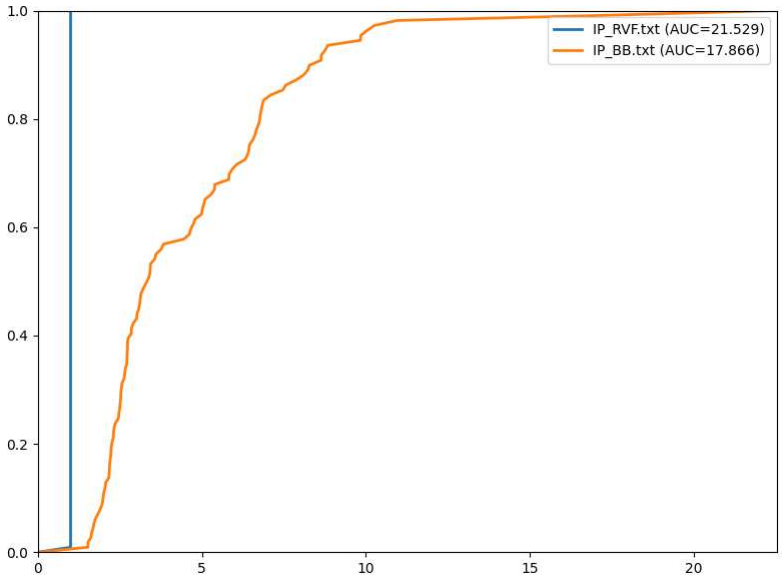


Figure N.1: Base performance profile comparing the execution times of the BB Algorithm and the RVF Algorithm (with Implementation 2) for all PIP instances, including SPP, KP, and randomly generated PIP instances.

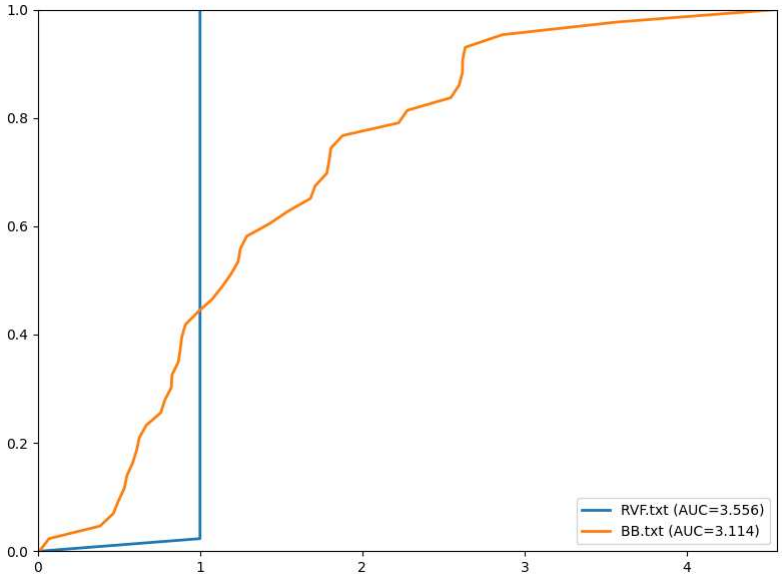


Figure N.2: Base performance profile comparing the execution times of RVF and BB for all MILPs.

Vita

Samira Fallah is a Ph.D. candidate in the Department of Industrial and Systems Engineering at Lehigh University. She earned her Bachelor's and Master's degrees in Industrial Engineering from Sharif University of Technology in Iran in 2015 and 2017, respectively. In 2018, she was awarded an ISE Department Fellowship from Lehigh University and has served as a teaching assistant since 2019. She has also completed multiple internships in data science and operations research. She is currently an Applied Scientist at Uber.