



**Titre:** Peel and Bound: Solving Discrete Optimization Problems with  
Title: Decision Diagrams and Separation

**Auteur:** Isaac Rue Rudich  
Author:

**Date:** 2024

**Type:** Mémoire ou thèse / Dissertation or Thesis

**Référence:** Rudich, I. R. (2024). Peel and Bound: Solving Discrete Optimization Problems with  
Citation: Decision Diagrams and Separation [Thèse de doctorat, Polytechnique Montréal].  
PolyPublie. <https://publications.polymtl.ca/59205/>

 **Document en libre accès dans PolyPublie**  
Open Access document in PolyPublie

**URL de PolyPublie:** <https://publications.polymtl.ca/59205/>  
PolyPublie URL:

**Directeurs de  
recherche:** Louis-Martin Rousseau  
Advisors:

**Programme:** Doctorat en mathématique  
Program:

**POLYTECHNIQUE MONTRÉAL**

affiliée à l'Université de Montréal

**Peel and Bound: Solving Discrete Optimization Problems with Decision  
Diagrams and Separation**

**ISAAC RUE RUDICH**

Département de mathématiques et de génie industriel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*  
Mathématiques appliquées

Mai 2024

**POLYTECHNIQUE MONTRÉAL**

affiliée à l'Université de Montréal

Cette thèse intitulée :

**Peel and Bound: Solving Discrete Optimization Problems with Decision  
Diagrams and Separation**

présentée par **Isaac Rue RUDICH**

en vue de l'obtention du diplôme de *Philosophiæ Doctor*  
a été dûment acceptée par le jury d'examen constitué de :

**Michel GENDREAU**, président

**Louis-Martin ROUSSEAU**, membre et directeur de recherche

**Quentin CAPPART**, membre et codirecteur de recherche

**Andrea LODI**, membre

**Laurent MICHEL**, membre externe

**DEDICATION**

*“It is exhausting, having to reason all the time  
in a universe which wasn’t meant to be reasonable.”*

*- Breakfast of Champions by Kurt Vonnegut*

## ACKNOWLEDGEMENTS

Many small interactions got me here, and I wish to express my gratitude to those who impacted my path, whether through brief encounters or prolonged engagements. In 2018, I was trying to figure out what to do next on my very unconventional career trajectory. I had no formal math background, but I found myself drawn to solving scheduling problems and seeking advice on my next steps. This search led me to the offices of Michael Trick and Willem-Jan van Hoes, both of whom generously recommended graduate programs and helped clarify my goals. I had never even heard the term *Operations Research* before those conversations, and I doubt I would have discovered this lovely field without them.

My advisors, Louis-Martin Rousseau and Quentin Cappart, significantly shaped my research and provided invaluable guidance throughout my journey. Before even arriving in Canada, Louis-Martin suggested I read *Decision Diagrams for Optimization* to determine if it aligned with my research interests. It did. This book not only captivated my attention but also led me to the 2018 Decision Diagrams Symposium. Although I was merely an observer with little to contribute at the time, I could not have anticipated how instrumental some of the attendees would be in influencing the direction of my thesis.

Reflecting on these crucial interactions, I am grateful to a broader community of peers, colleagues, and mentors who have supported, enlightened, and encouraged me. It is with deep appreciation that I extend my thanks to each of them for enriching my academic and personal growth. In particular, notable conversations with Willem-Jan van Hoes, Michael Römer, Andre Cire, Laurent Michel, Xavier Gillard, Manuel López-Ibáñez, Moira MacNeil, and Vianney Coppé sparked significant insights that greatly enriched my research.

I also cherish the deep friendships formed along the way. Augustin Parjadis, thank you for easing my transition to a French school, and for countless hours spent playing board games. Anthony Karahalios, you've been a dear friend, as well as an excellent sounding board for my myriad ideas. Dani Ripsman, though our collaboration and friendship are relatively new, they feel longstanding.

Special thanks to my family members Avi Rudich and Rachel Rue for the weekly Decision Diagram journal club, which evolved into a *Trying to Solve Stochastic Games* club, that has met nearly every week for four years.

Also thanks to Robin for being supportive. None of this would've been possible without my cats: Sir Sqwonk and Countess Pizazz. And finally to Zenith, for being part of the journey.

## RÉSUMÉ

Cette thèse présente des avancées significatives dans la théorie et l'implémentation des méthodes d'optimisation basées sur les diagrammes de décision : tout d'abord le développement et l'amélioration de l'algorithme Peel-and-Bound (PnB) comme une étude sur la construction d'un solveur utilisant des diagrammes de décision (DD), et l'introduction de DD relaxés implicites qui généralise une idée cruciale pour l'implémentation efficace de PnB.

Les DD relaxés sont un outil graphique pour construire des bornes de relaxation combinatoires pour des problèmes d'optimisation. L'algorithme PnB démontre le potentiel puissant des solveurs basés sur les DD exploitant la séparation, une méthode de construction de DD relaxés partant d'un petit DD représentant une relaxation initiale faible qui est ensuite améliorée par division de nœuds. PnB est un algorithme hautement parallélisable, construit autour de la production de DD relaxés par séparation, qui peut échanger toute la mémoire disponible contre de l'efficacité de calcul.

L'implémentation de PnB a été testée sur les 467 instances de référence du problème du voyageur de commerce avec fenêtre de temps (TSP-TW) utilisées pour tester le solveur branch-and-bound (BnB) basé sur les DD. Même lorsque BnB utilise le calcul parallèle, PnB surpasse BnB. De plus, PnB a résolu 15 instances qui, à notre connaissance, étaient ouvertes dans la littérature. Dans notre test final de PnB, nous avons exécuté la nouvelle implémentation de PnB sur la variante du makespan des 467 instances du TSP-TW. PnB a résolu 94 % des instances de makespan, et 3 % supplémentaires lorsqu'il était initialisé avec la meilleure solution connue. L'implémentation de l'algorithme PnB a montré des performances de pointe sur les instances de référence du TSP-TW, surpassant le solveur branch-and-bound basé sur les DD existant.

Cette thèse fait progresser davantage les méthodes d'optimisation basées sur les DD en généralisant le concept de DD relaxés implicites, une méthode générique de génération de DD qui ne nécessite pas que les arcs soient étiquetés. Au lieu de cela, cette information est déplacée vers les nœuds. Les DD relaxés implicites sont une méthode facile à implémenter pour obtenir des accélérations massives dans les solveurs basés sur les DD. Ce concept a été crucial pour atteindre des résultats de pointe avec notre implémentation de PnB.

L'utilité de PnB et des DD relaxés implicites a été démontrée par une application au problème de routage d'astéroïdes (ARP). Ce problème peut être considéré comme une variante du célèbre problème du voyageur de commerce où toutes les villes (astéroïdes) sont dans l'espace, et donc en mouvement constant. L'algorithme trouve des solutions réalisables de

haute qualité pour plusieurs instances de l'ARP, dont beaucoup sont optimales sous une légère hypothèse sur la qualité de l'optimiseur interne. C'est la première méthode dans la littérature plus efficace que la force brute pour trouver des solutions exactes aux problèmes d'optimisation de trajectoire globale comme l'ARP. Le cadre définissant l'algorithme est polyvalent, hautement évolutif et applicable à une large gamme de problèmes de séquençage exigeants en complexité.

Cette thèse non seulement fait progresser la compréhension et l'application des DD à travers l'introduction de méthodes innovantes comme le peel-and-bound et les DD relaxés implicites, mais elle ouvre également la voie à de futures percées dans des défis d'optimisation complexes qui nécessitent de résoudre un problème d'optimisation combinatoire externe comportant un problème d'optimisation interne coûteux en calcul.

## ABSTRACT

This thesis presents significant advancements in the theory and implementation of Decision-Diagram-based optimization methods: primarily the development and enhancement of the Peel-and-Bound (PnB) algorithm as a study in how to construct a solver using Decision Diagrams (DDs), and the introduction of implicit relaxed DDs which generalizes an idea critical to efficient implementation of PnB.

Relaxed DDs are a graphical tool for constructing combinatorial relaxed bounds for optimization problems. The PnB algorithm demonstrates the powerful potential of DD-based solvers that leverage separation, a method of constructing relaxed DDs by starting from a small DD representing a weak initial relaxation and improving that relaxation by splitting nodes. PnB is a highly parallelizable algorithm, built around constructing relaxed DDs by separation, which can trade memory for computational efficiency.

The PnB implementation was tested on the 467 benchmark instances of Traveling Salesman Problem with Time Windows (TSP-TW) used to test the DD-based branch-and-bound (BnB) solver (ddo). Even when ddo is using parallel processing, PnB outperforms ddo. Furthermore, PnB closed 15 instances that, to the best of our knowledge, were open in the literature. In our final test of PnB, we ran the new implementation of PnB on the makespan variant of the 467 TSP-TW instances. PnB closed 94% of the makespan instances, and an additional 3% when seeded with the best known solution. The implementation of the PnB algorithm had a cutting-edge performance on the benchmark instances of the TSP-TW, outperforming the existing DD-based branch-and-bound (BnB) solver.

This thesis further advances DD-based optimization methods by generalizing the concept of implicit relaxed DDs, a generic method of generating DDs that does not require the arcs to be labeled. Instead, information that would have been stored in arc labels is moved to the nodes. Implicit relaxed DDs are an easy-to-implement method for achieving massive speed-ups in DD-based solvers. This concept was critical to achieving cutting edge results with our PnB implementation.

The utility of both PnB and implicit relaxed DDs was demonstrated with an application to the Asteroid Routing Problem (ARP). This problem can be thought of as a variant of the well-known Travelling Salesman Problem where all of the cities (asteroids) are in Space, and thus in constant motion. The ARP has an outer combinatorial problem that requires finding the optimal permutation to visit the asteroids, and an inner non-linear non-convex optimization problem that requires computing the optimal trajectory between two asteroids



at specific points in time. The algorithm finds high-quality feasible solutions for several instances of the ARP, many of which are optimal under a mild assumption about the quality of the inner optimizer. This is the first method in the literature more efficient than brute force for finding exact solutions to global trajectory optimization problems like the ARP. The framework defining the algorithm is versatile, highly scalable, and applicable to a diverse range of computationally demanding sequencing problems.

This thesis not only advances the understanding and application of DDs through the introduction of innovative methods like peel-and-bound and implicit relaxed DDs, but also sets the stage for future breakthroughs in complex optimization challenges that require solving an outer combinatorial optimization problem that has a computationally expensive inner optimization problem.

## TABLE OF CONTENTS

DEDICATION . . . . .	iii
ACKNOWLEDGEMENTS . . . . .	iv
RÉSUMÉ . . . . .	v
ABSTRACT . . . . .	vii
TABLE OF CONTENTS . . . . .	ix
LIST OF TABLES . . . . .	xiii
LIST OF FIGURES . . . . .	xv
LIST OF SYMBOLS AND ACRONYMS . . . . .	xvi
LIST OF APPENDICES . . . . .	xviii
CHAPTER 1 INTRODUCTION . . . . .	1
1.1 Research Goal . . . . .	2
1.2 Contributions . . . . .	2
1.3 Publications . . . . .	3
1.4 Outline . . . . .	3
CHAPTER 2 DECISION DIAGRAMS FOR OPTIMIZATION . . . . .	5
2.1 Optimization Problems: Notation and Examples . . . . .	5
2.1.1 Knapsack Example . . . . .	6
2.1.2 Sequence Ordering Example . . . . .	6
2.2 Exact Decision Diagrams . . . . .	7
2.2.1 Merging Equivalent Nodes . . . . .	8
2.2.2 Finding a Compact Decision Diagram . . . . .	9
2.2.3 Reading a Solution . . . . .	11
2.2.4 Weighted & Multivalued Decision Diagrams . . . . .	12
2.2.5 Top-Down Compilation . . . . .	14
2.3 Restricted Decision Diagrams . . . . .	15
2.4 Relaxed Decision Diagrams . . . . .	18

2.4.1	Top-Down Relaxed Decision Diagrams . . . . .	19
2.4.2	Relaxed Decision Diagrams by Separation . . . . .	23
2.5	Decision Diagrams for Sequencing Problems . . . . .	27
2.5.1	Encoding the AllDifferent Constraint . . . . .	28
2.5.2	Encoding a Generalizable Optimality Constraint . . . . .	29
2.6	Branch-and-Bound with Decision Diagrams . . . . .	30
2.7	Other Innovations in DD-Based Solvers . . . . .	34
2.7.1	ddo . . . . .	34
2.7.2	Arc-Flow Formulation . . . . .	35
2.7.3	HADDOCK . . . . .	35
2.7.4	DIDP . . . . .	35
2.7.5	Variable Ordering . . . . .	36
CHAPTER 3	PEEL-AND-BOUND . . . . .	37
3.1	Motivation . . . . .	37
3.2	Algorithm . . . . .	38
3.2.1	Complexity Analysis . . . . .	38
3.3	Example and Visualization . . . . .	41
3.4	Advantages and Implementation Decisions . . . . .	41
3.4.1	Node Selection . . . . .	41
3.4.2	Limitations and Handling Memory . . . . .	43
3.4.3	Integrating Rough Relaxed Bounds . . . . .	43
3.4.4	Parallelization . . . . .	44
3.5	Experiments on the Sequence Ordering Problem . . . . .	45
3.5.1	Description of the Heuristics Considered . . . . .	46
3.5.2	Experimental Results . . . . .	46
3.6	Summary . . . . .	52
CHAPTER 4	IMPROVED PEEL-AND-BOUND . . . . .	53
4.1	Motivation . . . . .	53
4.2	Improvements to the Theory . . . . .	53
4.2.1	Handling Non-Separable Objective Functions . . . . .	53
4.2.2	Embedded Restricted Decision Diagrams . . . . .	54
4.2.3	Peel-and-Bound with Top-Down Compilation . . . . .	56
4.3	Improvements to the Heuristics . . . . .	56
4.3.1	Node Selection Heuristic . . . . .	56
4.3.2	Search Diversification . . . . .	56

4.4	Experiments with our Improved Implementation . . . . .	57
4.4.1	Node Selection Heuristic . . . . .	57
4.4.2	Traveling Salesman Problem with Time Windows . . . . .	59
4.4.3	Traveling Salesman Problem with Time Windows - Makespan . . . . .	61
4.5	Summary . . . . .	64
CHAPTER 5 IMPLICIT RELAXED DECISION DIAGRAMS . . . . .		65
5.1	Algorithm . . . . .	65
5.1.1	Complexity Analysis . . . . .	68
5.2	Ease of Implementation . . . . .	68
5.3	The Future of Peel-and-Bound . . . . .	69
CHAPTER 6 APPLICATION TO FINDING EXACT SOLUTIONS TO THE SPACE- TIME DEPENDENT TSP . . . . .		70
6.1	Motivation . . . . .	70
6.2	Background . . . . .	71
6.2.1	The Asteroid Routing Problem . . . . .	71
6.2.2	Trajectory Optimization . . . . .	74
6.2.3	Relaxed Decision Diagrams for the ARP . . . . .	75
6.3	The Initial Decision Diagram . . . . .	77
6.3.1	Initial Setup . . . . .	77
6.3.2	Relaxing the Black Box . . . . .	77
6.3.3	Calculating Valid Arc Bounds . . . . .	78
6.4	Heuristic Search with Embedded Restricted Decision Diagrams . . . . .	84
6.5	Using Peel-and-Bound for the ARP . . . . .	84
6.6	Implementation Details . . . . .	86
6.6.1	Memoization of B . . . . .	86
6.6.2	Heuristic Decisions . . . . .	88
6.6.3	Limitations of the Inner Optimizer . . . . .	88
6.7	Experimental Results . . . . .	89
6.7.1	Note on Optimality . . . . .	89
6.7.2	Initial Experiment: Determining Best Settings . . . . .	90
6.7.3	Second Experiment: Test of Smaller Instances . . . . .	93
6.7.4	Final Experiment: Test of Larger Instances . . . . .	98
6.8	Opportunities for Parallel Computing . . . . .	100
6.9	Conclusions on the Framework for Outer/Inner Optimization Problems . . . . .	101
6.10	Acknowledgements . . . . .	101

CHAPTER 7 CONCLUSION . . . . .	102
7.1 Some Ideas for Future Research . . . . .	102
7.1.1 Implicit Relaxed DDs Everywhere . . . . .	102
7.1.2 More Decision Diagrams by Separation . . . . .	102
7.1.3 Exact Solutions to Inner/Outer Optimization Problems . . . . .	103
7.2 Final Thoughts . . . . .	104
REFERENCES . . . . .	105
APPENDICES . . . . .	114

## LIST OF TABLES

Table 2.1	Knapsack instance values ( $\mathcal{P}_{sack}$ ) . . . . .	6
Table 2.2	SOP instance ( $\mathcal{P}_{SOP}$ ) . . . . .	7
Table 3.1	Summary Statistics: Percentage Improvement of Peel-and-Bound Over Branch-and-Bound . . . . .	47
Table 3.2	Summary Statistics: Percentage Improvement of Peel-and-Bound at $\omega = 2048$ over Peel-and-Bound at $\omega = 256$ . . . . .	49
Table 4.1	TSP-TW Results for Newly Closed Problems . . . . .	61
Table 4.2	Makespan Results for Problems Not Closed by the Unseeded Run . .	63
Table 6.1	ARP instances with $n = 15$ , a 2 day runtime, an embedded search width of 100, and $multi = 1$ . . . . .	92
Table 6.2	ARP instances with $n = 10$ . . . . .	93
Table 6.3	Best solutions for $n \in \{10, 15, 20\}$ . . . . .	96
Table 6.4	ARP instances with $n = 15$ and a 7 day runtime . . . . .	97
Table 6.5	ARP instances with $n = 20$ and a 7 day runtime . . . . .	98
Table 6.6	ARP instances with $n = 25$ , a relaxed DD-width of 2048, and a 3 day runtime . . . . .	99
Table 6.7	ARP instances with $n = 30$ , a relaxed DD-width of 2048, and a 3 day runtime . . . . .	100
Table 6.8	Best solutions for $n \in \{25, 30\}$ . . . . .	100
Table A.1	Comparison Data for $\omega = 64$ Experiments on SOP . . . . .	114
Table A.2	Comparison Data for $\omega = 256$ Experiments on SOP . . . . .	115
Table A.3	Comparison of PnB at $\omega = 2048$ over PnB at $\omega = 256$ on SOP . . . .	116
Table A.4	<b>(Part 1 of 2)</b> TSP-TW Results of PnB at $\omega = 2048$ on Open Problems: Seeded and Unseeded . . . . .	117
Table A.5	<b>(Part 2 of 2)</b> TSP-TW Results of PnB at $\omega = 2048$ on Open Problems: Seeded and Unseeded . . . . .	118
Table A.6	<b>(Part 1 of 6)</b> TSP-TW Results of PnB at $\omega = 2048$ on Closed Problems: Seeded and Unseeded . . . . .	119
Table A.7	<b>(Part 2 of 6)</b> TSP-TW Results of PnB at $\omega = 2048$ on Closed Problems: Seeded and Unseeded . . . . .	120
Table A.8	<b>(Part 3 of 6)</b> TSP-TW Results of PnB at $\omega = 2048$ on Closed Problems: Seeded and Unseeded . . . . .	121

Table A.9	<b>(Part 4 of 6)</b> TSP-TW Results of PnB at $\omega = 2048$ on Closed Problems: Seeded and Unseeded . . . . .	122
Table A.10	<b>(Part 5 of 6)</b> TSP-TW Results of PnB at $\omega = 2048$ on Closed Problems: Seeded and Unseeded . . . . .	123
Table A.11	<b>(Part 6 of 6)</b> TSP-TW Results of PnB at $\omega = 2048$ on Closed Problems: Seeded and Unseeded . . . . .	124
Table A.12	<b>(Part 1 of 7)</b> Makespan Results of PnB at $\omega = 2048$ on Closed Problems: Seeded and Unseeded . . . . .	125
Table A.13	<b>(Part 2 of 7)</b> Makespan Results of PnB at $\omega = 2048$ on Closed Problems: Seeded and Unseeded . . . . .	126
Table A.14	<b>(Part 3 of 7)</b> Makespan Results of PnB at $\omega = 2048$ on Closed Problems: Seeded and Unseeded . . . . .	127
Table A.15	<b>(Part 4 of 7)</b> Makespan Results of PnB at $\omega = 2048$ on Closed Problems: Seeded and Unseeded . . . . .	128
Table A.16	<b>(Part 5 of 7)</b> Makespan Results of PnB at $\omega = 2048$ on Closed Problems: Seeded and Unseeded . . . . .	129
Table A.17	<b>(Part 6 of 7)</b> Makespan Results of PnB at $\omega = 2048$ on Closed Problems: Seeded and Unseeded . . . . .	130
Table A.18	<b>(Part 7 of 7)</b> Makespan Results of PnB at $\omega = 2048$ on Closed Problems: Seeded and Unseeded . . . . .	131

# LIST OF FIGURES

Figure 2.1	Decision Tree for $\mathcal{P}_{sack}$ . . . . .	7
Figure 2.2	Shrinking the Decision Diagram for $\mathcal{P}_{sack}$ . . . . .	10
Figure 2.3	Reading the optimal solution for $\mathcal{P}_{sack}$ . . . . .	12
Figure 2.4	Exact Weighted MDD for $\mathcal{P}_{SOP}$ . . . . .	13
Figure 2.5	Restricted Decision Diagram for $\mathcal{P}_{sack}$ . . . . .	17
Figure 2.6	Restricted Decision Diagram for $\mathcal{P}_{SOP}$ . . . . .	18
Figure 2.7	Top-Down Compilation of Relaxed Decision Diagram for $\mathcal{P}_{sack}$ . . . . .	21
Figure 2.8	Top-Down Compilation of Relaxed Decision Diagram for $\mathcal{P}_{SOP}$ . . . . .	22
Figure 2.9	Compilation of Relaxed Decision Diagram by Separation for $\mathcal{P}_{sack}$ . . . . .	26
Figure 2.10	Compilation of Relaxed Decision Diagram by Separation for $\mathcal{P}_{SOP}$ . . . . .	27
Figure 2.11	DD-Based Branch-and-Bound on $\mathcal{P}_{sack}$ . . . . .	32
Figure 2.12	DD-Based Branch-and-Bound on $\mathcal{P}_{SOP}$ . . . . .	33
Figure 3.1	Example of a Repeated Sub-Graph During Branch-and-Bound . . . . .	42
Figure 3.2	Example of a Peel Operation . . . . .	48
Figure 3.3	Performance Profiles: Peel-and-Bound and Branch-and-Bound . . . . .	49
Figure 3.4	Dual bounds for ESC25 and ft70.1 . . . . .	51
Figure 4.1	Performance of Improved Peel-and-Bound on SOP . . . . .	58
Figure 4.2	Performance of Improved Peel-and-Bound on TSP-TW . . . . .	60
Figure 4.3	Performance of Improved Peel-and-Bound on Makespan . . . . .	62
Figure 5.1	Implicit Relaxed DD, Before and After Filtering . . . . .	67
Figure 6.1	Depiction of a transfer from $a$ to $a'$ . . . . .	72
Figure 6.2	Visualization of best known solution to instance with $n = 10$ and $seed = 8$ . . . . .	73
Figure 6.3	Relaxed decision diagrams for an ARP with asteroids $\{A, B, C\}$ . . . . .	76
Figure 6.4	ARP instances with $n = 15$ , a 2 day runtime, an embedded search width of 100, and $multi = 1$ (Table 6.1) . . . . .	91
Figure 6.5	ARP instances with $n = 10$ . Lines show mean value over ARP instances (Table 6.2) . . . . .	91
Figure 6.6	ARP instances with $n = 15$ and a 7 day runtime (Table 6.4) . . . . .	95
Figure 6.7	ARP instances with $n = 20$ and a 7 day runtime (Table 6.5) . . . . .	96



## LIST OF SYMBOLS AND ACRONYMS

### Acronyms:

ARP	Asteroid Routing Problem
BDD	Binary Decision Diagram
BnB	Branch-and-Bound
CP	Constraint Program(ming)
DD	Decision Diagram
DP	Dynamic Program(ming)
LP	Linear Program(ming)
MDD	Multivalued Decision Diagram
MIP	Mixed Integer Progam(ming)
OR	Operations Research
PnB	Peel-and-Bound
SAT	Boolean Satisfiability
SOP	Sequence Ordering Problem
TSP	Travelling Salesman Problem
TSP-TW	Travelling Salesman Problem with Time Windows
TSP-TD	Travelling Salesman Problem with Time Dependent Travel Times

### General Symbols:

$\mathcal{P}$	a discrete optimization problem
$Sol(\mathcal{P})$	the set of feasible solutions to $\mathcal{P}$
$\pi$	a solution to an optimization problem
$\pi^*$	the optimal solution to an optimization problem
$z^*$	the value of an optimal solution to an optimization problem
$z^*(\mathcal{P})$	the value of an optimal solution to $\mathcal{P}$
$d(x)$	the domain of variable $x$

## Decision Diagram Symbols:

$r$	a root node
$t$	a terminal node
$\oplus$	a merge operator for nodes
$\ominus$	a split operator for nodes
$a_{uv}$	an arc that goes from node $u$ to node $v$
$l$	an arc label
$l(a)$	the label on arc $a$
$w(a)$	the weight of arc $a$
$\ell$	a layer index of a Decision Diagram
$\ell(u)$	the layer index of node $u$
$L$	a layer of a Decision Diagram (a set of nodes)
$d(u)$	the domain of a node (a set of arc labels)
$f(u)$	the value of the objective function at $u$
$\omega$	the maximum width of a Decision Diagram
$\mathcal{M}$	a Decision Diagram
$\mathcal{M}(u)$	a partial Decision Diagram rooted at $u$
$\mathcal{M}(u, v)$	a partial Decision Diagram rooted at $u$ with terminal $v$
$\mathcal{M}_\omega$	a Decision Diagram with width $\omega$
$\mathcal{M}^-$	a restricted Decision Diagram
$\mathcal{M}^+$	a relaxed Decision Diagram
$d(\mathcal{M})$	the set of solutions encoded by Decision Diagram $\mathcal{M}$
$All_u^\downarrow$	the set of labels that appear on every path from $r$ to $u$
$Some_u^\downarrow$	the set of labels that appear on at least one path from $r$ to $u$
$All_u^\uparrow$	the set of labels that appear on every path from $u$ to $t$
$Some_u^\uparrow$	the set of labels that appear on at least one path from $u$ to $t$
$z(\mathcal{M})$	the value of the best path through $\mathcal{M}$
$z_u^\downarrow$	the value of the best path from $r$ to $u$
$z_u^\uparrow$	the value of the best path from $u$ to $t$

**LIST OF APPENDICES**

Appendix A	Experimental Data . . . . .	114
------------	-----------------------------	-----

## CHAPTER 1 INTRODUCTION

Operations Research (OR) is the science of applying mathematical tools to provide a quantitative foundation for decision-making. The use of these tools is widespread: logistics companies utilize them to schedule deliveries and route vehicles, hospitals employ them to optimize patient flow, schools apply them to organize course schedules, investment firms leverage them for risk management, and manufacturers rely on them to minimize resource waste. OR tools are essential for the functioning of modern society. However, the need for new and improved tools consistently surpasses their supply. With increasing data availability, access to computing power, and the desire to optimize new domains, there is a growing demand for OR that exceeds its capabilities.

Traditional techniques often struggle with computational efficiency when faced with intricate constraints or large data sets. Additionally, evolving challenges have created a need for tools that can represent and solve problems more flexibly. This environment set the stage for Multivalued Decision Diagrams (MDDs) to emerge as a new tool capable of addressing gaps in existing methodologies and offering novel ways to represent optimization problems.

OR provides a range of tools, such as Linear Programming (LP), Mixed Integer Programming (MIP), Dynamic Programming (DP), Constraint Programming (CP), Boolean Satisfiability (SAT), and an extensive range of heuristics. Each tool has a dedicated and active research community working to improve it. This diversity persists because no single tool can outperform the others across all problems. Each tool excels in certain areas, but falls short in others. In 2016, the book *Decision Diagrams for Optimization* organized and compiled several papers, supplemented them with examples and data, and made the case that MDDs could serve as a novel OR tool to bridge gaps left by existing techniques [1]. Decision Diagrams (DDs) have long been used in Computer Science, but the framework presented in this book demonstrated their potential as a tool for optimization as well.

Since 2016, research into leveraging DD-based techniques in other OR tools has surged, alongside the development of dedicated DD-based solvers. These two approaches are symbiotic: methods and insights from one often benefit the other, and both have evolved in parallel. A 2022 survey of DD research, which reviewed over 100 papers, offers just a glimpse of the ongoing work, and the community of DD researchers continues to expand each year [2]. While most DD research has focused on integrating DD techniques with other tools, a few groups are making significant efforts to develop state-of-the-art DD-based solvers. This thesis details one of these initiatives.

## 1.1 Research Goal

The goal of this thesis is to offer insight into how DD-based techniques can be applied to advanced optimization challenges, emphasizing the crucial decisions involved in implementing Decision Diagrams. In addition, I aim to provide practical advice for developing a DD-based solver, offering context for the rationale behind the decisions made during its implementation.

This work delves into the specific challenges faced when adapting DD-based methods to various applications and highlights best practices for ensuring the efficiency and effectiveness of a DD-based solver. By exploring the nuances of decision-making and design choices, I hope to guide researchers and practitioners in leveraging these techniques for cutting-edge optimization, ultimately expanding the scope and impact of Decision Diagram methodologies.

## 1.2 Contributions

The main contributions of this work are:

- **Peel-and-Bound Framework:** Development of a generic framework for a DD-based solver, known as *peel-and-bound* (PnB), and its open-source implementation in Julia [3]. This implementation demonstrates cutting-edge results on the Traveling Salesman Problem with Time Windows, closing 15 open benchmark instances, and making the framework a valuable resource for researchers and practitioners.
- **Embedded Restricted Decision Diagrams:** Introduction of a novel DD-based search technique for finding feasible solutions to optimization problems called *Embedded Restricted Decision Diagrams*, which is also referred to as *Embedded Search*. This work generalizes the concept of *rough bounding* [4,5] by demonstrating that Embedded Search can leverage rough bounding in a problem-agnostic manner.
- **Implicit Relaxed Decision Diagrams:** A generic algorithm for constructing relaxed DDs without needing to label the arcs. This construction is easier to implement than standard relaxed DDs, and substantially more efficient at run-time.
- **Framework for Finding Exact Solutions to Outer/Inner Optimization:** Establishment of a framework for using peel-and-bound to identify optimal solutions to optimization problems with an outer combinatorial problem and a complex or computationally expensive inner problem, under a mild assumption about the quality of the inner optimizer. The framework is demonstrated through its application to the Asteroid Routing Problem (ARP), marking the first instance where a global trajectory

optimization problem is solved exactly through a method more efficient than brute force.

### 1.3 Publications

A large portion of this thesis is based on work that has been previously published in the following papers. This is noted at the beginning of each relevant chapter.

- *Peel-and-Bound: Generating Stronger Relaxed Bounds with Multivalued Decision Diagrams* by Isaac Rudich, Quentin Cappart, and Louis-Martin Rousseau, was published at the 28th International Conference on Principles and Practice of Constraint Programming (CP 2022) where it won the Best Paper award [6]. In this paper, we introduced the peel-and-bound framework for making a DD-based solver, and we implemented a simple test showing that a naive implementation of PnB outperforms a naive implementation of the branch-and-bound (BnB) framework for a DD-based solver [7, 8].
- *Improved Peel-and-Bound: Methods for Generating Dual Bounds with Multivalued Decision Diagrams* by Isaac Rudich, Quentin Cappart, and Louis-Martin Rousseau, was published in the Journal of Artificial Intelligence Research in 2023 [9]. In this paper, significant effort was devoted to implementing PnB as efficiently as possible, and to detailing the steps required to achieve cutting-edge results. This paper closed several open instances of the Traveling Salesman Problem with Time Windows (TSP-TW).
- *An Exact Framework for Solving the Space-Time Dependent TSP* by Isaac Rudich, Quentin Cappart, Manuel López-Ibáñez, Michael Römer, and Louis-Martin Rousseau is currently only available as a preprint. In this paper, PnB was used to close several open instances of the Asteroid Routing Problem, demonstrating a framework for solving optimization problems with an outer combinatorial problem, and a computationally expensive inner optimization problem. This marks the first time a technique more efficient than brute force was applied to a global trajectory optimization problem.

### 1.4 Outline

Chapter 2 begins the thesis proper with a decision diagram tutorial. The tutorial includes figures new for this thesis demonstrating every concept with two different examples. Chapter 3 introduces peel-and-bound, which is the foundation for everything else in this thesis. Chapter 4 builds on the initial concept of peel-and-bound, improving the theory and demonstrating

that our PnB solver can achieve cutting edge results. Chapter 5 presents the novel concept of Implicit Relaxed DDs, which has not been published elsewhere. Chapter 6 explains how we used PnB to find exact solutions to the ARP under a mild assumption about the quality of the inner optimizer. Chapter 7 closes the thesis with my parting thoughts.

## CHAPTER 2 DECISION DIAGRAMS FOR OPTIMIZATION

The standard approach for explaining the use of Decision Diagrams (DDs) serves two main purposes: (1) to justify their use, and (2) to explain the methodology. A foundational tutorial is presented in the book *Decision Diagrams for Optimization* [1], while a more recent example is available in *Discrete optimization with decision diagrams : design of a generic solver, improved bounding techniques, and discovery of good feasible solutions with large neighborhood search* [5]. There are earlier works as well [10–14], but *Decision Diagrams for Optimization* unified and expanded on these earlier works. This standard approach justifies the use of DDs by illustrating how a DD representing a problem can be built from the Dynamic Programming (DP) model for that problem. This establishes an intuitive understanding of why DDs are valuable and provides insight into the origins of DD-based methods. However, this approach is widely available, and other intuitive methods for explaining DD-based techniques can provide equally valuable insight. Every DP model can be turned into a DD, but not every DD can be turned into a DP model [15].

This chapter provides model-agnostic motivation for using DDs for optimization. In other words, for the purposes of this explanation, a DD will be the problem model, constructed directly from the problem description. This will be of particular practical use in Chapter 6 where we solve the Asteroid Routing Problem (ARP). Writing out the objective function for even a small instance of the ARP is intractable. Thus, constructing a DD for the ARP from a DP model in the traditional way is impossible, and we instead construct the DD directly from the problem description.

The explanation detailed in this chapter is based almost entirely on ideas compiled in *Decision Diagrams for Optimization*, but differs in that it starts from decision trees instead of a Dynamic Program [1, 7, 8]. For further references to key literature, see Section 2.7.

### 2.1 Optimization Problems: Notation and Examples

Let  $\mathcal{P}$  be an instance of a discrete minimization problem with  $n$  variables  $\{x_1, \dots, x_n\}$ , let  $Sol(\mathcal{P})$  be the set of feasible solutions to  $\mathcal{P}$ , let  $\pi^*(\mathcal{P})$  be an optimal solution to  $\mathcal{P}$ , let  $z^*(\mathcal{P})$  be the value of an optimal solution to  $\mathcal{P}$ , and let  $d(x_i)$  be the domain of variable  $x_i, i \in \{1, \dots, n\}$ .



### 2.1.1 Knapsack Example

In this chapter, two examples will be used. The first example is an instance of the knapsack problem,  $\mathcal{P}_{sack}$  [16]. In the knapsack problem, there is a bag with a limited weight capacity, and several items that can be placed into the bag. Each item has a utility and a weight. The objective is to maximize the utility of items placed in the bag without exceeding the weight capacity. The instance we will be using is shown below in Table 2.1.

Table 2.1 Knapsack instance values ( $\mathcal{P}_{sack}$ )

**Knapsack Capacity: 10**

Item	Weight	Utility
A	3	6
B	4	8
C	5	7
D	8	9

Let the variables for  $\mathcal{P}_{sack}$  be  $\{x_A, x_B, x_C, x_D\}$  where  $x_i \in \{0, 1\}$  and  $i \in \{A, B, C, D\}$ ; a value of 0 means the item is not included in the sack, and a value of 1 means the item is included in the sack. For reference, the optimal solution to this instance is  $[x_A = 0, x_B = 1, x_C = 1, x_D = 0]$  with a value of 15.

### 2.1.2 Sequence Ordering Example

The second example is an instance of the Sequence Ordering Problem (SOP) ( $\mathcal{P}_{SOP}$ ) [17]. In the SOP, there is a list of tasks (or elements), and the goal is to find the least expensive ordering of those tasks in which each task is performed exactly once. Additionally, certain precedence constraints must be respected, meaning that some tasks must appear earlier than others for a sequence to be feasible. In other words, the SOP is an asymmetric TSP with precedence constraints.

Defining a SOP instance requires providing the cost  $c_{ij}$  of following item  $x_i$  with  $x_j$ , for each valid ordered pair  $\{i, j\}$ . The cost matrix in Table 2.2 defines  $\mathcal{P}_{SOP}$ . The precedence constraints are that A must come before everything else, and B must come before D; these are indicated by **X** in the cost matrix.

Let the variables for  $\mathcal{P}_{SOP}$  be  $\{x_1, x_2, x_3, x_4\}$ . The domain of each variable is in  $\{A, B, C, D\}$ , and  $x_i = A$  indicates that  $A$  is the  $i^{th}$  element in the sequence. For reference, the optimal solution to this instance is  $[x_1 = A, x_2 = B, x_3 = D, x_4 = C]$  with a cost of 17.

Table 2.2 SOP instance ( $\mathcal{P}_{SOP}$ )

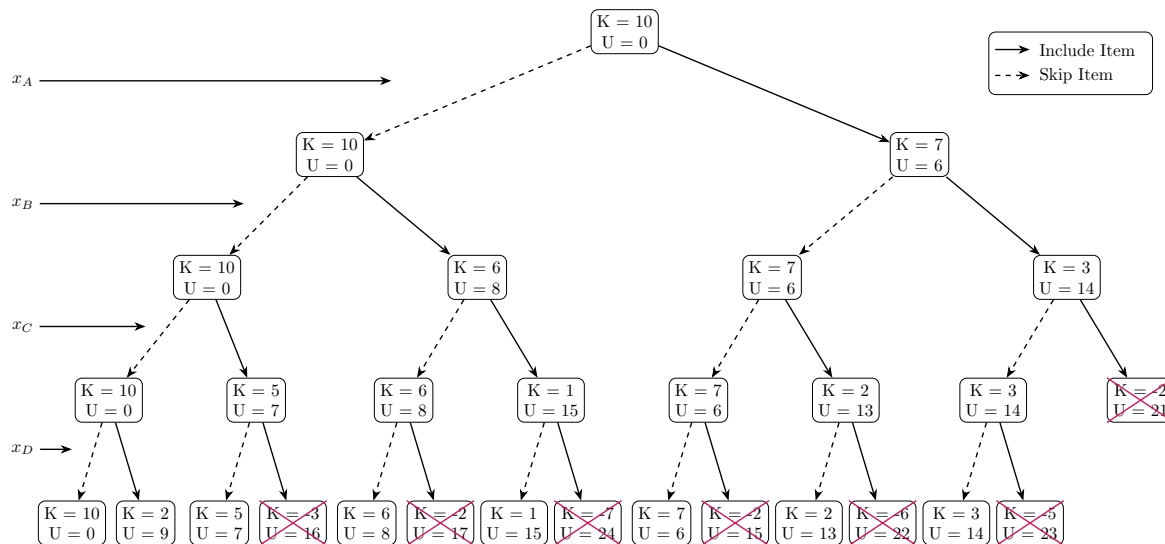
Transition cost of  $c_{row,col}$  to go from  $row$  to  $col$ .

From \ To	A	B	C	D
A	-	8	5	0
B	X	-	5	8
C	X	5	-	5
D	X	X	1	-

## 2.2 Exact Decision Diagrams

An exact DD for an optimization problem  $\mathcal{P}$ , is a directed layered acyclic graph with a root  $r$  and terminal  $t$ , such that every path from  $r$  to  $t$  encodes a feasible solution to  $\mathcal{P}$ , and every feasible solution to  $\mathcal{P}$  is encoded as a path from  $r$  to  $t$ . So a DD is an exact representation of  $\mathcal{P}$  if the set of solutions it encodes is equal to  $Sol(\mathcal{P})$ .

To build an intuition for what this looks like, we will construct an exact DD for  $\mathcal{P}_{sack}$  (see Section 2.1.1). The following method is inefficient, but educational. Begin by drawing the decision tree that represents every feasible solution to  $\mathcal{P}_{sack}$  as its own branch. The result is shown below in Figure 2.1. At each node in the tree  $K$  is the remaining capacity, and  $U$  is the utility. Nodes that exceed the total capacity are removed from the tree, because those solutions are infeasible.

Figure 2.1 Decision Tree for  $\mathcal{P}_{sack}$ 

K:= remaining capacity, U:= utility.

### 2.2.1 Merging Equivalent Nodes

Making a decision tree to store every feasible solution is possible for small problems, but quickly becomes intractable. An exact DD can often accomplish the task with substantially fewer nodes. We will now turn the decision tree into a decision diagram by merging nodes [18,19]. For the diagram to remain exact, merges must not add or remove any paths. This is done using a recursive procedure.

Let  $a$  be an arc, and let  $l(a)$  be the label on an arc, i.e., the value assigned to a decision variable by  $a$ . Let  $a_{uv}$  be an arc from  $u$  to  $v$ . In  $\mathcal{P}_{sack}$ , every arc has a label  $l \in \{0,1\}$ , indicated by a dashed line or solid line, respectively. Let two nodes  $u$  and  $v$  be *equivalent* if, for every out-arc of  $u$  there is an out-arc of  $v$  with the same label and destination node, and similarly for every out-arc of  $v$  there is an out-arc of  $u$  with the same label and destination node. So, if  $u$  and  $v$  are equivalent, then for each out-arc  $a_{uz}$  of  $u$  there is an out-arc  $a_{vz}$  of  $v$  such that  $l(a_{uz}) = l(a_{vz})$ , and *vice versa*. In addition,  $u$  and  $v$  must be on the same layer.

Merging two equivalent nodes requires defining a merge operator. Let  $u$  and  $v$  be two equivalent nodes to be merged. In general, when equivalent nodes are merged, the following happens: (1) a new node  $m_{uv}$  is created on the same layer as the merging nodes, (2) the destination of the in-arcs of  $u$  and the in-arcs of  $v$  is changed to be  $m_{uv}$ , and (3) chosen arbitrarily, the out-arcs of either  $u$  or  $v$  are changed so that their origin is  $m_{uv}$ . This is formalized in Algorithm 1.

---

**Algorithm 1:** Generic Merge Operation for Equivalent Nodes

---

```

1 Input: equivalent nodes  $u$  and  $v$ 
2 Create a new node  $m$ 
3 foreach in-arc  $a_{zu}$  of  $u$  do
4   | Adjust the arc so that its destination is  $m$  instead of  $u$ , becoming  $a_{zm}$ 
5 end
6 foreach in-arc  $a_{zv}$  of  $v$  do
7   | Adjust the arc so that its destination is  $m$  instead of  $v$ , becoming  $a_{zm}$ 
8 end
9 foreach out-arc  $a_{uz}$  of  $u$  do
10  | Adjust the arc so that its origin is  $m$  instead of  $u$ , becoming  $a_{mz}$ 
11 end
12 Delete  $u$  and  $v$ 
13 return  $m$ 

```

---

When two equivalent nodes are merged, a procedure for merging the information stored on the nodes must be defined. This definition will be specific to the optimization problem being solved and the information being stored. In  $\mathcal{P}_{sack}$ , each node is tracking the remaining

capacity of the knapsack  $K$ , and the utility  $U$  achieved so far. Let  $K_1$  and  $K_2$  be the merging  $K$  values,  $U_1$  and  $U_2$  be the merging  $U$  values,  $K'$  be the merged  $K$  value, and  $U'$  be the merged  $U$  value. If a new value for the remaining capacity stored on the merged node is defined as the smallest of the old values for remaining capacity  $K' = \min(K_1, K_2)$ , then  $K'$  represents the remaining capacity if the heaviest path to the merged node is taken. Note that it is equally valid to define the new value as the largest of the old values  $K' = \max(K_1, K_2)$ , but  $K'$  will instead represent the remaining capacity if the lightest path to that node is taken. In this example, we will choose the former definition ( $\min(K_1, K_2)$ ), because tracking the heaviest path to each node may provide insight about the domain of the node. When merging the utility values, we will always keep the largest value  $U' = \max(U_1, U_2)$ , so that  $U'$  represents the value of the highest-utility path to that node.

### 2.2.2 Finding a Compact Decision Diagram

With the merge operator and equivalency defined, we have all of the tools necessary to turn a decision tree into a much more compact exact DD. All of the nodes in the bottom layer of a decision tree are trivially equivalent because the set of out-arcs for each such node is  $\emptyset$ . Thus, they can all be merged together without adding or removing any paths from the diagram. After merging all of the nodes on the bottom layer into a single node, denoted  $t$ , all of the nodes on the previous layer have only one destination. Thus, many of the nodes on the layer above  $t$  may be equivalent, and can be merged together. This process can be repeated for each layer, from the bottom to the top, until all equivalent nodes have been merged. This is formalized in Algorithm 2, and demonstrated for  $\mathcal{P}_{sack}$  in Figure 2.2.

---

#### Algorithm 2: Shrink Decision Tree into Decision Diagram

---

```

1 Input: A decision tree  $T$  with  $n + 1$  layers, where  $n$  is the number of decision variables
2 Let  $Q$  be the set of nodes on layer  $n + 1$ 
3 Let  $\oplus$  be the merge operator defined in Algorithm 1
4 foreach index  $i$ , from  $n + 1$  to 1 do
5   foreach node  $u$  in layer  $i$  do
6     foreach node  $v \neq u$  in layer  $i$  do
7       if  $u$  and  $v$  are equivalent then
8          $u \leftarrow u \oplus v$ 
9       end
10    end
11  end
12 end
13 return  $T$ 

```

---

Note that the process of shrinking a DD in Algorithm 2 is focused on clarity, not efficiency. An actual implementation would take care not to repeat checks for pairs of nodes and may pre-sort the nodes to make identifying equivalency candidates faster. Also note that there is no need to start from a decision tree; the process can start from any exact DD and will try to generate a more compact exact DD. All three of the diagrams with a single terminal node shown in Figure 2.2 are valid exact DDs for  $\mathcal{P}_{sack}$ .

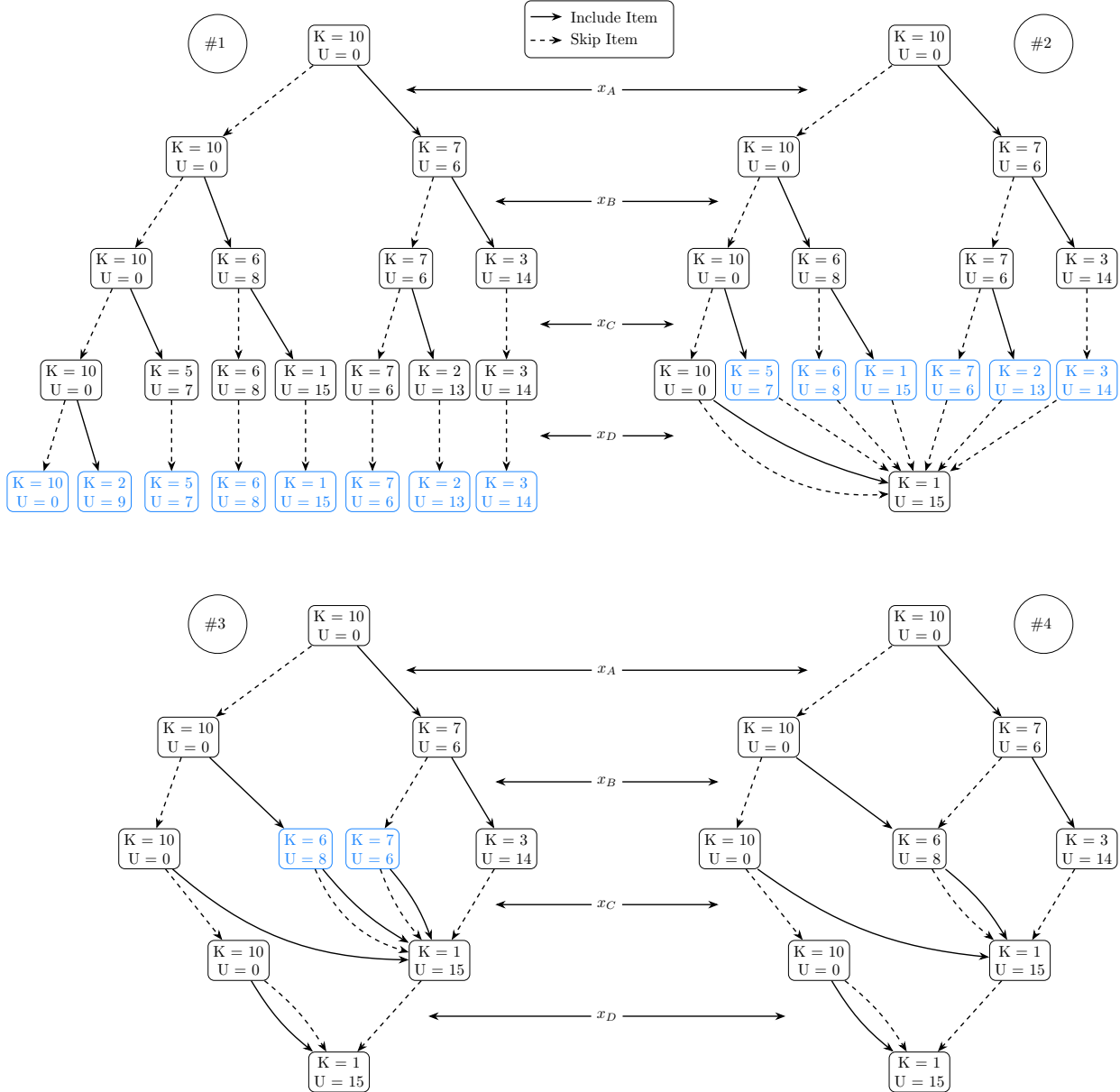


Figure 2.2 Shrinking the Decision Diagram for  $\mathcal{P}_{sack}$

K:= remaining capacity, U:= utility. Nodes in blue are merged.

### 2.2.3 Reading a Solution

Once you have a valid exact DD for an optimization problem  $\mathcal{P}$ , finding the optimal solution to  $\mathcal{P}$  is trivial. It can be done by recursively finding the best node at each layer starting from the bottom. *Best* means smallest for minimization problems, and largest for maximization problems.

Let  $l(a)$  be the label of arc  $a$ , and let  $\ell(u)$  be the index of the layer of node  $u$ . Let  $f(a_{uv})$  be the objective value stored at node  $u$ , plus the impact on the objective function of traversing arc  $a_{uv}$ . In other words,  $f(a_{uv})$  is the impact of setting  $x_{\ell(u)} = l(a_{uv})$ . Start from the terminal node  $t$ , and find the best  $f(a_{ut})$  value,  $f^* = f(a_{u^*t}^*)$ , where  $u^* \in \{\text{nodes with an arc to } t\}$ . The label  $l(a^*)$  is the value of the decision variable represented by that layer; as we are starting from the bottom,  $x_n = l(a^*)$ . Then the process is repeated, replacing  $t$  with  $u^*$ . This repeats layer by layer until the best path from the root to the terminal is identified. This path encodes the optimal solution. This process is formalized in Algorithm 3, and demonstrated in Figure 2.3 using  $\mathcal{P}_{sack}$ .

---

#### Algorithm 3: Read Solution from Decision Diagram

---

```

1 NOTE: This is normalized for minimization problems. For maximization problems
   change  $\infty$  to  $-\infty$  on Line 6, and change  $f^* > f(a_{uq})$  to  $f^* < f(a_{uq})$  on Line 9.
2 Input: A decision diagram with terminal node  $t$  and root node  $r$ 
3 Initialize  $\pi^* \leftarrow \emptyset$ 
4  $q \leftarrow t$ 
5 while  $q \neq r$  do
6    $f^* \leftarrow \infty$ 
7   foreach node  $u$  with an arc to  $q$  do
8     foreach arc  $a_{uq}$  from  $u$  to  $q$  do
9       if  $f^* > f(a_{uq})$  then
10          $f^* \leftarrow f(a_{uq})$ 
11          $u^* \leftarrow u$ 
12          $a^* \leftarrow a_{uq}$ 
13       end
14     end
15   end
16    $\pi^* \leftarrow \pi^* \cup \{l(a^*)\}$ 
17    $q \leftarrow u^*$ 
18 end
19 Reverse the order of  $\pi^*$ 
20 return  $\pi^*$ 

```

---

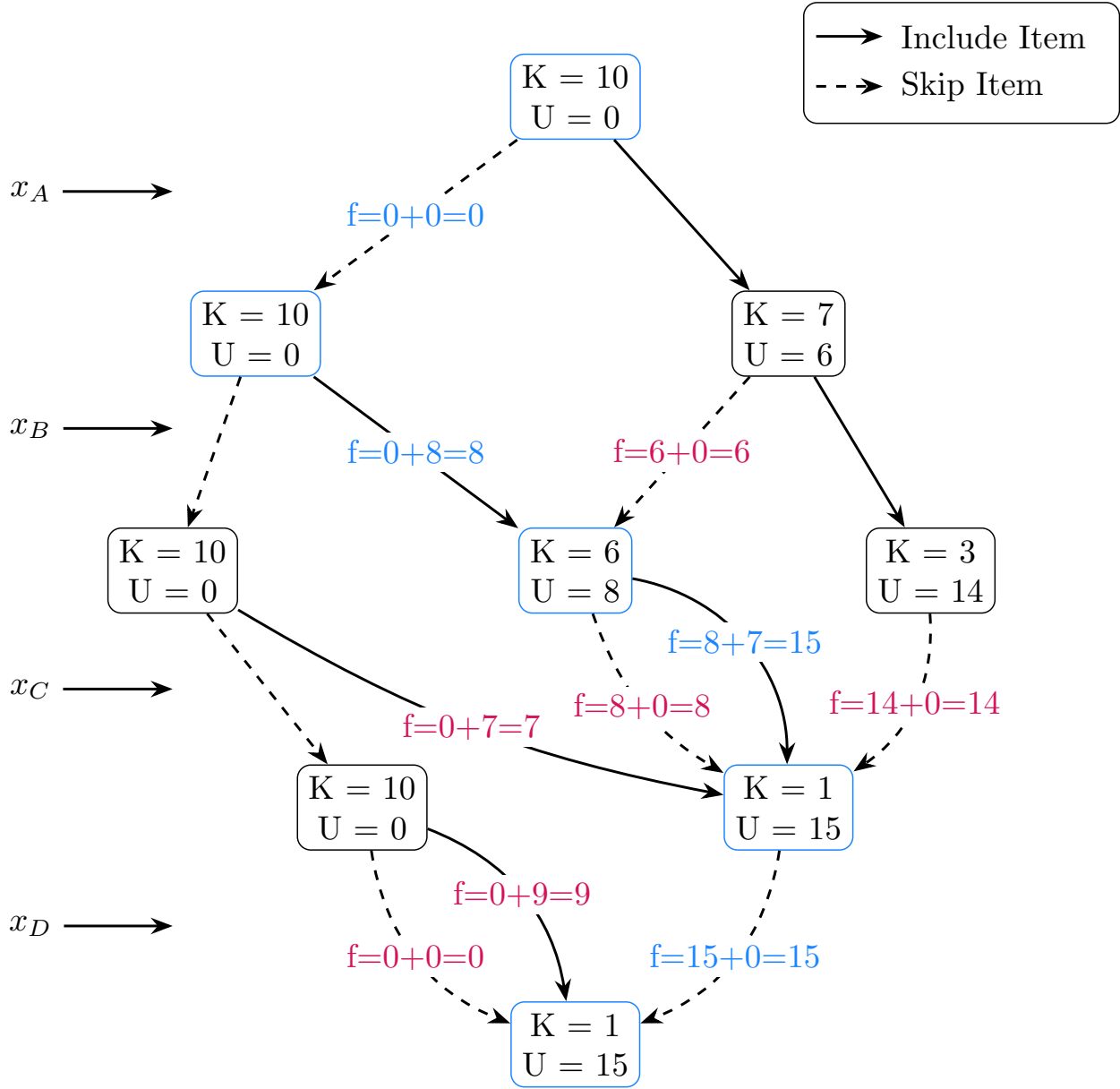


Figure 2.3 Reading the optimal solution for  $\mathcal{P}_{sack}$

K:= remaining capacity, U:= utility.

Nodes and arcs in blue are part of the optimal solution.

Arcs in red are read but not explored.

#### 2.2.4 Weighted & Multivalued Decision Diagrams

Up to this point, we have only considered an example with binary decision variables. Each item in  $\mathcal{P}_{sack}$  is either included (1) or excluded (0). A DD that has binary arcs only is called a *Binary Decision Diagram* (BDD). So the DDs for  $\mathcal{P}_{sack}$  in Figure 2.2 are all BDDs.

However, the methods described thus far also work when the domains of the decision variables are larger, as long as they are discrete. The only difference is that each node can have an out-arc for each element in its domain. Decision Diagrams with non-binary arcs are called *Multivalued Decision Diagrams* (MDDs). We are introducing this terminology because it is common in the literature. In this thesis, we often use DDs to refer to both BDDs and MDDs.

An example of an MDD is shown in Figure 2.4 using the  $\mathcal{P}_{SOP}$  example from Section 2.1.2. In the  $\mathcal{P}_{SOP}$  example, the only difference between the decision tree and the exact DD is that the nodes on the bottom layer have been merged into a single terminal node. This happens because the solution space is highly constrained by the precedence constraints. Note that each arc is storing two pieces of information. The first is the label of the arc, the second is the impact on the objective function of assigning the label of that arc to the decision variable on its layer. In this thesis, we will refer to this impact sometimes as the *weight* of the arc, and sometimes as the *cost* of the arc depending on the situation. When the arcs have weights, the DD is called *weighted* [20, 21].

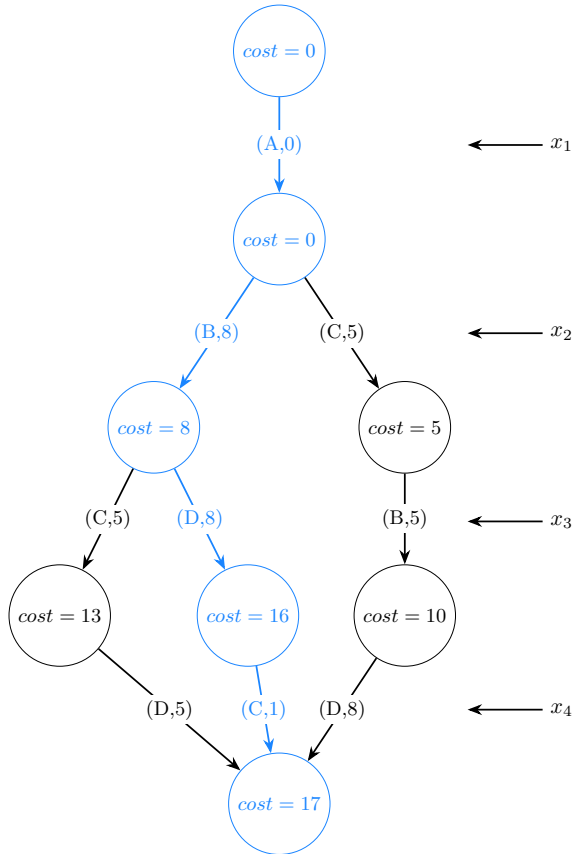


Figure 2.4 Exact Weighted MDD for  $\mathcal{P}_{SOP}$

Optimal solution indicated in blue.



### 2.2.5 Top-Down Compilation

So far, this section has shown how a DD can exactly represent the solution space for an optimization problem  $\mathcal{P}$ , but the method described for generating an exact DD is useless because it requires a brute force exploration of the entire solution space. However, an exact DD can be constructed top-down without exploring the entire solution space using a technique inspired by Dynamic Programming [20–22].

This process begins as before; for each feasible initial decision, add a new node  $u$ , and add an arc  $a_{ru}$  from the root  $r$  to  $u$ . Let  $d(u)$  be the *domain* of node  $u$ . The domain of a node  $u$  is the set of assignments of values to the decision variable  $x_{\ell(u)}$  that can be made on paths that pass through  $u$ . Equivalently, the domain of a node  $u$  is the set of labels on out-arcs of  $u$ . Let the *aggregate domain* of a node  $u$  be the set of every feasible partial solution that starts at  $u$  and ends at  $t$ , i.e., the set of paths from  $u$  to  $t$  in a complete version of the DD.

If two nodes have the same aggregate domain, they are equivalent and can be merged. In this way, equivalent nodes can be merged top-down as the DD is created, instead of bottom-up after the fact. Thus, if  $\mathcal{P}$  is a problem where the aggregate domain of a node pair can be proved equivalent without a brute force exploration of the solution space, then  $\mathcal{P}$  may be a good candidate for generating an exact DD top-down. Algorithm 4 formalizes this process.

The  $\mathcal{P}_{sack}$  example is a good example of a problem where an exact DD can be compiled top-down. The aggregate domain of a node in a knapsack problem is dependent only on its remaining capacity. Thus, if two nodes have the same remaining capacity, they have the same aggregate domains, and they are equivalent. Furthermore, there may be ranges of remaining capacity at each layer with matching aggregate domains that are easy to calculate. For example, at layer 3, the weights of the remaining items are 5 and 8 for items  $C$  and  $D$ , respectively. In order to put both items in the knapsack, the knapsack would have to have a remaining capacity  $K$  of 13, which exceeds the total weight limit of 10. Thus, having the maximum remaining capacity of 10 will only allow one of those items to be placed into the knapsack. Also, having a remaining capacity of 4 or less prevents item  $C$  from being added to the knapsack, and having a remaining capacity of 7 or less prevents item  $D$  from being added to the knapsack. Therefore, on layer 3, nodes with  $K \in [0, 4]$  are equivalent, as are nodes with  $K \in [5, 7]$  and nodes with  $K \in [8, 10]$ . Thus, if you generated an exact DD for  $\mathcal{P}_{sack}$  top-down, the two highlighted nodes in DD #3 in Figure 2.2 could be proven equivalent without a brute force exploration of the solution space, because  $K = 6$  and  $K = 7$  are equivalent on layer 3.

In general, for the SOP, two nodes have the same aggregate domain if their in-arcs all have

---

**Algorithm 4:** Top-Down Compilation of Exact Decision Diagrams

---

```

1 Input: An unfinished Decision Diagram  $\mathcal{M}$  with exactly one node, the root node  $r$ 
2 Let  $\oplus$  be the merge operator defined in Algorithm 1
3 Initialize  $Q \leftarrow \{r\}$ 
4 Initialize  $Q_{next} \leftarrow \emptyset$ 
5 while  $Q \neq \emptyset$  do
6    $Q_{next} \leftarrow \emptyset$ 
7   foreach node  $u \in Q$  do
8     foreach label  $l \in d(u)$  do
9       Create a new node  $v$ 
10       $Q_{next} \leftarrow Q_{next} \cup \{v\}$ 
11      Add arc  $a_{uv}$  from  $u$  to  $v$ 
12    end
13  end
14  foreach pair of nodes  $\{u, v\} \in Q_{next}$  do
15    if  $u$  and  $v$  have the same aggregate domain then
16       $Q_{next} \leftarrow Q_{next} \setminus \{u, v\}$ 
17       $Q_{next} \leftarrow Q_{next} \cup \{u \oplus v\}$ 
18    end
19  end
20   $Q \leftarrow Q_{next}$ 
21 end
22 return  $\mathcal{M}$ 

```

---

the same label and all incoming paths have visited the same set of elements. For example, a node with exactly one incoming path  $[A, B, C]$  could be merged with a node in the same layer with exactly one incoming path  $[B, A, C]$ .

### 2.3 Restricted Decision Diagrams

In the previous section (2.2), we discussed what an exact DD is, and how one might be constructed. However, for most meaningful problems, constructing an exact DD is intractable. Exact DDs have their uses and remain an active area of research, but they are not a panacea. Restricted DDs are an alternative method of leveraging DDs [1].

A restricted DD for an optimization problem  $\mathcal{P}$  encodes only feasible solutions to  $\mathcal{P}$ , but it may not encode every feasible solution to  $\mathcal{P}$ . Let  $\mathcal{M}^-$  be a restricted DD for  $\mathcal{P}$ , and let  $d(\mathcal{M})$  be the set of solutions encoded by Decision Diagram  $\mathcal{M}$ . A restricted DD for an optimization problem  $\mathcal{P}$ , is one where  $d(\mathcal{M}^-) \subseteq \text{Sol}(\mathcal{P})$ . Thus, the best path through  $\mathcal{M}^-$ , shortest for minimization and longest for maximization, is easy to find, feasible, and the best solution

encoded in the DD. Methods of generating restricted DDs are methods of generating a primal bound on  $\mathcal{P}$  by searching for feasible solutions.

---

**Algorithm 5:** Top-Down Compilation of Restricted Decision Diagrams

---

```

1 Input: A unfinished Decision Diagram  $\mathcal{M}^-$  with exactly one node, the root node  $r$ 
2 Recall:  $\ell(u)$  is the layer of node  $u$ 
3 Let  $\oplus$  be the merge operator defined in Algorithm 1
4 Initialize  $Q \leftarrow \{r\}$ 
5 Initialize  $Q_{next} \leftarrow \emptyset$ 
6 while  $Q \neq \emptyset$  do
7    $Q_{next} \leftarrow \emptyset$ 
8   foreach node  $u \in Q$  do
9     foreach label  $l \in d(u)$  do
10      Create a new node  $v$ 
11       $Q_{next} \leftarrow Q_{next} \cup \{v\}$ 
12      Add arc  $a_{uv}$  from  $u$  to  $v$ 
13    end
14  end
15  foreach pair of nodes  $\{u, v\} \in Q_{next}$  do
16    if  $u$  and  $v$  have the same aggregate domain then
17       $Q_{next} \leftarrow Q_{next} \setminus \{u, v\}$ 
18       $Q_{next} \leftarrow Q_{next} \cup \{u \oplus v\}$ 
19    end
20  end
21  if  $|Q_{next}| > \omega$  then
22    Sort  $u \in Q_{next}$  from best to worst by  $f(u)$ 
23    Trim  $Q_{next}$  to length  $\omega$  by removing the last elements
24  end
25   $Q \leftarrow Q_{next}$ 
26 end
27 return  $\mathcal{M}^-$ 

```

---

Constructing restricted DDs is a straightforward process. Let  $f(u)$  be the value of the objective function at node  $u$ . In other words,  $f(u)$  is the sum of the arc weights on the best path from  $r$  to  $u$ . First, a maximum width  $\omega$  is selected. Then, you proceed as if generating a decision tree, but every time the width of the DD exceeds  $\omega$ , you remove the least promising nodes. The definition of *least promising* is heuristic, but for the sake of simplicity, we will assume in this section that the nodes with the least optimal objective function values are the least promising. Essentially, a restricted DD is just a generalized greedy heuristic [4, 23–25]. Note that we can still merge nodes with the same aggregate domains, because doing so will never remove or add any paths from the DD, just store them more efficiently. This process

is formalized in Algorithm 5.

The process of building a restricted DD with  $\omega = 2$  is demonstrated for  $\mathcal{P}_{sack}$  in Figure 2.5 and for  $\mathcal{P}_{SOP}$  in Figure 2.6. In Section 2.2.5, we defined a method of determining if two nodes were equivalent for the SOP. However, in our experiments, we observed that the work of finding equivalent nodes in a restricted DD for a SOP often outweighed the benefit of being able to merge nodes. Thus, our implementation in Chapter 3 does not leverage equivalent node merging for the SOP.

In Figure 2.5, observe that the restricted DD contains the optimal solution, but unlike an exact DD, does not constitute proof that the solution is optimal. By contrast, in Figure 2.6, the optimal solution is excluded from the DD as it is constructed.

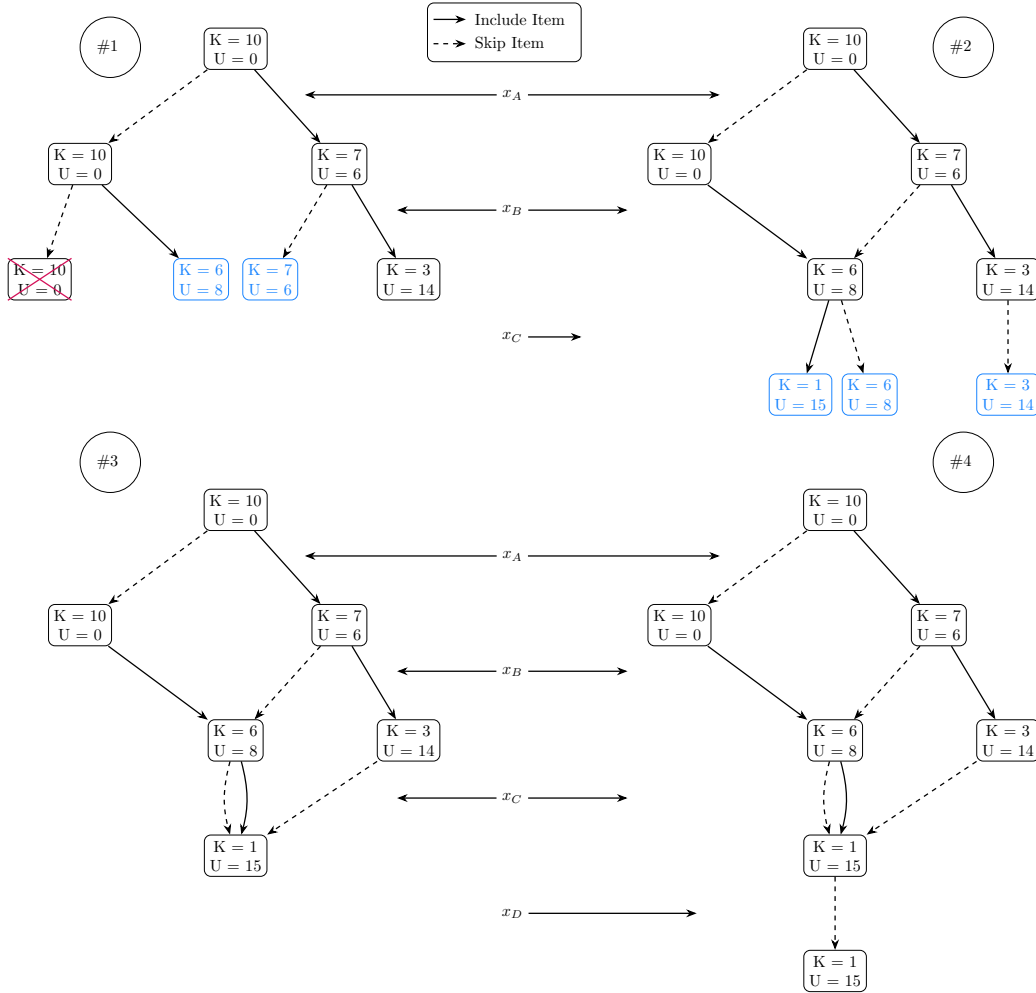
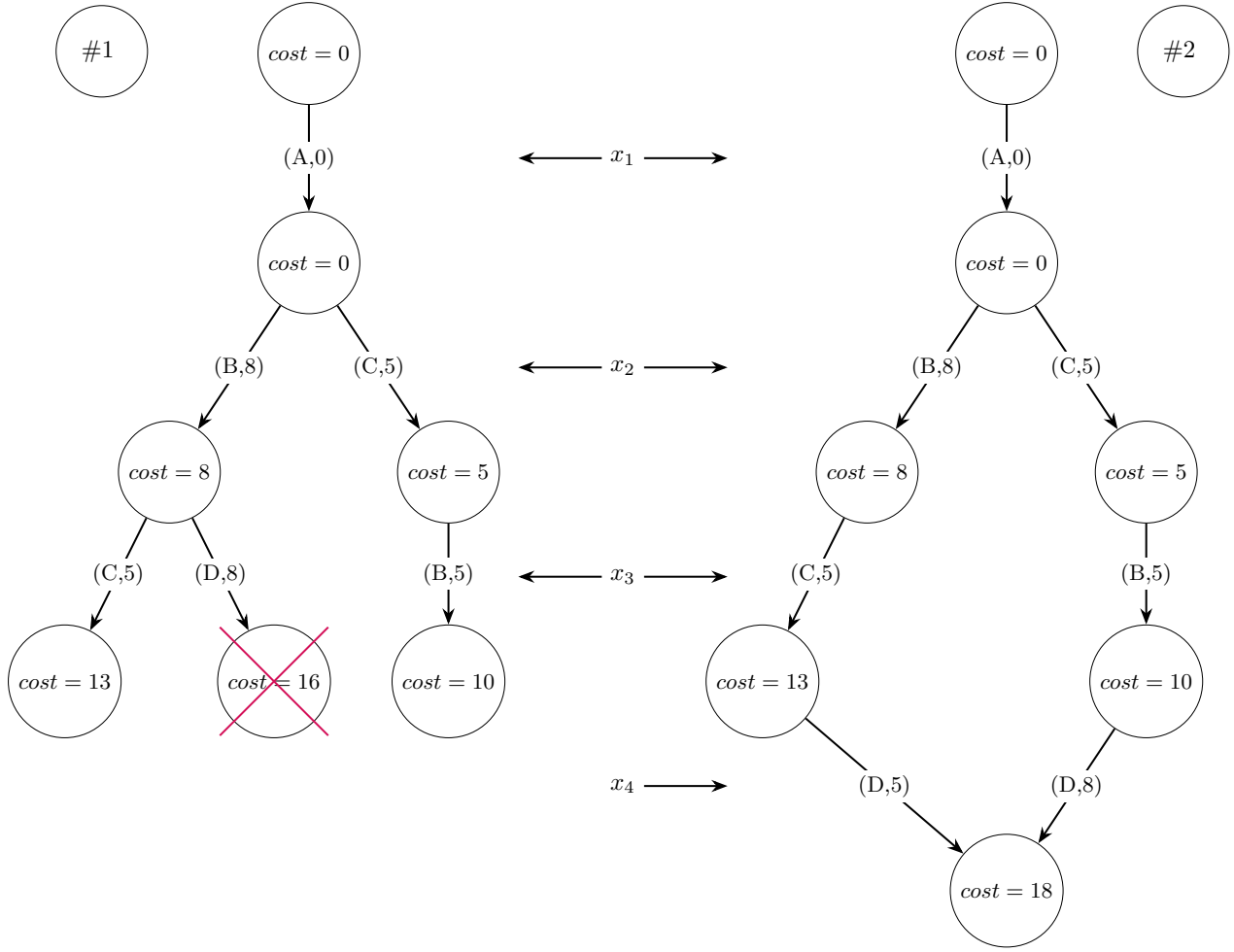


Figure 2.5 Restricted Decision Diagram for  $\mathcal{P}_{sack}$

K:= remaining capacity, U:= utility. Nodes in blue are merged.

Figure 2.6 Restricted Decision Diagram for  $\mathcal{P}_{SOP}$ 

## 2.4 Relaxed Decision Diagrams

As discussed in Section 2.3, restricted DDs provide a structure for searching for feasible solutions by encoding a subset of the feasible solutions. Relaxed DDs instead provide a relaxed (dual) bound on an optimization problem  $\mathcal{P}$  [10–12, 26].

A relaxed DD for  $\mathcal{P}$  encodes every feasible solution to  $\mathcal{P}$ , but it might also encode infeasible solutions to  $\mathcal{P}$ . Let  $\mathcal{M}^+$  be a relaxed DD for  $\mathcal{P}$ . A relaxed DD for  $\mathcal{P}$  is one where  $Sol(\mathcal{P}) \subseteq d(\mathcal{M}^+)$ . Thus, the best path through  $\mathcal{M}^+$  is easy to find, potentially infeasible, and provides a dual bound on the best possible solution  $\mathcal{P}$ .

### 2.4.1 Top-Down Relaxed Decision Diagrams

Constructing a relaxed DD top-down requires more effort than constructing a restricted DD, but is conceptually similar. The algorithm is almost entirely the same, except that when the maximum width  $\omega$  is exceeded, two nodes are merged together, as opposed to a node being removed. When nodes that are not equivalent are merged, potentially infeasible solutions are added to the DD, but no feasible solutions are removed. This is how adding infeasible solutions to the DD allows the DD to store all of the feasible solutions compactly [1, 10, 12, 26, 27].

Before we construct a relaxed DD, it is necessary to redefine the merge operator. In Algorithm 1, we detail a method of merging equivalent nodes, but now we will be merging non-equivalent nodes. The new procedure is almost the same as before, but instead of copying the out-arc from one of the nodes arbitrarily, we must now copy every out-arc of both nodes onto the new merged node. DDs are deterministic, so an arc is defined by its label and any repeat arc labels will be merged into a single arc. The new procedure is detailed in Algorithm 6. Note that this procedure also works on equivalent nodes, but the *if* statement on Line 13 will never trigger because  $d(u) = d(v)$  by definition for equivalent nodes.

---

**Algorithm 6:** Generic Merge Operation for Non-Equivalent Nodes

---

```

1 Input: nodes  $u$  and  $v$ 
2 Create a new node  $m$ 
3 foreach in-arc  $a_{zu}$  of  $u$  do
4   | Adjust the arc so that its destination is  $m$  instead of  $u$ , becoming  $a_{zm}$ 
5 end
6 foreach in-arc  $a_{zv}$  of  $v$  do
7   | Adjust the arc so that its destination is  $m$  instead of  $v$ , becoming  $a_{zm}$ 
8 end
9 foreach out-arc  $a_{uz}$  of  $u$  do
10  | Adjust the arc so that its origin is  $m$  instead of  $u$ , becoming  $a_{mz}$ 
11 end
12 foreach out-arc  $a_{vz}$  of  $v$  do
13   | if  $l(a_{vz}) \notin d(m)$  then
14     | Adjust the arc so that its origin is  $m$  instead of  $v$ , becoming  $a_{mz}$ 
15   | end
16 end
17 Delete  $u$  and  $v$ 
18 return  $m$ 

```

---

Now, we have the tools to build a relaxed DD. Recall that  $f(u)$  is the sum of the arc weights on the best path from  $r$  to  $u$ . First, a maximum width  $\omega$  is selected. Then, the algorithm

proceeds as if generating a decision tree, but every time the width of the DD exceeds  $\omega$ , two nodes are selected and merged. The selection of those nodes is heuristic. Selecting the nodes strategically may result in a stronger relaxed bound, but the result will still be valid if you select randomly. In this section, for simplicity, we will use a heuristic that always merges the least promising nodes. Note that we can still merge nodes with the same aggregate domains, because doing so will never remove or add any paths from the DD, just store them more efficiently. The process of constructing a relaxed DD top-down is formalized in Algorithm 7.

---

**Algorithm 7:** Top-Down Compilation of Relaxed Decision Diagrams

---

```

1 Input: A unfinished Decision Diagram  $\mathcal{M}^+$  with exactly one node, the root node  $r$ 
2 Recall:  $\ell(u)$  is the layer of node  $u$ 
3 Let  $\oplus$  be the merge operator defined in Algorithm 6
4 Initialize  $Q \leftarrow \{r\}$ 
5 Initialize  $Q_{next} \leftarrow \emptyset$ 
6 while  $Q \neq \emptyset$  do
7    $Q_{next} \leftarrow \emptyset$ 
8   foreach node  $u \in Q$  do
9     foreach label  $l \in d(u)$  do
10      Create a new node  $v$ 
11       $Q_{next} \leftarrow Q_{next} \cup \{v\}$ 
12      Add arc  $a_{uv}$  from  $u$  to  $v$ 
13    end
14  end
15  foreach pair of nodes  $\{u, v\} \in Q_{next}$  do
16    if  $u$  and  $v$  have the same aggregate domain then
17       $Q_{next} \leftarrow Q_{next} \setminus \{u, v\}$ 
18       $Q_{next} \leftarrow Q_{next} \cup \{u \oplus v\}$ 
19    end
20  end
21  while  $|Q_{next}| > \omega$  do
22    Sort  $u \in Q_{next}$  from best to worst by  $f(u)$  // heuristic
23    Let  $u$  and  $v$  be the last two elements  $\in Q_{next}$ 
24     $Q_{next} \leftarrow Q_{next} \setminus \{u, v\}$ 
25     $Q_{next} \leftarrow Q_{next} \cup \{u \oplus v\}$ 
26  end
27   $Q \leftarrow Q_{next}$ 
28 end
29 return  $\mathcal{M}^+$ 

```

---

In the  $\mathcal{P}_{sack}$  example, two pieces of information stored on the nodes need to be merged: remaining capacity and utility. In order for the relaxed DD to yield a valid dual bound on





$\mathcal{P}_{SOP}$  is a minimization problem instead of a maximization problem. Here, cost takes the place of utility, and the lower cost is kept when two nodes are merged. This way, the shortest path to each node is tracked. Another piece of information was being implicitly tracked that we will now explicitly track. When generating the arcs for a node  $u$ , the available arc labels  $d(u)$  are the subset of labels that have not appeared on the path to  $u$ , and are not ruled out by precedence constraints. Therefore, going forward, we will track this information explicitly on the nodes. When we merge nodes  $u$  and  $v$  in a relaxed DD for  $\mathcal{P}_{SOP}$ , the domain of the new node is  $d(u) \cup d(v)$ . As a final step, observe that it is possible for the weight of an arc to be unclear. If one path to a node ends in  $A$  and the other in  $B$ , then an out-arc with a  $C$  label could take on either transition cost  $c_{AC}$  or  $c_{BC}$ . To maintain the property that the DD is a valid relaxation, we will always use the smaller transition cost. Top-down compilation of a relaxed DD for  $\mathcal{P}_{sack}$  is shown in Figure 2.8. The dual bound on  $\mathcal{P}_{SOP}$  from the relaxed DD, with  $cost = 14$ , is valid but not feasible. It uses the solution  $[x_1 = A, x_2 = B, x_3 = C, x_4 = C]$  which repeats  $C$  twice.

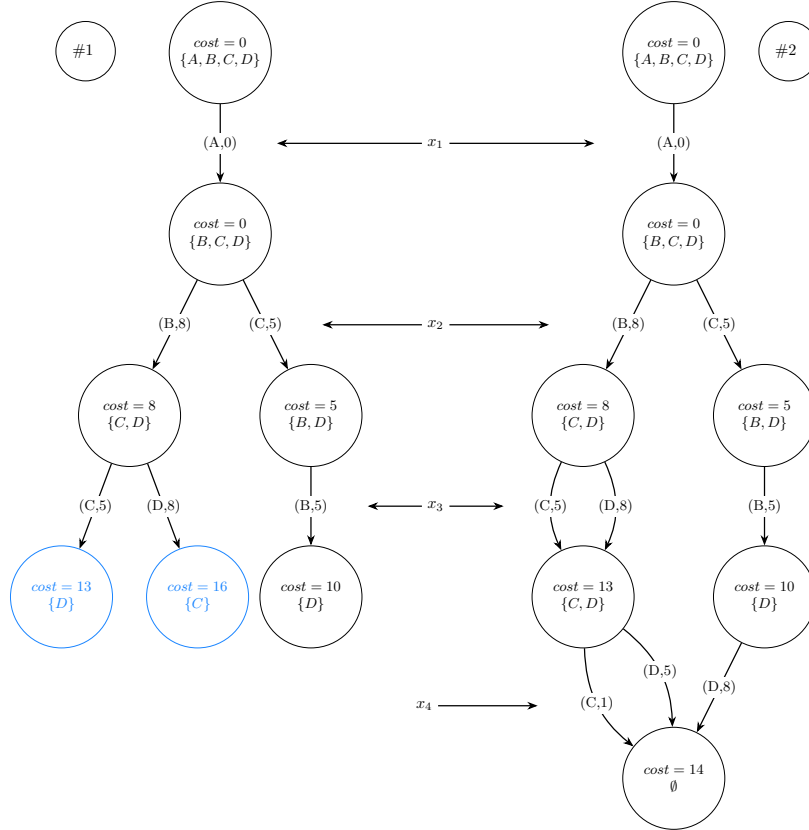


Figure 2.8 Top-Down Compilation of Relaxed Decision Diagram for  $\mathcal{P}_{SOP}$

Node merges shown in blue.

### 2.4.2 Relaxed Decision Diagrams by Separation

Top-down generation of relaxed DDs is appealing because it is conceptually simple and quite powerful. However, there is another approach that allows for substantially more structural manipulation of the DD. A relaxed DD can instead be constructed by separation [12, 26]. The fundamental insight in making the shift from top-down to separation is recognizing that it is trivial to construct a relaxed DD with  $\omega = 1$ . If you were to make a  $\omega = 1$  DD top-down, you would merge every layer into a single node. If  $u$  is a node in that diagram on layer  $\ell$ , then  $d(u) = d(x_\ell)$ . Thus,  $u$  will have one out-arc for each element  $\in d(x_\ell)$ . Figures 2.9 and 2.10 show  $\omega = 1$  relaxed DDs for  $\mathcal{P}_{sack}$  and  $\mathcal{P}_{SOP}$  respectively.

To construct a relaxed DD top-down, you merge nodes on each layer until they do not exceed  $\omega$ . To construct a relaxed DD by separation, you instead split nodes on each layer until you have reached  $\omega$ , or there are no valid splits left to make. A split is a merge in reverse. A merge combines two nodes into a single node, reducing the size of the DD, and introducing potentially infeasible solutions. A split turns a single node into two nodes, increasing the size of the DD and removing infeasible solutions in the process. To perform a split, first, a node  $u$  is selected to be split into  $u$  and a new node  $u'$ . Then, a Boolean function is used to determine which in-arcs go to  $u$  and which go to  $u'$ . All of the out-arcs of  $u$  are copied onto  $u'$ . Finally, a filtering function is used to remove out-arcs of  $u$  and  $u'$  that are now infeasible.

The choice of Boolean function is heuristic, but we use the same one for all of the work in this thesis [28]. Normally, when a node  $u$  is selected to be split, it is because a path being removed passes through  $u$ . Let  $\phi$  be the label of the in-arc of  $u$  on that path. In order to choose a subset of the arcs pointing at the old node  $u$  during a split, we need a Boolean function for the in-arcs of  $u$ . Let  $All_v^\downarrow$  be the set of labels visited by every path from the root node  $r$  to node  $v$ . We use the function  $h(a_{vu}, \phi)$ , defined as follows:

$$h(a_{vu}, \phi) = \begin{cases} \mathbf{true} & \text{if } \phi \in (All_v^\downarrow \cup label(a_{vu})) \\ \mathbf{false} & \text{otherwise} \end{cases} \quad (2.1)$$

The result of using this function when splitting is that we now know that  $\phi$  is on every path to  $u'$ . Using this function, node splitting is formalized in Algorithm 8.

Let an *exact node* be a node  $u$  such that every path from  $r$  to  $u$  would result in a node with the same aggregate domain in a decision tree for the problem being solved. In other words, an exact node is a node that has only ever merged with equivalent nodes, and has only exact nodes as parents and ancestors. An exact node never needs to be split because the information encoded by the node is exact and not relaxed. With that in mind, to construct

---

**Algorithm 8:** Node Splitting [28]

---

```

1 Input: a node  $u$  on layer  $L$  in relaxed DD  $\mathcal{M}^+$ 
2 Let  $in(u)$  be the set of arcs that end at  $u$ 
3 Let  $out(u)$  be the set of arcs that originate from  $u$ 
4 Let  $a_{vu}$  be an arc going from node  $v$  to node  $u$ 
5 Let  $h(a_{vu}, \dots)$  be a function that takes in an arc (as well as any other relevant
   information) and returns true or false (see Eq. 2.1)
6 Create a new node  $u'$ 
7  $L \leftarrow L \cup \{u'\}$ 
8 foreach  $a_{vu} \in in(u)$  do
9   | if  $h(a_{vu}, \dots)$  then
10  |   | Redirect  $a$  such that  $a_{vu'}$ 
11  | end
12 end
13 foreach  $a_{uv} \in out(u)$  do
14  | Create and add arc  $a_{u'v}$  such that  $l(a_{uv}) = l(a_{u'v})$ 
15  |  $filter(a_{uv}), filter(a_{u'v})$  //  $filter(a)$  removes arcs that violate constraints
16 end
17 return  $\mathcal{M}^+$ 

```

---

a relaxed DD by separation, you split nodes on each layer until you have reached  $\omega$ , or until all remaining nodes in the layer are exact nodes. In both the original work [28] and in this work, we split the nodes in each layer going from the top down. There is no reason that it must be done in this order, but our efforts to find an alternative order that performs better, or even as well, have been unsuccessful.

There are two more heuristic choices to be made before finalizing a procedure for constructing a relaxed DD by separation. Each time a node is split, that node must have been chosen from its layer. The heuristic we use for node selection is to pick the node containing the best path through the DD. For the sake of this explanation, let  $\mathcal{P}_{\min}$  be a minimization problem. In a relaxed DD  $\mathcal{M}^+$  for  $\mathcal{P}_{\min}$ , the length  $z$  of the shortest path  $\pi$  is a dual bound on the value of the optimal solution  $z^*$ . If  $\pi$  is feasible, then  $\pi$  is the optimal solution, and  $\mathcal{P}_{\min}$  has been solved. If  $\pi$  is infeasible, then removing it from  $\mathcal{M}^+$  will cause a new path  $\pi'$  with length  $z'$  to become the new shortest path through  $\mathcal{M}^+$ . This new path has the property that  $z \leq z' \leq z^*$ . Thus, by selecting the node to split that contains the best path, we are making an attempt to improve the dual bound generated by  $\mathcal{M}^+$ .

The second heuristic decision follows directly from node selection. Once a node  $u$  is selected to be split, a decision must be made about which arc label to split on. The heuristic we just described for selecting a node to split was premised on a desire to remove the best path from  $\mathcal{M}^+$ . In order to follow through on that premise, we will simply select the arc label on an in-arc of  $u$  that contains that path. Now all of the heuristic decisions have been addressed, and Algorithm 9 formalizes this procedure for constructing relaxed DDs by separation.

---

**Algorithm 9:** Compilation of Relaxed Decision Diagrams by Separation [28]

---

```

1 Input: A  $\omega = 1$  relaxed Decision Diagram  $\mathcal{M}^+$ 
2 Recall:  $\ell(u)$  is the layer of node  $u$ 
3 Recall:  $f(a_{vu}) = f(v)$  plus the impact on the objective function of setting  $x_{\ell(u)} = l(a_{vu})$ 
4 Let  $\ominus(u, a)$  be the split operation defined by Algorithm 8 that splits on node  $u$  and
   passes arc  $a$  to a Boolean function to sort the arcs
5 Let  $in(u)$  be the set of arcs that end at  $u$ 
6 foreach layer  $L$  of  $\mathcal{M}^+$  do
7   while  $U = \{u : u \in L, u \text{ is not exact}\} \neq \emptyset$  AND  $|L| < \omega$  do
8     Select a node  $u \in U$  with the best  $f(u)$ 
9     Select the arc  $a_{vu} \in in(u)$  with the best  $f(a_{vu})$ 
10     $u' \leftarrow \ominus(u, a_{vu})$ 
11     $L \leftarrow L \cup \{u'\}$ 
12  end
13 end
14 return  $\mathcal{M}^+$ 

```

---

Figure 2.9 demonstrates the construction of a relaxed DD by separation for  $\mathcal{P}_{sack}$ . Note that this DD is different from the one constructed top-down (Figure 2.7). Despite being different, both DDs yield a valid dual bound on  $\mathcal{P}_{sack}$ . The DD constructed by separation provides a stronger bound in this instance. There is no theoretical guarantee of this, but in practice, there is often more information available about the impact a split might have during separation, than about the impact a merge might have during top-down construction. This is because when constructing a DD top-down, you only have information about the layers you have already constructed, but when constructing a DD by separation, you have information about every layer.

Figure 2.10 demonstrates the construction of a relaxed DD by separation for  $\mathcal{P}_{SOP}$ . This DD is the same as the one generated top-down (Figure 2.8), but this is merely a coincidence. As shown by  $\mathcal{P}_{sack}$ , there is no theoretical guarantee that this will happen.

Note that if you did not pick a maximum width  $\omega$  when constructing a relaxed DD by separation, and instead continued to split nodes until every layer contained only exact nodes,

then the resulting DD would be exact. This method works well on some problems, but it is not typically powerful enough on its own to solve problems of a useful size.

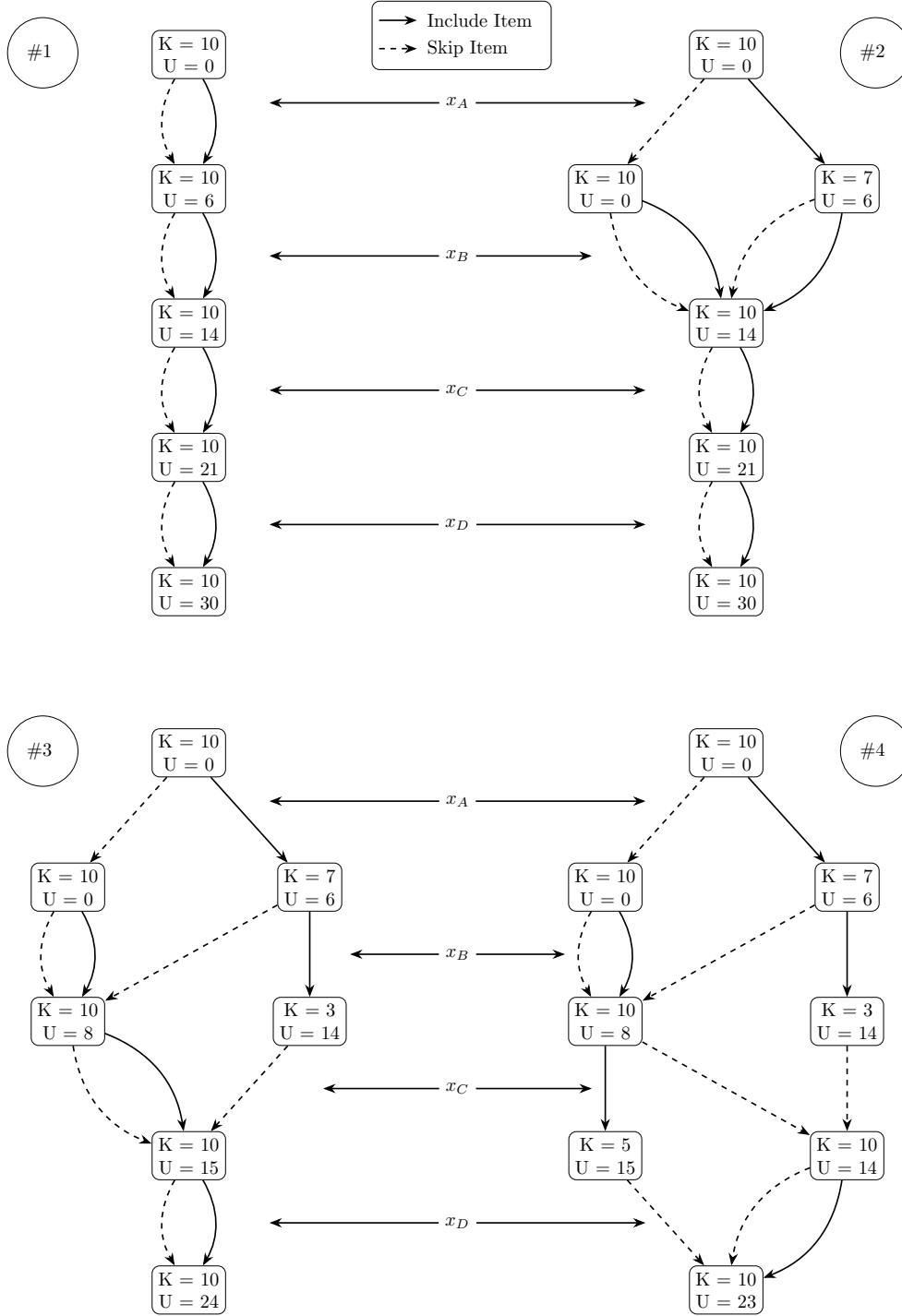


Figure 2.9 Compilation of Relaxed Decision Diagram by Separation for  $\mathcal{P}_{sack}$

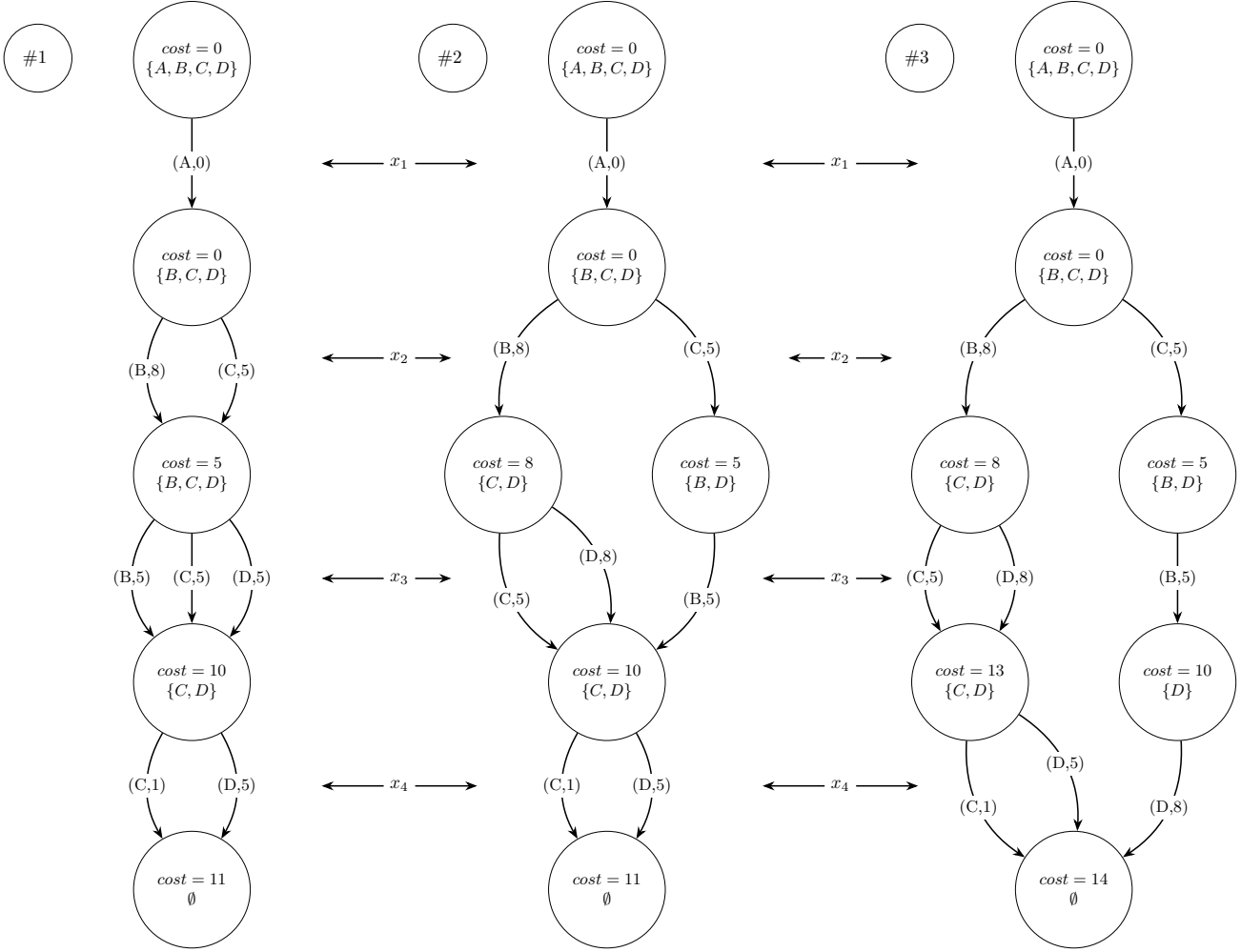


Figure 2.10 Compilation of Relaxed Decision Diagram by Separation for  $\mathcal{P}_{SOP}$

## 2.5 Decision Diagrams for Sequencing Problems

In this section, we will dive into specific methods for using relaxed DDs to solve sequencing problems. While the content of this section is included in *Decision Diagrams for Optimization* [1, 10, 28], it is not typically included in a general explanation of Decision Diagrams. It is included here because the work in this thesis makes extensive use of these techniques. Furthermore, understanding these techniques is worthwhile, because they clearly demonstrate the utility of constructing relaxed DDs by separation.

Sequencing problems have the constraint that each variable must take a different value. For example, in the well-known Traveling Salesman Problem (TSP), the salesman must visit each city in a list of  $n$  cities exactly once. Let  $i, j \in \{1, \dots, n\}$  and  $i \neq j$ , and let  $x_i$  be the  $i^{th}$  city

visited. In any solution where  $x_i = x_j$ , then there is a city being visited more than once, and so the solution is infeasible. More concretely, every variable in the TSP must take a different value. This is often referred to as the *AllDifferent* constraint by the Constraint Programming (CP) community [29].

### 2.5.1 Encoding the AllDifferent Constraint

The presence of an AllDifferent constraint, which includes all of the decision variables, allows for substantial trimming of nodes and arcs; this is the form constraint propagation takes when generating relaxed DDs by separation. In Section 2.4.2, we introduced the notation  $All_u^\downarrow$  as the set of labels visited by every path from the root node  $r$  to node  $u$ . We now also define the symmetric notation  $All_u^\uparrow$  as the set of labels visited by every path from  $u$  to  $t$ . Similarly, we define  $Some_u^\downarrow$  as the set of labels visited by at least one path from the root node  $r$  to node  $u$ , and  $Some_u^\uparrow$  as the set of labels visited by at least one path from  $u$  to  $t$ .

When constructing a relaxed DD top-down, you can track  $All_u^\downarrow$  and  $Some_u^\downarrow$ , but the symmetric information,  $All_u^\uparrow$  and  $Some_u^\uparrow$ , about paths to  $t$ , is not available. By contrast, when constructing a relaxed DD by separation, you start from a valid relaxed DD with  $\omega = 1$ . All four pieces of information can be recursively stored on each node in this initial DD.  $All_u^\downarrow$  and  $Some_u^\downarrow$  are recursively calculated for each node going top-down, and then  $All_u^\uparrow$  and  $Some_u^\uparrow$  are recursively calculated for each node going bottom-up. When a node is split, the values of these pieces of information for the new nodes can be recalculated easily from the values stored in their parents and children. Algorithm 10 explains how to calculate the initial values for  $Some_u^\downarrow$ , and can be trivially modified to handle  $Some_u^\uparrow$ ,  $All_u^\downarrow$ , and  $All_u^\uparrow$  as well.

---

#### Algorithm 10: Calculating Initial $Some_u^\downarrow$ Values [28]

---

```

1 Input: A relaxed Decision Diagram  $\mathcal{M}^+$  with root node  $r$ 
2 foreach layer  $L \in \mathcal{M}^+$  do
3   foreach  $u \in L$  do
4      $Some_u^\downarrow \leftarrow \emptyset$ 
5     foreach parent  $v$  of  $u$  do
6        $Some_u^\downarrow \leftarrow Some_u^\downarrow \cup Some_v^\downarrow$ 
7       foreach arc  $a_{vu}$  from  $v$  to  $u$  do
8          $Some_u^\downarrow \leftarrow Some_u^\downarrow \cup \{l(a_{vu})\}$ 
9       end
10    end
11  end
12 end
13 return  $\mathcal{M}^+$ 

```

---

Once the four path-based pieces of information are known, they can be used to trim nodes and arcs. We will not enumerate every method of doing so here, but we will provide two examples. In a DD with the AllDifferent constraint, a feasible path must contain every label exactly once. Given a node  $u$ , if  $|Some_u^\downarrow \cup Some_u^\uparrow| \neq n$ , then there is at least one label that does not appear on any path passing through  $u$ . Thus, there are no feasible paths that pass through  $u$ , and  $u$  can be removed from the DD. Alternatively, if  $|All_u^\downarrow \cap All_u^\uparrow| \geq 1$ , then there is at least one label on every path from  $r$  to  $u$  that is also on every path from  $u$  to  $t$ . Again, there are no feasible paths that pass through  $u$ , and  $u$  can be removed from the DD.

As a final note, recall that in Section 2.4.2, we defined an exact node as a node  $u$  such that every path from  $r$  to  $u$  results in a node with the same aggregate domain. For sequencing problems with the AllDifferent constraint, a node is exact if, and only if,  $Some_u^\downarrow = All_u^\downarrow$ , and all in-arcs of  $u$  originate from exact nodes. In other words, every label that is on any path to  $u$  must be on every path to  $u$ .

### 2.5.2 Encoding a Generalizable Optimality Constraint

Path-based information is not the only information that can be leveraged. Specific problems may have other specific information worth storing, but some information is relevant to any optimization problem  $\mathcal{P}$  being solved with a relaxed DD constructed by separation. For example, using the same recursive method that was used to calculate the path-based pieces of information, you can also calculate values that track the objective function. Let  $z_u^\downarrow$  be the length of the best path from  $r$  to  $u$ , and let  $z_u^\uparrow$  be the length of the best path from  $u$  to  $t$ . Let  $z$  be the value of the best known solution to a minimization problem  $\mathcal{P}_{\min}$ . If there is a node  $u$  such that  $z_u^\downarrow + z_u^\uparrow \geq z$ , then there is no path through  $u$  with a value lower than the value of the best known solution. Thus,  $u$  can be removed from the DD, because every path through  $u$  is worse than, or equivalent to, the best known solution. The *filter* function, referred to in Algorithm 8, leverages all of the constraints encoded using methods like the ones described in this section.

In general, any constraint that can be encoded using methods like those described in the preceding paragraphs can be used to trim nodes from the DD. The *filter* function, referred to in Algorithm 8, is adapted to each specific optimization problem for the purpose of removing nodes that violate constraints, both general constraints that apply to any optimization problem and constraints specific to the individual problem.



## 2.6 Branch-and-Bound with Decision Diagrams

Branch-and-bound (BnB) is a well-known method for solving optimization problems, particularly Mixed Integer Programs (MIPs). This algorithm divides the problem into smaller sub-problems (branching) and explores the sub-problems by estimating their bounds. If the bound of a particular sub-problem proves it cannot possibly contain the optimal solution, that sub-problem is not further explored (bounding). This approach helps significantly reduce the number of calculations by eliminating large parts of the search space.

Traditionally, BnB solvers use a linear relaxation in the form of an LP to generate dual bounds on sub-problems. A DD-based BnB solver, instead, uses a combinatorial relaxation in the form of a relaxed DD. This idea inspired a new field of research studying the potential and limitations of such a solver. This thesis is part of that effort, focused on making a cutting-edge DD-based solver that leverages DDs constructed by separation.

In a typical BnB algorithm, the branching takes place by splitting on the domain of the variables. With decision diagrams, the branching takes place on the nodes themselves by selecting a set of exact nodes that represents the problem [1, 7, 8]. The solver outline from [1] defines an exact node as a node  $u$  for which every path from  $r$  to  $u$  ends in an equivalent state. This is the language of Dynamic Programming and is equivalent to the definition given in Section 2.4.2 that every such path results in an equivalent aggregate domain.

An *exact cutset* is defined as a set of exact nodes that contain every path from  $r$  to  $t$ . Let  $\mathcal{M}^+(u)$  be a relaxed decision diagram with root  $u$ , and let  $\mathcal{M}^-(u)$  be a restricted decision diagram with root  $u$ . The branch-and-bound algorithm for DDs proceeds by selecting an exact cutset of  $\mathcal{M}^+$ , and using each node  $u$  in the cutset as the root for a new restricted decision diagram  $\mathcal{M}^-(u)$ , and relaxed decision diagram  $\mathcal{M}^+(u)$ . A node can be removed from the queue if the relaxation of that node is not better than the best known solution to  $\mathcal{P}$ , otherwise the exact cutset of the new node is added to the queue, and the process repeats until the queue is empty. This is detailed by Algorithm 11.

Figure 2.11 demonstrates DD-based BnB with  $\omega = 2$  for  $\mathcal{P}_{sack}$ . The initial restricted DD is the one from Figure 2.5, and the initial relaxed DD is the one from figure 2.7. In step #1, the initial restricted DD  $\mathcal{M}^-$  is constructed, but it is not exact, so the best solution in  $\mathcal{M}^-$  is stored:  $z_{opt} = z(\mathcal{M}^-)$ . In step #2, the initial relaxed DD  $\mathcal{M}^+$  is constructed. The upper bound on utility provided by  $\mathcal{M}^+$  of  $U = 24 > z_{opt}$ , so there may still be a better solution. An exact cutset of  $\mathcal{M}^+$  is identified, and then in steps #3 and #4, restricted DDs are generated for each node in the cutset. In this case, they are both exact, and no better solution was found, so  $z_{opt} = z^*(\mathcal{P}_{sack}) = 15$ . If one of the restricted DDs had not been

---

**Algorithm 11:** Decision Diagram based Branch-and-Bound (BnB) [8]

---

```

1 Note: This is normalized for minimization
2 Let  $\mathcal{M}(uu')$  be a partial DD with root  $u$  and terminal  $u'$ 
3 Let  $z(\mathcal{M})$  be the length of the best (shortest) path through  $\mathcal{M}$ 
4 Let  $z_{opt}$  be the value of the best known solution
5  $Q = \{r\}$ 
6  $v(r) \leftarrow 0$ 
7  $z_{opt} \leftarrow \infty$ 
8 while  $Q \neq \emptyset$  do
9    $u \leftarrow \text{selectNode}(Q)$ ,  $Q \leftarrow Q \setminus \{u\}$ ; // selects a node to process arbitrarily
10   $\mathcal{M}^- \leftarrow \mathcal{M}^-(u)$ 
11  if  $z(\mathcal{M}^-) + v(u) < z_{opt}$  then
12     $z_{opt} \leftarrow z(\mathcal{M}^-) + v(u)$ 
13  end
14  if  $\mathcal{M}^-$  is not an exact DD then
15     $\mathcal{M}^+ \leftarrow \mathcal{M}^+(u)$ 
16    if  $z(\mathcal{M}^+) + v(u) < z_{opt}$  then
17       $S \leftarrow \text{exactCutset}(\mathcal{M}^+)$ 
18      foreach  $u' \in S$  do
19         $v(u') \leftarrow v(u) + z(\mathcal{M}^+(uu'))$ 
20         $Q \leftarrow Q \cup u'$ 
21      end
22    end
23  end
24 end
25 return  $z_{opt}$ 

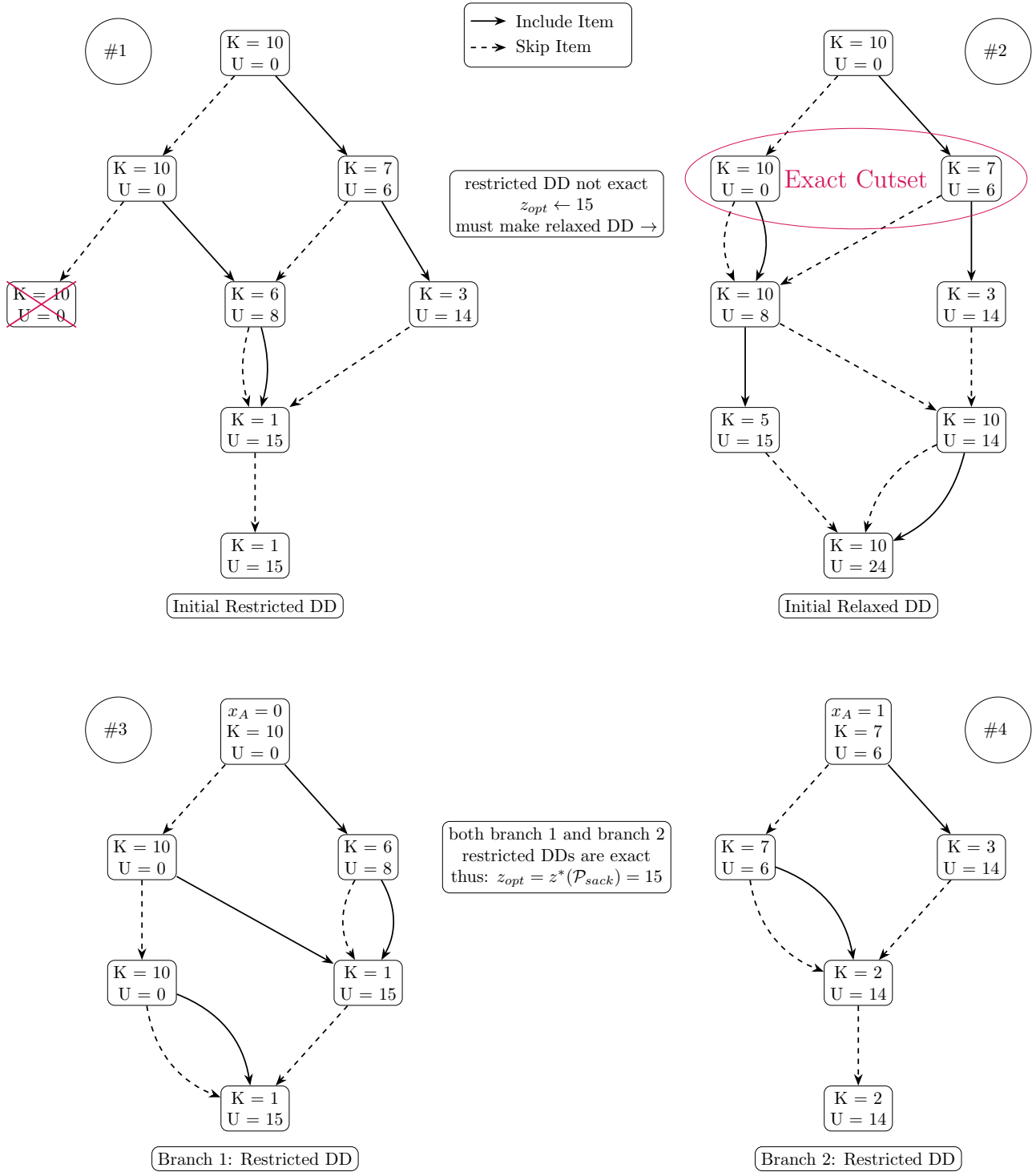
```

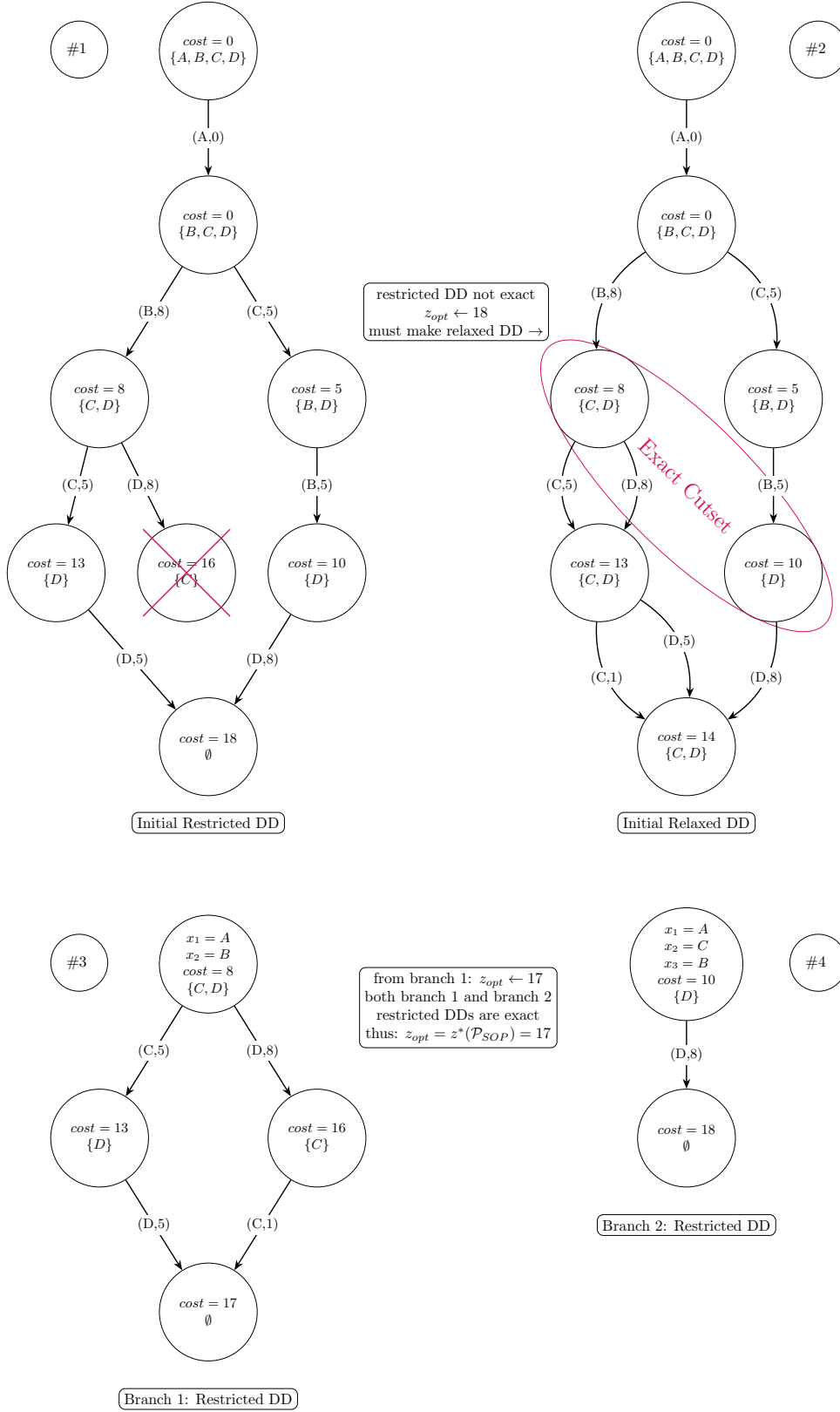
---

exact, a relaxed DD would have been generated for that branch, and the process would have continued for another iteration.

Figure 2.12 demonstrates DD-based BnB with  $\omega = 2$  for  $\mathcal{P}_{SOP}$ . The initial restricted DD is the one from Figure 2.6, and the initial relaxed DD is the one from Figure 2.10. The process unfolds almost exactly as it does with BnB for  $\mathcal{P}_{sack}$ .

A notable advantage of this procedure is that memory use is kept minimal. At each step, the previous DD is removed from memory, and the next one takes its place. The only thing that needs to be stored is the processing queue of exact nodes. Thus, even on large problems, memory use can be managed by changing  $\omega$ . However, memory use cannot be strictly defined, because the size of the processing queue is unpredictable.

Figure 2.11 DD-Based Branch-and-Bound on  $\mathcal{P}_{sack}$

Figure 2.12 DD-Based Branch-and-Bound on  $\mathcal{P}_{SOP}$

## 2.7 Other Innovations in DD-Based Solvers

In recent years, there have been significant advancements in DD-based solvers and related techniques. This research has proven the utility of DDs, applying them effectively to a variety of complex problems. This exploration has been symbiotic with other fields, with researchers continually discovering innovative ways to integrate DDs with other types of solvers. These novel methods often enhance DD-based solvers as well. As the field is still relatively new, it holds tremendous opportunity for substantial improvements in solver capabilities. This section explores some of the major efforts to leverage and enhance DD-based techniques.

### 2.7.1 ddo

An effort closely aligned with the work in this thesis is the development of the *decision-diagram-based optimization* solver (ddo) [5, 30]. To the best of our knowledge, ddo currently represents the only other initiative exploring techniques for implementing the original DD-based branch-and-bound algorithm (Algorithm 11) in a generic way. The principal difference between the ddo approach and the one detailed in this thesis is that the work on the ddo focuses on what can be achieved with relaxed DDs that are constructed top-down, compared to our emphasis on DDs constructed by separation. Although our implementations have diverged significantly, some of their ideas have directly inspired aspects of our work. These influences are discussed in the relevant chapters, but we will briefly touch on the pertinent techniques here.

In an early paper on this topic, Algorithm 11 is improved by incorporating a local search [4]. The modified algorithm uses a heuristic to quickly compute a *rough relaxed bound* at each node. If the sum of the shortest path length to a node with the rough relaxed bound for that node exceeds the best known solution, the node can be removed. More formally, let  $\mathcal{P}_{\min}$  be a minimization problem, let  $rrb(u)$  be a rough relaxed bound on the distance between  $u$  and  $t$ , let  $v(u)$  be the shortest distance from  $r$  to  $u$ , and let  $z_{opt}$  be the value of best known solution to  $\mathcal{P}_{\min}$ . If  $v(u) + rrb(u) > z_{opt}$ , the node can be removed. The authors demonstrate that if  $rrb(u)$  is inexpensive to compute, it can be useful for filtering nodes in  $\mathcal{M}^+$  and  $\mathcal{M}^-$ . However, their approach necessitates the creation of problem-specific heuristics. In Section 4.2.2, we introduce a generalized version of this concept, termed *embedded restricted decision diagrams*. This concept was developed independently from, but concurrently with, their work on dominance-based pruning [25, 31], which also uses rough relaxed bounding techniques. Thus, embedded restricted decision diagrams share some similarities with this work.

### 2.7.2 Arc-Flow Formulation

The arc-flow formulation presents a different approach to utilizing DDs. In these algorithms, each path through the DD encodes a partial solution, and achieving a complete solution involves iteratively solving a linear programming problem (LP) [32, 33]. For instance, in graph coloring [34], a DD is constructed where each path represents a single color. Then an LP is solved, treating the DD as a maximum flow graph where each unit of flow corresponds to a color. If the resulting flow is non-integer, the DD is refined, and the LP is rerun until the coloring is feasible.

This technique is quite powerful and also offers an easy way to integrate continuous variables into a DD-based solver. In a collaboration with Danielle Ripsman that has not been published yet, we applied the arc-flow formulation to design radiation therapy treatments with limited apertures. In radiation therapy, a beam is directed through a custom aperture with the objective of targeting all harmful cells while minimizing damage to healthy ones. First, we construct a DD, then we solve a MIP flow model where each positive flow on an arc signifies its use in an aperture, and finally, we refine the DD until a feasible aperture design is achieved using the MIP.

### 2.7.3 HADDOCK

The HADDOCK project, short for Handling Automatically Decision Diagrams Over Constraint Kernels, introduces a language and framework designed to integrate Decision Diagrams (DDs) with Constraint Programming (CP) techniques. This integration aims to guide and enhance the CP search process [35]. A significant focus of this work has been to fully embed a DD-based solver within a CP solver [36], leading to the development of numerous encodings, similar to those detailed in Section 2.5.

These efforts underscore the vast potential of research into blending DD-based methods with concepts from other disciplines, showcasing the innovative ways these integrations can advance work in multiple fields. The HADDOCK project also has particularly well-thought out implementations of DDs for use with other solvers. It provides a strong starting point for considering how DDs are handled in memory, and how to make DD-based operations efficient.

### 2.7.4 DIDP

Domain Independent Dynamic Programming (DIDP) is a new area of research parallel to work on DD-based solvers [37]. The ideas for DD-based solvers were directly inspired by

Dynamic Programming (DP), and DIDP is similarly inspired by DP. There are multiple implementations of the DIDP solver, all using the Dynamic Programming Description Language (DyPDL), a formalism introduced to define DP models based on a state transition system inspired by AI planning.

Since DIDP and DDs are both inspired by DP, DIDP solver techniques incorporate ideas from the DD literature for guiding searches. DIDP also integrates methods from AI planning with those DD-based techniques. This exploration will undoubtedly yield new methods of incorporating ideas from other fields alongside DDs.

### 2.7.5 Variable Ordering

It is also worth mentioning some of the relevant literature on variable ordering. The success of a DD-based algorithm can vary greatly based on the variable order used to construct the underlying DDs [21]. The study of how to generate good orderings is an active area of research.

There is an enormous amount of work on variable ordering for DDs that predates their use for optimization [38–41]. However, there has also been some research dedicated to variable ordering techniques for DD-based optimization. Two notable works include using reinforcement learning to find better orderings for a given problem [42] and using a portfolio approach to run multiple orderings in parallel and select the most successful one [43]. Further research on general improvements to variable ordering may lead to substantially more efficient DD-based solvers.

## CHAPTER 3 PEEL-AND-BOUND

### 3.1 Motivation

The motivation for peel-and-bound (PnB) stems from an observation about how DD-based branch-and-bound (BnB) (Algorithm 11) interacts with the generation of relaxed DDs by separation (Algorithm 9). When a relaxed DD  $\mathcal{M}^+$  is constructed by separation in a BnB structure, a large portion of the work done while generating each  $\mathcal{M}^+$  is repeated at every iteration. To create the relaxed DD  $\mathcal{M}_\omega^+(u)$  rooted at exact node  $u$  with width  $\omega$ , the relaxed DD  $\mathcal{M}_1^+(u)$  is constructed and then refined. To refine  $\mathcal{M}_1^+(u)$ , you iterate over each layer from the top down, splitting nodes until each layer either reaches the maximum width  $\omega$ , or has no valid splits remaining.

Consider the usual scenario where the nodes are split in a predetermined order. The static order of node splits means that for each node  $v \in \mathcal{M}_1^+(u)$ , the first split performed on  $v$  is the same in both  $\mathcal{M}_1^+(r)$  and  $\mathcal{M}_1^+(u)$ . The existing in-arcs of  $v$  will be sorted in the same way in both  $\mathcal{M}_1^+(r)$  and  $\mathcal{M}_1^+(u)$ . The only difference between the two DDs is the possibility of filtering arcs in  $\mathcal{M}_1^+(u)$  that could not be filtered in  $\mathcal{M}_1^+(r)$  due to the added constraint that all paths, in every valid construction of  $\mathcal{M}^+(u)$ , pass through  $u$ . The extra filtered arcs are the reason that  $\mathcal{M}^+(u)$  may produce a stronger bound than  $\mathcal{M}^+(r)$ . However, because splits are performed in the same order and manner each time, many arcs that were filtered while constructing  $\mathcal{M}^+(r)$  will be filtered again while constructing  $\mathcal{M}^+(u)$ .

There is a sub-graph of  $\mathcal{M}^+(r)$ , induced by node  $u$ , that contains all of the paths that will be encoded in  $\mathcal{M}^+(u)$ , but does not contain the arcs that are filtered from both DDs during construction. Thus, less work needs to be performed at each iteration of BnB by starting from that sub-graph instead of  $\mathcal{M}_1^+(u)$ .

If the split order is static, the same DD is generated starting from either  $\mathcal{M}_1^+(u)$ , or the sub-graph induced by  $u$ . If the split order changes between BnB iterations, the sub-graph induced by  $u$  is still a valid construction of  $\mathcal{M}^+(u)$ , but the generated DD will differ from the one that was refined from  $\mathcal{M}_1^+(u)$ .

Later in Section 3.3, a repeated sub-graph during BnB is visualized in Figure 3.1.



### 3.2 Algorithm

The mechanism for starting each relaxed DD after  $\mathcal{M}^+(r)$  from a sub-graph of a previously generated DD, can be embedded into a slightly modified version of DD-based BnB (Algorithm 11). In PnB, the queue stores DDs instead of nodes. After the initial relaxation  $\mathcal{M}^+(r)$  is generated, the entire DD is placed into the queue  $Q$  such that  $Q = \{\mathcal{M}^+(r)\}$ . Then, a DD  $\mathcal{M}^+(u)$  is selected from  $Q$  (for the first iteration  $\mathcal{M}^+(u) = \mathcal{M}^+(r)$ ). However, instead of selecting an exact cutset of  $\mathcal{M}^+(u)$ , only a single exact node  $e$  from  $\mathcal{M}^+(u)$  is selected. Then,  $e$  will be *peeled* from  $\mathcal{M}^+(u)$ . The process of selecting a DD and exact node are heuristic decisions that are discussed in Section 3.4.

The process of peeling  $e$  to construct  $\mathcal{M}^+(e)$  is as follows. Remove  $e$  from  $\mathcal{M}^+(u)$ , and then let  $e$  be the root node of a new DD  $\mathcal{M}^+(e)$ . Note that  $e$  still has out-arcs that end in  $\mathcal{M}^+(u)$ . For each node  $v$  in  $\mathcal{M}^+(u)$  with an in-arc that originates in  $\mathcal{M}^+(e)$ , a new node  $v'$  is made and added to  $\mathcal{M}^+(e)$ . Each in-arc  $a_{ov}$  of  $v$  that originates in  $\mathcal{M}^+(e)$  is removed, and then arc  $a_{ov'}$  is added to  $\mathcal{M}^+(e)$ . Then, a copy of each out-arc of  $v$  is made with origin  $v'$  instead of  $v$ , and the out-arcs of  $v$  and  $v'$  are filtered using filter functions like the ones described in Section 2.5. The process of removing and adding arcs is repeated until there are no arcs ending in  $\mathcal{M}^+(u)$  that originate in  $\mathcal{M}^+(e)$ . Notably, this procedure not only accomplishes a top-down reading of the sub-graph induced by  $e$ , it also potentially strengthens  $\mathcal{M}^+(u)$  by removing nodes and arcs in the process. The peel operation is formalized in Algorithm 12.

PnB simply performs peels and refinements until optimality can be proven. If the shortest path through the modified  $\mathcal{M}^+(u)$  is better than the best known solution,  $\mathcal{M}^+(u)$  is put back into  $Q$ . Then  $\mathcal{M}^+(m)$  is refined using Algorithm 9, and, if the shortest path through the refined  $\mathcal{M}^+(m)$  is better than the best known solution,  $\mathcal{M}^+(m)$  is added to  $Q$ . The whole procedure is repeated until there are no nodes left in the queue ( $Q = \emptyset$ ). PnB is formalized in Algorithm 13.

#### 3.2.1 Complexity Analysis

Separating each node  $u$  during a peel requires creating a new node  $u'$ , moving the in-arcs of  $u$  that originate in the peeled DD  $u$  to  $u'$ , copying the out-arcs of  $u$  to  $u'$ , and then filtering the out-arcs of  $u$  and  $u'$ . Creating a new node in our implementation has a time in  $\mathcal{O}(n)$ , where  $n$  is the number of possible decisions, due to storing state information that has a size in  $\mathcal{O}(n)$  (such as  $All_u^\downarrow$ ). However, it is possible that, in other applications, the size of a node is in  $\mathcal{O}(1)$ . The number of in-arcs of  $u$  is at most  $\omega$ , although this worst case is unlikely in practice because it requires  $\mathcal{M}^+$  to have width  $\omega$ , and for each node in  $\mathcal{M}^+$  on layer  $\ell(u) - 1$

---

**Algorithm 12:** Peeling Procedure
 

---

```

1 Input: A relaxed DD  $\mathcal{M}^+$  with root  $r$  and terminal  $t$ , and a node  $u$  in  $\mathcal{M}^+$ 
2 Output: a relaxed DD  $\mathcal{M}^+(u)$  peeled from  $\mathcal{M}^+$ , and what remains of  $\mathcal{M}^+$ 

3 Notation Reference:
4  $in(u)$  is the set of arcs that end at node  $u$ 
5  $out(u)$  is the set of arcs that originate from node  $u$ 
6  $in(\mathcal{M}^+)$  is the set of arcs that end in DD  $\mathcal{M}^+$ 
7  $out(\mathcal{M}^+)$  is the set of arcs that originate in DD  $\mathcal{M}^+$ 

8 Let  $\mathcal{M}^+(u)$  be an empty DD
9  $in(u) \leftarrow \emptyset$  // remove the in-arcs of  $u$ 
10  $\mathcal{M}^+ \leftarrow \mathcal{M}^+ \setminus u$ 
11  $\mathcal{M}^+(u) \leftarrow u$  // initialize  $\mathcal{M}^+(u)$  to be just  $u$ 
12 while  $in(\mathcal{M}^+) \cap out(\mathcal{M}^+(u)) \neq \emptyset$  do
13   foreach node  $m \in \mathcal{M}^+$  with an in arc that originates in  $\mathcal{M}^+(u)$  do
14     Create a new node  $m'$ , and add it to  $\mathcal{M}^+(u)$ 
15     foreach arc  $a_{md} \in out(m)$  do
16       | Add arc  $a_{m'd}$ 
17     end
18     foreach arc  $a \in in(m)$  that originates in  $\mathcal{M}^+(u)$  do
19       | Change the destination of  $a$  to  $m'$ 
20       | filter( $a$ )
21     end
22     foreach arc  $a \in out(m)$  do
23       | filter( $a$ )
24     end
25   end
26 end
27 while  $\exists m \in \mathcal{M}^+$  with  $in(m) = \emptyset \vee out(m) = \emptyset$  (excluding  $r$  and  $t$ ) do
28    $in(m) \leftarrow \emptyset$ 
29    $out(m) \leftarrow \emptyset$ 
30    $\mathcal{M}^+ \leftarrow \mathcal{M}^+ \setminus \{m\}$ 
31 end
32 return  $(\mathcal{M}^+(u), \mathcal{M}^+)$ 

```

---

to have an arc ending at  $u$ . Thus, moving the in-arcs of  $u$  has a time in  $\mathcal{O}(\omega)$ . The number of out-arcs of  $u$  is at most  $n$ , and each arc has a size in  $\mathcal{O}(1)$ , so copying the out-arcs has a time in  $\mathcal{O}(n)$ . Each individual filtering process has a time in  $\mathcal{O}(1)$  as it uses only existing state information from  $u$  and  $u'$ , and it is performed on the (at most  $2n$ ) out-arcs of  $u$  and  $u'$ . Thus, filtering the out-arcs has a time in  $\mathcal{O}(n)$ . Therefore, separating one node during the peel process has a time in  $\mathcal{O}(n + \omega)$ . Separations during a standard relaxation procedure require selecting a node ( $\mathcal{O}(\omega)$ ), making a new node ( $\mathcal{O}(n)$ ), partitioning the in-arcs ( $\mathcal{O}(n\omega)$ ),

---

**Algorithm 13:** Peel-and-Bound (PnB) Algorithm

---

```

1 Input: The initial relaxed DD  $\mathcal{M}^+(r)$ 
2 Notation Reference:
3  $z(\mathcal{M})$  is the length of the best (shortest) path through  $\mathcal{M}$ 
4  $Q \leftarrow \{\mathcal{M}^+(r)\}$ 
5  $z_{opt} \leftarrow \infty$  // Value of the best known solution.
6 while  $Q \neq \emptyset$  do
7    $\mathcal{D} \leftarrow \text{selectDiagram}(Q)$ ,  $Q \leftarrow Q \setminus \{\mathcal{D}\}$ 
8    $u \leftarrow \text{selectExactNode}(\mathcal{D})$ 
9    $\mathcal{M}^+(u), \mathcal{D}^* \leftarrow \text{peel}(\mathcal{D}, u)$  // See Algorithm 12
10  if  $z(\mathcal{D}^*) < z_{opt}$  then
11     $Q \leftarrow Q \cup \{\mathcal{D}^*\}$ 
12  end
13   $\mathcal{M}^- \leftarrow \mathcal{M}^-(u)$ . // See Algorithm 5
14  if  $z(\mathcal{M}^-) < z_{opt}$  then
15     $z_{opt} \leftarrow z(\mathcal{M}^-)$ 
16  end
17  if  $\mathcal{M}^-$  is not exact then
18     $\mathcal{M}^+ \leftarrow \mathcal{M}^+(u)$  // See Algorithm 9
19    if  $z(\mathcal{M}^+) < z_{opt}$  then
20       $Q \leftarrow Q \cup \{\mathcal{M}^+\}$ 
21    end
22  end
23 end
24 return  $z_{opt}$ 

```

---

copying the out-arcs ( $\mathcal{O}(n)$ ), and filtering the out-arcs ( $\mathcal{O}(n)$ ). The reason that there can be more in-arcs during a standard relaxation procedure is because the nodes in a  $\omega = 1$  DD can have in-arcs with different labels coming from the same node, whereas the structure of the DD during a peel guarantees that each node  $u$  can have only one in-arc from each node on the layer  $\ell(u) - 1$ . Thus, the total time for a separation in a standard relaxation is in  $\mathcal{O}(n\omega)$ .

The maximum number of separations during a peel is the maximum number of nodes in the peeled DD. A peeled DD can have at most  $(n-3) \times \omega + 2$  nodes, and thus the number of nodes is in  $\mathcal{O}(n\omega)$ . Therefore, the entire peel process has a time in  $\mathcal{O}(n^2\omega + n\omega^2)$ . The maximum number of separations during a standard relaxation is the exact same as during a peel, since the resulting DD will be the same size. Thus, the standard relaxation has a total time in  $\mathcal{O}(n^2\omega^2)$ . However, PnB uses a peel to generate some fraction of the nodes, then a standard relaxation to generate the rest. Let  $\alpha$  be the percent of nodes that are peeled during the peel. It follows that the total time for an iteration of PnB is in  $\mathcal{O}(\alpha(n^2\omega + n\omega^2) + (1 - \alpha)(n^2\omega^2))$ .

Therefore, the larger that  $\alpha$  grows, the more time PnB saves over BnB.

### 3.3 Example and Visualization

To visualize a peel operation, we use a SOP instance, but not the one described in Section 2.1.2, because it has too few arcs to yield an informative visualization. Consider a new SOP instance where the goal is to order the elements  $[A, B, C, D]$ , subject to the precedence constraint that  $A$  must precede  $D$ , an alphabetical ordering heuristic, and  $\omega = 3$ . For the sake of simplicity, we ignore arc costs.

First, we motivate PnB by identifying where BnB is repeating computations. Figure 3.1 shows  $\mathcal{M}^+(r)$ , and identifies an exact node  $e$ . Then, it shows  $\mathcal{M}^+(e)$  in three stages. The first stage is the initial  $\omega = 1$  DD. The second stage is after one split on each layer, and the third stage is the complete DD. The sub-graph shared by  $\mathcal{M}^+(r)$  and  $\mathcal{M}^+(e)$  is highlighted in blue, indicating that, in this case, the first two splits could have been read from  $\mathcal{M}^+(r)$  instead of being re-created from scratch.

Figure 3.2 demonstrates a peel operation. Observe that the subgraph from Figure 3.1 is identified and peeled from the graph, without the need to redo work. Additionally, note that the DD the peel was initiated from was refined during the peel process. There is now less remaining work needed for PnB to process the DD than there was before the peel.

### 3.4 Advantages and Implementation Decisions

#### 3.4.1 Node Selection

The original DD-based BnB algorithm [8] requires selecting an exact cutset of  $\mathcal{M}^+$ . PnB requires selecting a DD from the queue and an exact node to start the peel process. The choice of node has a substantial impact on how quickly the process converges to an optimal solution, because it serves two purposes simultaneously. As discussed earlier, the first purpose of peeling is to avoid recreating a portion of the DD at each iteration. The second purpose is to strengthen the overall relaxation.

Let  $\mathcal{M}^+(u)$  be a DD peeled from  $\mathcal{M}^+$ , and let  $\mathcal{M}_*^+$  be  $\mathcal{M}^+$  after the peel operation. If  $Sol(\mathcal{P}) \subseteq Sol(\mathcal{M}^+)$  then  $Sol(\mathcal{P}) \subseteq Sol(\mathcal{M}_*^+) \cup Sol(\mathcal{M}^+(u))$ . The only step of PnB that removes paths is the *filter* step, which only removes an arc if no feasible solutions can pass through that arc. If the node the peel is induced from contains the shortest path through  $\mathcal{M}^+$ , then there will be a new shortest path through  $\mathcal{M}_*^+$  with  $z(\mathcal{M}_*^+) \geq z(\mathcal{M}^+)$ . Similarly after peeling, the peeled DD is going to be strengthened and  $z(\mathcal{M}_\omega^+(u)) \geq z(\mathcal{M}^+(u))$ . Therefore,

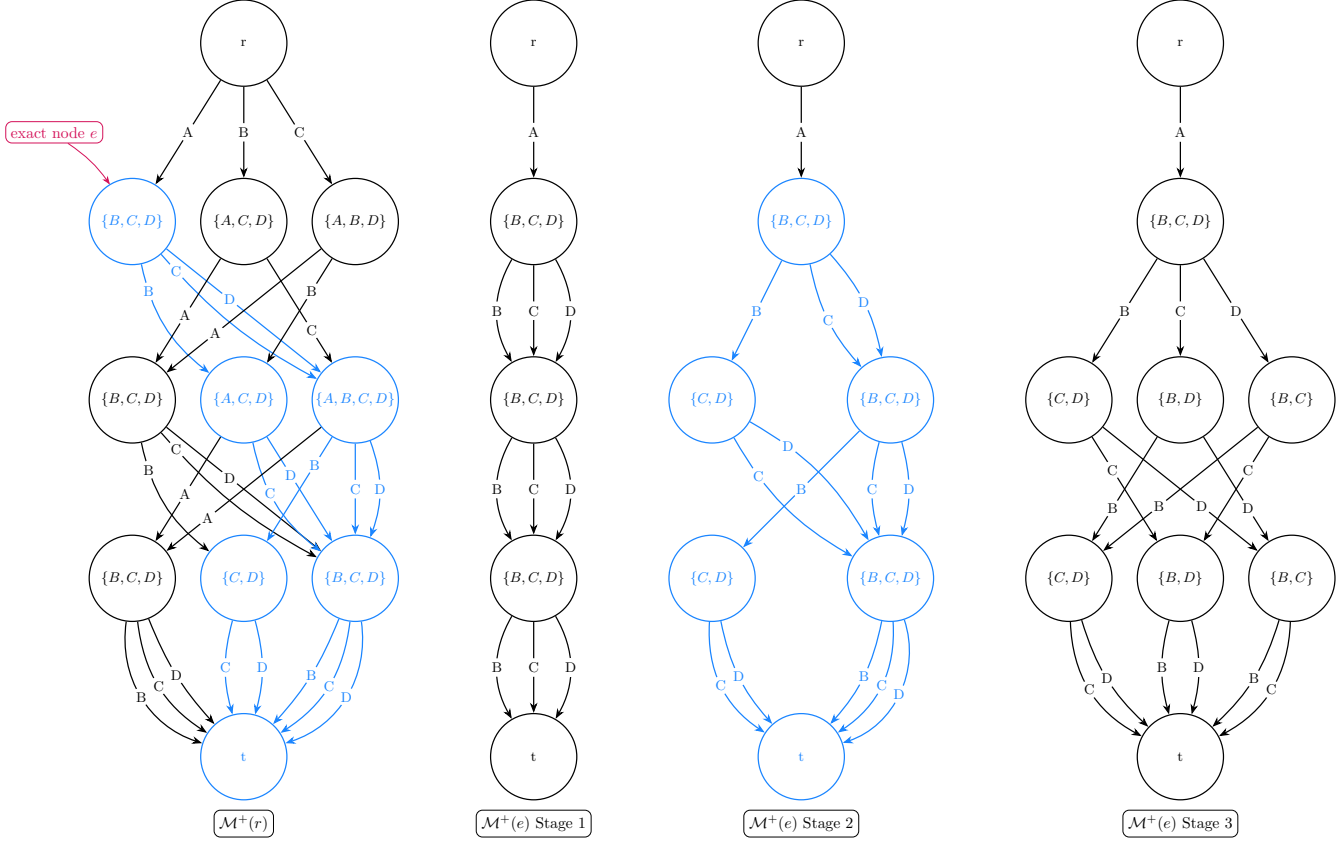


Figure 3.1 Example of a Repeated Sub-Graph During Branch-and-Bound

A sub-graph (shown in blue), and the associated relaxed DD with the same root, for a SOP instance. The objective is to order the elements  $[A, B, C, D]$ , subject to the precedence constraint that  $A$  must precede  $D$ , an alphabetical ordering heuristic, and  $\omega = 3$ .

when implementing the *selectDiagram* and *selectExactNode* functions from Algorithm 13, we propose selecting the DD  $\mathcal{D}$  with the weakest bound, and an exact node from  $\mathcal{D}$  that contains  $z^*(\mathcal{D})$  at each iteration. Using these parameters, the peel step of PnB strengthens the relaxed bound of  $\mathcal{P}$ , in addition to providing a stronger initial DD to use when generating  $\mathcal{M}^+(u)$ .

We originally proposed two heuristics for selecting a node from  $\mathcal{D}$  that contains  $z(\mathcal{D})$ . The first heuristic picks the first node in the shortest path through the DD with at least one child that is not exact, we call this the *last exact node*. The second heuristic picks the *frontier* node, the highest-index exact node that contains  $z(\mathcal{D})$ . Taking the last exact node is more of a breadth-first search that peels a large set of nodes where all of the paths to the nodes in the set share a beginning with  $z^*(\mathcal{D})$ . In contrast, taking the frontier node is more of a depth-first search, taking fewer nodes and exploring those nodes at greater depth.

### 3.4.2 Limitations and Handling Memory

The experiments we performed are on sequencing problems, but PnB can be easily applied to other optimization problems. However, some existing DD-based methods conflict with PnB. For example, some DD algorithms use a dynamic variable order [43], such that the variables the layers on  $\mathcal{M}^-$  are mapped to in one iteration of BnB, are different in the next. PnB as it is presented in this work cannot be paired with a dynamic variable order. Furthermore, the method in this work is specific to DDs generated using separation. We explain how to extend the framework to DDs that use a merge operator in Section 4.2.3.

Memory limitations present a problem for PnB in theory, but not in practice. Each open DD remains in the queue, and thus must be stored in memory. However, this problem can be handled in many ways; two are given here. A dynamic method of handling the problem is to start targeting large DDs with bounds close to  $z_{opt}$  as memory limitations start to become a problem. Such DDs can usually be closed quickly, and subsequently removed from memory, freeing up space for the algorithm to continue. Alternatively, the DDs with bounds closest to  $z_{opt}$  can be deleted in favor of storing just the root node, then when they need to be processed, initial DDs are generated for those once again. This method essentially falls back to BnB until memory limitations cease to be a problem. Additional approaches for working with memory limitations, and evidence that the problem can be handled efficiently have also explored [44].

### 3.4.3 Integrating Rough Relaxed Bounds

This implementation incorporates the rough relaxed bounding method (Section 2.7.1) [4]. Rough relaxed bounding was used to trim the domain of each node during construction of the restricted DDs, and was also added as a check to the *filter* function in Algorithm 9. When the initial model is created, a map is also created from each node  $u$ , to a list of the other nodes sorted by their distance from  $u$ . For the SOP (Section 2.1.2), the rough relaxed bound  $rrb(a)$  of an arc  $a_{fg}$  was calculated as follows. For each node  $u$  that has not necessarily been visited ( $u \notin All_g^\downarrow$ ), look up the shortest distance from that node to a different node that has also not been visited. Then, sort the resulting list, and repeatedly remove the largest value until the list has a length equal to the number of remaining decisions. The sum of the values in the list, plus the value of the shortest path from  $r$  to the end of  $a$ , is the rough relaxed bound of  $a$ . If  $rrb(a)$  is worse than the best known solution, the arc is removed.

### 3.4.4 Parallelization

The DD-based BnB shown in Algorithm 11 is particularly amenable to parallelization [5, 7]. Algorithms seeking to parallelize must overcome the data-race problem. In other words, if multiple processors are working on a problem simultaneously, then there must be a process in place to stop them from trying to write to the same place in memory at the same time. For many algorithms, this poses a substantial challenge or creates substantial overhead. However, for both DD-based BnB, and PnB, the solution is simple. As a problem is being solved, nodes (or DDs in the case of PnB) are placed into a processing queue. Each node/DD represents a discrete problem that needs to be solved, and can be processed separately. Given access to a sufficiently large number of processors, each node/DD added to the queue could be immediately dispatched to an available processor for processing. The only communication required between the processors is the current value of the best known solution.

In theory, this method of parallel processing could result in a linear improvement in time spent solving a problem when increasing the number of processors available, because  $k$  processors may be able to process  $k$  nodes, in the time it takes 1 processor to process 1 node. This process could also result in a superlinear speedup due to the non-deterministic processing order of elements in the queue. In the parallel implementation, one processor may find an improved bound which can then be utilized by the other processors. Therefore, elements from the queue could be processed using bounds that are better than the bounds used by the single-thread deterministic implementation. The ability to identify better bounds earlier while parallel processing could yield less elements that need to be processed overall. In practice, however, there are heuristic decisions that must be made that can have a large impact on solve time.

The dilemma one encounters in implementation is the existence of a critical path in the solution finding process. To demonstrate this with an extreme example, consider a problem that does the following when solved using PnB on a single processor. Each time a node is peeled, one of the two resulting DDs is solved and closed without requiring any additional peel operations. Then, the single remaining DD is processed again, a node is peeled, and the whole process repeats itself  $m$  times. In this example, there are roughly  $2m$  DDs that need to be processed, but only 2 are ever available at the same time. So in this case,  $k$  processors would take exactly as long as 2 processors to solve the problem. We propose two methods of handling this dilemma. The first is simple; reduce the maximum width of the DDs. When solving a problem that is encountering this critical path problem, the work can be divided more equitably among the available processors by reducing the amount of work done at each iteration. When using a single processor, a higher maximum width is almost always more

desirable as long as the DDs can still be generated quickly, because the extra space makes it more likely that the DD will be solved instead of producing more DDs to add to the queue. Reducing the maximum width will increase the number of DDs that need to be processed to solve the problem, but also produces those DDs more quickly. A width too low can generate an enormous number of DDs without closing any of them. The optimal width to use is one that generates enough DDs for all available processors to have consistent work, but does not generate a large backlog of work.

The second method of handling the critical path issue is to use the peel process to redistribute work as needed. Each time a processor is available and not being used, the peel procedure can peel off a sub-graph for that processor to work on. The downside of this is that often, a lot of the work that only needed to be performed once will occur on both DDs. The first method of simply lowering the maximum width accomplishes the same goal, but each DD reaches its assigned maximum width before it is further processed, so arcs that can be processed out only need to be processed out of one DD. Any implementation of this second method would likely require more heuristic decisions to ensure the task scheduler distributes the work in a useful way.

### 3.5 Experiments on the Sequence Ordering Problem

The goal of this section is to assess the performances of PnB (Algorithm 13) as it compares to the standard DD-based BnB algorithm (BnB, Algorithm 11). Both algorithms were implemented in Julia and are open-source<sup>1</sup>.

To ensure a fair comparison, both algorithms resort to the same function for generating relaxed DDs (Algorithm 9), and the same function for generating restricted DDs (Algorithm 5). While the functions being called are the same, there are two differences at run-time. At the end of line 11 in Algorithm 9, an additional operation runs during BnB where the values of the arcs leaving layer  $j$  are updated. The second difference is that BnB starts each relaxation from a  $\omega = 1$  DD, while PnB passes a partially completed DD to the relaxation function as a starting point.

The testing environment was built from scratch to ensure a fair comparison, so it lacks the many propagators used by cutting-edge solvers like CPO to remove nodes from the PnB/BnB queue [28, 45]. However, it provides a clean comparison of the two algorithms by requiring that every function used by both BnB and PnB is exactly the same between the two, with the only differences arising due to PnB's ability to ensure that all arcs are exact from the

---

<sup>1</sup><https://github.com/IsaacRudich/ImprovedPnB>



beginning. All of the heuristic decisions that were made are identical for both algorithms.

### 3.5.1 Description of the Heuristics Considered

The SOP can be considered an asymmetric TSP with precedence constraints (Section 2.1.2). The objective is to find a minimum cost path that visits each of the  $n$  elements exactly once, and respects the precedence constraints. The method used for generating relaxed DDs requires creating a heuristic ordering of all possible arc assignments by importance. The arc values in this case are representative of the  $n$  elements in the path. The ordering used was generated by sorting the  $n$  elements, first by their average distance from the other elements, and then by the number of elements each element must precede. The resulting order places a higher importance on elements that are far away from other elements and must precede many other elements.

The BnB algorithm processes nodes in an order designed to try and improve the existing relaxed bound at each iteration. When a node  $u$  is added to the BnB queue, it is assigned a value equal to the value of the shortest path from the root  $r$  to the terminal  $t$ , that passes through  $u$ . The best known relaxed bound on the problem is the smallest value of a node in the queue, and that node is always chosen to be processed. PnB is implemented with the same goal of improving bounds at each iteration. However, PnB stores DDs, not nodes. Let the value of a DD be the value of the shortest path to the terminal. At each iteration of PnB, the DD with the lowest value is selected, and then a node is chosen from that DD to induce the peel process. All of the experiments in this chapter used a process where the selected node is the first node in the shortest path from  $r$  to  $t$  with a child node that is not exact (the last exact node). Testing was done to determine whether using the last exact node or the frontier node would perform better for the problem being considered, but there was not a significant difference between the two during any of the tests. Several of the benchmark problems were run using various DD widths, and the last exact node was chosen because it sometimes showed a very slight improvement over the frontier node. While it is likely that this choice makes a difference on some problems, it does not matter for the SOP.

### 3.5.2 Experimental Results

The experiments were performed on a computer equipped with an AMD Rome 7532 at 2.40 GHz with 64Gb RAM. The solver was tested using DD widths of 64, 128, and 256 on the 41 SOP problems available in TSPLIB [46]. For comparisons between PnB and BnB, a timestamp, new bounds, and the length of the remaining queue were recorded each time the bounds on a problem were improved. Another experiment was performed to test the

scalability of PnB at width 2048, for which only the final bounds were recorded. Execution time was limited to 3,600 seconds.

The smallest DD width tested for both methods was 64, and the largest DD width tested was 256. Table 3.1 has summary statistics for those widths as the percentage improvement demonstrated by PnB. A positive percentage always indicates that PnB performed better than BnB in that category, while a negative percentage indicates that BnB performed better. Figure 3.3 shows performance profiles for all of the experiments. Table 3.2 contains summary statistics comparing PnB at width 256 to PnB at width 2048, where a positive percentage always indicates that the width of 2048 performed better.

Table 3.1 Summary Statistics: Percentage Improvement of Peel-and-Bound Over Branch-and-Bound

RB = Relaxed Bound, BS = Best Solution, OG = Optimality Gap, QL = Queue Length.

Tables A.1 and A.2 in Appendix A show the comprehensive results.

	Width: 64				Width: 256			
	RB	BS	OG	QL	RB	BS	OG	QL
Average % Improvement	114%	0.5%	22.8%	1,647%	545%	3.3%	181%	308%
Median % Improvement	26%	0.05%	17.4%	734%	80%	1.7%	35%	141%

In (1),  $e$  is selected to induce the peel process and removed from the the original DD ( $\mathcal{M}^+(r)$  from Figure 3.1). In (2) the arcs that connect  $e$  to the original DD are moved to copies of the nodes they originally ended at, and infeasible arcs are filtered. In (3) and (4) the process is repeated until the DDs are disconnected.

Table 3.2 Summary Statistics: Percentage Improvement of Peel-and-Bound at  $\omega = 2048$  over Peel-and-Bound at  $\omega = 256$

Table A.3 in Appendix A shows the comprehensive results.

	PnB: 2048 vs PnB: 256		
	Relaxed Bound	Best Solution	Optimality Gap
Average % Improvement	19.5%	0.8%	18.6%
Median % Improvement	16.3%	0.5%	13.7%

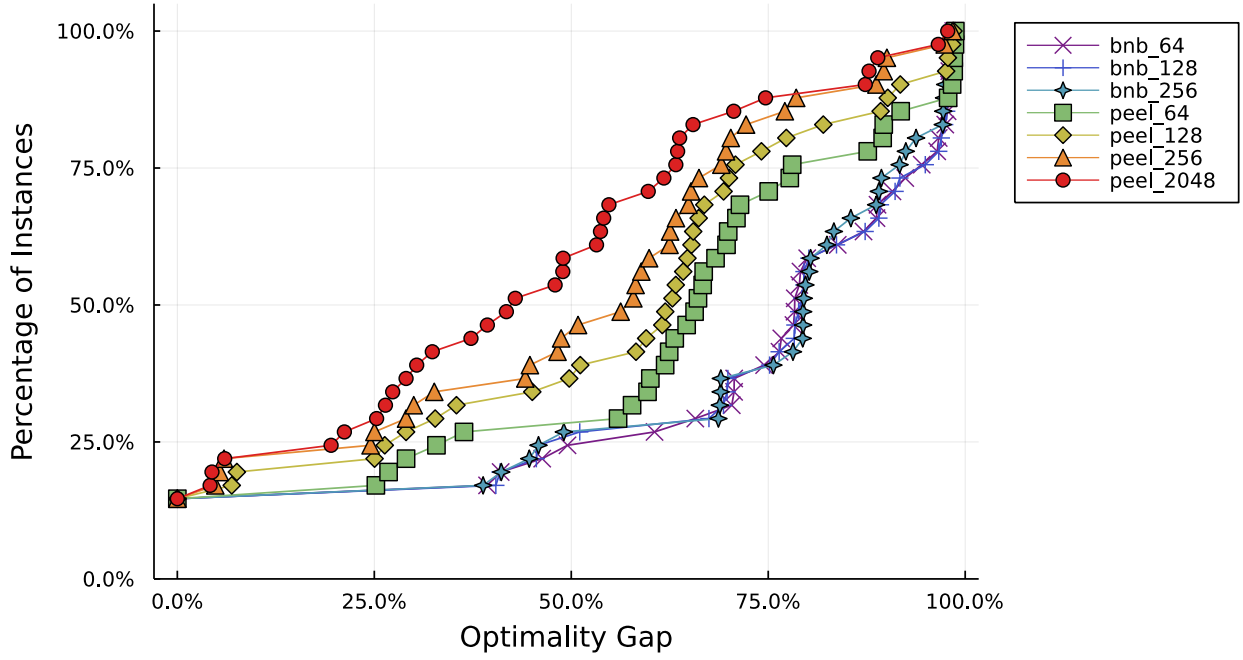


Figure 3.3 Performance Profiles: Peel-and-Bound and Branch-and-Bound

$$\text{optimality gap} = \frac{\text{upper\_bound} - \text{lower\_bound}}{\text{upper\_bound}}$$

As shown in Table 3.1, PnB vastly outperforms BnB in these experiments. The average and median improvements from using PnB at both widths are significant in terms of the relaxed bound, the remaining optimality gap, and the number of nodes that still need to be processed. The best solution found by the end of the runtime also tends to be slightly better with PnB, but the found solutions are often so close to the real optimal solutions that there is little room for improvement. At both widths, six of the problems were solved to optimality. BnB was faster in only one of those cases, and, in that case, the difference was .04 seconds. The median time for PnB to close in these cases was 191% faster at a width of 64, and 580% faster at a width of 256. The relaxed bound produced by PnB at a width of 64 was better for 28 of the remaining 35 problems, and at a width of 256 was better for 34 of the remaining 35 problems. The optimality gap was similarly better for PnB on every problem except the ones where BnB found a better relaxed bound. However, of the problems where BnB had a better optimality gap, the improvement was less than 1% for all but one problem.

Figure 3.3 reinforces that even though there are some instances where a specific BnB setting slightly outperforms a specific PnB setting, the gap in those cases is small relative to the general gap between all PnB settings and all BnB settings. As shown in Table 3.2, increasing the width to 2048 from 256 led to an 19.5% average improvement (16.3% median improvement) in the relaxed bound. Figure 3.3 also shows that the performance of PnB nearly uniformly increases with the maximum allowable width. Similar to the difference between BnB and PnB, some specific instances see a small out-performance of the PnB running at a smaller width, but the gap is small relative to the usual gap between the 2048-width experiment and the rest of the experiments. Finally, Figure 3.3 shows that `peel_2048` solved 50% of instances to within a 42% optimality gap, `peel_64` solved 50% of instances to within a 67% optimality gap, and the best performing branch and bound (`bnb_64`) solved 50% of instances to within only a 79% optimality gap. The overall performance of PnB improves as more problems are considered, especially as the maximum allowable width for the DDs is increased.

The selected graphs shown in Figure 3.4 are representative of the two main types of behavior observed over the problem set. On problems where the underlying relaxation method works well, the relaxed bound moves quickly towards convergence with the best found solution. On problems where the underlying relaxation does not work well, both algorithms slowly improve the relaxed bound, but PnB starts stronger as it can use exact arc values, and it maintains the advantage throughout. It is clear from the time-series data that to be competitive with cutting-edge solvers, PnB must be combined with other constraint programming propagators. However, it is also clear that PnB can have a significant edge over a propagator that generates the required DDs from scratch at each iteration.

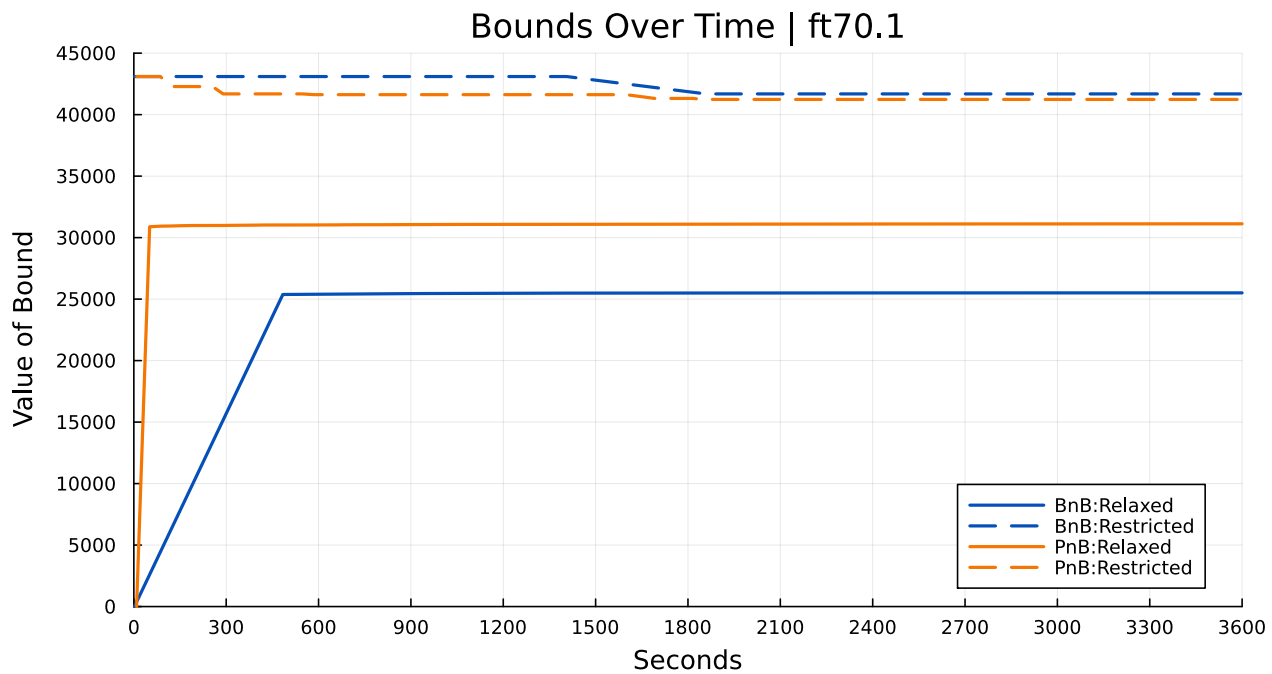
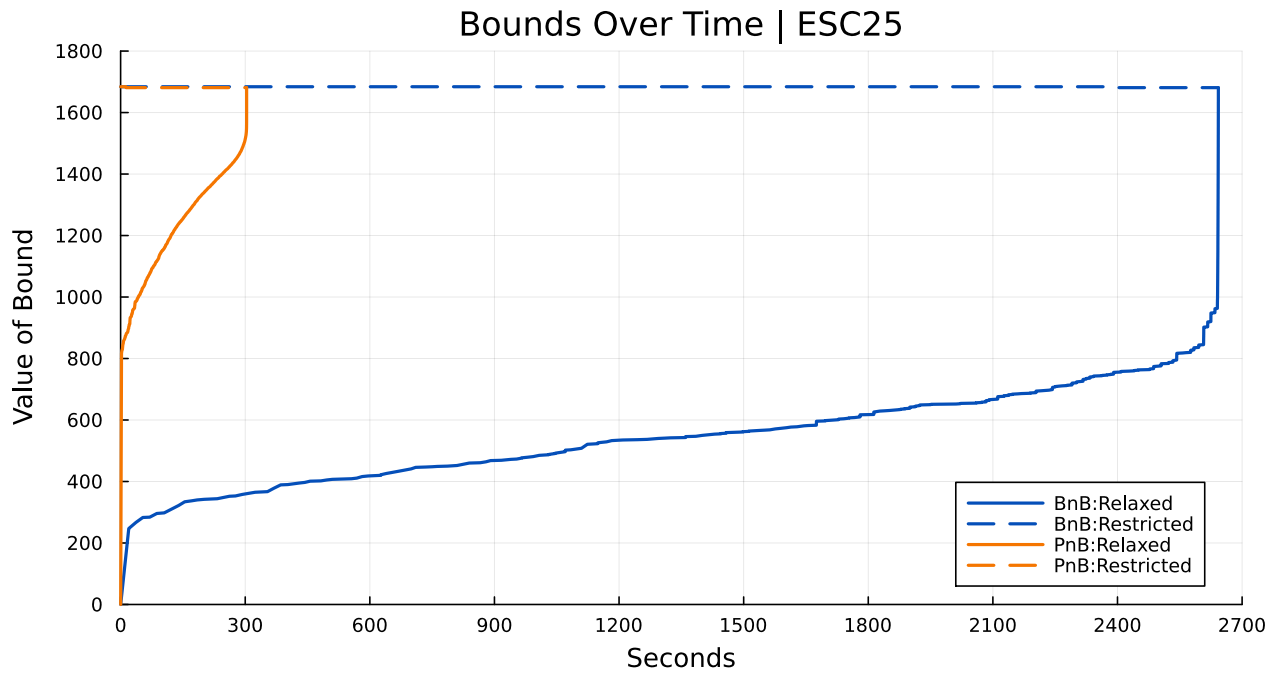


Figure 3.4 Dual bounds for ESC25 and ft70.1

ESC25 was solved within the time limit, and ft70.1 was not solved within the time limit.  
Both use  $\omega = 256$ .

### 3.6 Summary

In this chapter, we introduced the PnB algorithm as a novel alternative to DD-based BnB. In PnB, constructed DDs are repeatedly reused to avoid unnecessary computation. We discussed various heuristic strategies that can optimize PnB, and explored its potential applications to other problems.

We compared the performance of a PnB scheme to a BnB scheme using the same DD-based propagator. We tested both algorithms on the 41 instances of the SOP from TSPLIB. The results demonstrated that PnB significantly outperforms BnB on the SOP by generating substantially stronger relaxed bounds on instances that were not closed during the experiment, and reaching optimality faster when the instances were closed. This research underscores the advantages of re-using work in DD-based solvers. Additionally, PnB benefits from scaling the maximum allowable width. Thus, relaxed DDs that yield strong bounds at scale, but are too costly to generate iteratively, only need to be constructed once.

In the following chapter, we will delve into the transition from this initial implementation of PnB, to an implementation that has achieved cutting-edge results. We will outline the enhancements and optimizations that were integral to evolving the algorithm from a theoretical model into an efficient tool.

## CHAPTER 4 IMPROVED PEEL-AND-BOUND

### 4.1 Motivation

The initial inspiration for implementing PnB was that DD-based BnB repeats numerous computations while constructing DDs. PnB aims to avoid re-computations. However, the goal of the initial implementation was to simply make a clean comparison between a naive PnB implementation and a naive DD-based BnB implementation. That test (Section 3.5.2) showed that PnB was worth pursuing. We then built a new implementation from scratch, carefully considering how to make the algorithm efficient and scalable, so we could test it against other solvers.

Our primary target for improvement involved identifying and addressing bottlenecks in the algorithm. The slowest part of the first implementation is handling arcs. Benchmarking the code made it clear that having the computer perform read and write operations for every arc in the DD was consuming most of the runtime. In Section 4.2.1, we discuss a method of making all of the arc values implicit for sequencing problems. This change represents the most significant component of the speed-up we achieved. The second major improvement arose from our new search procedure detailed in Section 4.2.2, which reads a restricted DD from an existing relaxed DD, instead of generating it from scratch.

### 4.2 Improvements to the Theory

#### 4.2.1 Handling Non-Separable Objective Functions

When a problem is represented with an Integer Program, the value or cost of making a decision is separate for each variable; in other words the objective function is separable. However, when using DDs to represent a problem, the objective function can be non-separable. In the example SOP used in Section 2.1.2, the cost of an arc leaving a node  $u$  is dependent on the labels of the arcs that end at  $u$ . If the arcs have conflicting labels, then arcs leaving  $u$  may not have an exact cost. The algorithm for using DDs to solve sequencing problems [28] proposed that each iteration of Algorithm 9 starts from a  $\omega = 1$  DD. However, for PnB with a non-separable objective function, starting from a  $\omega = 1$  DD poses a problem. The arcs in such a DD do not have exact values, because they are dependent on the state of the node they originate from. As nodes are peeled, the values of those arcs must be updated, and the operation becomes computationally expensive at scale.



When solving sequencing problems where the non-separability of transition costs arises from their dependency on the previous element in the sequence, this problem can be avoided by creating the initial DD using a structure where all of the arcs ending at a given node have the same label. The resulting initial DD has a width of  $n$ , and each node on the layer is assigned to one state  $s \in \{1, \dots, n\}$ . Then every possible feasible arc between consecutive layers is added. Thus, the nodes of  $\mathcal{M}^+$  do not have relaxed states, and each arc can only take one possible value. Starting from such a DD not only removes the need to update arc values, it ensures that every arc generated during PnB is an exact copy of an arc that exists in the initial DD, since arcs are only copied or removed, never updated or added. Using this structure makes implementation easier by removing the need to store any information on the arcs at all. As each node will only have one state, the label and weight of an arc is implied by the state of the node it originates from, and the node it points to. This allows all information in a DD to be stored on the nodes, and thus only the information on the nodes ever needs to be read or updated.

When transition costs are dependent on factors other than the previous element in a sequence, such as the Time-Dependent TSP (TSP-TD), modifications to the described method become necessary. In general an exact cost may not be computable, but a relaxed bound on cost can be made computable by storing the relevant values as a range at each node. Consider a DD for the TSP-TD where nodes  $u$  and  $v$  are connected by an arc. Let  $[a_u, b_u]$  be the range of possible times one can arrive at  $u$ . While an exact cost for the arc from  $u$  to  $v$  is not available, a relaxed bound can be found by varying the time from  $a_u$  to  $b_u$ , and then using the best value found as the cost. As the DD is peeled, the gap between  $a_u$  and  $b_u$  may grow smaller, but it will never grow larger. When the gap decreases, the previously calculated cost is still a valid relaxed bound. Thus, the cost can be updated to retrieve an improved bound, but it is not necessary to do so with every peel operation. Many problems will require modifying this concept to fit their particular constraints, but this method is easy to adapt in most cases. An alternative method of handling non-separable objective functions has also been explored [15, 47, 48].

#### 4.2.2 Embedded Restricted Decision Diagrams

In both DD-based BnB (Algorithm 11), and PnB (Algorithm 13), at each branch a restricted DD is created before the relaxed DD. This is useful not just for solution finding, but also because the process of creating a restricted DD is often several orders of magnitude faster than the process of refining a relaxed DD. When the restricted DD is exact, the relaxed DD can be closed without any additional processing. PnB provides an opportunity to leverage relaxed

DDs to improve the associated restricted DDs. As discussed in Section 2.7.1, this concept was developed independently from, but concurrently with, work on dominance-based pruning, which was also inspired by rough relaxed bounding techniques [25, 31]. Thus, embedded restricted DDs share some similarities with this work.

A path in both a restricted DD and a relaxed DD represents a sequence of decisions. In a restricted DD, each path from the root to a node in the DD represents a feasible partial solution to the problem being solved. A relaxed DD contains every feasible solution, partial or otherwise. Thus, each path in a restricted DD will also exist in the associated relaxed DD. Furthermore, DDs are deterministic, so each path in a DD maps to exactly one node. This means that any possible path (sequence of decisions) in a restricted DD, will map to exactly one node in the associated relaxed DD. In other words, every possible restricted DD for an optimization problem will be embedded in a relaxed DD for that problem.

Formally, each path to a node  $u^-$  in a restricted DD  $\mathcal{M}^-$  can be mapped to exactly one node  $u^+$  in a relaxed DD  $\mathcal{M}^+$ . Recall that the domain of a node  $u$  is the set of feasible arc labels on out-arcs of  $u$ . When generating  $\mathcal{M}^-$ , each node in  $\mathcal{M}^-$  creates a child node on the next layer for every element in its domain, and then the new layer is trimmed down to a pre-chosen maximum width  $\omega$ .

The mapping from restricted DDs to relaxed DDs can be leveraged to improve the restricted DD as it is being generated. Let  $d(u^-)$  be the domain of  $u^-$ . Set  $d(u^-) \leftarrow d(u^-) \cap d(u^+)$  before generating the child nodes of  $u^-$ . This way, if an arc has been proven to be sub-optimal in  $\mathcal{M}^+$ , it will not be created when generating  $\mathcal{M}^-$ .

This also allows for easy intensification of the search when combined with peel operations. If a node  $u$  has been peeled from  $\mathcal{M}^+$  into  $\mathcal{M}^+(u)$ , then  $\mathcal{M}^-$  will not include any solutions that pass through  $u$ . Similarly, if a restricted DD  $\mathcal{M}^-(u)$  is generated that is embedded in  $\mathcal{M}^+(u)$ , it will only explore solutions that pass through  $u$ .

Without using this intersection operation as a way to trim the domain,  $\mathcal{M}^-$  will search the entire solution space that starts from the same root as  $\mathcal{M}^+$ , even if the peeled DDs have already been fully explored and are known to be sub-optimal. Using this method, each restricted DD will only search the solutions encoded within the matching relaxed DD. This means that each restricted DD has a significantly improved chance of finding the best solution embedded in the relaxed DD it maps to, because the solution space it must explore becomes smaller with each peeled node.

### 4.2.3 Peel-and-Bound with Top-Down Compilation

The structure of PnB is designed to take advantage of relaxed DDs that are compiled by separation. Here we propose a method for applying PnB to relaxed DDs that are compiled top-down. However, we have not tested this method, and it remains a topic of future research to determine if it would be useful in practice. The goal of PnB is to re-use work already done by reusing DDs. When performing top-down compilation, nodes are merged instead of separated. The peel procedure can be used exactly as before, but after a node is peeled, the remaining DDs must be altered so that new nodes can be added top-down using a merge procedure. However, there are no nodes to add, the DDs already represent feasible bounds on the problem, and so some nodes must be removed. Begin by selecting a relaxed node  $u$  (in other words some node  $u$  that is not exact), and removing it from the DD. Then remove any arcs in the DD that are sub-optimal or no longer feasible due to the removed node. For each arc  $a_{vu}$  that used to point to  $u$ , create a new arc  $a_{vu'}$  that points to a new node  $u'$  created by following the top-down compilation rules being used. Finally, proceed to perform top-down compilation using the set of new nodes as root nodes for the procedure.

## 4.3 Improvements to the Heuristics

### 4.3.1 Node Selection Heuristic

Recall from Section 3.4.1 that we initially only discussed, and experimented with, two heuristics for selecting which node to peel from a DD. After our initial experiments we have added a third node selection heuristic. We pick what we call the *maximal* node. The maximal node is simply the node on the second layer that contains  $z(\mathcal{D})$ . This peels as many nodes as possible while still picking a node that contains  $z(\mathcal{D})$ .

### 4.3.2 Search Diversification

The structure of PnB yields another method of searching for solutions that forces increased diversification. The embedded restricted DDs described in Section 4.2.2 take advantage of the reduced search space embedded in the relaxed DD, but make no effort to explore substantially different regions, and thus are at risk of getting stuck in a local optimum. Here we propose a simple method of diversifying the solutions explored. Starting from the root, and moving down layer by layer, map each node  $u$  in the relaxed DD to the best feasible path that ends at  $u$ , and is a continuation of a path a parent of  $u$  maps to. The obvious drawback is that many paths will become infeasible or sub-optimal, and many of the relaxed nodes might not

map to a feasible path using this method, simply because the paths that were being explored in their parents were bad paths. To fix this, the number of paths stored can be expanded. Let  $k$  be any positive integer; if each node maps to the  $k$  best paths to that node, then as the value of  $k$  increases there is a much higher likelihood of new, and better, solutions being found. However, as the number of nodes in the DD can be quite large, even small values of  $k$  can be computationally expensive. So this method can be a powerful tool for diversification, but it comes with a significant drawback in terms of compute time. It has the potential to be valuable if used just once at the beginning of the PnB process to search for initial solutions, but is unlikely to be useful if repeated often. This idea may also benefit from being combined with the large neighborhood search using restricted DDs [23].

#### 4.4 Experiments with our Improved Implementation

The first implementation of PnB, which was used to generate the results in Section 3.5.2, was designed to create a fair comparison of PnB and BnB; it was also limited to the SOP. The re-implementation of PnB is generic, and the new version is similarly open-source<sup>1</sup>. All of the raw data from the following experiments can be found with the code. In this section, we further explore the performance of the algorithm. The experiments were performed on a computer equipped with an AMD Rome 7532 at 2.40 GHz with 186Gb RAM.

##### 4.4.1 Node Selection Heuristic

In Sections 3.4.1 & 4.3.1, we propose three heuristics for selecting a node to be peeled: frontier, last exact node, and maximal. Here we compare the performance of those three different settings on the same set of SOP instances we tested in Section 3.5.2; we similarly limit the runtime of the solver to 3600 seconds. We include the results from the most successful run performed by the original implementation of PnB to show how much the new implementation has improved in general. The results are shown in Figure 4.1. The first graph is a scatter-plot displaying the solve time of each problem that was solved to optimality, for each of the three node selection settings. The second graph shows performance profiles for the same tests, and includes data from the best run of the first implementation of PnB to show the overall improvement of the solver. All of the new tests were performed using  $\omega = 2048$ , and included a diversified search with  $k = 5$  using the procedure described in Section 4.3.2. Figure 4.1 makes it clear that choosing between the node selection heuristics has little effect on solving the SOP. The number of instances solved to optimality is identical, with maximal

---

<sup>1</sup><https://github.com/IsaacRudich/ImprovedPnB>

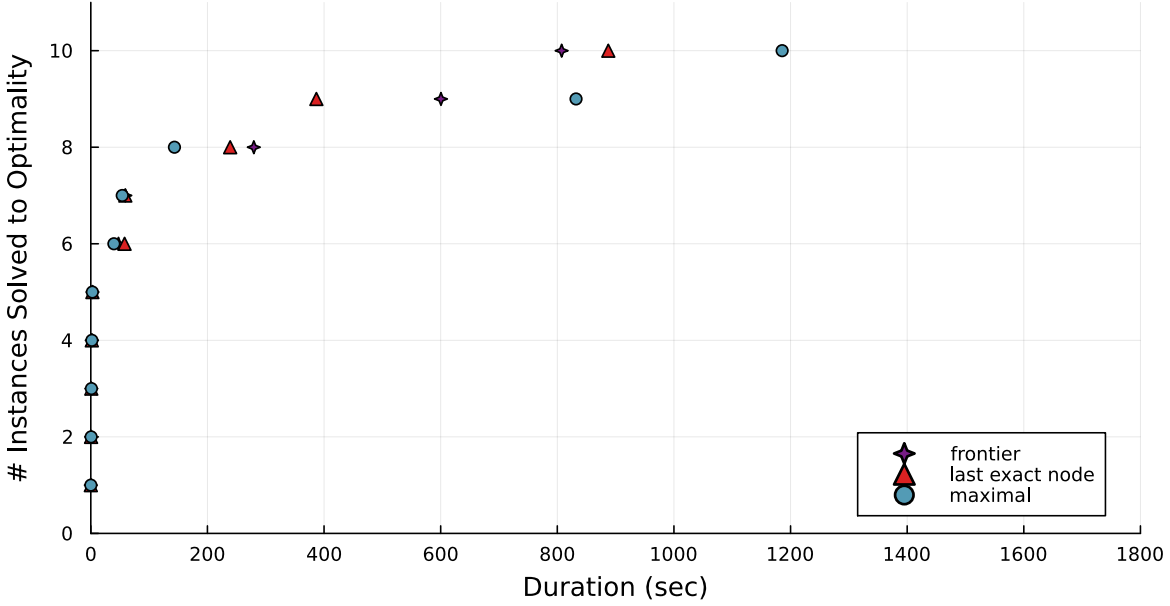
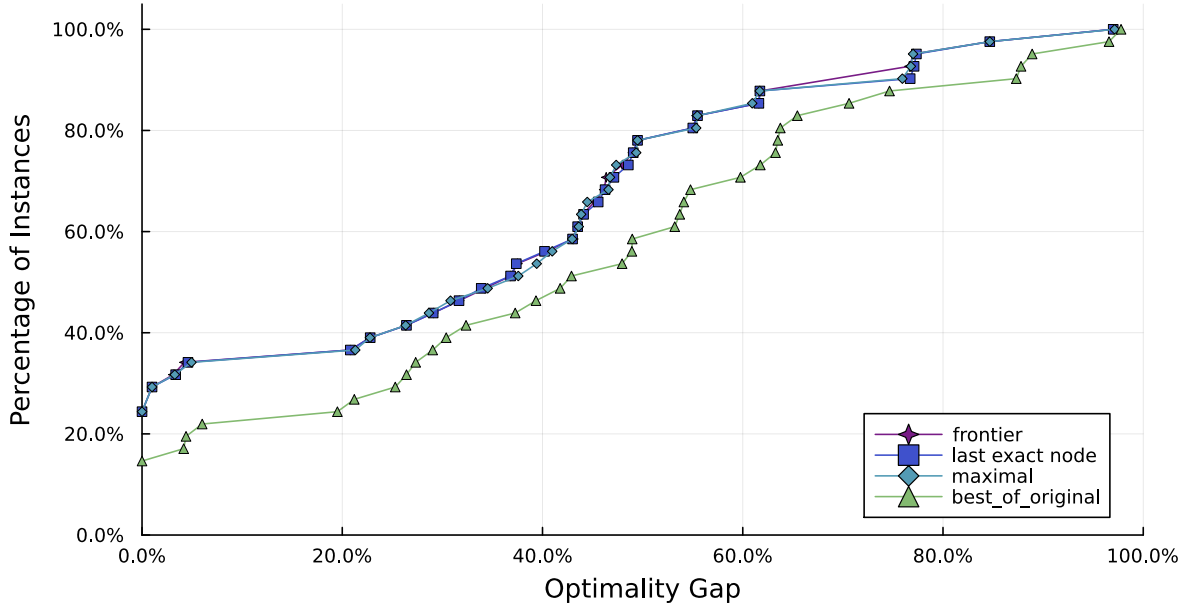
SOP Instances Solved over Time:  $\omega = 2048$ Performance Profiles on the SOP:  $\omega = 2048$ 

Figure 4.1 Performance of Improved Peel-and-Bound on SOP

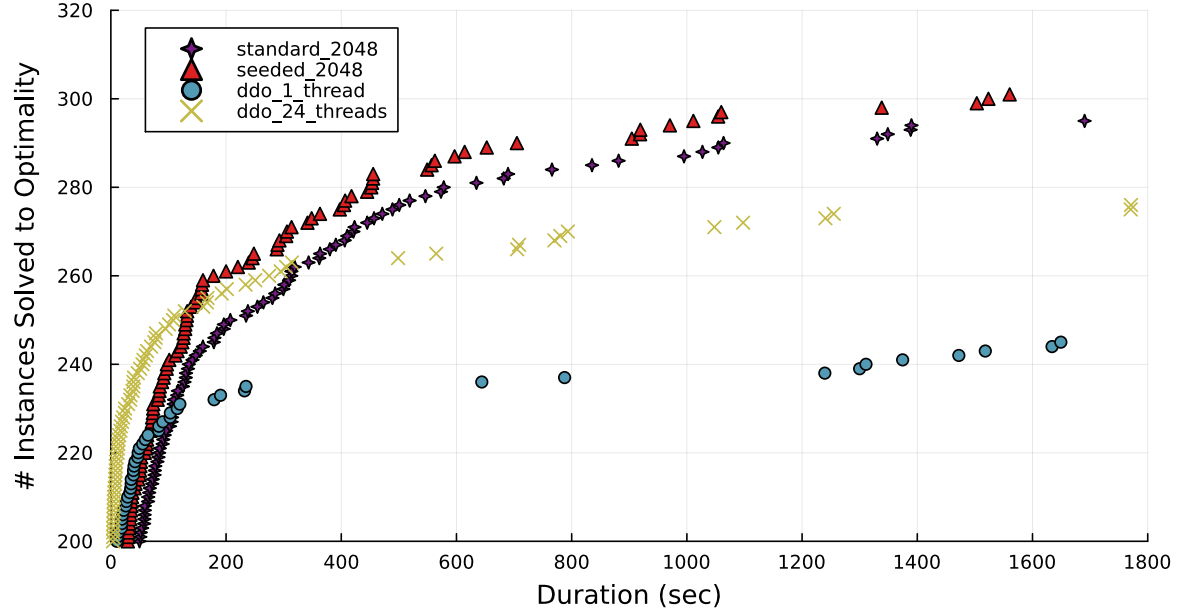
having a slight lag on the final two problems, and the performance profiles are nearly identical. The performance profiles also serve to demonstrate the progress of the solver, with the percentage of instances closed to any given optimality gap being about 10% higher.

#### 4.4.2 Traveling Salesman Problem with Time Windows

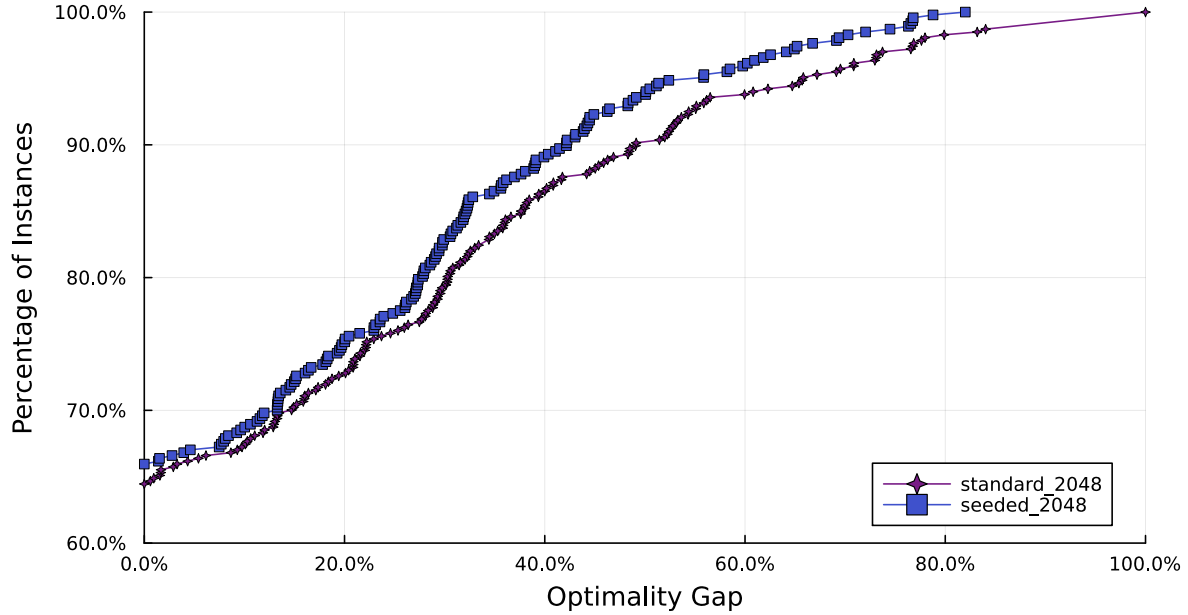
The Traveling Salesman Problem with Time Windows (TSP-TW) is a variation of the TSP where a salesman must find the shortest cycle that visits each customer once, but each customer may have an added constraint requiring they be visited within a specific time window. It provides a useful metric for benchmarking the performance of the implementation of PnB, as DD focused algorithms are more likely to outperform traditional methods on highly constrained problems than highly unconstrained problems, and the TSP-TW instances are highly constrained. For the following tests we include both a standard and seeded run of the solver, where seeded means that the solver started out knowing the value of the best known solution, and skipped the initial diversified search. The seeded version is of interest because the solver can leverage heuristically generated solutions to reach proof of optimality faster. The difference between the standard and seeded run may help to determine if taking that step is worthwhile when solving a specific problem. We test our solver on the same set of 467 benchmark instances used by ddo [4] to test their implementation of DD-based BnB. These instances are available online [49], and include the following sets: AFG [50], Dumas [51], GendreauDumas [52], Langevin [53], Ohlmann-Thomas [54], Solomon-Pesant [55], and Solomon-Potvin-Bengio [56]. Figure 4.2 shows a time to solve graph for the closed instances, once with PnB on a single thread, once with seeded PnB on a single thread, once with ddo on a single thread, and once with ddo using 24 threads. The experiments were limited to 3600 seconds. Figure 4.2 also shows the performance profiles of PnB using the same data. Since the solution space of TSP-TW tends to be drastically more constrained than SOP, at least for the benchmark instances, it is more difficult to find feasible solutions, and we use a width of 100 for the initial diversified search.

It is clear from Figure 4.2, that PnB outperforms ddo on the TSP-TW benchmark set. Although ddo is faster to start, PnB on a single thread solves about 60 more instances than ddo on a single thread, and about 20 more instances than ddo using 24 threads. This experiment plainly demonstrates the advantages of trading memory for speed when using DDs. The performance profiles show that PnB solved about 65% of the instances to optimality. The seeded version of the solver performed only slightly better. Looking at the raw data (available in the repository), it is also clear that performance degrades when PnB has trouble finding good solutions, and when the size of the time windows is large (causing the problem to be more unconstrained than others in the benchmark set).

To the best of our knowledge, the last paper to report relaxed bounds on the instances in these benchmark sets is from 2012 [57]. We use those results as a reference for comparison. The standard run closes 14 open instances, and the seeded run closes 1 additional instance.



TSP-TW Instances Solved over Time: DDO vs. Peel-and-Bound



TSP-TW Performance Profiles

Figure 4.2 Performance of Improved Peel-and-Bound on TSP-TW

The results for these instances, and the previously known relaxed bounds, are reported in Table 4.1. A table with every problem from the benchmark set that remains open to the best of our knowledge (in other words, those unsolved by both the reference paper and PnB), is available in Appendix A: Table A.4. The full data is available in Appendix A: Tables

A.6-A.11. The raw data is also available in the repository with the solver.

Table 4.1 TSP-TW Results for Newly Closed Problems

LB = Lower Bound, UB = Upper Bound, OG = Optimality Gap. In all cases the existing best known solution was proven optimal by Peel and Bound.

Set	Problem Information		Baldacci et al. (2012) LB	Single Thread: 2048				Seeded Single Thread: 2048			
	Name	Best Known Solution		LB	UB	Time	OG	LB	UB	Time	OG
AFG	rbg086a.tw	8400	8399	8400	8400	254.47	-	8400	8400	132.49	-
	rbg092a.tw	7158	7156.6	7158	7158	1917.74	-	7158	7158	288.02	-
GendreauDumas	n20w200.004.txt	293	289.484	293	293	27.97	-	293	293	15.47	-
	n40w200.002.txt	303	302.091	303	303	1388.47	-	303	303	417.75	-
	n80w100.004.txt	649	645.408	649	649	2372.02	-	649	649	904.33	-
SolomonPasant	rc203.1	726.99	726.66	726.99	726.99	299.58	-	726.99	726.99	200.01	-
SolomonPotvinBengio	rc_202.3.txt	837.72	835.87	837.72	837.72	9.66	-	837.72	837.72	8.67	-
	rc_202.4.txt	793.03	791.54	793.03	793.03	108.08	-	793.03	793.03	73.79	-
	rc_205.4.txt	760.47	756.95	760.47	760.47	13.65	-	760.47	760.47	9.74	-
	rc_206.2.txt	828.06	826.66	828.06	828.06	102.63	-	828.06	828.06	68.91	-
	rc_206.4.txt	831.67	827.54	831.67	831.67	107.39	-	831.67	831.67	84.51	-
	rc_207.1.txt	732.68	731.57	732.68	732.68	195.9	-	732.68	732.68	157.17	-
	rc_207.2.txt	701.25	694.22	701.25	701.25	1690.43	-	701.25	701.25	92.84	-
	rc_207.3.txt	682.40	677.23	682.40	682.40	1054.62	-	682.40	682.40	596.7	-
	rc_208.1.txt	789.25	785.69	751.20	794.17	-	5.41	789.25	789.25	2511.58	-

#### 4.4.3 Traveling Salesman Problem with Time Windows - Makespan

Makespan adjusts the objective function of TSP-TW so the total time includes any idle time spent waiting for a customer to be available when the salesman arrives early, as opposed to just the distance traveled. To test this problem, we use the same benchmark instances that we did in Section 4.4.2, and simply adjust the objective function. The tests all use the same settings as the test for TSP-TW, and the results are presented in the same way in Figure 4.3.

While we were unable to find a benchmark comparison in the literature with relaxed bounds for the makespan variant of these benchmark sets, the results from testing PnB on makespan speak for themselves. Just over 94% of the instances were closed to optimality by the unseeded run of the solver, and just over 97% were closed by the seeded run. For the unseeded run, just over 97.5% were closed to 1% optimality gap, and 99% for the seeded run. In total, 26 instances were not closed by the unseeded run, and 12 of those were not closed by the seeded run. The results and bounds for those 26 instances are shown in Table 4.2. We are unsure if any of the makespan problems are considered to be open, but we can say that the only ones that aren't definitively closed are the 12 not in bold that were not solved during the seeded run of the solver. Of those, only 2 have an optimality gap larger than 2%. For the 455 closed instances, the best existing solution was reported [49]. A table with full results for the benchmark sets, and time to solve for all of the closed problems, is shown in Appendix A: Tables A.12-A.18. The raw data is also available in the repository with the solver.



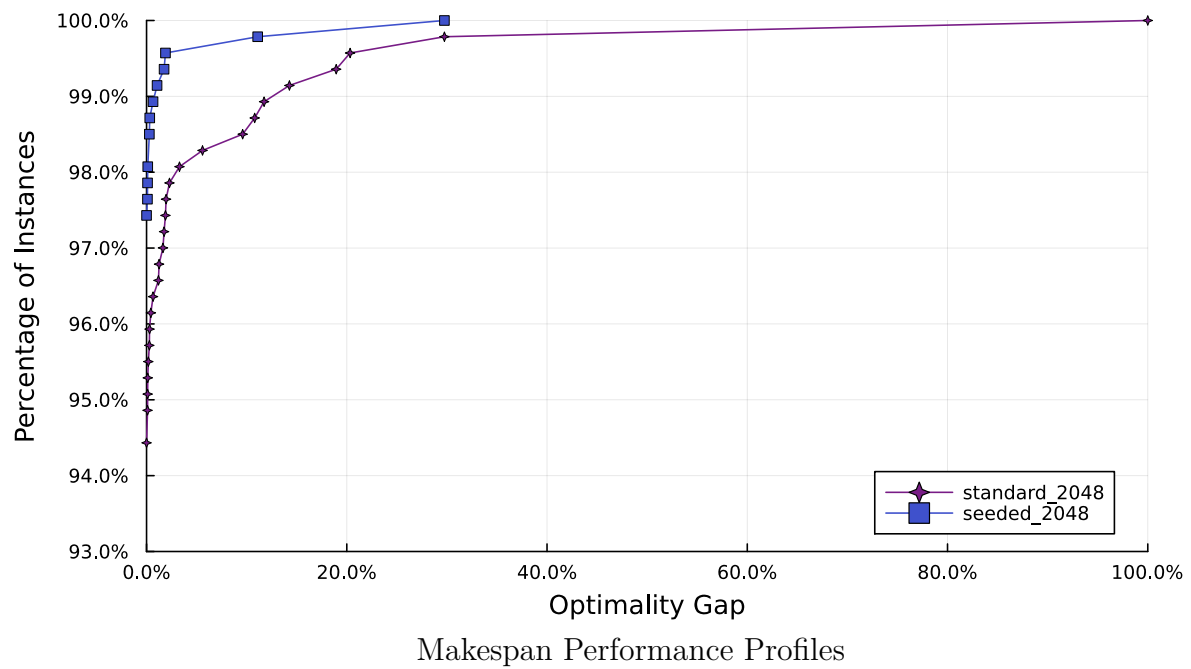
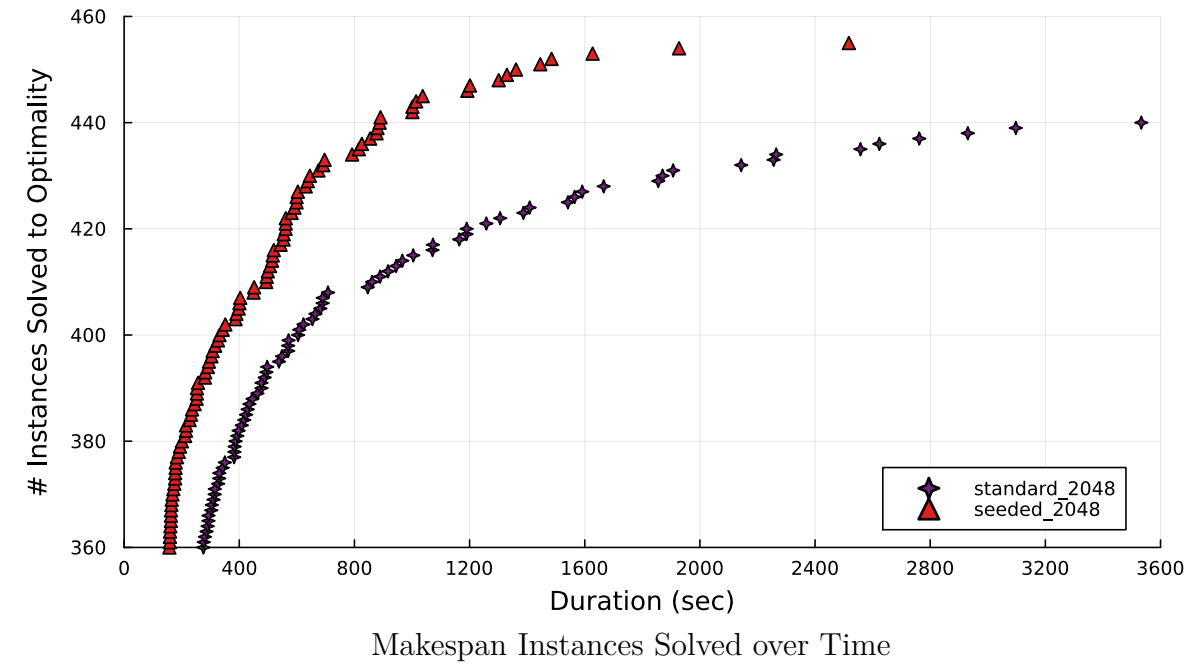


Figure 4.3 Performance of Improved Peel-and-Bound on Makespan

Table 4.2 Makespan Results for Problems Not Closed by the Unseeded Run

LB = Lower Bound, UB = Upper Bound, OG = Optimality Gap. The bolded instances were closed by the seeded run.

Set	Problem Information		Single Thread: 2048			Seeded Single Thread: 2048		
	Name	Best Known Solution	LB	UB	OG	LB	UB	OG
AFG	rbg050b.tw	11957	11748	11957	1.75	11748	11957	1.75
	rbg172a.tw	17783	17766	17784	0.10	17766	17783	0.10
Dumas	<b>n150w60.002.txt</b>	<b>940</b>	<b>940</b>	<b>941</b>	<b>0.11</b>	<b>940</b>	<b>940</b>	-
	<b>n200w40.002.txt</b>	<b>1137</b>	<b>1137</b>	-	<b>100</b>	<b>1137</b>	<b>1137</b>	-
GendreauDumas	n100w100.003.txt	819	818	819	0.12	818	819	0.12
	<b>n100w80.003.txt</b>	<b>829</b>	<b>829</b>	<b>839</b>	<b>1.19</b>	<b>829</b>	<b>829</b>	-
	<b>n60w200.003.txt</b>	<b>560</b>	<b>560</b>	<b>561</b>	<b>0.18</b>	<b>560</b>	<b>560</b>	-
	n80w140.005.txt	739	725	739	1.89	725	739	1.89
	<b>n80w160.002.txt</b>	<b>654</b>	<b>652</b>	<b>665</b>	<b>1.95</b>	<b>654</b>	<b>654</b>	-
	n80w180.002.txt	633	631	639	1.25	631	633	0.32
	n80w180.005.txt	632	444	632	29.75	444	632	29.75
OhlmannThomas	n80w200.004.txt	667	660	671	1.64	660	667	1.05
	<b>n150w120.004.txt</b>	<b>925</b>	<b>925</b>	<b>929</b>	<b>0.43</b>	<b>925</b>	<b>925</b>	-
	<b>n150w140.002.txt</b>	<b>1020</b>	<b>1020</b>	<b>1021</b>	<b>0.10</b>	<b>1020</b>	<b>1020</b>	-
	<b>n150w140.004.txt</b>	<b>898</b>	<b>898</b>	<b>919</b>	<b>2.29</b>	<b>898</b>	<b>898</b>	-
	<b>n150w140.005.txt</b>	<b>926</b>	<b>826</b>	<b>926</b>	<b>10.8</b>	<b>926</b>	<b>926</b>	-
	<b>n150w160.002.txt</b>	<b>890</b>	<b>861</b>	<b>912</b>	<b>5.59</b>	<b>890</b>	<b>890</b>	-
	<b>n150w160.004.txt</b>	<b>912</b>	<b>912</b>	<b>943</b>	<b>3.29</b>	<b>912</b>	<b>912</b>	-
	n200w120.001.txt	1089	1086	1089	0.28	1086	1089	0.28
	n200w120.002.txt	1072	1065	1072	0.65	1065	1072	0.65
	<b>n200w140.001.txt</b>	<b>1138</b>	<b>929</b>	<b>1146</b>	<b>18.94</b>	<b>1138</b>	<b>1138</b>	-
	n200w140.003.txt	1083	979	1083	9.60	1082	1083	0.09
	<b>n200w140.005.txt</b>	<b>1121</b>	<b>961</b>	<b>1121</b>	<b>14.27</b>	<b>1121</b>	<b>1121</b>	-
SolomonPesant	<b>rc204.2</b>	<b>870.52</b>	<b>728.94</b>	<b>914.89</b>	<b>20.33</b>	<b>870.52</b>	<b>870.52</b>	-
SolomonPotvinBengio	rc_204.1.txt	917.83	915.22	918.01	0.30	915.22	917.83	0.28
	rc_208.3.txt	686.80	606.15	686.80	11.74	610.58	686.80	11.10

## 4.5 Summary

This work improved on the previous implementation of peel-and-bound, but it also improved the theory behind the algorithm. We introduced a method of generating the initial DD where nodes are labeled instead of arcs. This not only allows for efficient handling of problems with non-separable objective functions, it also provides a method for implementing DDs without needing to store or update arcs. We will expand on this in Chapter 5. We also introduced embedded restricted DDs, a method of reading restricted DDs from existing relaxed DDs instead of generating them separately.

This chapter also presented PnB as an algorithm capable of achieving cutting edge results. We re-implemented the algorithm to be both more efficient and generic. We tested the new implementation on the 467 benchmark instances of TSP-TW used to test the DD-based BnB solver (ddo) [4]. The results show that PnB outperforms ddo on TSP-TW, even when ddo is using parallel processing. Furthermore, PnB closed 15 instances that, to the best of our knowledge, are open in the literature. In our final test, we ran the new implementation of PnB on the makespan variant of the 467 TSP-TW instances. PnB closed 94% of the makespan instances, and an additional 3% when seeded with the best known solution. We provide best known bounds for all TSP-TW and makespan instances that we believe to be open.

The next chapter generalizes the idea for creating a relaxed DD with exact arcs to any discrete optimization problem. It also discusses how this impacts our implementation of a PnB solver, and what the next steps are in a path to making the implementation capable of taking in a generic model without requiring specific knowledge of the solver.

## CHAPTER 5    IMPLICIT RELAXED DECISION DIAGRAMS

This chapter explores how one of the techniques presented in this thesis offer insights applicable to other implementations of DDs, not just PnB. It aims to equip researchers pursuing other DD-based solving methods with an essential insight from my experience that can be readily integrated into their own implementations.

### 5.1    Algorithm

In Section 4.2.1, we discussed a method of handling non-separable objective functions for sequencing problems by constructing an initial relaxed DD with a width equal to the number of variables. However, the method described works for any discrete optimization problem, is remarkably efficient, and has a straightforward implementation. This section will formalize the generalized procedure, then discuss its theoretical and practical benefits.

Let  $\mathcal{P}$  be a discrete optimization problem with variables  $X = \{x_1, \dots, x_n\}$ , and let  $d(x_i)$  be the domain of variable  $x_i$ . Create a root node  $r$ , and then for each value  $l$  in the domain of  $x_1$ , add a node  $u$  with label  $l$ . Then, repeat this in a new layer for each subsequent variable, adding a terminal node  $t$  at the end. Let a pointer be a reference to the memory location of another object. Let the phrase *add an implicit arc from  $u$  to  $v$*  mean: add a pointer at  $u$  to  $v$  indicating  $v$  is a child of  $u$ , and add a pointer at  $v$  to  $u$  indicating  $u$  is a parent. Then, add an implicit arc from every node on a layer  $i$  to every node on layer  $i + 1$ . Note that  $t$  is a dummy node, and all of the implicit arcs to  $t$  will have a weight of 0, because there is no associated decision variable. This construction is a valid relaxed DD for any  $\mathcal{P}$ . Once it is constructed, the arcs can be filtered using problem specific constraint propagation (See Section 2.5 for examples). Algorithm 14 formalizes this procedure.

The result of using this data structure is a DD where the arc labels are stored on the nodes instead of the arcs. In optimization problems with separable objective functions, the weight of an arc is usually dependent on the state information of the parent node and the label of the arc. Now the weight of an arc is dependent on the state information of the parent node and the label of the child node. In optimization problems with non-separable objective functions, the weight of an arc is now dependent on the state information of the parent node, the label of the parent node, and the label of the child node. In either case, the weight of an arc can be retrieved using only information stored at its origin and destination. The arc itself can be stored as just a pointer, it just needs to be known that it exists. Hence, these decision

diagrams are implicit, because their state information is reconstructed from information on the nodes.

---

**Algorithm 14:** Generating Implicit Relaxed Decision Diagrams

---

```

1 Input: An optimization problem  $\mathcal{P}$  with variables  $X = \{x_1, \dots, x_n\}$ 
2 Notation Reference:
3  $d(x_i)$  is the domain of variable  $x_i$ 
4 Let a pointer be a reference to the memory location of another object
5 Let  $ptr(u)$  be a pointer to node  $u$ 
6 Redefine  $in(u)$  to be a set of pointers containing the implicit parents of  $u$ 
7 Redefine  $out(u)$  to be a set of pointers containing the implicit children of  $u$ 
8 Create a root node  $r$ 
9  $out(r) \leftarrow \emptyset$ 
10  $\mathcal{M}^+ \leftarrow \{r\}$  // Initialize the DD to be just  $r$ 
11 foreach  $x_i \in X$  from 1 to n do
12    $\alpha \leftarrow$  the set of nodes on layer  $i - 1$ 
13   foreach  $l \in d(x_i)$  do
14     Create a node  $u$  with layer index  $i$  and label  $l$ 
15      $\mathcal{M}^+ \leftarrow \mathcal{M}^+ \cup \{u\}$ 
16      $in(u) \leftarrow \emptyset$ 
17      $out(u) \leftarrow \emptyset$ 
18     foreach  $v \in \alpha$  do
19        $in(u) \leftarrow ptr(v)$ 
20        $out(v) \leftarrow ptr(u)$ 
21     end
22   end
23 end
24 Create a terminal node  $t$ 
25  $\mathcal{M}^+ \leftarrow \mathcal{M}^+ \cup \{t\}$ 
26  $in(t) \leftarrow \emptyset$ 
27 foreach node  $u$  on layer  $n$  do
28    $in(t) \leftarrow ptr(u)$ 
29    $out(u) \leftarrow ptr(t)$ 
30 end
31 foreach implicit arc  $a \in \mathcal{M}^+$  do
32    $filter(a)$  // problem-specific constraint propagation
33 end
34 return  $\mathcal{M}^+$ 

```

---

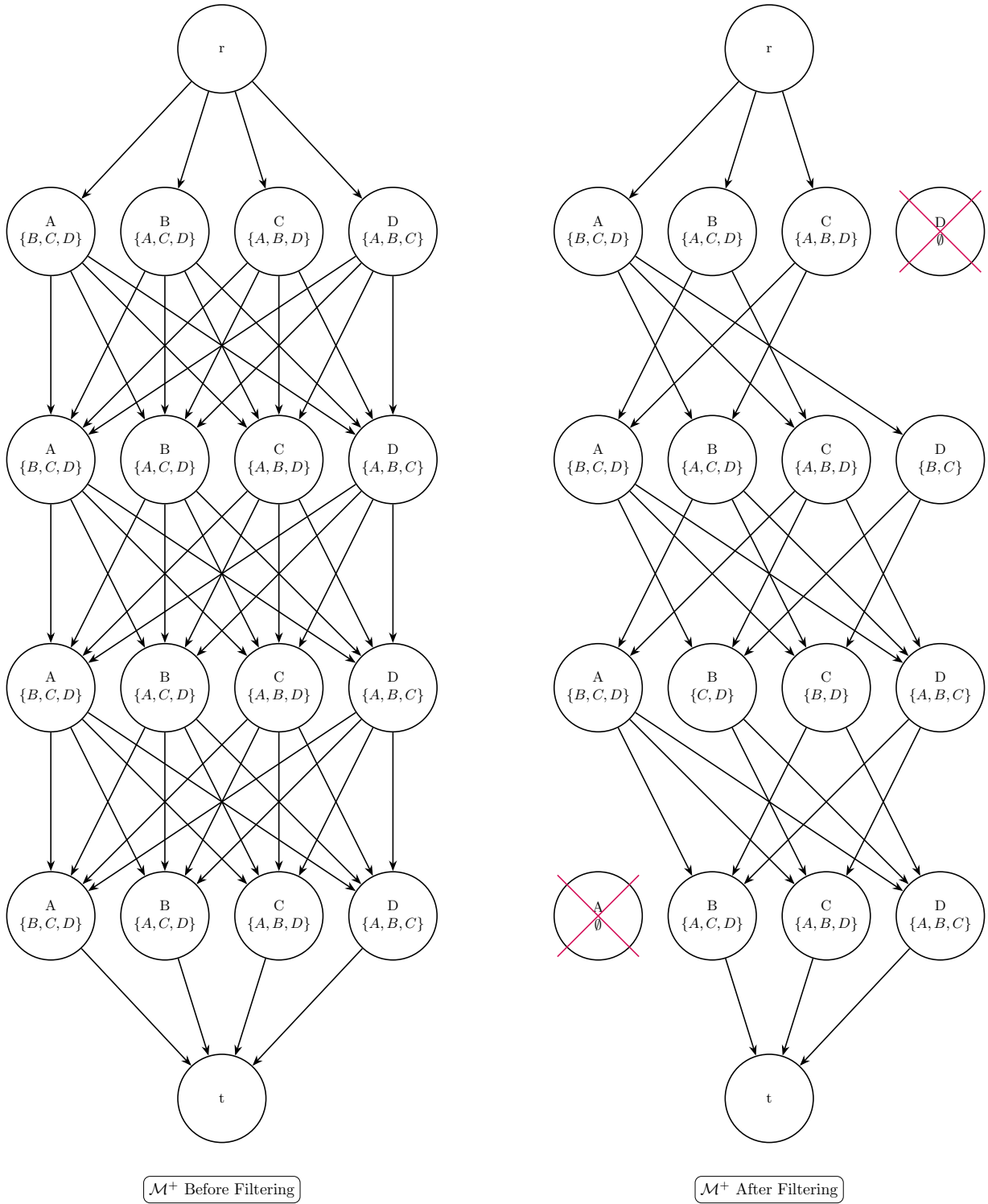


Figure 5.1 Implicit Relaxed DD, Before and After Filtering

SOP example with elements  $[A, B, C, D]$ , and the constraint that  $A$  must precede  $D$ .

Implicit arcs are drawn, but not labeled or weighted.

In the SOP example from Section 3.3, the goal is to order the elements  $[A, B, C, D]$ , subject to the precedence constraint that  $A$  must precede  $D$ . For the sake of simplicity, we ignore arc costs. Figure 3.1 showed a  $\omega = 3$  construction of  $\mathcal{M}^+(r)$  for this example. Figure 5.1 shows the implicit initial relaxed DD for this example, both before and after filtering the arcs.

### 5.1.1 Complexity Analysis

Assume you are generating a relaxed DD by separation starting from a  $\omega = 1$  diagram. Let  $\omega^*$  be the maximum allotted width. Every time a node  $u$  is split, its out-arcs are copied onto the new node  $u'$ . Recall that the layer index of  $u$  is  $\ell(u)$ . The maximum number of possible out-arcs to be copied during a split of  $u$  is  $|d(x_{\ell(u)})|$ . Let  $|d(X)|$  be  $\sum_{i=1}^n |d(x_i)|$ . The maximum number of splits is  $n(\omega^* - 1)$ . So the maximum number of arcs copied is at most  $n(\omega^* - 1) \cdot \frac{|d(X)|}{|X|}$ .

Now, assume you are generating a relaxed DD by separation starting from the initial implicit DD. The number of arcs copied during each split is the same, but the number of splits is slightly smaller at  $n(\omega^* - \frac{|d(X)|}{|X|})$ . However, this difference is clearly negligible. The maximum number of arcs copied in either case is at most  $\mathcal{O}(n \cdot \omega^* \cdot \frac{|d(X)|}{|X|})$ . In the case of problems like the SOP, where the average domain size is  $n - 1$ , this simplifies to  $\mathcal{O}(n^2 \cdot \omega^*)$ .

In general, reading and writing to memory are among the most expensive operations when implementing anything with a standard CPU. Whether or not an implicit DD is being used, pointers connecting the parents and children are required. However, when using an implicit DD, arc labels do not need to be read or written, saving  $\mathcal{O}(n \cdot \omega^* \cdot \frac{|d(X)|}{|X|})$  reads and writes.

## 5.2 Ease of Implementation

A hidden benefit of structuring relaxed DDs this way is that they are much easier to implement. When a node  $u$  is split into  $u$  and a new node  $u'$ , the state information of both  $u$  and  $u'$  needs to be updated. If the arcs are explicit and weighted, this can require iterating over the in-arcs of both nodes and updating their weights. If the arcs are implicit, all of the information is available on the nodes. You do not need to write code to handle updating arc information, you do not need to write code defining a data structure for your arcs, and you do not need to consider the trade-offs and implications of where you store your arcs in memory. Implicit relaxed DDs are not just more efficient, they require less work to implement and maintain in a solver.

### 5.3 The Future of Peel-and-Bound

At the time I am writing this thesis, the PnB solver is not capable of taking inputs that are restricted to a transition function and a split function. Using the solver requires writing several custom functions. Almost all of those functions fall into the category of being tedious to automate, but conceptually straightforward. The challenges involved with those functions are software engineering challenges, not mathematical challenges. However, a challenge I had viewed as a more significant hurdle was automating the generation of the initial relaxed DD. Now with Algorithm 14 worked out, that challenge has been solved. The next time I have a project that the PnB solver is a good tool for, I intend to generalize the initial relaxed DD generation using Algorithm 14. This will take the solver one enormous step closer to being as easy to use as any other general purpose DD-based solver.



## CHAPTER 6 APPLICATION TO FINDING EXACT SOLUTIONS TO THE SPACE-TIME DEPENDENT TSP

### 6.1 Motivation

In various real-world scenarios, we must determine the optimal sequence for visiting a number of locations, or scheduling jobs, where the travel cost between two locations is defined by an expensive black-box function without a closed form. These scenarios include a class of bi-level optimization problems. The *outer* problem involves finding an optimal permutation, while evaluating the cost of a given, possibly partial, permutation requires solving a different *inner* optimization problem that depends on the permutation. When the inner problem is computationally expensive and/or black-box, finding exact solutions to the bi-level optimization problem is difficult in practice.

Examples of this problem are time-dependent routing problems [58–60], especially variants where the travel costs must be computed on-demand [61] or are given by solving an inner optimization problem that, for example, determines vehicle speed [62]. This problem class also arises when planning missions in Space [63], such as finding a tour of the Jupiter Galilean moons [64], global optimization for multiple gravity assist trajectories [65–68], active space debris removal [58] and spacecraft formation control [69]. These problems are often solved by means of bi-level heuristics that combine numerical optimization methods with tree search methods such as Beam-Search [70], Lazy Race Tree Search [64], Monte-Carlo Tree Search [71] and Beam-ACO [72]. Other approaches treat the inner problem as a black-box function whose evaluation is expensive [73–76]. To the best of our knowledge, no exact method has ever been proposed to find optimal solutions for such problems. In the context of Space exploration, solutions to global trajectory optimization problems represent huge economic costs, hundreds of millions of dollars, and long mission horizons, from years to decades. Thus, optimality is highly desirable.

We propose a framework for finding exact solutions by representing the outer problem using decision diagrams (DDs) [1, 28] and then solving that model using PnB (Algorithm 13) with DD-based search techniques [5]. Our proposed approach treats the inner problem as an expensive black-box function evaluated on-demand, but it uses problem-specific knowledge about this black-box function to speed-up the search, and reduce the number of black-box evaluations needed to solve the problem.

As a case study, we consider the Asteroid Routing Problem (ARP) [77], which is a benchmark

problem inspired by the 11<sup>th</sup> Global Trajectory Optimization Competition (<https://gtoc11.nudt.edu.cn>). The ARP is a space-time dependent variant of the TSP. In the ARP, a spacecraft launched from Earth is tasked with visiting a specific set of asteroids. The goal is to find the permutation of asteroids that minimizes both fuel consumption and total travel time, which are aggregated into a single objective function. Travelling from one asteroid to another requires identifying optimal departure and travel times, which constitutes the inner problem. The ARP treats this inner problem as a black-box function that is solved by a pre-defined deterministic optimizer. Thus, in the ARP, the evaluation of a given permutation of the asteroids always yields the same objective function value. These characteristics make the ARP a simplified benchmark for the class of problems described above, and allow researchers to focus on the *outer* problem of optimizing the permutation of asteroids. As in the problems mentioned above, a brute force search is the only known method for identifying an optimal solution to the ARP, which is only computationally tractable for a very small number of asteroids. All approaches proposed so far are heuristic and time-consuming [76, 77].

In this chapter, we propose the first exact approach for solving the ARP. Our framework proves optimality of the outer problem under the assumption that the inner problem is optimized with sufficient quality. We address the quality of the inner optimizer in our experimental results (Section 6.7). We also discuss how this technique can be made scalable, and how it can be used to quickly generate strong heuristic solutions to the ARP. We provide exact solutions, and new best known solutions for several ARP instances.

## 6.2 Background

In this section we detail the ARP and how DDs can be applied to the ARP.

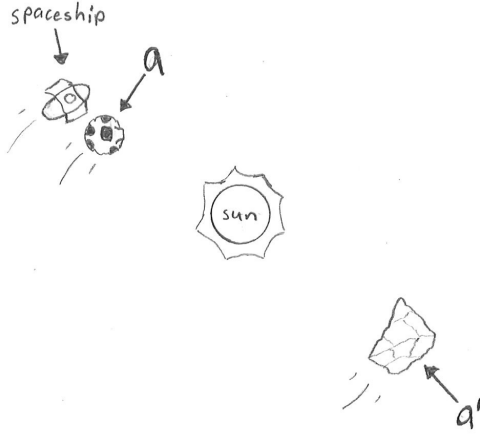
### 6.2.1 The Asteroid Routing Problem

In the ARP [77], we are given a set of  $n$  asteroids  $A = \{a_1, \dots, a_n\}$  orbiting around the Sun that a spacecraft launched from Earth ( $a_0$ ) must visit. The spacecraft does not need to return to Earth. For simplicity, the ARP treats the transfer of the spacecraft from Earth to the first-visited asteroid as any other transfer between asteroids, i.e., there is no escape velocity of Earth.

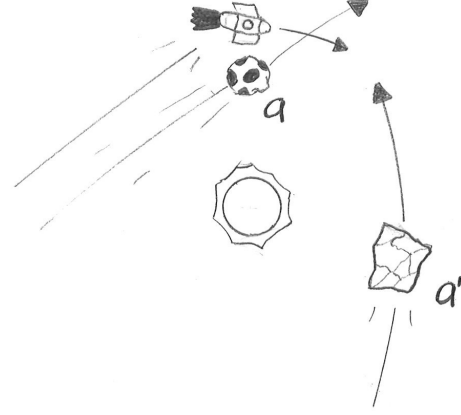
A transfer of the spacecraft between two asteroids  $a$  and  $a'$  that starts no earlier than time (epoch)  $\eta$  is computed with the following black-box function:

$$\mathcal{B}(a, a', \eta): A \times A \times \mathbb{R}^+ \rightarrow [0, \tau_{\max}] \times [1, t_{\max}] \times \mathbb{R}^+ \quad (6.1)$$

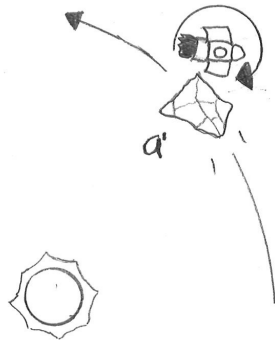
which returns three values  $(\tau, t, z)$ , where  $\tau$  is how long the spacecraft waits at  $a$  before departing,  $t$  is how long the spacecraft travels until it arrives at  $a'$ , such that arrival time is  $\eta + \tau + t$ , and  $z$  is the cost of this transfer. A transfer is depicted in Figure 6.1.



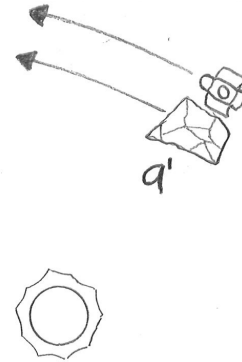
(a) At time  $\eta$ , the spaceship is in  $a$ 's orbit.



(b) After  $\tau$  days, at time  $\eta + \tau$ , the spaceship initiates a transfer to  $a'$ .



(c) After  $t$  days, at time  $\eta + \tau + t$ , the spaceship intersects  $a'$  and performs a maneuver to align itself with  $a'$ 's orbit.



(d) Still at time  $\eta + \tau + t$ , the spaceship is now in the same orbit as  $a'$ .

Figure 6.1 Depiction of a transfer from  $a$  to  $a'$

As shown above, the wait time  $\tau$  and travel time  $t$  are bounded by  $[0, \tau_{\max}]$  and  $[1, t_{\max}]$ , respectively. The wait time  $\tau$  has a lower bound of 0 because it is possible to depart immediately without waiting. The travel time  $t$  has a lower bound of 1 because a transfer will always require a nonzero time (at least 1 day in the ARP). The upper bounds  $\tau_{\max}$  and  $t_{\max}$

are set to 730 days in the original definition of the ARP [77].

Given a solution  $\pi = (\pi_0 = a_0, \pi_1, \dots, \pi_n)$ , where the subsequence  $(\pi_1, \dots, \pi_n)$  is a permutation of the asteroids in  $A$  that indicates the order in which they will be visited, the objective function of the ARP is:

$$\text{Minimize } f(\pi) = \sum_{i=1}^n z_i \quad \text{where } (\tau_i, t_i, z_i) = \mathcal{B}(\pi_{i-1}, \pi_i, \eta_i) \quad (6.2)$$

Besides,  $\eta_i = \eta_{i-1} + \tau_{i-1} + t_{i-1}$  is the arrival time at  $\pi_{i-1}$  and  $\eta_1$  is a given mission start time, which can be simplified to be zero. A solution is visualized in Figure 6.2.

The black-box function  $\mathcal{B}$  depends not only on the two asteroids being visited, but also on the arrival time  $\eta_i$ . Thus the ARP is an example of the class of time-dependent problems discussed in the introduction.

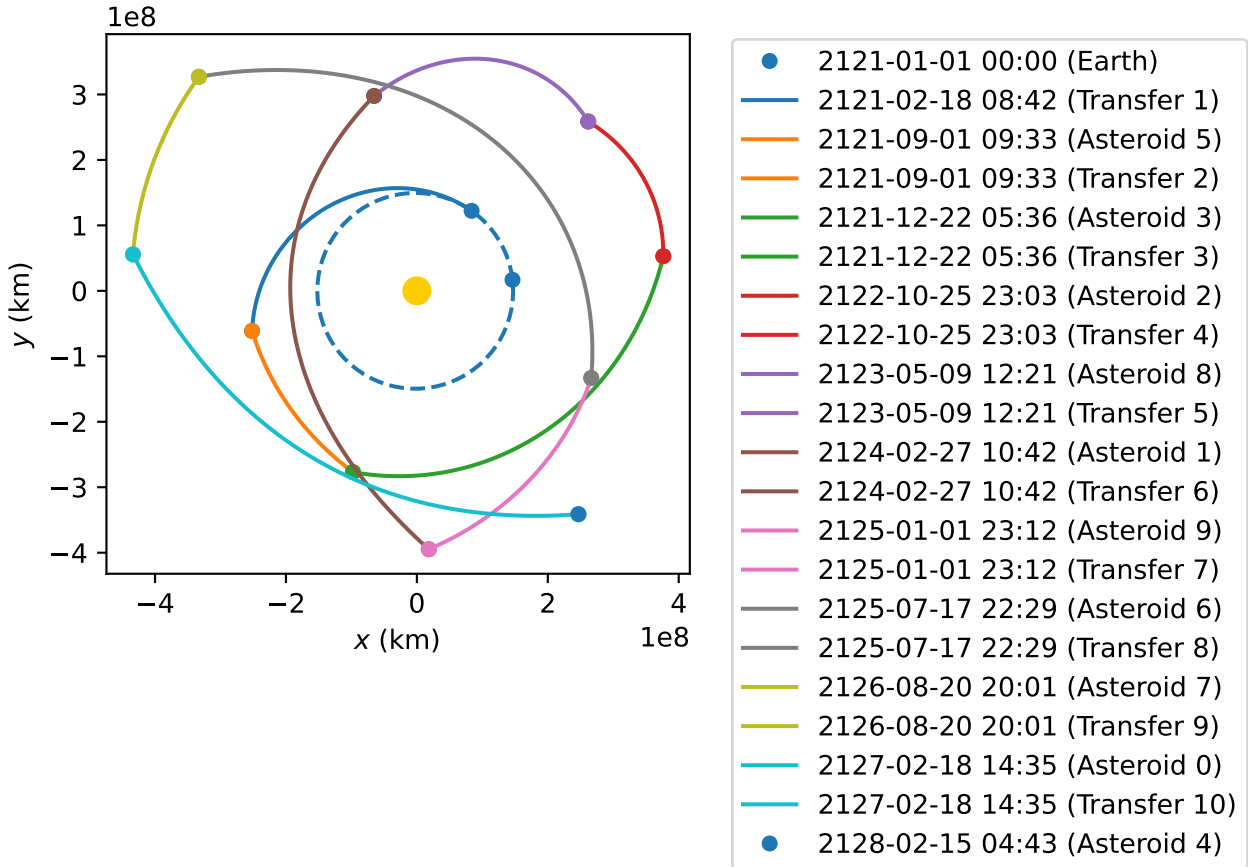


Figure 6.2 Visualization of best known solution to instance with  $n = 10$  and  $seed = 8$ .

### 6.2.2 Trajectory Optimization

In the ARP, and in many trajectory optimization problems, the spacecraft only uses *impulsive* maneuvers. An impulsive maneuver changes the spacecraft velocity in a negligible amount of time, thereby simplifying the calculation of the maneuver's effect on the spacecraft's orbit. As a result, the effect of the maneuver can be modeled as an instant change in the spacecraft's velocity vector  $\Delta\vec{v}$ , without accounting for the dynamics of the propulsion system, or the influence of external forces during the thrust application.

Lambert's Problem [78] calculates an orbit that connects two points in space-time departing at time  $\eta + \tau$  and arriving at time  $\eta + \tau + t$ . Using the solution to Lambert's Problem, a transfer between the orbits of two asteroids  $a$  and  $a'$  can be achieved with just two impulsive maneuvers. Initially, at time (epoch)  $\eta$ , the spacecraft follows the same orbit around the Sun as asteroid  $a$ . *Waiting* in this orbit does not consume any fuel, but does change its relative distance to other asteroids. The first impulse  $\Delta\vec{v}_1$  happens  $\tau$  days after the current time  $\eta$  (epoch), and moves the spacecraft from its current orbit to an orbit that will intercept the target asteroid  $a'$  after traveling for  $t$  days. When the spacecraft intercepts  $a'$ , the second impulse  $\Delta\vec{v}_2$  modifies its orbit to match the orbit of  $a'$ , so that the spacecraft follows the same orbit as  $a'$  without consuming any additional fuel. Usually the total magnitude of velocity change (the Euclidean L2 norm) is used as a surrogate for fuel consumption/energy costs:

$$\Delta V = \|\Delta\vec{v}_1\|_2 + \|\Delta\vec{v}_2\|_2 \quad \text{where} \quad (\Delta\vec{v}_1, \Delta\vec{v}_2) = \text{Lambert}(a, a', \eta + \tau, t) \quad (6.3)$$

We still need to decide  $\tau$ , how long should the spacecraft wait at  $a$ , and  $t$ , how long the transfer should take until arriving at  $a'$ . Waiting changes the relative distance to other asteroids, which may reduce the fuel or total time needed to reach the next asteroid. The optimal values of  $\tau$  and  $t$  depend on which objectives are being optimized.

In trajectory optimization there are many possible objectives. Two typical objectives are the minimization of fuel (energy) consumption and of the total mission time ( $\tau + t$ ). These two objectives are often in conflict. The ARP aggregates the two objectives as shown below:

$$\begin{aligned} z = f_{\text{inner}}(a, a', \eta, \tau, t) &= \Delta V + \frac{2 \text{ km/s}}{30 \text{ days}} \cdot (\tau + t) \\ \text{s.t.} \quad &\tau \in [0, \tau_{\text{max}}], \quad t \in [1, \tau_{\text{max}}] \\ \text{where} \quad &\Delta V = \|\Delta\vec{v}_1\|_2 + \|\Delta\vec{v}_2\|_2 \quad \text{and} \quad (\Delta\vec{v}_1, \Delta\vec{v}_2) = \text{Lambert}(a, a', \eta + \tau, t) \end{aligned} \quad (6.4)$$

where the trade-off constant  $\frac{2\text{km/s}}{30\text{days}}$  was chosen by [77].

The above becomes an *inner* optimization problem whose solution gives the values  $\tau$ ,  $t$  and  $z$  returned by  $\mathcal{B}$  (Eq. 6.2).

In the ARP, this inner problem is solved by using Sequential Least Squares Programming (SLSQP) [79], which is a deterministic optimizer. In this way, each permutation of the asteroids corresponds to a unique solution. SLSQP iteratively calculates the position and velocity of the asteroids, and then solves Lambert’s Problem in order to evaluate the above objective. Even if time is discretized into days, this inner optimization would be too slow to allow us to calculate the cost of every optimal trajectory for every pair of asteroids at every point in time.

A mission with  $n$  asteroids may have up to  $n(\tau_{\max} + t_{\max})$  possible values of  $\eta$ . In a mission where  $n = 10$ , calculating every possible trajectory from just one asteroid to one other asteroid for every possible departure time when time is discretized into days would take about 10 minutes using the inner optimizer in our implementation. Thus, calculating the full cost matrix for every ordered pair of asteroids ( $n(n - 1)$  pairs) at every possible departure day would take about 15 hours. Furthermore, departure time is continuous, not discrete, so this cost matrix would not even fully represent the problem, although it would certainly provide information that is useful for solving it.

### 6.2.3 Relaxed Decision Diagrams for the ARP

To initialize a relaxed DD for the ARP we are going to denote the root as *Earth* and the terminal as *done*. We use the implicit DD construction explained in Chapter 5. This means that in our implementation, the arcs are stored as pointers. However, in our figures here we include the implied arc labels for the sake of clarity. Note that we temporarily assume we have valid arc weights; we explain how to compute these in Section 6.3.3.

Figure 6.3a displays a valid relaxed DD for an ARP with asteroids  $\{A, B, C\}$ . Each arc is labeled with the decision about which asteroid is the next destination, and each node is labeled with the current asteroid. Every feasible permutation of asteroids is trivially encoded as a path from the root to the terminal, but so are several infeasible solutions, such as  $A \rightarrow B \rightarrow A$ . Figure 6.3b displays a weighted version of Figure 6.3a. Each arc still assigns the same decision, but the arc weights now display the lower bounds on the cost of transferring between those two asteroids at that point in the sequence. Each node is now also labeled with the length of the shortest path to that node. Thus, the length of the shortest path ( $A \rightarrow B \rightarrow C$ , which is highlighted), is a bound on this ARP. In other words, no matter

what permutation of asteroids you use, and no matter how well you optimize the trajectories between them, there is no route for the spacecraft that admits a cost less than 10 for this example.

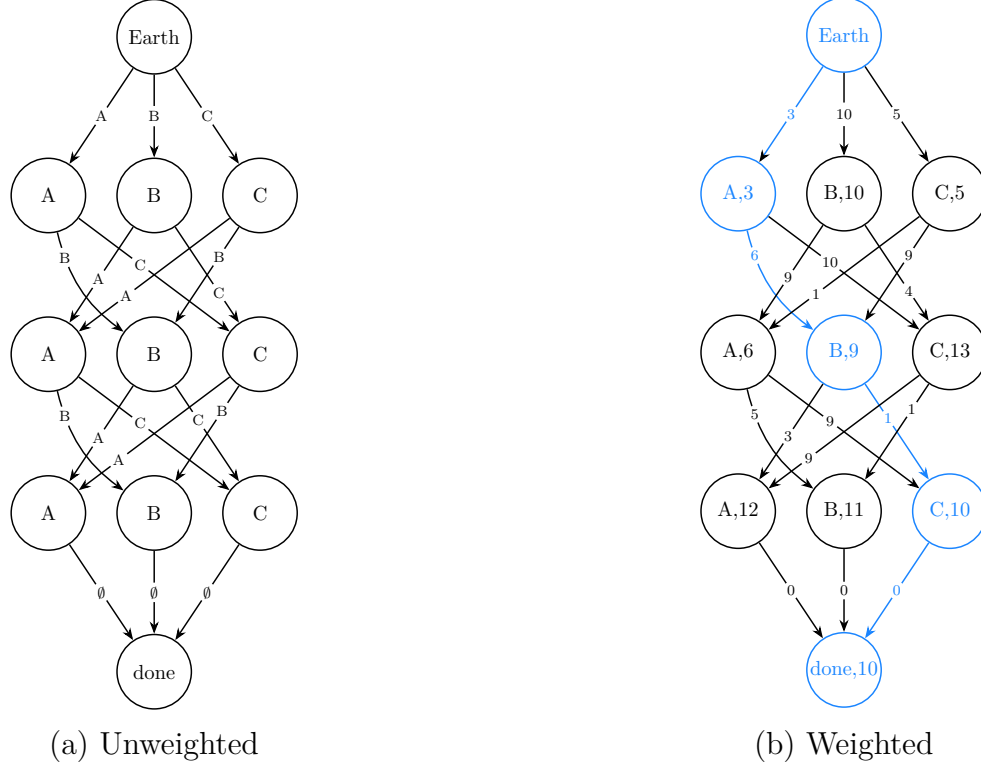


Figure 6.3 Relaxed decision diagrams for an ARP with asteroids  $\{A, B, C\}$ .

In the well-known Travelling Salesman Problem (TSP), the weights on the arcs would be the transition costs of going from one city to the next. However, in the TSP the arc weights are not dependent on all of the preceding cities, and the transition costs are easy to calculate. In this thesis, we are handling a problem where calculating arc weights requires calling a black-box function  $\mathcal{B}$  with non-negligible computation time. Furthermore, the exact arc weight for arc  $a$  is dependent on the path taken to reach  $a$ . In principle, this dependency requires the evaluation of all possible paths from  $r$  to  $a$ , in order to find the optimal one. However, if we can find, via relaxation of the  $\mathcal{B}$  function, a globally valid bound on the arc weights, that bound can be used to prune sub-optimal paths.

In the case of the ARP, this relaxation is given by a no-wait variant of  $\mathcal{B}$  where the cost of waiting is removed from the objective function; this is defined in Section 6.3.2. This no-wait variant can be used to find a globally valid bound on the cost of any transition between two

asteroids. Thus for the ARP, the arcs will be weighted with valid lower bounds on their transition costs instead of exact values. This idea is explored and formalized in Section 6.3. To refine the diagram we will be using the methods for solving sequencing problems with DDs discussed in Section 2.5 [28] together with PnB (Algorithm 11).

### 6.3 The Initial Decision Diagram

This section details a technique for constructing a relaxed DD for the ARP that yields a valid relaxed bound.

#### 6.3.1 Initial Setup

We begin by constructing a relaxed DD  $\mathcal{M}$  exactly as shown in Figure 6.3a. We start with a root node at layer 0, a terminal node at layer  $n + 1$ , and  $n$  nodes on each layer with index  $i$ , where  $1 \leq i \leq n$ . Each node  $u$  in a layer  $i$  is given a label  $l$  unique to that layer. Remember that node labels are conceptually the same as arc labels. Each node and each arc has exactly one label, always representing either *Earth* or an asteroid.

An arc represents the transition from one node to another, and so in the ARP, an arc represents a transfer of the spacecraft. The arc label is the asteroid being transferred to. A node represents a state, and in the ARP, part of that state is the location of the spacecraft (either *Earth* or an asteroid). Thus, the node label is either *Earth* for the root, or an asteroid for the other non-terminal nodes. The terminal node is a dummy node. Then, each node  $v$  in layer  $i - 1$  is the origin for an arc ending at  $u$ , as long as  $v$  has a different label. An arc with a null label is added from each node in layer  $n$  to the terminal. The result is a relaxed DD that encodes every feasible sequence of asteroids as a path from the root to the terminal.

#### 6.3.2 Relaxing the Black Box

In real-world problems that depend on a black-box function or an inner problem, it is often possible to define relaxed versions of such functions that provide useful bounds or heuristic values [61, 62].

In the ARP, we can define a relaxed version of Eq. 6.4 that removes the cost of waiting as



follows:

$$\begin{aligned}
f'_{\text{inner}}(a, a', \eta, \tau, t) &= \Delta V + \frac{2 \text{ km/s}}{30 \text{ days}} \cdot t \\
\text{s.t. } \tau &\in [0, \tau_{\text{f}}], \quad t \in [1, \tau_{\text{max}}] \\
\text{where } \Delta V &= \|\Delta \vec{v}_1\|_2 + \|\Delta \vec{v}_2\|_2 \quad \text{and} \quad (\Delta \vec{v}_1, \Delta \vec{v}_2) = \text{Lambert}(a, a', \eta + \tau, t)
\end{aligned} \tag{6.5}$$

Since  $f'_{\text{inner}}$  removes the cost of waiting, we also relax the upper bound of  $\tau$  to a parameter  $\tau_{\text{f}}$ . We can now define the following relaxed variant of  $\mathcal{B}$ :

$$\mathcal{B}'(a, a', \eta, \tau_{\text{f}}): A \times A \times \mathbb{R}^+ \rightarrow [0, \tau_{\text{f}}] \times [1, t_{\text{max}}] \times \mathbb{R}^+ \tag{6.6}$$

which returns three values  $(\tau, t, z)$ , where  $\tau$  and  $t$  are the values that minimize  $f'_{\text{inner}}$  given  $a, a', \eta$  and  $\tau_{\text{f}}$ , and  $z$  is its minimal value. Note that this is only a relaxation in the sense that it removes part of the cost from the objective function, it does not change the method used to solve it. The procedure for solving  $\mathcal{B}'$  is exactly the same as the procedure for solving  $\mathcal{B}$ .

Note that we will only ever use  $\mathcal{B}'$  to compute bounds on costs, we will never use it to evaluate the cost of a permutation of asteroids. We always use  $\mathcal{B}$  to evaluate the cost of solutions and partial solutions, so that the values we report can be compared with the values reported by [77].

### 6.3.3 Calculating Valid Arc Bounds

The examples in Section 6.2 assumed the availability of bounds on the arc weights. In this section, we will explain how they can be calculated. We also mentioned that calculating arc bounds is expensive because it requires calculating the black-box function  $\mathcal{B}$ . In the ARP, this means that the inner problem must be solved using SLSQP. The algorithm we propose is designed to minimize the number of times we need to calculate the value of  $\mathcal{B}$ .

When using constraint propagation, arcs are filtered and removed, but never added. When splits and peels are performed, the number of arcs in the DD may grow, but only when the out-arcs of a node are copied. When this occurs, the asteroid that an arc is traveling from, and the asteroid that the arc is traveling to, remain unchanged. Therefore, when splitting or peeling, a valid bound on the cost of an arc is also a valid bound on its copy. While it may be desirable to recalculate the bound on an arc after a split to get a tighter bound, peels and splits can be carried out as many times as memory limitations allow, without the need to re-evaluate  $\mathcal{B}$ . We propose a two-phase algorithm for creating an initial relaxed DD. This

initial construction is intentionally computationally expensive, because we will make enough evaluations of  $\mathcal{B}$ , that  $\mathcal{B}$  does not need to be evaluated again after the initial relaxed DD is constructed, except when evaluating feasible solutions.

### Phase One

Now we weight the DD. The arcs going from layer 0 to layer 1 are straightforward. A valid weight for an arc must be a valid lower bound on the cost of making the transfer represented by that arc. So a valid weight on arc  $a_{uv}$  going from node  $u$  to  $v$  with labels  $l(u) = \textit{Earth}$  and  $l(v)$  is given by the  $z$  value returned by  $\mathcal{B}(\textit{Earth}, l(v), \eta = 0)$ . There is no possibility for  $\eta$  other than 0, so that  $z$  value will be the smallest possible cost of going from Earth to  $l(v)$ .

In the ARP, these costs from layer 0 to layer 1 are exact, because the inner problem is only solved for one pair of asteroids at a time for simplicity [77]. In other words,  $\mathcal{B}$  does not depend on the path from node  $v$  to the terminal. Thus, we can assume that the value of  $\mathcal{B}$  for an arc only depends on the earliest start time at the starting node, which only depends on the path from the root node to that starting node. In the general trajectory optimization problem, it is possible to globally optimize the inner problem for a whole sequence of asteroids by optimizing all waiting and transfer times simultaneously. This simultaneous optimization may further reduce the cost of the sequence; for example, it may be that increasing the waiting time between early transfers leads to a reduction in the transfer cost of later transfers. In that case, the value of  $\mathcal{B}$  at a given arc may also depend on the path from the destination node to the terminal.

Our proposed method works for either type of problem because the sum of the arc weights on a path is still a globally valid bound on the cost of the sequence represented by that path. A dependence on the path from a node to the terminal simply makes the evaluation of feasible solutions more expensive because we need to re-optimize all waiting and transfer times simultaneously every time we weight an arc. In the ARP, if there is only one path from the root to a node  $u$ , and we know both the cost of the path from the root to  $u$  and the earliest start time at the previous node, then the exact cost of that path can be calculated with a single evaluation of  $\mathcal{B}$  that only optimizes the wait and transfer times of the arc that ends at  $u$ . In our implementation, we leverage this property by updating the weights of arcs on such a path with their exact cost, and track the earliest start time for each node. This is not necessary to find an optimal solution, but it is a useful heuristic.

The arcs going from layers  $i$  to  $i + 1$  for  $i \in \{1, n - 1\}$  remain to be weighted. We begin by finding a valid bound on the transfer between each ordered pair of asteroids  $a, a' \in A$ , with  $a \neq a'$ . We can use the no-wait  $\mathcal{B}'$  (Eq. 6.6) for that purpose by noticing that the maximum

time that any solution requires is trivially bounded by the sum of the upper bounds on the waiting times and travel times of the trajectories, i.e.,  $n(\tau_{\max} + t_{\max})$ . Therefore, the latest start time for the final transfer is bounded by  $n(\tau_{\max} + t_{\max}) - t_{\max}$ . That means that the transfer from an asteroid  $a$  to an asteroid  $a'$  can wait  $\tau \in [est(a), n(\tau_{\max} + t_{\max}) - t_{\max}]$ , where  $est(a)$  is the earliest start time at  $a$ . Initially,  $est(a) = \tau^* + t^*$ , where  $\tau^*$  and  $t^*$  are the values returned by  $\mathcal{B}(Earth, a, \eta = 0)$  as described above. We can now define:

$$z_{\min}(a, a') := z \quad \text{where} \quad (\tau, t, z) = \mathcal{B}'(a, a', \eta = est(a), \tau_f = n(\tau_{\max} + t_{\max}) - t_{\max}) \quad (6.7)$$

Because  $\mathcal{B}'$  does not penalize waiting time, the waiting time  $\tau$  returned by  $\mathcal{B}'$  above gives an optimal  $\eta$  for minimizing the actual cost of the transfer using  $\mathcal{B}$  (Eq. 6.1). In other words, the cost  $z_{\min}(a, a')$  is the smallest possible cost of the transfer from  $a$  to  $a'$ , and thus a lower bound of the  $z$  value returned by  $\mathcal{B}(a, a', \eta = est(a) + \tau)$ . Computing  $z_{\min}(a_i, a_j)$  for all pairs  $a_i, a_j$ , where  $i, j \in \{1, \dots, n\}$  and  $i \neq j$  requires  $n^2 - n$  calls to  $\mathcal{B}'$ .

We add the pre-calculated  $z_{\min}(a, a')$  values as arc weights. A valid weight for an arc  $a_{uv}$  going from node  $u$  with label  $l(u)$  to  $v$  with label  $l(v)$ , is simply  $z_{\min}(l(u), l(v))$ , which is the best possible transfer between  $l(u)$  and  $l(v)$  when ignoring waiting time at  $l(u)$ . The resulting relaxed DD contains valid weights such that the sum of the costs of the arcs on any path from the root to the terminal provide a lower bound on the true cost of the permutation represented by the labels of the nodes on that path. However, this lower bound is quite weak, so we perform a second phase to strengthen the arc weights.

## Phase Two

In this phase, we will use a heuristic solution  $z^-$  to strengthen the arc weights. There are many methods available in the literature for finding a strong heuristic solution in trajectory optimization problems such as the ARP [80, 81]. In the ARP, any valid permutation of the asteroids is a feasible solution. For simplicity, our implementation uses a Euclidean distance heuristic to find the initial  $z^-$ . It picks asteroids one by one, always moving to the closest unvisited asteroid as measured by Euclidean distance at the time of arrival to the last visited asteroid. This simple heuristic is exceptionally effective and our proposed algorithm has no trouble rapidly finding strong feasible solutions. In Section 6.4, we will describe the method we use for finding feasible solutions in more detail.

Before we can make use of  $z^-$ , we need to store some information on the nodes. We begin by recursively updating each node  $u$  with the length of the shortest path from the root to  $u$ , denoted by  $z_{\downarrow}(u)$ , as well as the length of the shortest path from  $u$  to the terminal, denoted

by  $z_{\uparrow}(u)$  [28]. Recall that the length of a path from the root to the terminal is a valid lower bound on the cost of the associated permutation. It follows that  $z_{\downarrow}(u) + z_{\uparrow}(u)$  is a valid lower bound on the cost of any path that passes through  $u$ , because the shortest path that passes through  $u$  is trivially the shortest path from the root to  $u$  merged with the shortest path from  $u$  to the terminal. Note that this implies that if  $z_{\downarrow}(u) + z_{\uparrow}(u) > z^-$ , then the optimal solution cannot pass through  $u$ , and we can remove  $u$  from the DD because the shortest path that passes through  $u$  has a higher cost than a known feasible solution.

Now consider an arc  $a_{uv}$  connecting nodes  $u$  to node  $v$  with weight  $w(a_{uv})$ . Similarly,  $z_{\downarrow}(u) + w(a_{uv}) + z_{\uparrow}(v)$  is a valid lower bound on the cost of any path that can pass through  $a_{uv}$ . This is because the shortest path passing through an arc is the shortest path to the origin of that arc, then the arc itself, then the shortest path from the destination of the arc to the terminal. As before, we know that if  $z_{\downarrow}(u) + w(a_{uv}) + z_{\uparrow}(v) > z^-$ , then the optimal solution cannot pass through  $a_{uv}$ , and thus  $a_{uv}$  can be removed from the graph. We will leverage this constraint to calculate new stronger arc weights.

Let us consider arc  $a_{uv}$ , and let the layer of  $u$  be  $1 \leq i \leq n-1$ . The latest possible start time  $\eta$  for arcs going from layer  $i$  to  $i+1$  is  $i \cdot (\tau_{\max} + t_{\max})$ . Thus, a naive approach to improving the quality of the weight of  $a_{uv}$  is  $\mathcal{B}'(l(u), l(v), \eta = est(u), \tau_f = i \cdot (\tau_{\max} + t_{\max}) + \tau_{\max})$ , which returns the optimal transfer  $(\tau, t, z)$  between two asteroids when waiting is free.

This  $z$  value is a valid weight for each such arc  $a_{uv}$ , and greatly improves the weights on arcs on higher indexed layers over the weights calculated in phase one. However, this will do very little to improve the weights of arcs at lower indexed layers. This method is valid, but as we will see next, it is extremely inefficient. Consequently, we will now use the concepts we covered in this section so far to motivate a method that scales better with the layer index.

Recall the constraint that for arc  $a_{uv}$ , it must be that  $z_{\downarrow}(u) + w(a_{uv}) + z_{\uparrow}(v) \leq z^-$ , or we can remove the arc. In the previous method, we used  $i \cdot (\tau_{\max} + t_{\max})$  as a bound on the free waiting time an arc can leverage. A better bound can be retrieved by recognizing that the true cost of  $w(a_{uv})$  includes the cost of wait time as a component. Let  $\tau_a$  be the wait time used by  $a_{uv}$ , and recall that the cost of time in the *inner* objective function (Eq. 6.4) is  $\frac{2}{30}$ . It must also be the case that  $z_{\downarrow}(u) + \frac{2}{30}\tau_a + z_{\uparrow}(v) \leq z^-$ . This is because if  $\tau_a$  is large enough to cause that constraint to be violated, then  $w(a_{uv})$  will also be large enough to cause its constraint to be violated, because  $w(a_{uv})$  is a sum of components that include  $\frac{2}{30}\tau_a$ . This yields the constraint that:

$$\tau_a \leq \frac{30}{2}(z^- - z_{\downarrow}(u) - z_{\uparrow}(v)) \quad (6.8)$$

where the right-hand-side of Eq. (6.8) is a constant because  $z_{\downarrow}(u)$  and  $z_{\uparrow}(v)$  are known. This

means that we can use this bound for  $\tau_a$  to define a stronger bound per layer:

$$z_{\min}(u, v, i) := z \text{ where } (\tau, t, z) = \mathcal{B}'(l(u), l(v), \eta = est(u), \tau_f) \text{ and } \tau_f = \frac{30}{2}(z^- - z_{\downarrow}(u) - z_{\uparrow}(v)) \quad (6.9)$$

Starting with  $i = 1$ , we calculate a new bound for every arc on a layer ( $z_{\min}(u, v, i)$ ), then we update the values of  $z_{\downarrow}(v)$  for each  $v$  (in other words, each node on layer  $i + 1$ ), then we set  $i \leftarrow i + 1$ , and repeat until  $i = n + 1$ . Therefore, we do not update the arcs with a weight of 0 that go to the terminal. After this process is done, the  $z_{\uparrow}$  values of the nodes need to be updated as they are dependent on the weights of the arcs.

Phase two requires performing a total of  $n^3 - 2n^2 + n$  evaluations of  $\mathcal{B}'$ , but results in strong enough initial bounds that, after this initial setup, our algorithm for the ARP only needs to call the black-box function  $\mathcal{B}$  to evaluate the true cost of partial feasible solutions; it does not need to evaluate  $\mathcal{B}$  or  $\mathcal{B}'$  for the purposes of bounding.

The weights are improved based on the  $z_{\downarrow}$  and  $z_{\uparrow}$  values, but the values of  $z_{\downarrow}$  and  $z_{\uparrow}$  are updated based on the new arc weight. It is possible to run phase two repeatedly until the values converge. However, each additional update requires a huge amount of computation, and in practice yields minuscule changes. So the decision of how many repetitions to perform while constructing the initial DD is heuristic. In our implementation, we found that this process almost never improves the bound enough to be useful, so we did not include it.

### Earliest Start and Arrival Times

So far we have assumed that the black-box function, and thus the inner optimization, only depends on the transfer locations and the arrival time at the origin location (Eq. 6.1). In practice, this is often not the case, and it is in fact possible to optimize all of the inner problem variables for a given permutation at once to find a better solution. More generally, the objective function  $f(\pi)$  may be completely black-box and not linearly separable as in Eq. 6.2. Even in such cases, it may still be possible to apply our proposed method.

In the case of the ARP, and many global trajectory optimization problems, given a permutation of the asteroids or celestial bodies to be visited, it is possible in principle to globally optimize all times  $\tau_i$  and  $t_i$  for all  $i = 1, \dots, n$  in Eq. 6.2, and further improve the objective function value. The original definition of the ARP [77] does not perform this global inner optimization because it does not guarantee finding a better objective function value, and it makes evaluating each permutation significantly more time-consuming. Nevertheless, in practical global trajectory optimization problems, performing such global inner optimization may be beneficial if enough computation time is available. Thus, we explain here how to

adapt our proposed method to handle such cases.

For the ARP as originally defined, calculating earliest start times is an unnecessary but useful heuristic. As explained in Section 6.3.3,  $\mathcal{B}(\text{Earth}, l(u), \eta = 0)$  returns the  $z$  value for an arc going from Earth to a node  $u$  in layer 1 as well as the exact  $\tau + t$  value that the transfer requires, which is the exact earliest start time at  $u$  for any valid sequence. However, if we optimize all of the inner problem decision variables for a valid sequence at the same time, the earliest start time may be different. We describe next a valid alternative.

First, we define a variant of  $\mathcal{B}$  that limits the maximum total time  $\theta$  for the transfer:

$$\begin{aligned} \tilde{\mathcal{B}}(a, a', \eta, \theta) &= (\tau^*, t^*, z^*) \\ \text{where } z^* &= f_{\text{inner}}(a, a', \tau^*, t^*) \quad \text{and} \quad (\tau^*, t^*) = \arg \min f_{\text{inner}}(a, a', \tau, t) \\ \text{s.t. } \quad &\tau + t \leq \theta, \tau \in [0, \tau_{\max}], t \in [1, t_{\max}] \end{aligned} \quad (6.10)$$

The above function  $\tilde{\mathcal{B}}$  also returns three values  $(\tau, t, z)$  with the same meaning as they have for  $\mathcal{B}$ , but the inner problem has the additional constraint that  $\tau + t \leq \theta$ .

Given a relaxed DD where the nodes store  $z_{\downarrow}(u)$  and  $z_{\uparrow}(u)$ , we can use the above function to find the earliest start time ( $est$ ) for any node  $u$  in layer 1. In other words, we can use  $\tilde{\mathcal{B}}$  to find the  $est$  for visiting an asteroid  $l(u)$  from Earth. This is accomplished by performing a binary search with a starting range of  $[0, \tau_{\max} + t_{\max}]$  to find the value  $\theta$  such that:

$$z^- - z_{\uparrow}(u) = \tilde{z} \quad \text{where} \quad (\tau, t, \tilde{z}) = \tilde{\mathcal{B}}(\text{Earth}, a, \eta = 0, \theta) \quad (6.11)$$

However, the  $z$  values returned by  $\mathcal{B}$ , and thus  $\tilde{\mathcal{B}}$ , are not necessarily smooth, so the equality may not be strictly satisfiable because the target value may not be in the range of  $\tilde{\mathcal{B}}$ . When this is the case, the goal of the binary search should be modified to find the value of  $\theta$  that produces the largest value of  $\tilde{z} \leq z^- - z_{\uparrow}(u)$ .

The earliest arrival time ( $eat$ ) to reach the end of an arc  $a_{uv}$  can be similarly found by performing a binary search to find the value of  $\theta$  such that:

$$\tilde{z} = z^- - z_{\downarrow}(u) - z_{\uparrow}(v) \quad \text{where} \quad (\tau, t, \tilde{z}) = \tilde{\mathcal{B}}(l(u), l(v), \eta = est(u), \theta) \quad (6.12)$$

The  $est$  of a node  $v$  is the minimum  $eat$  of arcs ending at  $v$ , which allows us to repeat the search for the nodes in the next layer. However, performing this search on a node  $v$  requires running a binary search on every single arc ending at  $v$ , which is significantly time-consuming. If using this method, we recommend only using it on nodes with one parent.

## 6.4 Heuristic Search with Embedded Restricted Decision Diagrams

A benefit of leveraging embedded restricted DDs, and a critical benefit for solving ARPs, is that each node  $u$  in a relaxed DD can also be labeled with the length of the shortest path from  $u$  to the terminal, which we have already denoted as  $z_{\uparrow}(u)$  in this thesis. This provides a simple but powerful heuristic for deciding which nodes to explore. This heuristic is directly inspired by, but substantially different from, the rough bounding proposed in [30].

Let  $\omega_s$  be the maximum width allotted to the search. Given a node to be processed  $u^-$  in a partially constructed restricted DD  $\mathcal{M}^-$ , the typical method for figuring out which of  $u^-$ 's children to explore would be to generate all of them and then repeatedly throw away the node  $v^-$  in the layer being constructed with the highest  $z_{\downarrow}(v)$  until the width of the layer is  $\omega_s$ . However, for the ARP, each creation of a child requires a transfer from the last asteroid visited to an unvisited one, which requires a call to  $\mathcal{B}$ , causing such a search to become rapidly intractable.

In order for the embedded restricted DDs to be generated quickly, it is critical to limit the number of calls to  $\mathcal{B}$ . Evaluating a single feasible solution to the ARP requires  $n - 1$  calls to  $\mathcal{B}$ . Our goal is to perform a heuristic search that requires at most  $\omega_s(n - 1)$  calls to  $\mathcal{B}$ .

Let  $V^-$  be the set of potential child nodes that could be created during the procedure. Instead of generating every possible child node  $v^- \in V^-$ , we calculate a bound on the true cost of a solution passing through each  $v^-$ . Then, we generate the  $\omega_s$  child nodes in  $v^-$  with the lowest bounds. For a potential node  $v^-$  with parent  $u^-$ , associated nodes in the relaxed DD  $v^+$  and  $u^+$  respectively, and arc  $a_{u^+v^+}$  with weight  $w(a_{u^+v^+})$ , the bound  $b(v^-)$  is calculated as:

$$b(v^-) = z_{\downarrow}(u^-) + w(a_{u^+v^+}) + z_{\uparrow}(v^+) \quad (6.13)$$

For any node  $u^-$  in a restricted DD  $\mathcal{M}^-$ , let  $u^+$  be the associated node in  $\mathcal{M}^+$ . Let  $r^+$  be the root of an existing relaxed DD  $\mathcal{M}^+$ . The procedure begins by creating  $r^-$ , without creating its children. Then it iterates over the nodes in  $\mathcal{M}^+$  associated with the potential children of  $r^-$ , and creates the  $\omega_s$  best candidate children of  $r^-$ , where candidacy is determined by the value returned by Eq. 6.13. Then this is repeated for each layer of the restricted DD. This procedure is formalized in Algorithm 15.

## 6.5 Using Peel-and-Bound for the ARP

In many problems, such as the TSP, the arc weights are exact, thus if the shortest path through the relaxed diagram is feasible, then it is also optimal. However, if the arc weights

---

**Algorithm 15:** Heuristic Search Procedure
 

---

```

1 Input: maximum width  $\omega_s$ ; relaxed DD  $\mathcal{M}^+$  with root  $r$  on layer 0
2 Let  $w(a_{xy})$  be the weight of the arc connecting  $x$  to  $y$ .
3 Let  $\mathcal{M}^-$  be a restricted DD with the same number of layers as  $\mathcal{M}^+$ , where each layer is
   initialized to  $\emptyset$  except layer 0 which is initialized to  $\{r\}$ .
4 For any node  $u^- \in \mathcal{M}^-$ , let  $u^+$  be the associated node in  $\mathcal{M}^+$ .
5 foreach layer of nodes  $L$  in  $\mathcal{M}^-$  do
6    $q \leftarrow \emptyset$  //  $q$ : empty list of nodes.
7    $b \leftarrow \emptyset$  //  $b$ : empty mapping of nodes to values.
8   foreach node  $u^- \in L$  do
9     Let the domain of  $u^-$  be  $d(u^-)$ , initialized to be the set of node labels on the
       path from  $r$  (Earth) to  $u^-$ 
10    Let  $d(u^+)$  be the set of node labels used by children of  $u^+$ 
11     $d(u^-) \leftarrow d(u^-) \cap d(u^+)$ 
12    foreach label  $l \in d(u^-)$  do
13      Create a new node  $v^-$  as a child of  $u^-$  with label  $l$ , but don't weight arc  $a_{u^-v^-}$ 
14       $b(v^-) \leftarrow z_{\downarrow}(u^-) + w_{a_{u^+v^+}} + z_{\uparrow}(v^+)$ 
15       $q \leftarrow q \cup \{v^-\}$ 
16    end
17  end
18  Sort the elements  $v^- \in q$  from least to greatest by their bounds  $b(v^-)$ 
19  while  $|q| > \omega_s$  do
20    Remove the last element of  $q$  (the largest bound) from both  $q$  and  $\mathcal{M}^-$ 
21  end
22  foreach node  $v^- \in q$  do
23     $\tau^*, t^*, z^* \leftarrow \mathcal{B}(l(u^-), l(v^-), est(u^-))$ 
24     $w(a_{u^-v^-}) \leftarrow z^*$ 
25     $est(v^-) \leftarrow est(u^-) + \tau^* + t^*$ 
26  end
27 end
28 return  $\mathcal{M}^-$ 

```

---

are just lower bounds, as in the ARP, the length of the shortest path might be a relaxed bound on the true cost, even if the path is feasible.

The only change required to use PnB for the ARP is in the decision of when to add a DD to the queue. In the original presentation of PnB, when the shortest path through a relaxed DD is a feasible solution, that DD is not placed back into the processing queue because the best known feasible solution encoded in that DD is known. However, the shortest path through a relaxed DD for the ARP being a feasible solution does not prove that path to be the optimal path in that DD because the arc weights are not exact. Thus, in order to refrain from adding a DD back into the queue for the ARP, there must be a known feasible solution with a cost



that is not larger than the length of the shortest path in that DD.

## 6.6 Implementation Details

This section details the data structures and heuristic decisions that we used to design an efficient implementation of the algorithms in this thesis.

### 6.6.1 Memoization of $\mathcal{B}$

The number of arcs stored in the processing queue during PnB can grow extremely rapidly. A core optimization when implementing PnB is refraining from performing unneeded writes to memory. As mentioned before, when new arcs are generated during PnB, they are copies of existing arcs, and have the same cost. Therefore, we only want to store the cost once. Instead of storing costs on the arcs, we need a single location in memory where all such costs are stored. Storing feasible solutions can be accomplished in a straightforward way, but storing arc bounds efficiently requires a more complex data structure.

#### Storing Feasible Solutions

The arc costs for feasible solutions are stored in a basic decision tree to avoid redundant evaluations of  $\mathcal{B}$ . The tree is initialized with a root node  $r$  having cost  $c(r) = 0$  and a label of *Earth*. When evaluating the cost of a partial (or complete) solution, the root node is queried for the first element (asteroid)  $a$  in the sequence after *Earth*. If the root node has a child  $v$  whose label is also  $a$ , then the cost of going from Earth to  $a$  is  $c(v)$ , and the next element in the sequence is compared with the labels of the children of  $v$ . This process is repeated for each element of the given sequence, adding each individual node cost to a sum total, until either the whole sequence is found in the tree and the total cost is returned, or the next element  $a$  in the sequence is not among the labels of the children of the corresponding tree node  $u$ . If  $u$  does not have a child with the same label as  $a$ , then node  $v$  is added to the tree and its fields are set as follows:

$$\begin{aligned}\tau^*, t^*, z^* &\leftarrow \mathcal{B}(l(u), l(v), est(u)) \\ c(v) &\leftarrow z^* \\ est(v) &\leftarrow est(u) + \tau^* + t^*\end{aligned}\tag{6.14}$$

The above process ensures that the cost of any feasible solution, partial or complete, is only computed once. Although a complete tree for an instance of size  $n$  has  $1 + \sum_{i=1}^n \prod_{j=0}^{i-1} (n - j)$

nodes, in practice only a very small part of the tree is generated. Even with a complete tree where the children of each node are kept sorted, it would take at most  $\mathcal{O}(k \log n)$  comparisons to retrieve the cost of a sequence of length  $k \leq n$ , which is much faster than evaluating the sequence.

## Memoizing Arc Bounds

During PnB, we need to calculate arc weights every time we peel a DD (Algorithm 12), and every time we need to compute the shortest path through a DD. However, we would like to reuse the values already computed by the  $z_{\min}$  functions of phase one and two (Eqs. 6.7 and 6.9, respectively). Thus, we wish to look-up a pre-computed value of  $z_{\min}$  that would be valid for the new arc.

Both definitions of  $z_{\min}$  only depend on the evaluation of  $\mathcal{B}'(a, a', \eta, \tau_f)$ . Moreover, by the definition of the inner optimization problem (Eq. 6.5), we also know that  $\mathcal{B}'(a, a', \eta', \tau'_f)$  cannot return a cost  $z$  better than  $\mathcal{B}'(a, a', \eta, \tau_f)$  if  $[\eta', \eta' + \tau'_f] \subseteq [\eta, \eta + \tau_f]$ , because the inner optimization problem is the same but with tighter bounds on  $\tau$  for the former.

During a run of PnB, the time intervals  $[\eta, \eta + \tau_f]$  implied by the arcs can only decrease in the above manner. Let  $u$  be a node that is split into  $u'_1$  and  $u'_2$ , and let  $c(u)$  be the cost of node  $u$ . The bounds on the new nodes must be at least as tight as the bounds on the old nodes so  $c(u) \leq c(u'_1), c(u) \leq c(u'_2)$ . The earliest starting time of the nodes can increase but not decrease, so  $est(u) \leq est(u'_1), est(u) \leq est(u'_2)$ . This means that the interval start can get larger but not smaller. Let  $\tau_u$  be the wait time associated with  $u$ , the wait time of the peeled nodes can decrease but not increase,  $\tau_u \geq \tau_{u'_1}, \tau_u \geq \tau_{u'_2}$ , which means that the interval end can get smaller but not larger. Thus, the time interval  $[\eta, \eta + \tau_f]$  used to compute the arc weight associated with a node  $u$ , must contain the time intervals that would be required to compute the arc weights associated with  $u'_1$  and  $u'_2$ .

Let us store in an interval tree the  $z_{\min}$  values resulting from any evaluation of  $\mathcal{B}'$  with origin  $a$  and destination  $a'$  together with the time intervals used in the evaluation. Given an arc that has an associated time interval and whose origin and destination correspond to  $a$  and  $a'$ , every stored  $z_{\min}$  value with a time interval that contains the time interval of the arc is a valid bound for the arc. The highest value among those stored, provides the strongest bound.

Interval trees are a standard data structure designed for efficiently finding all intervals that overlap with any given interval or point. They have a lookup time for finding overlapping intervals in  $\mathcal{O}(n + m)$  where  $n$  is the number of intervals in the tree, and  $m$  is the number of intervals that satisfy the query. A standard interval tree is optimized for finding overlapping

intervals; each tree-node stores the maximum value of its descendants. In our case, we only ever need to find containing intervals, thus we also store the minimum value (interval start) of its descendants in each tree-node. This prevents wasted time from looking for containing intervals on a branch of the tree when the lower bound on that branch is higher than the start time of our interval. We also adjusted the lookup function to only return containing intervals, instead of all overlapping intervals.

### 6.6.2 Heuristic Decisions

In our implementation, we test two different methods for selecting a DD from  $Q$  (line 7 in Algorithm 13): worst bound, and largest index. Selecting the DD with the worst bound is similar to performing a breadth-first-search; it is making a decision to try to always improve the current lower bound on the problem, but can lead to a very large  $Q$ . Selecting the DD with a root node whose index in the initial DD was highest is similar to performing a depth-first search; it prioritizes keeping  $Q$  small over improving the lower bound as quickly as possible.

We also test two different methods for selecting a node to peel from DD  $Q$  (line 7 in Algorithm 13). The first method is to pick the last exact node, which is the highest indexed exact node on the shortest path through  $Q$ . The second method is to pick the maximal node, which is the lowest indexed node on the shortest path through  $Q$ . These methods are discussed in detail in [9].

### 6.6.3 Limitations of the Inner Optimizer

The inner problem is non-convex and non-linear, making it impossible in general to prove that a local optimum is also a global optimum. In  $\mathcal{B}$ , the search space is small enough that local optima are likely to be global optima as well. Even if this is not the case, the inner optimizer is deterministic, meaning that the same permutation will yield the same local optima for the inner optimization problems. The goal is to find the globally optimal permutation. When using  $\mathcal{B}'$ , the inner optimizer searches within a superset of the search space induced by  $\mathcal{B}$ , potentially finding a local optimum that is worse than the one returned by  $\mathcal{B}$ . In such cases, the outer optimization might be misled about which permutation is optimal. However, the permutation found is still a solution (possibly suboptimal) to the original ARP, because feasible solutions are always evaluated using  $\mathcal{B}$ , not  $\mathcal{B}'$ .

More concretely, the inner optimization that computes  $\mathcal{B}$  in [77] consists of a single deterministic run of SLSQP, as explained in Section 6.2.1. A single run of SLSQP almost always

returns a global optimum when  $\tau_f \leq \tau_{\max}$  as set in the original paper. However, SLSQP may return a locally optimal solution when  $\tau_f \gg \tau_{\max}$ , as we set it here when computing  $\mathcal{B}'$  (Eq. 6.6). That is,  $\mathcal{B}'(a, a', \eta = 0, \tau_f)$  may return  $(\tau', t', z')$  as optimal yet  $\mathcal{B}'(a, a', \eta = \tau', \tau_f)$  may return a better solution even though the time interval  $[\tau', \tau_f]$  is contained within  $[0, \tau_f]$ . This problem can be alleviated by performing multiple restarts of SLSQP for each evaluation of  $\mathcal{B}'$ . In other words, when optimizing the relaxed inner problem (Eq. 6.5), we can partition the range  $[\eta, \eta + \tau_f]$  into disjoint intervals, run SLSQP for each interval, and keep the best solution. We report experiments with different number of restarts that show their effect on runtime and solution quality.

SLSQP could be replaced with a more effective optimizer, such L-BFGS-B [82] or CMA-ES [83]. For other inner problems, it may be possible to ensure that the inner optimizer returns a global optimum. In this thesis, we decided to keep SLSQP as the inner optimizer in order to compare our results on the ARP with those reported by [77]. The following experiments (Section 6.7) show that increasing the quality of the inner optimizer does lead to better solutions, but not always, and not by much, so it is likely that we are already finding the optimal solutions for several of the ARP instances.

## 6.7 Experimental Results

An ARP instance with  $n$  asteroids is constructed by randomly selecting  $n$  asteroids from a database containing 83,453 asteroids. The selection process is controlled by a *seed* value, ensuring that specific instances can be consistently replicated. In [77], they test their algorithms with  $n \in \{10, 15, 20, 25, 30\}$  and  $seed \in \{42, 73\}$ . We ran experiments with the same  $n$  values, but added three randomly chosen seeds: 8, 22, and 59. The experiments were performed on a computer equipped with an AMD Rome 7532 at 2.40 GHz with 64Gb RAM. Our code and raw experimental data are available at <https://zenodo.org/doi/10.5281/zenodo.12675217>. Several of the results tables in this section have a *queue* column. This value is the number of DDs remaining in the processing queue when the solver ran out of time.

### 6.7.1 Note on Optimality

As discussed in Section 6.6.3, changing the quality of the inner optimizer may impact the quality of solutions returned by our framework. For several of the problems in this section, we report different optimal solutions using different settings. These different solutions are the result of the inner optimizer not being of sufficient quality for some settings, and thus are intentional, and do not represent a mistake in the implementation. *multi* is the setting

we used to change the quality of the inner solver. It is the number of restarts that the inner solver uses when searching for an optimal solution.

### 6.7.2 Initial Experiment: Determining Best Settings

In our first test, shown in Table 6.1, we sought to determine which peel setting would be the most effective, which DD-widths  $\omega$  would be the most effective, and the effect of using depth-first search (dfs, see Section 6.6.2). The value of  $\omega$  used can have a large impact on the solve time because relaxed DDs with larger widths generally yield better bounds, but also take longer to compute. The best value for  $\omega$  is the one that balances this quality/time trade-off.

We tested  $n = 15$  with all 5 seeds, 2 days of runtime, and no SLSQP restarts (*multi* = 1 means one SLSQP run for solving the inner optimization problem). Figure 6.4 summarizes these results; the lines show the mean gap (or mean time) over the ARP instances.

We tested two peel settings: maximal and last exact node. Although there is no clear winner, maximal generally performs better and performs only slightly worse when it does not. Consequently, we will use the maximal setting in all future experiments. The effectiveness of the depth-first-search (dfs) queue ordering appears random based on summary data. The raw data, which is also available in the repository, contains a timestamp and bounds for each improvement to the bounds made during a run of PnB. The raw data suggests the success of dfs mainly depends on whether the best solution is found in an early branch. It also suggests that dfs is slightly worse for the *maximal* peel setting and both  $\omega = 512$  and  $\omega = 2048$ , as also shown in the left plot of Fig. 6.4. We disabled dfs ordering in the remaining experiments, opting for the default ordering that processes the DD with the weakest bound, likely reducing the optimality gap in unresolved problems. Regarding relaxed DD-width, we tested 512, 1024, and 2048. While 2048 showed the greatest success in closing problems and narrowing the optimality gap, 512 resolved specific settings significantly faster. Moving forward, we will retest 512 and 2048, and introduce a new width of 256. We do not include tests with widths larger than 2048 because in our experience with DDs, the slowdown in construction time from increasing the width starts to become substantial around 2048. This happens because the number of arcs at each layer is at most  $\mathcal{O}(\omega^2 n)$ , and small increases in the width beyond 2048 can lead to enormous increases in the construction time.

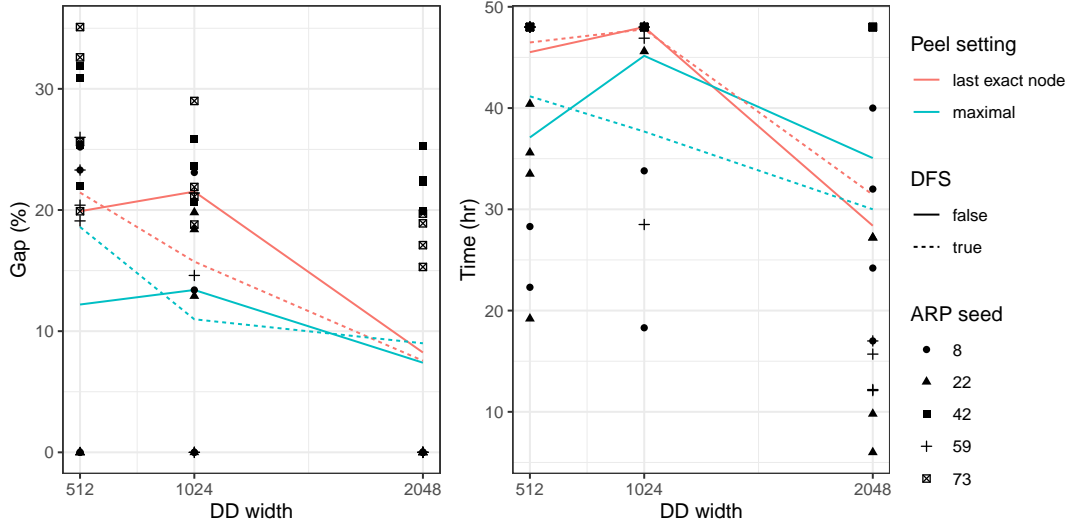


Figure 6.4 ARP instances with  $n = 15$ , a 2 day runtime, an embedded search width of 100, and  $multi = 1$  (Table 6.1)

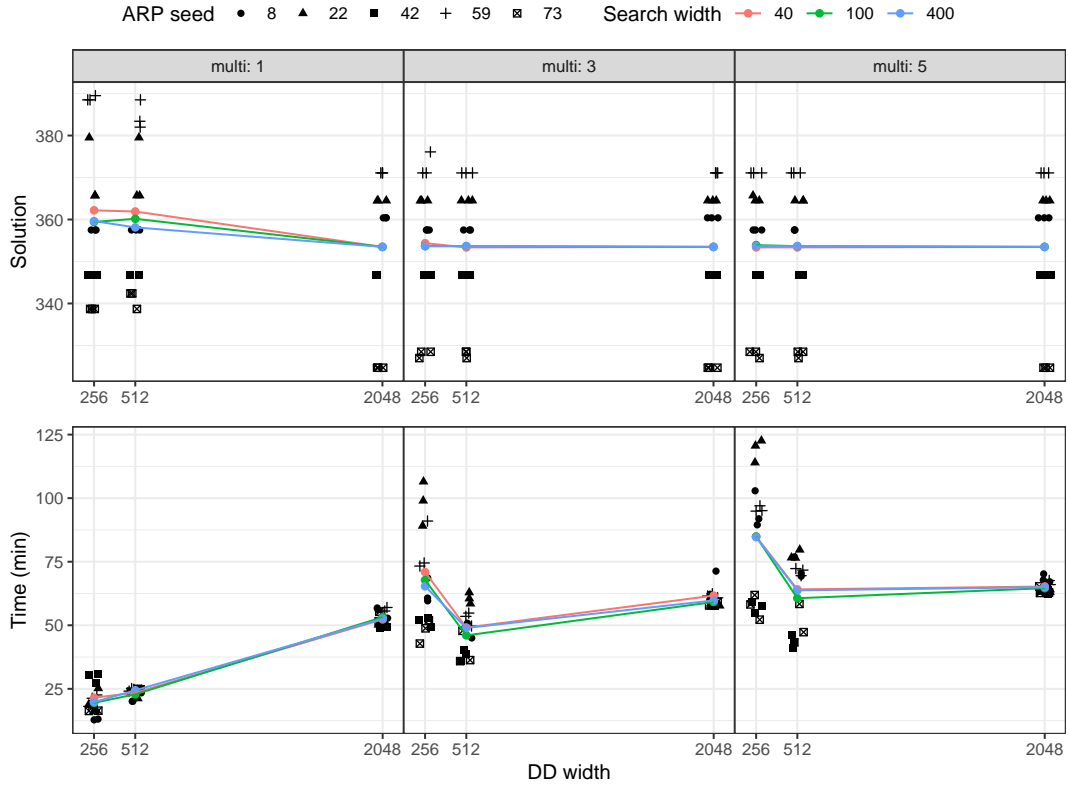


Figure 6.5 ARP instances with  $n = 10$ . Lines show mean value over ARP instances (Table 6.2)

Table 6.1 ARP instances with  $n = 15$ , a 2 day runtime, an embedded search width of 100, and  $multi = 1$

$dfs$  = depth first search (queue processing order),  $DD\ width$  = width of relaxed DD,  
 $lb$  = lower bound,  $ub$  = upper bound,  $gap$  = optimality gap,  $time$  = time to solve

seed	peel setting	dfs	DD width	lb	ub	gap (%)	time (hr)
8	last exact node	true	512	406.0	528.9	23.3	-
			1024	433.3	500.4	13.4	-
			2048	499.5	499.5	0.0	17.0
	maximal	true	512	499.5	499.5	0.0	28.3
			1024	500.1	500.1	0.0	18.3
			2048	504.5	504.5	0.0	32.0
	last exact node	false	512	411.3	550.0	25.2	-
			1024	412.6	536.5	23.1	-
			2048	500.9	500.9	0.0	24.2
	maximal	false	512	499.5	499.5	0.0	22.3
			1024	499.5	499.5	0.0	33.8
			2048	503.8	503.8	0.0	40.0
22	last exact node	true	512	472.4	472.4	0.0	40.4
			1024	400.9	500.0	19.8	-
			2048	501.7	501.7	0.0	27.2
	maximal	true	512	472.4	472.4	0.0	33.5
			1024	501.7	501.7	0.0	45.6
			2048	481.9	481.9	0.0	9.8
	last exact node	false	512	472.4	472.4	0.0	35.6
			1024	406.6	498.5	18.4	-
			2048	482.3	482.3	0.0	6.0
	maximal	false	512	472.4	472.4	0.0	19.2
			1024	438.9	504.0	12.9	-
			2048	501.7	501.7	0.0	27.2
42	last exact node	true	512	351.2	508.2	30.9	-
			1024	382.3	500.1	23.6	-
			2048	393.4	507.5	22.5	-
	maximal	true	512	346.0	508.2	31.9	-
			1024	376.7	508.2	25.9	-
			2048	379.6	508.2	25.3	-
	last exact node	false	512	379.3	508.2	25.4	-
			1024	8.1	508.2	23.6	-
			2048	394.5	507.5	22.3	-
	maximal	false	512	395.3	506.9	22.0	-
			1024	402.2	506.9	20.7	-
			2048	406.0	506.9	19.9	-
59	last exact node	true	512	416.6	523.2	20.4	-
			1024	523.2	523.2	0.0	46.9
			2048	530.4	530.4	0.0	17.0
	maximal	true	512	403.7	545.8	26.0	-
			1024	523.2	523.2	0.0	28.5
			2048	530.4	530.4	0.0	12.2
	last exact node	false	512	420.9	548.9	23.3	-
			1024	426.3	542.7	21.4	-
			2048	530.4	530.4	0.0	15.7
	maximal	false	512	440.3	543.9	19.1	-
			1024	446.7	523.2	14.6	-
			2048	530.4	530.4	0.0	12.1
73	last exact node	true	512	333.9	495.2	32.6	-
			1024	389.2	498.0	21.9	-
			2048	419.5	495.2	15.3	-
	maximal	true	512	326.3	502.5	35.1	-
			1024	351.7	495.2	29.0	-
			2048	402.3	500.8	19.7	-
	last exact node	false	512	373.8	502.5	25.6	-
			1024	396.3	502.5	21.1	-
			2048	407.6	502.5	18.9	-
	maximal	false	512	398.7	498.1	19.9	-
			1024	408.1	502.5	18.8	-
			2048	416.3	502.1	17.1	-

Table 6.2 ARP instances with  $n = 10$ 

*search* = width of embedded search, *DD width* = width of relaxed dd  
*time* = time to solve

seed	search	DD width	<i>multi</i> = 1		<i>multi</i> = 3		<i>multi</i> = 5	
			solution	time(min)	solution	time(min)	solution	time(min)
8	40	256	357.5	13.1	357.5	60.7	357.5	89.5
		512	357.5	20.1	357.5	49.6	357.5	69.3
		2048	360.4	52.8	360.4	71.3	360.4	70.2
	100	256	357.5	12.8	357.5	59.6	357.5	91.9
		512	357.5	20.1	357.5	45.0	357.5	70.5
		2048	360.4	56.1	360.4	60.0	360.4	67.8
	400	256	357.5	16.1	357.5	68.6	357.5	102.9
		512	357.5	23.4	357.5	50.8	357.5	76.9
		2048	360.4	56.8	360.4	62.2	360.4	67.1
22	40	256	379.5	25.2	364.5	99.0	364.5	120.6
		512	379.5	24.5	364.5	58.5	364.5	79.7
		2048	364.5	50.2	364.5	57.7	364.5	63.0
	100	256	365.7	16.2	364.5	106.5	365.7	122.6
		512	365.7	21.2	364.5	60.5	364.5	76.6
		2048	364.5	51.4	364.5	58.5	364.5	64.1
	400	256	365.7	18.9	364.5	89.1	364.5	114.0
		512	365.7	23.8	364.5	62.9	364.5	76.5
		2048	364.5	51.2	364.5	58.8	364.5	64.0
42	40	256	346.7	30.3	346.7	51.9	346.7	57.5
		512	346.7	23.7	346.7	36.0	346.7	40.9
		2048	346.7	50.1	346.7	59.3	346.7	63.9
	100	256	346.7	30.9	346.7	49.5	346.7	54.9
		512	346.7	24.8	346.7	38.6	346.7	43.3
		2048	346.7	49.3	346.7	58.0	346.7	62.4
	400	256	346.7	27.2	346.7	52.9	346.7	59.3
		512	346.7	25.1	346.7	40.4	346.7	46.3
		2048	346.7	49.1	346.7	57.7	346.7	63.3
59	40	256	388.5	22.6	376.1	91.0	371.1	94.9
		512	383.4	25.1	371.1	53.5	371.1	72.3
		2048	371.1	57.0	371.1	60.8	371.1	63.8
	100	256	388.5	21.3	371.1	74.5	371.1	97.0
		512	388.5	24.1	371.1	49.7	371.1	69.5
		2048	371.1	53.9	371.1	61.5	371.1	66.0
	400	256	389.5	18.1	371.1	73.3	371.1	95.1
		512	382.0	24.8	371.1	54.8	371.1	71.7
		2048	371.1	55.6	371.1	62.4	371.1	67.6
73	40	256	338.7	16.4	327.0	52.0	327.0	61.9
		512	342.4	23.9	327.0	47.8	327.0	58.4
		2048	324.7	51.5	324.7	59.2	324.7	65.3
	100	256	338.7	16.3	328.5	48.8	328.5	58.2
		512	342.4	24.0	328.5	36.3	328.5	43.3
		2048	324.7	55.3	324.7	57.8	324.7	62.7
	400	256	338.7	18.7	328.5	42.8	328.5	52.2
		512	338.7	25.0	328.5	35.9	328.5	47.3
		2048	324.7	50.2	324.7	57.8	324.7	62.9

### 6.7.3 Second Experiment: Test of Smaller Instances

Our first round of experiments gave us a rough idea of what settings might be successful. Our second round of experiments makes up the bulk of our tests. As discussed in the previous paragraph we test relaxed DD-widths of 256, 512, and 2048. For the embedded search widths ( $\omega_s$  from Algorithm 15) we tested 40, 100, and 400, where previously we just used 100. In typical DD literature [1], the size of the search for feasible solutions would be as big or bigger than the size of the relaxed DD, because it is relatively computationally cheap. However, for



the ARP, the search for feasible solutions is computationally expensive because of the inner-optimization that requires evaluating the black-box function  $\mathcal{B}$ . This suggested to us that larger embedded search widths are unlikely to be worth it. However, the following results indicate that might not be true. We also tested  $multi \in \{1, 3, 5\}$ . Table 6.2 (Fig. 6.5) show the results of the second round of experiments for  $n = 10$ , Table 6.4 (Fig. 6.6) for  $n = 15$ , and Table 6.5 (Fig. 6.7) for  $n = 20$ . In Table 6.2, every solution is optimal, so it does not include the *gap* columns.

The largest search settings, i.e., an embedded search width of 400 and a relaxed DD-width of 2048, clearly dominate the results, with tighter optimality gaps and more problems solved. For the remainder of the discussion, we will be referring primarily to those results. Recall from Section 6.6.3 that different *multi* values can have different optimal solutions because the inner optimizer may return a local optimum instead of the global optimum. When  $n = 10$  (Table 6.2 and Fig. 6.5) and  $multi = 1$ , all of the problems are solved to optimality in under 1 hour, and when  $multi = 5$ , they are solved to optimality in under 2 hours. In some cases the largest search settings alleviates some of the negative effects of  $multi = 1$  (in Table 6.2, seed 22, DD width 512, compare search width 40 to search width 100). This is likely because a larger search provides more opportunities to stumble on improved solutions that will later be trimmed as a result of using a low *multi*. For the the largest search settings, the effect of *multi* is only to increase the runtime, however settings with smaller DD widths make it clear that *multi* can have a strong impact on the underlying arc calculations. For example, seed 59, with search width 40, and relaxed DD-width 256, has a different exact solution with  $multi = 1$  than it does with  $multi = 5$ ; the cost drops from 388.5 to 371.1. Recall that because the inner problem in the ARP is nonlinear and nonconvex, it is not possible to prove optimality in general. In lieu of that, *multi* can be increased arbitrarily to increase confidence in the solutions. If *multi* is increased, and the algorithm returns the same permutation as it did for a lower *multi* value, that provides evidence that the optimal solution is stable. The larger the increase in *multi*, the stronger the evidence.

We now focus on the results obtained with an embedded search width of 400 and a relaxed DD-width of 2048 on instances of size  $n \in \{15, 20\}$  (Tables 6.4 and 6.5) with a maximum runtime of 7 days. With  $multi = 1$ , three of the problems were solved in under a day, with the other two taking around two days. However, many of the solutions are improved with a higher *multi* (see the top row of Figures 6.6 and 6.7). For  $n = 15$ , the setting  $multi = 1$  is unlikely to yield optimal solutions. With  $multi = 5$ , only seeds 8 and 22 were solved exactly, but they yielded much better solutions. In seed 22 for example, the best solution drops from 501.7 to 466.3. With  $n = 20$ , none of the problems were solved to optimality within the 7 day limit.

The only available comparison we can make is with [77]. However, because they assume that the inner problem is a total black box, and we take advantage of the structure of the black box, any time/computation comparison is meaningless. So we will merely be comparing our solutions to theirs, to show that our method yields high quality solutions. They only used seeds 42 and 73, so for the other problems we can only report our solutions. These results are shown in Table 6.3. Notably the solution reported by Peel-and-Bound's preferred setting ( $\omega_s = 400$ ,  $\omega = 2048$ ,  $multi=5$ ) is the same as the best previously found for  $n = 10$ , and better than those previously reported for  $n = 15$  and  $n = 20$ . In some cases, the gap is quite large, for example with  $n = 20$ , seed= 42, our solution is 12.0% better. We also report the best observed solution from any setting, and for 13 of the 15 instances the preferred setting found the same solution as the best solution found by any setting. All of these results use  $\mathcal{B}$  to evaluate the cost of a permutation, and thus they are directly comparable. In other words, the values listed are not dependent on the optimality of the inner problem.

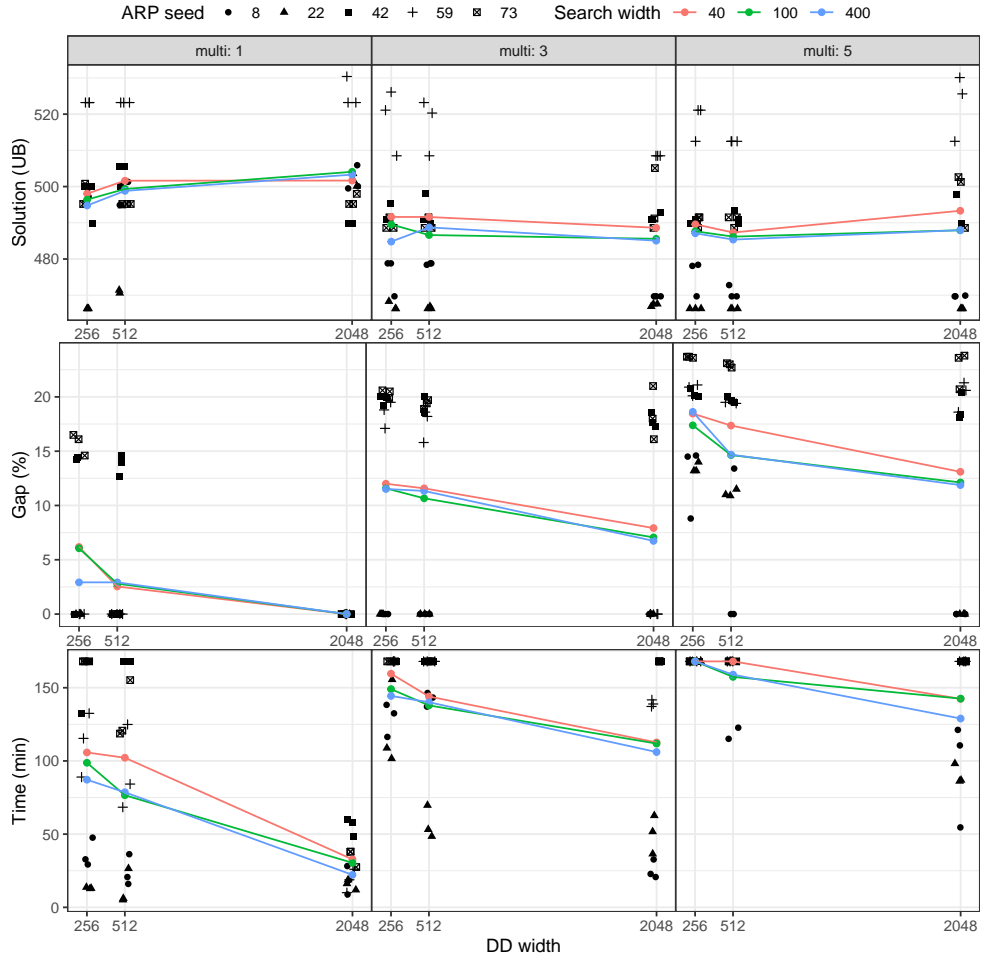
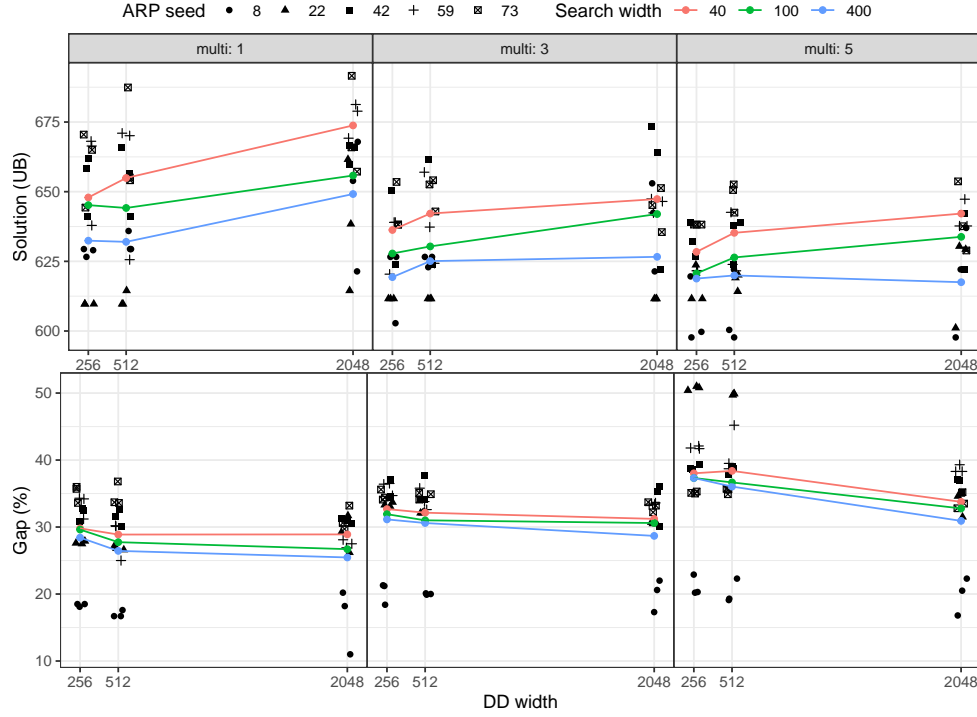


Figure 6.6 ARP instances with  $n = 15$  and a 7 day runtime (Table 6.4)

Figure 6.7 ARP instances with  $n = 20$  and a 7 day runtime (Table 6.5)Table 6.3 Best solutions for  $n \in \{10, 15, 20\}$ 

n	seed	[77]*		Peel-and-Bound: Preferred Settings**		Peel-and-Bound: Best Found***	
		Average Value	Best Value	Value	Optimality Gap (%)	Value	Optimality Gap (%)
10	8	-	-	360.4	0.0	357.5	0.0
	22	-	-	364.5	0.0	364.5	0.0
	42	374.9	346.7	346.7	0.0	346.7	0.0
	59	-	-	371.1	0.0	371.1	0.0
	73	355.9	324.7	324.7	0.0	324.7	0.0
15	8	-	-	469.7	0.0	469.7	0.0
	22	-	-	466.3	0.0	466.3	0.0
	42	497.2	490.9	489.7	18.1	489.7	0.0
	59	-	-	525.6	20.6	<b>508.5</b>	0.0
	73	525.6	519.9	488.6	20.7	488.6	16.1
20	8	-	-	597.7	16.8	597.7	20.3
	22	-	-	601.1	31.5	601.1	31.5
	42	737.0	707.2	622.2	35.2	622.2	30.1
	59	-	-	637.7	38.3	<b>619.9</b>	41.7
	73	661.8	652.5	628.9	32.8	628.9	32.8

\* The results from [77] in the *Average Found* column are the average solution cost over several runs of the same stochastic algorithm. *Best Found* reports the best solution found in any run of the algorithm.

\*\* The preferred settings for Peel-and-Bound are:  $\omega_s = 400$ ,  $\omega = 2048$ , *multi* = 5.

\*\*\* The bold values are the only instances where the best found solution by any setting is better than the solution found using the preferred setting.

Table 6.4 ARP instances with  $n = 15$  and a 7 day runtime

$search$  = width of embedded search,  $DD\ width$  = width of relaxed dd  
 $lb$  = lower bound,  $ub$  = upper bound,  $gap$  = optimality gap,  
 $time$  = time to solve if solved,  $queue$  = nodes in the processing queue

seed	search	DD width	$multi = 1$					$multi = 3$					$multi = 5$				
			lb	ub	gap (%)	time(hr)	queue	lb	ub	gap (%)	time(hr)	queue	lb	ub	gap (%)	time(hr)	queue
8	40	256	499.8	499.8	0.0	47.6	0	478.8	478.8	0.0	138.3	0	408.1	478.1	14.6	-	1172
		512	494.9	494.9	0.0	36.3	0	478.4	478.4	0.0	146.4	0	409.3	472.8	13.4	-	579
		2048	500.1	500.1	0.0	28.2	0	469.7	469.7	0.0	22.8	0	469.7	469.7	0.0	121.2	0
	100	256	497.2	497.2	0.0	29.3	0	478.8	478.8	0.0	132.5	0	428.4	469.7	8.8	-	311
		512	501.3	501.3	0.0	20.7	0	478.8	478.8	0.0	137.1	0	469.7	469.7	0.0	115.1	0
		2048	505.9	505.9	0.0	18.8	0	469.7	469.7	0.0	32.7	0	469.9	469.9	0.0	110.6	0
	400	256	499.5	499.5	0.0	32.8	0	469.7	469.7	0.0	116.4	0	409.2	478.4	14.5	-	1011
		512	499.5	499.5	0.0	16.0	0	478.8	478.8	0.0	143.3	0	469.7	469.7	0.0	122.7	0
		2048	499.5	499.5	0.0	8.8	0	469.7	469.7	0.0	20.7	0	469.7	469.7	0.0	54.6	0
22	40	256	466.3	466.3	0.0	12.8	0	466.3	466.3	0.0	155.6	0	404.6	466.3	13.2	-	1152
		512	494.7	494.7	0.0	26.5	0	466.9	466.9	0.0	69.7	0	415.0	466.3	11.0	-	179
		2048	500.1	500.1	0.0	11.9	0	466.9	466.9	0.0	62.7	0	466.3	466.3	0.0	86.9	0
	100	256	466.3	466.3	0.0	13.0	0	468.3	468.3	0.0	108.8	0	404.7	466.3	13.2	-	1099
		512	471.4	471.4	0.0	5.1	0	466.3	466.3	0.0	48.5	0	415.6	466.3	10.9	-	167
		2048	503.6	503.6	0.0	18.8	0	467.6	467.6	0.0	51.7	0	466.3	466.3	0.0	98.2	0
	400	256	466.3	466.3	0.0	13.7	0	466.3	466.3	0.0	101.6	0	401.2	466.3	14.0	-	1058
		512	470.6	470.6	0.0	6.1	0	466.3	466.3	0.0	53.2	0	412.9	466.3	11.5	-	206
		2048	501.7	501.7	0.0	16.2	0	467.6	467.6	0.0	36.5	0	466.3	466.3	0.0	86.2	0
42	40	256	428.0	500.1	14.4	-	1259	396.5	495.4	20.0	-	1672	392.6	490.9	20.0	-	1634
		512	436.7	500.1	12.7	-	316	398.2	498.0	20.0	-	747	394.6	493.5	20.0	-	712
		2048	489.7	489.7	0.0	59.7	0	401.4	492.9	18.6	-	141	396.4	497.9	20.4	-	147
	100	256	428.9	500.1	14.2	-	1203	396.6	490.9	19.2	-	1643	391.4	489.7	20.1	-	1537
		512	434.8	505.6	14.0	-	414	399.8	490.9	18.6	-	752	395.2	490.9	19.5	-	709
		2048	489.7	489.7	0.0	57.7	0	406.0	490.9	17.3	-	105	399.6	489.7	18.4	-	127
	400	256	489.7	489.7	0.0	132.3	0	392.9	490.9	19.9	-	1312	388.0	489.7	20.8	-	1232
		512	431.7	505.6	14.6	-	397	398.3	489.7	18.7	-	671	393.1	489.7	19.7	-	646
		2048	489.7	489.7	0.0	48.5	0	404.4	490.9	17.6	-	109	400.8	489.7	18.1	-	126
59	40	256	523.2	523.2	0.0	132.5	0	423.4	526.1	19.5	-	1617	412.0	521.1	20.9	-	1490
		512	523.2	523.2	0.0	124.9	0	427.8	523.2	18.2	-	721	412.9	512.5	19.4	-	661
		2048	523.2	523.2	0.0	26.0	0	508.5	508.5	0.0	141.7	0	417.1	530.1	21.3	-	156
	100	256	523.2	523.2	0.0	115.4	0	423.1	521.1	18.8	-	1517	411.0	521.1	21.1	-	1386
		512	523.2	523.2	0.0	68.4	0	428.1	508.5	15.8	-	655	411.7	512.5	19.7	-	591
		2048	523.2	523.2	0.0	18.8	0	508.5	508.5	0.0	138.9	0	417.4	512.5	18.6	-	144
	400	256	523.2	523.2	0.0	89.0	0	421.5	508.5	17.1	-	1330	409.5	512.5	20.1	-	1208
		512	523.2	523.2	0.0	84.2	0	423.6	520.3	18.6	-	543	412.6	512.5	19.5	-	613
		2048	530.4	530.4	0.0	10.0	0	508.5	508.5	0.0	137.2	0	417.6	525.6	20.6	-	147
73	40	256	418.2	500.8	16.5	-	1,253	390.9	491.5	20.5	-	1,653	375.7	491.5	23.6	-	1,557
		512	495.2	495.2	0.0	155.2	0	394.5	491.5	19.7	-	737	378.4	491.5	23.0	-	689
		2048	495.2	495.2	0.0	38.1	0	399.3	505.1	21.0	-	152	382.9	502.6	23.8	-	149
	100	256	415.5	495.2	16.1	-	1,010	391.2	488.6	19.9	-	1,672	375.1	491.5	23.7	-	1,509
		512	495.2	495.2	0.0	120.7	0	396.3	488.6	18.9	-	741	378.0	491.5	23.1	-	671
		2048	498.0	498.0	0.0	37.9	0	403.0	491.2	18.0	-	134	383.2	501.3	23.6	-	144
	400	256	422.7	495.2	14.6	-	1,098	387.7	488.6	20.6	-	1,361	372.9	488.6	23.7	-	1,284
		512	495.2	495.2	0.0	118.7	0	393.9	488.6	19.4	-	658	377.6	488.6	22.7	-	641
		2048	495.2	495.2	0.0	27.5	0	410.1	488.6	16.1	-	77	387.5	488.6	20.7	-	135

Table 6.5 ARP instances with  $n = 20$  and a 7 day runtime

*search* = width of embedded search, *DD width* = width of relaxed dd  
*lb* = lower bound, *ub* = upper bound, *gap* = optimality gap,  
*queue* = nodes in the processing queue

seed	search	DD width	<i>multi</i> = 1				<i>multi</i> = 3				<i>multi</i> = 5			
			lb	ub	gap (%)	queue	lb	ub	gap (%)	queue	lb	ub	gap (%)	queue
8	40	256	512.7	629.4	18.5	1,012	493.5	626.6	21.2	708	477.4	619.6	22.9	639
		512	524.0	635.9	17.6	557	501.4	626.6	20.0	360	483.2	622.1	22.3	332
		2048	533.2	667.9	20.2	90	509.7	653.0	22.0	70	495.0	637.0	22.3	70
	100	256	513.3	626.6	18.1	1,101	492.9	626.6	21.3	679	478.2	599.7	20.3	678
		512	524.2	629.4	16.7	598	500.4	626.6	20.1	353	484.7	600.4	19.3	369
		2048	535.2	653.9	18.2	102	510.3	642.4	20.6	76	494.7	622.1	20.5	75
	400	256	512.7	629.0	18.5	948	492.1	602.8	18.4	626	476.7	597.7	20.2	581
		512	524.3	629.4	16.7	509	499.3	622.9	19.9	328	483.4	597.7	19.1	344
		2048	552.9	621.4	11.0	84	514.1	621.4	17.3	94	497.5	597.7	16.8	91
22	40	256	442.3	609.7	27.5	710	408.8	611.6	33.1	644	305.6	623.7	51.0	629
		512	447.5	614.5	27.2	380	415.6	611.6	32.1	331	310.9	620.1	49.9	320
		2048	451.8	661.6	31.7	69	423.9	611.6	30.7	69	410.6	630.4	34.9	66
	100	256	441.2	609.7	27.6	649	407.7	611.6	33.3	590	303.5	611.6	50.4	578
		512	447.5	609.7	26.6	389	415.4	611.6	32.1	319	310.3	619.1	49.9	314
		2048	453.8	614.5	26.2	80	425.4	611.6	30.5	69	411.5	629.3	34.6	66
	400	256	439.8	609.7	27.9	582	405.6	611.6	33.7	515	301.3	611.6	50.8	496
		512	446.2	609.7	26.8	357	414.2	611.6	32.3	297	308.8	614.2	49.7	284
		2048	451.8	638.4	29.2	69	425.4	611.6	30.5	68	412.1	601.1	31.5	67
42	40	256	445.6	661.9	32.7	616	409.0	650.6	37.1	613	387.8	638.8	39.3	598
		512	448.7	665.8	32.6	321	412.3	661.5	37.7	331	391.0	638.8	38.8	331
		2048	457.6	666.5	31.3	66	430.6	673.5	36.1	65	404.3	642.3	37.1	65
	100	256	444.4	658.2	32.5	591	409.2	623.7	34.4	597	386.9	632.2	38.8	575
		512	448.8	656.5	31.6	327	411.3	623.7	34.1	301	389.1	638.0	39.0	297
		2048	459.3	665.8	31.0	67	429.8	664.2	35.3	63	404.8	642.3	37.0	63
	400	256	443.1	641.2	30.9	548	408.6	623.7	34.5	522	384.9	626.7	38.6	499
		512	448.4	641.2	30.1	290	411.0	623.7	34.1	276	388.1	623.7	37.8	273
		2048	458.3	659.6	30.5	66	435.0	622.2	30.1	65	403.2	622.2	35.2	63
59	40	256	439.0	668.1	34.3	612	406.4	639.0	36.4	594	361.9	621.7	41.8	589
		512	468.4	671.0	30.2	331	422.1	657.0	35.8	316	352.4	642.6	45.2	311
		2048	489.6	681.3	28.1	68	429.9	647.4	33.6	70	392.7	647.3	39.3	67
	100	256	438.5	666.4	34.2	592	405.8	639.0	36.5	568	359.7	621.6	42.1	560
		512	468.2	670.1	30.1	325	421.9	637.3	33.8	308	377.4	623.9	39.5	301
		2048	492.6	678.9	27.5	71	430.7	646.5	33.4	68	393.3	637.7	38.3	69
	400	256	438.9	637.9	31.2	595	404.8	620.4	34.7	514	361.7	619.9	41.7	469
		512	469.2	625.6	25.0	351	420.9	624.3	32.6	293	381.1	621.6	38.7	283
		2048	489.3	669.2	26.9	72	428.4	642.4	33.3	67	393.4	637.7	38.3	66
73	40	256	429.0	670.5	36.0	630	420.8	653.5	35.6	614	414.0	638.2	35.1	559
		512	434.1	687.4	36.8	312	424.6	654.1	35.1	314	419.5	652.6	35.7	313
		2048	462.0	691.6	33.2	74	431.6	651.3	33.7	68	423.9	653.7	35.2	66
	100	256	427.8	665.0	35.7	562	420.3	638.2	34.1	571	415.0	638.2	35.0	558
		512	434.6	655.1	33.7	321	424.6	652.6	34.9	316	419.4	650.6	35.6	309
		2048	462.2	665.9	30.6	69	430.7	645.2	33.2	66	423.9	637.6	33.5	68
	400	256	427.5	644.3	33.6	543	418.9	638.2	34.4	485	412.9	638.2	35.3	507
		512	434.1	654.1	33.6	300	423.8	642.9	34.1	291	418.2	642.5	34.9	288
		2048	462.2	657.2	29.7	70	431.1	635.5	32.2	63	422.7	628.9	32.8	65

#### 6.7.4 Final Experiment: Test of Larger Instances

In our final round of experiments we run the larger instances with  $n \in \{25, 30\}$ . Due to limited resource availability we had to use a slightly less powerful computer equipped with an Intel E5-2650 v4 Broadwell 2.2GHz CPU with 64Gb RAM. We use a relaxed DD-width of 2048, and we test larger embedded search sizes than before: 400, 1024, and 2048. We limit the runtime to 3 days, again due to resource availability, but this has the added benefit of showing that the algorithm works well within a smaller time-frame than the 7 days used in

## Section 6.7.3.

The results, shown in Tables 6.6 and 6.7, suggest that increasing the width of the embedded search may be worth the extra computational cost of constructing the additional solutions, but the results are not enough to be conclusive.  $\omega_s = 2048$  finds the best solutions more often than the other settings, so as before we will choose the largest search setting to be the preferred setting. More experiments would be needed to make a strong recommendation between the search settings, but it is clear that all of the settings used in this section are effective.

We again compare our results with [77] where possible, and record the best solution found using our preferred setting as well as the best solutions found by any setting. This is shown in Table 6.8. As before, the solutions from our preferred setting are better than the best solutions found by [77] for all of the problems. The preferred solution only matches the best found solution for 5 of the 10 problems. However, the gap between the two solutions is quite small for the other 5 instances. The largest difference, which occurs with  $n = 25$  and seed= 42, is only 2.4%.

Table 6.6 ARP instances with  $n = 25$ , a relaxed DD-width of 2048, and a 3 day runtime

*search* = width of embedded search, *lb* = lower bound, *ub* = upper bound,  
*gap* = optimality gap, *queue* = nodes in the processing queue

Seed	Search	<i>multi</i> = 1				<i>multi</i> = 3				<i>multi</i> = 5			
		lb	ub	gap (%)	queue	lb	ub	gap (%)	queue	lb	ub	gap (%)	queue
8	400	538.1	781.5	31.2	4	510.3	778.7	34.5	10	503.2	760.6	33.8	9
	1024	538.1	778.9	30.9	2	510.1	780.1	34.6	9	503.2	760.6	33.8	9
	2048	538.1	778.9	30.9	2	510.3	774.0	34.1	9	499.7	763.3	34.5	7
22	400	506.8	842.9	39.9	10	475.4	774.4	38.6	10	463.8	791.2	41.4	11
	1024	506.8	787.0	35.6	11	475.4	774.0	38.6	10	463.7	785.3	41.0	7
	2048	506.8	784.6	35.4	10	475.4	774.0	38.6	9	463.7	784.6	40.9	7
42	400	513.9	783.5	34.4	2	497.5	761.1	34.6	9	467.2	733.3	36.3	9
	1024	513.9	776.3	33.8	2	497.5	755.0	34.1	9	467.2	751.3	37.8	9
	2048	513.9	778.7	34.0	2	497.5	755.0	34.1	9	467.2	751.3	37.8	9
59	400	508.6	794.6	36.0	6	486.0	747.6	35.0	10	466.8	725.2	35.6	9
	1024	508.8	795.6	36.1	11	486.0	751.5	35.3	10	466.1	724.5	35.7	7
	2048	508.6	781.3	34.9	6	481.3	738.2	34.8	7	466.8	718.2	35.0	9
73	400	494.7	797.3	38.0	9	461.2	793.8	41.9	11	447.4	743.4	39.8	7
	1024	493.6	790.3	37.5	7	459.4	784.9	41.5	8	447.4	743.4	39.8	7
	2048	493.6	782.0	36.9	7	459.1	768.0	40.2	7	447.4	743.4	39.8	7

Table 6.7 ARP instances with  $n = 30$ , a relaxed DD-width of 2048, and a 3 day runtime

*search* = width of embedded search, *lb* = lower bound, *ub* = upper bound,  
*gap* = optimality gap, *queue* = nodes in the processing queue

Seed	Search	<i>multi</i> = 1				<i>multi</i> = 3				<i>multi</i> = 5			
		lb	ub	gap (%)	queue	lb	ub	gap (%)	queue	lb	ub	gap (%)	queue
8	400	523.5	955.5	45.2	4	486.9	906.9	46.3	7	471.8	904.2	47.8	7
	1024	523.5	935.7	44.1	4	486.9	894.8	45.6	7	464.1	898.7	48.4	6
	2048	523.5	935.7	44.1	4	486.9	892.2	45.4	7	464.1	898.7	48.4	6
22	400	608.8	901.8	32.5	5	597.5	896.2	33.3	8	503.7	908.5	44.6	7
	1024	608.8	896.7	32.1	5	593.5	889.6	33.3	7	503.7	905.6	44.4	7
	2048	608.8	897.9	32.2	5	597.5	900.6	33.6	6	503.5	892.8	43.6	5
42	400	515.9	906.6	43.1	9	486.7	880.1	44.7	7	465.7	858.6	45.8	6
	1024	513.7	911.8	43.7	8	486.7	873.1	44.3	7	465.7	854.8	45.5	6
	2048	512.9	861.1	40.4	7	486.7	871.3	44.1	7	465.7	835.1	44.2	6
59	400	552.0	923.7	40.2	6	537.4	906.2	40.7	7	449.1	872.7	48.5	3
	1024	552.0	910.6	39.4	6	535.7	858.1	37.6	6	449.1	864.5	48.1	3
	2048	552.0	894.8	38.3	6	534.6	868.3	38.4	2	449.1	864.1	48.0	3
73	400	504.1	896.6	43.8	5	495.1	903.9	45.2	4	480.2	904.3	46.9	5
	1024	504.1	975.2	48.3	5	495.1	932.8	46.9	3	480.2	906.3	47.0	5
	2048	504.1	945.1	46.7	5	495.1	908.5	45.5	2	478.5	882.9	45.8	2

Table 6.8 Best solutions for  $n \in \{25, 30\}$ 

n	seed	[77]*		Peel-and-Bound: Preferred Settings**		Peel-and-Bound: Best Found***	
		Average Value	Best Value	Value	Optimality Gap (%)	Value	Optimality Gap (%)
25	8	-	-	763.3	34.5	<b>760.6</b>	33.8
	22	-	-	784.6	40.9	<b>774.0</b>	38.6
	42	881.5	865.7	751.3	37.8	<b>733.3</b>	36.3
	59	-	-	718.2	35	718.2	35.0
	73	873.6	863.7	743.4	39.8	743.4	39.8
30	8	-	-	898.7	48.4	<b>892.2</b>	45.4
	22	-	-	892.8	43.6	892.8	43.6
	42	1084.6	1065.2	835.1	44.2	835.1	44.2
	59	-	-	864.1	48.0	<b>858.1</b>	37.6
	73	967.7	952.1	882.9	45.8	882.9	45.8

\* The results from [77] in the *Average Found* column are the average solution cost over several runs of the same stochastic algorithm. *Best Found* reports the best solution found in any run of the algorithm.

\*\* The preferred settings for Peel-and-Bound are:  $\omega_s = 2048$ ,  $\omega = 2048$ , *multi*= 5.

\*\*\* The bold values are the instances where the best found solution by any setting is better than the solution found using the preferred setting.

## 6.8 Opportunities for Parallel Computing

Consider the number of diagrams left in the queues from the results tables. Large values indicate that: (1) a significant amount of work remains to be done before closing those instances, and (2) that work is extremely parallelizable because each DD in the queue represents a sub-problem to be solved that is totally independent from the rest.

We ran all of our experiments on a single thread, but as long as there are unused threads available, each of those DDs could be assigned to a different thread without the need for them to communicate [7, 9, 44]. Thus, an implementation that included parallel processing could handle problems at a much larger scale if many CPUs were available. Additionally, setting  $multi > 1$  is parallelizable, as it represents  $multi$  independent restarts of SQSLP at disjoint areas of the search space.

## 6.9 Conclusions on the Framework for Outer/Inner Optimization Problems

In this chapter, we study outer-inner optimization problems, where the outer problem is combinatorial and the inner problem is numerical and black-box. We introduce the first optimization framework designed to find exact solutions for such problems under mild assumptions about the inner problem. Global trajectory optimization is one real-world example of such problems, and we use the Asteroid Routing Problem (ARP) as a case study. Although the inner optimizer in the ARP returns a local optimum, we show how to control the likelihood that this local optimum is also global. From the perspective of the outer problem, our proposed method successfully solves instances of up to 15 celestial bodies, and finds high-quality solutions for larger problems. Notably, we find new best-known solutions for several instances, many of them likely to be optimal. Additionally, we have made our implementation, data, and a detailed guide on how to use the solver, available in a public repository.

In the domain of global trajectory optimization, our methods represent a pioneering approach not only for finding optimal solutions but also for discovering high-quality feasible solutions. This work opens the door to future research that could adapt our solver to tackle other complex challenges, such as those involving multiple gravity assists. More broadly, we provide a robust framework for addressing sequencing problems where the cost function is computationally demanding. The proposed approach is highly scalable and future work should explore how to set its parameters according to the number of CPUs available and problem size. The principles underlying our approach are versatile, promising applicability to a diverse range of problems in the future.

## 6.10 Acknowledgements

We received advice about global trajectory optimization from the Advanced Concepts Team at the European Space Agency. We would like to specifically thank Dario Izzo and Emmanuel Blazquez for their input on this project.



## CHAPTER 7 CONCLUSION

When I began the work in this thesis, I had just familiarized myself with DD-based BnB, and I was entranced by the idea. However, it left me wondering: Why do they throw away the DDs at every iteration? That is the question that led to PnB, an algorithm that can trade memory for efficiency. The goal when I started implementing my ideas was to build on the BnB work to prove that a DD-based solver could compete with a traditional cutting edge solver. Around the same time, ddo and Haddock were in pursuit of the same goal. There has since been an enormous increase in the number of researchers using DDs, applying them to a variety of topics, and improving the theory underlying them. However, DD-based solvers are still in their infancy compared to modern MIP and CP solvers which have had decades of refinement. There are so many unanswered questions, and so many simple ideas that have not been explored yet with DD-solvers.

### 7.1 Some Ideas for Future Research

#### 7.1.1 Implicit Relaxed DDs Everywhere

The largest single performance boost that I encountered while improving the PnB implementation was the switch to implicit relaxed DDs (Chapter 5). Following that, I gleefully deleted all of the code responsible for managing arc labeling, arc storage, and arc weighting. This is an idea that is easy to test in other applications. It merely requires making the shift from thinking about labeled arcs to thinking about labeled nodes. This combination of being both easy to implement and highly efficient makes it a great target for incorporation with other solvers and applications. There are also unexplored questions about what constraint propagation techniques become available, or worthwhile, when all the nodes in a relaxed DD have an exact label.

#### 7.1.2 More Decision Diagrams by Separation

Constructing relaxed DDs by separation has some huge advantages over using top-down compilation. Aside from the advantages laid out in this thesis pertaining to PnB, using separation allows you to mirror your constraint propagators, one working top-down, the other bottom-up. Separation also allows you to start from a state where all equivalent nodes have already been merged, making it easier to generate compact representations. However, there has been significantly less work leveraging separation, apparently due to its perceived

complexity.

Some notable directions for future research:

1. There seems to be a competition among DD researchers to define a modeling language for using DDs, and then get everyone else to use it. However, to the best of my knowledge, all of them account for node merging, and only one of them accounts for node splitting (HADDOCK). More work on generic modeling for DDs that incorporates node splitting would be worthwhile.
2. Is there a dynamic method of splitting nodes in a relaxed DD that is more efficient than top-down or bottom-up? I have been unable to find one thus far, but it seems likely that such a method exists. The challenge to overcome is that if you begin with the first layer and proceed top-down, then it is computationally inexpensive to keep all of the state information on the nodes in the layers above and below the splits up to date. However, if instead the nodes are split dynamically, with a new layer being selected for each node split, then many nodes have to be updated to leverage the information gained by the split.
3. Can separation and merging be combined into something better than the sum of its parts? In Section 4.2.3, we explored the idea of a version of PnB that uses top-down compilation at each iteration. This would split at some parts of the algorithm, and merge at others. Whether or not that specific idea is useful, an exploration of techniques that leverage both separation and top down compilation could yield new algorithms that benefit both compilation techniques.
4. Separation provides such a perfect template for rewriting CP constraints as DD-based constraints. There has been work in this area [2, 84, 85], but there is room for substantially more.

### 7.1.3 Exact Solutions to Inner/Outer Optimization Problems

In Chapter 6, we provided a method of finding exact solutions to a problem that was generally thought not to admit them. This is part of a broader class of problems that have an outer combinatorial optimization problem combined with an inner computationally expensive black-box problem. We showed that for the Asteroid Routing Problem, the black-box could be modified to create a version of the black-box capable of generating valid bounds on the weights of arcs in a relaxed DD. This opens up a huge opportunity to apply this concept to other optimization problems with expensive black-box functions.

## 7.2 Final Thoughts

This thesis includes a tutorial to get up to speed with decision diagrams, followed by a thorough discussion of peel-and-bound. In the papers I drew from, we demonstrated that peel-and-bound can achieve cutting edge results even though, like the other DD-based solvers, it is still in its infancy. This thesis serves as a demonstration of how powerful a DD-based solver that leverages separation can be, and calls for more work in this area. It also introduced implicit relaxed DDs, which are not only highly efficient, but will also help make separation easier to implement, which will hopefully broaden their appeal.

The final demonstration in this thesis is the application of peel-and-bound to find exact solutions to global trajectory optimization problems. The framework described has the potential to make an impact not just on trajectory-related research, but also on other areas where the outer combinatorial optimization problem requires solving an expensive inner optimization problem.

This thesis not only advances the understanding and application of decision diagrams through the introduction of innovative methods like peel-and-bound and implicit relaxed DDs, but also sets the stage for future breakthroughs in complex optimization challenges across various fields.

## REFERENCES

- [1] D. Bergman, A. Cire, W.-J. van Hoeve, and J. Hooker, *Decision Diagrams for Optimization*. Springer International Publishing, 01 2016.
- [2] M. P. Castro, A. A. Cire, and J. C. Beck, “Decision diagrams for discrete optimization: A survey of recent advances,” *INFORMS Journal on Computing*, vol. 34, no. 4, pp. 2271–2295, 2022.
- [3] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, “Julia: A fresh approach to numerical computing,” *SIAM review*, vol. 59, no. 1, pp. 65–98, 2017. [Online]. Available: <https://doi.org/10.1137/141000671>
- [4] X. Gillard, V. Coppé, P. Schaus, and A. A. Cire, “Improving the filtering of branch-and-bound mdd solver,” in *Integration of Constraint Programming, Artificial Intelligence, and Operations Research, 18th International Conference, CPAIOR 2021*, ser. Lecture Notes in Computer Science. Springer International Publishing, 2021.
- [5] X. Gillard, “Discrete optimization with decision diagrams: design of a generic solver, improved bounding techniques, and discovery of good feasible solutions with large neighborhood search,” Ph.D. dissertation, UCL-Université Catholique de Louvain, 2022.
- [6] I. Rudich, Q. Cappart, and L.-M. Rousseau, “Peel-and-bound: Generating stronger relaxed bounds with multivalued decision diagrams,” in *International Conference on Principles and Practice of Constraint Programming*, 2022.
- [7] D. Bergman, A. A. Cire, A. Sabharwal, H. Samulowitz, V. Saraswat, and W.-J. van Hoeve, “Parallel combinatorial optimization with decision diagrams,” in *Proceedings of the International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 2014, pp. 351–367.
- [8] D. Bergman, A. Cire, W.-J. van Hoeve, and J. Hooker, “Discrete optimization with decision diagrams,” *INFORMS Journal on Computing*, vol. 28, pp. 47–66, 02 2016.
- [9] I. Rudich, Q. Cappart, and L.-M. Rousseau, “Improved peel-and-bound: Methods for generating dual bounds with multivalued decision diagrams,” *Journal of Artificial Intelligence Research*, vol. 77, pp. 1489–1538, 2023.

- [10] H. Andersen, T. Hadzic, J. Hooker, and P. Tiedemann, “A constraint store based on multivalued decision diagrams,” in *Bessière, C. (eds) Principles and Practice of Constraint Programming – CP 2007*, ser. Lecture Notes in Computer Science, vol. 4741, 09 2007, pp. 118–132.
- [11] T. Hadzic, J. Hooker, B. O’Sullivan, and P. Tiedemann, “Approximate compilation of constraints into multivalued decision diagrams,” in *Stuckey, P.J. (eds) Principles and Practice of Constraint Programming. CP 2008*, ser. Lecture Notes in Computer Science, 09 2008, pp. 448–462.
- [12] S. Hoda, W.-J. van Hoeve, and J. Hooker, “A systematic approach to mdd-based constraint programming,” in *Cohen, D. (eds) Principles and Practice of Constraint Programming – CP 2010. CP 2010*, ser. Lecture Notes in Computer Science, vol. 6308, 09 2010, pp. 266–280.
- [13] T. Hadzic and J. Hooker, “Discrete global optimization with binary decision diagrams,” in *Workshop on Global Optimization: Integrating Convexity, Optimization, Logic Programming, and Computational Algebraic Geometry (GICOLAG). Vienna, 2006*.
- [14] T. Hadzic and J. Hooker, “Postoptimality analysis for integer programming using binary decision diagrams,” in *GICOLAG Workshop (Global Optimization: Integrating Convexity, Optimization, Logic Programming, and Computational Algebraic Geometry), Vienna. Technical report, Carnegie Mellon University, 2006*.
- [15] J. Hooker, “Decision diagrams and dynamic programming,” in *Gomes, C., Sellmann, M. (eds) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. CPAIOR 2013*, ser. Lecture Notes in Computer Science, vol. 7874, 05 2013.
- [16] H. M. Salkin and C. A. De Kluyver, “The knapsack problem: a survey,” *Naval Research Logistics Quarterly*, vol. 22, no. 1, pp. 127–144, 1975.
- [17] L. F. Escudero, “An inexact algorithm for the sequential ordering problem,” *European Journal of Operational Research*, vol. 37, no. 2, pp. 236–249, 1988.
- [18] Bryant, “Graph-based algorithms for boolean function manipulation,” *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, 1986.
- [19] I. Wegener, *Branching programs and binary decision diagrams: theory and applications*. SIAM, 2000.

- [20] D. Bergman, W.-J. van Hoeve, and J. Hooker, “Manipulating mdd relaxations for combinatorial optimization,” in *Lecture Notes in Computer Science*, vol. 6697, 05 2011, pp. 20–35.
- [21] D. Bergman, A. A. Cire, W.-J. Van Hoeve, and J. N. Hooker, “Variable ordering for the application of bdds to the maximum independent set problem,” in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 9th International Conference, CPAIOR 2012, Nantes, France, May 28–June 1, 2012. Proceedings 9*. Springer, 2012, pp. 34–49.
- [22] D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. N. Hooker, “Optimization bounds from binary decision diagrams,” *INFORMS Journal on Computing*, vol. 26, no. 2, pp. 253–268, 2014.
- [23] X. Gillard and P. Schaus, “Large neighborhood search with decision diagrams,” in *International Joint Conference on Artificial Intelligence*, 2022.
- [24] V. Coppé, X. Gillard, and P. Schaus, “Decision diagram-based branch-and-bound with caching for dominance and suboptimality detection,” *INFORMS Journal on Computing*, 2024.
- [25] V. Coppé, “Advances in discrete optimization with decision diagrams: Dominance, caching and aggregation-based heuristics,” Ph.D. dissertation, Carnegie Mellon University, USA, 2024.
- [26] T. Hadzic, J. Hooker, B. O’Sullivan, and P. Tiedemann, “Approximate compilation of constraints into multivalued decision diagrams,” in *International Conference on Principles and Practice of Constraint Programming*. Springer, 2008, pp. 448–462.
- [27] T. Hadzic, J. N. Hooker, and P. Tiedemann, “Propagating separable equalities in an mdd store,” in *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*. Springer, 2008, pp. 318–322.
- [28] A. A. Cire and W.-J. van Hoeve, “Multivalued decision diagrams for sequencing problems,” *Operations Research*, vol. 61, no. 6, pp. 1259, 1462, 2013.
- [29] J.-C. Régin, “A filtering algorithm for constraints of difference in csps,” in *AAAI*, vol. 94, 1994, pp. 362–367.

- [30] X. Gillard, P. Schaus, and V. Coppé, “Ddo, a generic and efficient framework for mdd-based optimization,” in *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, 2021, pp. 5243–5245.
- [31] V. Coppé, X. Gillard, and P. Schaus, “Branch-and-bound with barrier: Dominance and suboptimality detection for dd-based branch-and-bound,” *ArXiv*, vol. abs/2211.13118, 2022.
- [32] Z. Tang and W.-J. van Hoeve, “Dual bounds from decision diagram-based route relaxations: An application to truck-drone routing,” *Transportation Science*, vol. 58, no. 1, pp. 257–278, 2024.
- [33] A. Karahalios and W.-J. van Hoeve, “Column elimination for capacitated vehicle routing problems,” in *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 2023, pp. 35–51.
- [34] W.-J. Hoeve, “Graph coloring with decision diagrams,” *Mathematical Programming*, vol. 192, pp. 631–674, 05 2021.
- [35] R. Gentzel, L. Michel, and W.-J. v. Hoeve, “Haddock: A language and architecture for decision diagram compilation,” in *International Conference on Principles and Practice of Constraint Programming*. Springer, 2020, pp. 531–547.
- [36] R. Gentzel, L. Michel, and W.-J. van Hoeve, “Optimization bounds from decision diagrams in haddock,” in *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 2023, pp. 150–166.
- [37] R. Kuroiwa and J. C. Beck, “Domain-independent dynamic programming: Generic state space search for combinatorial optimization,” in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 33, 2023, pp. 236–244.
- [38] F. A. Aloul, I. L. Markov, and K. A. Sakallah, “Force: a fast and easy-to-implement variable-ordering heuristic,” in *Proceedings of the 13th ACM Great Lakes symposium on VLSI*, 2003, pp. 116–119.
- [39] B. Bollig and I. Wegener, “Improving the variable ordering of obdds is np-complete,” *IEEE Transactions on computers*, vol. 45, no. 9, pp. 993–1002, 1996.
- [40] M. Rice and S. Kulhari, “A survey of static variable ordering heuristics for efficient bdd/mdd construction,” *University of California, Tech. Rep*, p. 130, 2008.

- [41] R. Rudell, “Dynamic variable ordering for ordered binary decision diagrams,” in *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*. IEEE, 1993, pp. 42–47.
- [42] A. Parjadis, Q. Cappart, L.-M. Rousseau, and D. Bergman, “Improving branch-and-bound using decision diagrams and reinforcement learning,” in *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 2021, pp. 446–455.
- [43] A. Karahalios and W.-J. Hoeve, “Variable ordering for decision diagrams: A portfolio approach,” *Constraints*, vol. 27, no. 1-2, pp. 1–18, 01 2022.
- [44] G. Perez and J.-C. Régin, “Parallel algorithms for operations on multi-valued decision diagrams,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018.
- [45] P. Baptiste, C. Le Pape, and W. Nuijten, *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*, ser. International Series in Operations Research and Management Science, Kluwer. Springer International Publishing, 2001.
- [46] G. Reinelt, “TspLib. a traveling salesman problem library,” *INFORMS Journal on Computing*, vol. 3, pp. 376–384, 11 1991.
- [47] J. N. Hooker, “Job sequencing bounds from decision diagrams,” in *Beck, J. (eds) Principles and Practice of Constraint Programming. CP 2017*, ser. Lecture Notes in Computer Science, 08 2017, pp. 565–578.
- [48] J. Hooker, “Improved job sequencing bounds from decision diagrams,” in *Schiex, T., de Givry, S. (eds) Principles and Practice of Constraint Programming. CP 2019*, ser. Lecture Notes in Computer Science, vol. 11802, 09 2019, pp. 268–283.
- [49] M. López-Ibáñez and C. Blum, “Benchmark instances for the travelling salesman problem with time windows (tsptw),” <https://lopez-ibanez.eu/tsptw-instances>, Nov 2022, accessed: 2022-01-22. [Online]. Available: <https://lopez-ibanez.eu/tsptw-instances>
- [50] N. Ascheuer, “Hamiltonian path problems in the on-line optimization of flexible manufacturing systems,” Ph.D. dissertation, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1996.
- [51] Y. Dumas, J. Desrosiers, E. Gelinas, and M. M. Solomon, “An optimal algorithm for the traveling salesman problem with time windows,” *Operations research*, vol. 43, no. 2, pp. 367–371, 1995.



- [52] M. Gendreau, A. Hertz, G. Laporte, and M. Stan, “A generalized insertion heuristic for the traveling salesman problem with time windows,” *Operations Research*, vol. 46, no. 3, pp. 330–335, 1998.
- [53] A. Langevin, M. Desrochers, J. Desrosiers, S. G  linas, and F. Soumis, “A two-commodity flow formulation for the traveling salesman and the makespan problems with time windows,” *Networks*, vol. 23, no. 7, pp. 631–640, 1993.
- [54] J. W. Ohlmann and B. W. Thomas, “A compressed-annealing heuristic for the traveling salesman problem with time windows,” *INFORMS Journal on Computing*, vol. 19, no. 1, pp. 80–90, 2007.
- [55] G. Pesant, M. Gendreau, J.-Y. Potvin, and J.-M. Rousseau, “An exact constraint logic programming algorithm for the traveling salesman problem with time windows,” *Transportation Science*, vol. 32, no. 1, pp. 12–29, 1998.
- [56] J.-Y. Potvin and S. Bengio, “The vehicle routing problem with time windows part ii: genetic search,” *INFORMS journal on Computing*, vol. 8, no. 2, pp. 165–172, 1996.
- [57] R. Baldacci, A. Mingozzi, and R. Roberti, “New state-space relaxations for solving the traveling salesman problem with time windows,” *INFORMS Journal on Computing*, vol. 24, no. 3, pp. 356–371, 2012.
- [58] D. Izzo, I. Getzner, D. Hennes, and L. F. Sim  es, “Evolving solutions to tsp variants for active space debris removal,” in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, 2015, pp. 1207–1214.
- [59] M. Gendreau, G. Ghiani, and E. Guerriero, “Time-dependent routing problems: A review,” *Computers & operations research*, vol. 64, pp. 189–197, 2015.
- [60] T. Adamo, M. Gendreau, G. Ghiani, and E. Guerriero, “A review of recent advances in time-dependent vehicle routing,” *European Journal of Operational Research*, 2024.
- [61] J. F. Ehmke, A. M. Campbell, and B. W. Thomas, “Vehicle routing to minimize time-dependent emissions in urban areas,” *European Journal of Operational Research*, vol. 251, no. 2, pp. 478–494, 2016.
- [62] H. Andersson, K. Fagerholt, and K. Hobbessland, “Integrated maritime fleet deployment and speed optimization: Case study from ro-ro shipping,” *Computers & Operations Research*, vol. 55, pp. 233–240, 2015.

- [63] A. Shirazi, J. Ceberio, and J. A. Lozano, “Spacecraft trajectory optimization: A review of models, objectives, approaches and solutions,” *Progress in Aerospace Sciences*, vol. 102, pp. 76–98, 2018.
- [64] D. Izzo, L. F. Simões, M. Mörtens, G. C. de Croon, A. Heritier, and C. H. Yam, “Search for a grand tour of the jupiter galilean moons,” in *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, 2013, pp. 1301–1308.
- [65] O. Abdelkhalik and A. Gad, “Dynamic-size multiple populations genetic algorithm for multigravity-assist trajectory optimization,” *Journal of Guidance, Control, and Dynamics*, vol. 35, no. 2, pp. 520–529, 2012.
- [66] M. Ceriotti and M. Vasile, “Automated multigravity assist trajectory planning with a modified ant colony algorithm,” *Journal of Aerospace Computing, Information, and Communication*, vol. 7, no. 9, pp. 261–293, 2010.
- [67] D. Izzo, V. Becerra, D. Myatt, S. Nasuto, and J. Bishop, “Search space pruning and global optimisation of multiple gravity assist spacecraft trajectories,” *Journal of Global Optimization*, vol. 38, pp. 283–296, 2007.
- [68] M. Vasile and P. De Pascale, “Preliminary design of multiple gravity-assist trajectories,” *Journal of Spacecraft and Rockets*, vol. 43, no. 4, pp. 794–805, 2006.
- [69] T. Lee, M. Leok, and N. H. McClamroch, “A combinatorial optimal control problem for spacecraft formation reconfiguration,” in *2007 46th IEEE Conference on Decision and Control*. IEEE, 2007, pp. 5370–5375.
- [70] A. E. Petropoulos, E. P. Bonfiglio, D. J. Grebow, T. Lam, J. S. Parker, J. Arrieta, D. F. Landau, R. L. Anderson, E. D. Gustafson, G. J. Whiffen *et al.*, “Gtoc5: results from the jet propulsion laboratory,” *Acta Futura*, vol. 8, no. 1, pp. 21–27, 2014.
- [71] D. Hennes and D. Izzo, “Interplanetary trajectory planning with monte carlo tree search,” in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.
- [72] L. F. Simões, D. Izzo, E. Haasdijk, and A. Eiben, “Multi-rendezvous spacecraft trajectory optimization with beam p-aco,” in *Evolutionary Computation in Combinatorial Optimization: 17th European Conference, EvoCOP 2017, Amsterdam, The Netherlands, April 19-21, 2017, Proceedings 17*. Springer, 2017, pp. 141–156.

- [73] M. Zaefferer, J. Stork, M. Frieze, A. Fischbach, B. Naujoks, and T. Bartz-Beielstein, “Efficient global optimization for combinatorial problems,” in *Proceedings of the 2014 annual conference on genetic and evolutionary computation*, 2014, pp. 871–878.
- [74] E. Irurozki and M. López-Ibáñez, “Unbalanced mallows models for optimizing expensive black-box permutation problems,” in *Proceedings of the genetic and evolutionary computation conference*, 2021, pp. 225–233.
- [75] V. Santucci and M. Baiocchi, “A fast randomized local search for low budget optimization in black-box permutation problems,” in *2022 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2022, pp. 1–8.
- [76] F. Chicano, B. Derbel, and S. Verel, “Fourier transform-based surrogates for permutation problems,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2023, pp. 275–283.
- [77] M. López-Ibáñez, F. Chicano, and R. Gil-Merino, “The asteroid routing problem: A benchmark for expensive black-box permutation optimization,” in *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*. Springer, 2022, pp. 124–140.
- [78] D. Izzo, “Revisiting Lambert’s problem,” *Celestial Mechanics and Dynamical Astronomy*, vol. 121, pp. 1–15, 2015.
- [79] D. Kraft, “A software package for sequential quadratic programming,” *Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt*, 1988.
- [80] L. F. Simões, D. Izzo, E. Haasdijk, and A. Eiben, “Multi-rendezvous spacecraft trajectory optimization with beam p-aco,” in *Evolutionary Computation in Combinatorial Optimization: 17th European Conference, EvoCOP 2017, Amsterdam, The Netherlands, April 19-21, 2017, Proceedings 17*. Springer, 2017, pp. 141–156.
- [81] D. Hennes, D. Izzo, and D. Landau, “Fast approximators for optimal low-thrust hops between main belt asteroids,” in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2016, pp. 1–7.
- [82] J. Nocedal and S. Wright, “Numerical optimization. 2nd edn springer,” *New York*, 2006.
- [83] N. Hansen and A. Ostermeier, “Completely derandomized self-adaptation in evolution strategies,” *Evolutionary computation*, vol. 9, no. 2, pp. 159–195, 2001.

- [84] J. Kinable, A. A. Cire, and W.-J. van Hoeve, “Hybrid optimization methods for time-dependent sequencing problems,” *European Journal of Operational Research*, vol. 259, no. 3, pp. 887–897, 2017.
- [85] L. C. Riascos-Álvarez, M. Bodur, and D. M. Aleman, “A branch-and-price algorithm enhanced by decision diagrams for the kidney exchange problem,” *Manufacturing & Service Operations Management*, vol. 26, no. 2, pp. 485–499, 2024.

# APPENDIX A EXPERIMENTAL DATA

Table A.1 Comparison Data for  $\omega = 64$  Experiments on SOP

RB = Relaxed Bound, BS = Best Solution, T = Time in Seconds, OG = Optimality Gap, QL = Queue Length. Full time series data is available in the GitHub repository.

Problem Name	Info n	BnB: width 64					PnB: width 64					Percent Improvements				
		RB	BS	T	OG	QL	RB	BS	T	OG	QL	RB	BS	T	OG	QL
ESC07	9	2,125	2,125	0.03	0%	-	2,125	2,125	0.07	0%	-			-57%		
ESC11	13	2,075	2,075	0.65	0%	-	2,075	2,075	0.42	0%	-			55%		
ESC12	14	1,675	1,675	1.99	0%	-	1,675	1,675	0.64	0%	-			211%		
ESC25	27	1,681	1,681	956	0%	-	1,681	1,681	353	0%	-			171%		
ESC47	49	334	1,542		78%	8,842	368	1,676		78%	1,295	10.2%	-8.0%		0.4%	583%
ESC63	65	8	62		87%	2,756	44	62		29%	15	450.0%	0.0%		200.0%	18273%
ESC78	80	2,230	19,800		89%	1,040	5,000	20,045		75%	316	124.2%	-1.2%		18.2%	229%
br17.10	18	55	55	260	0%	-	55	55	5	0%	-			4652%		
br17.12	18	55	55	138	0%	-	55	55	21	0%	-			546%		
ft53.1	54	1,785	8,478		79%	8,841	3,324	8,244		60%	917	86.2%	2.8%		32.3%	864%
ft53.2	54	1,946	8,927		78%	7,356	3,450	8,633		60%	938	77.3%	3.4%		30.3%	684%
ft53.3	54	2,546	12,179		79%	5,594	4,234	12,327		66%	1,147	66.3%	-1.2%		20.5%	388%
ft53.4	54	3,780	14,811		74%	11,907	6,500	14,753		56%	2,372	72.0%	0.4%		33.1%	402%
ft70.1	71	25,444	41,926		39%	4,781	31,123	41,607		25%	412	22.3%	0.8%		56.0%	1060%
ft70.2	71	25,239	42,805		41%	3,998	31,195	42,623		27%	427	23.6%	0.4%		53.1%	836%
ft70.3	71	25,810	48,073		46%	4,036	31,872	47,491		33%	475	23.5%	1.2%		40.8%	750%
ft70.4	71	28,593	56,644		50%	8,642	35,974	56,552		36%	1,087	25.8%	0.2%		36.1%	695%
krol24p.1	101	10,773	46,158		77%	2,173	17,579	46,158		62%	105	63.2%	0.0%		23.8%	1970%
krol24p.2	101	11,061	46,930		76%	1,898	17,633	46,930		62%	109	59.4%	0.0%		22.4%	1641%
krol24p.3	101	12,110	55,991		78%	1,055	18,586	55,991		67%	117	53.5%	0.0%		17.3%	802%
krol24p.4	101	13,838	85,533		84%	2,990	24,388	85,316		71%	244	76.2%	0.3%		17.4%	1125%
p43.1	44	630	29,450		98%	12,945	380	29,380		99%	1,022	-39.7%	0.2%		-0.9%	1167%
p43.2	44	440	29,000		98%	8,519	420	29,080		99%	1,125	-4.5%	-0.3%		-0.1%	657%
p43.3	44	595	29,530		98%	12,802	490	29,530		98%	1,122	-17.6%	0.0%		-0.4%	1041%
p43.4	44	1,370	83,855		98%	21,105	1,050	83,890		99%	4,694	-23.4%	0.0%		-0.4%	350%
prob.42	42	99	289		66%	16,742	97	286		66%	2,613	-2.0%	1.0%		-0.5%	541%
prob.100	100	170	1,841		91%	1,731	182	1,760		90%	117	7.1%	4.6%		1.2%	1379%
rbg048a	50	76	379		80%	12,938	47	380		88%	1,551	-38.2%	-0.3%		-8.8%	734%
rbg050c	52	63	566		89%	11,480	154	512		70%	1,481	144.4%	10.5%		27.1%	675%
rbg109a	111	91	1,196		92%	2,773	379	1,196		68%	612	316.5%	0.0%		35.3%	353%
rbg150a	152	63	1,874		97%	241	565	1,865		70%	222	796.8%	0.5%		38.6%	9%
rbg174a	176	119	2,157		94%	809	626	2,156		71%	117	426.1%	0.0%		33.1%	591%
rbg253a	255	113	3,181		96%	403	708	3,180		78%	39	526.5%	0.0%		24.1%	933%
rbg323a	325	89	3,519		97%	437	289	3,529		92%	17	224.7%	-0.3%		6.2%	2471%
rbg341a	343	68	3,038		98%	366	321	3,064		90%	8	372.1%	-0.8%		9.2%	4475%
rbg358a	360	69	3,359		98%	289	73	3,373		98%	6	5.8%	-0.4%		0.1%	4717%
rbg378a	380	52	3,429		98%	266	50	3,429		99%	5	-3.8%	0.0%		-0.1%	5220%
ry48p.1	49	5,201	17,555		70%	10,480	6,171	17,454		65%	1,395	18.7%	0.6%		8.9%	651%
ry48p.2	49	5,291	18,046		71%	9,286	6,577	17,840		63%	1,445	24.3%	1.2%		12.0%	543%
ry48p.3	49	6,207	21,161		71%	9,039	6,985	20,962		67%	1,707	12.5%	0.9%		6.0%	430%
ry48p.4	49	13,610	34,517		61%	15,819	14,293	33,804		58%	3,217	5.0%	2.1%		4.9%	392%

Table A.2 Comparison Data for  $\omega = 256$  Experiments on SOP

RB = Relaxed Bound, BS = Best Solution, T = Time in Seconds, OG = Optimality Gap, QL = Queue Length. Full time series data is available in the GitHub repository.

Problem Name	Info n	BnB: width 256					PnB: width 256					Percent Improvements				
		RB	BS	T	OG	QL	RB	BS	T	OG	QL	RB	BS	T	OG	QL
ESC07	9	2,125	2,125	0.04	0%	-	2,125	2,125	0.04	0%	-			0%		
ESC11	13	2,075	2,075	0.48	0%	-	2,075	2,075	0.41	0%	-			17%		
ESC12	14	1,675	1,675	1.66	0%	-	1,675	1,675	0.34	0%	-			388%		
ESC25	27	1,681	1,681	2,643	0%	-	1,681	1,681	303	0%	-			771%		
ESC47	49	312	1,590		80%	720	658	1,339		51%	740	110.9%	18.7%		58.0%	-3%
ESC63	65	9	62		85%	53	44	62		29%	3	388.9%	0.0%		194.4%	1667%
ESC78	80	2,230	20,345		89%	59	5,600	20,135		72%	109	151.1%	1.0%		23.3%	-46%
br17.10	18	55	55	275	0%	-	55	55	3	0%	-			9468%		
br17.12	18	55	55	105	0%	-	55	55	5	0%	-			2146%		
ft53.1	54	1,708	8,424		80%	760	4,603	8,244		44%	271	169.5%	2.2%		80.5%	180%
ft53.2	54	1,856	9,059		80%	632	3,555	8,648		59%	272	91.5%	4.8%		35.0%	132%
ft53.3	54	2,493	12,598		80%	477	4,852	11,095		56%	390	94.6%	13.5%		42.6%	22%
ft53.4	54	3,619	14,867		76%	1,240	7,560	14,611		48%	797	108.9%	1.8%		56.8%	56%
ft70.1	71	25,507	41,686		39%	373	31,122	41,235		25%	108	22.0%	1.1%		58.3%	245%
ft70.2	71	25,261	42,901		41%	297	31,630	42,182		25%	123	25.2%	1.7%		64.4%	141%
ft70.3	71	25,891	47,806		46%	377	32,539	46,488		30%	151	25.7%	2.8%		52.8%	150%
ft70.4	71	31,186	56,366		45%	958	37,984	56,366		33%	356	21.8%	0.0%		37.0%	169%
kro124p.1	101	10,683	48,866		78%	152	19,224	45,643		58%	43	79.9%	7.1%		35.0%	253%
kro124p.2	101	10,706	52,038		79%	125	19,299	48,102		60%	43	80.3%	8.2%		32.6%	191%
kro124p.3	101	12,078	58,562		79%	64	20,145	57,358		65%	45	66.8%	2.1%		22.3%	42%
kro124p.4	101	14,511	82,672		82%	281	25,002	82,364		70%	102	72.3%	0.4%		18.4%	175%
p43.1	44	610	29,460		98%	1,033	27,255	28,635		5%	146	4368%	2.9%		1932%	608%
p43.2	44	460	29,020		98%	547	27,455	29,020		5%	391	5868%	0.0%		1725%	40%
p43.3	44	750	29,530		97%	1,016	27,780	29,530		6%	764	3604%	0.0%		1545%	33%
p43.4	44	1,425	83,880		98%	1,365	28,195	83,435		66%	1,380	1879%	0.5%		48.5%	-1%
prob.42	42	90	289		69%	1,166	103	275		63%	617	14.4%	5.1%		10.1%	89%
prob.100	100	157	1,886		92%	113	178	1,721		90%	45	13.4%	9.6%		2.3%	151%
rbg048a	50	80	389		79%	794	80	373		79%	534	0.0%	4.3%		1.1%	49%
rbg050c	52	62	583		89%	810	175	503		65%	442	182.3%	15.9%		37.0%	83%
rbg109a	111	89	1,181		92%	394	406	1,106		63%	204	356.2%	6.8%		46.1%	93%
rbg150a	152	115	1,845		94%	406	571	1,845		69%	100	396.5%	0.0%		35.8%	306%
rbg174a	176	362	2,172		83%	337	646	2,171		70%	57	78.5%	0.0%		18.6%	491%
rbg253a	255	359	3,177		89%	139	727	3,176		77%	22	102.5%	0.0%		15.0%	532%
rbg323a	325	99	3,476		97%	114	346	3,480		90%	14	249.5%	-0.1%		7.9%	714%
rbg341a	343	84	3,016		97%	120	340	3,016		89%	7	304.8%	0.0%		9.6%	1614%
rbg358a	360	88	3,280		97%	92	88	3,382		97%	5	0.0%	-3.0%		-0.1%	1740%
rbg378a	380	44	3,385		99%	35	53	3,385		98%	6	20.5%	0.0%		0.3%	483%
ry48p.1	49	5,470	17,464		69%	897	9,432	17,071		45%	377	72.4%	2.3%		53.5%	138%
ry48p.2	49	5,606	18,060		69%	834	6,615	17,627		62%	383	18.0%	2.5%		10.4%	118%
ry48p.3	49	6,558	21,142		69%	859	8,723	20,850		58%	513	33.0%	1.4%		18.6%	67%
ry48p.4	49	17,359	34,074		49%	1,557	17,322	33,773		49%	990	-0.2%	0.9%		0.7%	57%

Table A.3 Comparison of PnB at  $\omega = 2048$  over PnB at  $\omega = 256$  on SOP

RB = Relaxed Bound, BS = Best Solution, OG = Optimality Gap.

Problem Name	Info n	PnB: width 256			PnB: width 2048			Percent Improvements		
		RB	BS	OG	RB	BS	OG	RB	BS	OG
ESC47	49	658	1,339	51%	882	1,304	32%	34.0%	2.7%	57.2%
ESC63	65	44	62	29%	44	62	29%	0.0%	0.0%	0%
ESC78	80	5,600	20,135	72%	6,025	20,505	71%	7.6%	-1.8%	2.2%
ft53.1	54	4,603	8,244	44%	5,167	8,237	37%	12.3%	0.1%	18.5%
ft53.2	54	3,555	8,648	59%	4,910	8,598	43%	38.1%	0.6%	37.3%
ft53.3	54	4,852	11,095	56%	7,722	11,092	30%	59.2%	0.0%	85.2%
ft53.4	54	7,560	14,611	48%	7,466	14,618	49%	-1.2%	0.0%	-1.4%
ft70.1	71	31,122	41,235	25%	33,382	41,476	20%	7.3%	-0.6%	25.7%
ft70.2	71	31,630	42,182	25%	32,964	41,833	21%	4.2%	0.8%	18.0%
ft70.3	71	32,539	46,488	30%	34,366	46,001	25%	5.6%	1.1%	18.6%
ft70.4	71	37,984	56,366	33%	40,919	56,310	27%	7.7%	0.1%	19.3%
kro124p.1	101	19,224	45,643	58%	21,954	47,425	54%	14.2%	-3.8%	7.8%
kro124p.2	101	19,299	48,102	60%	22,746	49,571	54%	17.9%	-3.0%	10.7%
kro124p.3	101	20,145	57,358	65%	25,566	54,633	53%	26.9%	5.0%	21.9%
kro124p.4	101	25,002	82,364	70%	29,377	81,050	64%	17.5%	1.6%	9.2%
p43.1	44	27,255	28,635	5%	27,755	28,960	4%	1.8%	-1.1%	16%
p43.2	44	27,455	29,020	5%	27,725	29,000	4%	1.0%	0.1%	23%
p43.3	44	27,780	29,530	6%	27,755	29,530	6%	-0.1%	0.0%	-1%
p43.4	44	28,195	83,435	66%	28,680	83,020	65%	1.7%	0.5%	1.2%
prob.42	42	103	275	63%	152	261	42%	47.6%	5.4%	49.8%
prob.100	100	178	1,721	90%	220	1,735	87%	23.6%	-0.8%	2.7%
rbg048a	50	80	373	79%	93	367	75%	16.3%	1.6%	5.2%
rbg050c	52	175	503	65%	184	501	63%	5.1%	0.4%	3.1%
rbg109a	111	406	1,106	63%	453	1,126	60%	11.6%	-1.8%	5.9%
rbg150a	152	571	1,845	69%	672	1,841	63%	17.7%	0.2%	8.7%
rbg174a	176	646	2,171	70%	1,104	2,121	48%	70.9%	2.4%	46.5%
rbg253a	255	727	3,176	77%	1,186	3,101	62%	63.1%	2.4%	24.9%
rbg323a	325	346	3,480	90%	421	3,449	88%	21.7%	0.9%	2.6%
rbg341a	343	340	3,016	89%	329	2,965	89%	-3.2%	1.7%	-0.2%
rbg358a	360	88	3,382	97%	107	3,131	97%	21.6%	8.0%	0.8%
rbg378a	380	53	3,385	98%	74	3,338	98%	39.6%	1.4%	0.7%
ry48p.1	49	9,432	17,071	45%	10,386	17,124	39%	10.1%	-0.3%	13.7%
ry48p.2	49	6,615	17,627	62%	7,896	17,461	55%	19.4%	1.0%	14.0%
ry48p.3	49	8,723	20,850	58%	10,558	20,686	49%	21.0%	0.8%	18.8%
ry48p.4	49	17,322	33,773	49%	24,248	32,953	26%	40.0%	2.5%	84.4%

Table A.4 (**Part 1 of 2**) TSP-TW Results of PnB at  $\omega = 2048$  on Open Problems: Seeded and Unseeded

LB = Lower Bound, UB = Upper Bound, OG = Optimality Gap.

Set	Problem Information		Baldacci et. al. (2012)	Single Thread: 2048			Seeded Single Thread: 2048		
	Name	Best Known Solution	LB	LB	UB	OG	LB	UB	OG
Dumas	n100w60.001.txt	655	NA	602	659	8.65	602	655	8.09
Dumas	n100w60.003.txt	744	NA	619	749	17.36	620	744	16.67
Dumas	n100w60.004.txt	764	NA	657	-	100.00	656	764	14.14
Dumas	n150w40.001.txt	918	NA	810	919	11.86	812	918	11.55
Dumas	n150w40.002.txt	941	NA	804	945	14.92	803	941	14.67
Dumas	n150w40.003.txt	727	NA	528	737	28.36	525	727	27.79
Dumas	n150w40.004.txt	764	NA	678	782	13.30	678	764	11.26
Dumas	n150w40.005.txt	824	NA	585	-	100.00	566	824	31.31
Dumas	n150w60.001.txt	859	NA	645	906	28.81	635	859	26.08
Dumas	n150w60.002.txt	782	NA	530	829	36.07	529	782	32.35
Dumas	n150w60.003.txt	793	NA	541	825	34.42	540	793	31.90
Dumas	n150w60.004.txt	819	NA	578	-	100.00	574	819	29.91
Dumas	n150w60.005.txt	840	NA	581	849	31.57	570	840	32.14
Dumas	n200w40.001.txt	1023	NA	643	1038	38.05	638	1023	37.63
Dumas	n200w40.002.txt	948	NA	605	983	38.45	605	948	36.18
Dumas	n200w40.003.txt	933	NA	645	-	100.00	568	933	39.12
Dumas	n200w40.004.txt	980	NA	731	1030	29.03	714	980	27.14
Dumas	n200w40.005.txt	1037	NA	659	1056	37.59	607	1037	41.47
Dumas	n60w100.005.txt	451	NA	390	460	15.22	391	451	13.3
Dumas	n80w80.002.txt	592	NA	508	605	16.03	506	592	14.53
Dumas	n80w80.003.txt	589	NA	500	598	16.39	500	589	15.11
Dumas	n80w80.004.txt	594	NA	506	620	18.39	505	594	14.98
OhlmannThomas	n150w120.001.txt	734	725.50	324	809	59.95	324	734	55.86
OhlmannThomas	n150w120.002.txt	677	668.40	209	717	70.85	209	677	69.13
OhlmannThomas	n150w120.003.txt	747	746.40	416	807	48.45	416	747	44.31
OhlmannThomas	n150w120.004.txt	763	761.60	381	835	54.37	381	763	50.07
OhlmannThomas	n150w120.005.txt	689	684.70	266	754	64.72	274	689	60.23
OhlmannThomas	n150w140.001.txt	762	754.00	394	893	55.88	394	762	48.29
OhlmannThomas	n150w140.002.txt	755	752.00	407	855	52.40	416	755	44.90
OhlmannThomas	n150w140.003.txt	613	608.50	215	738	70.87	215	613	64.93
OhlmannThomas	n150w140.005.txt	663	662.00	197	750	73.73	197	663	70.29
OhlmannThomas	n150w160.001.txt	706	701.40	293	777	62.29	293	706	58.50
OhlmannThomas	n150w160.002.txt	711	709.70	286	826	65.38	286	711	59.77
OhlmannThomas	n150w160.003.txt	608	603.20	170	772	77.98	170	608	72.04
OhlmannThomas	n150w160.004.txt	672	672.00	336	749	55.14	344	672	48.81
OhlmannThomas	n150w160.005.txt	658	655.00	320	736	56.52	320	658	51.37
OhlmannThomas	n200w120.001.txt	799	793.30	312	910	65.71	312	799	60.95
OhlmannThomas	n200w120.002.txt	721	713.90	184	822	77.62	184	721	74.48
OhlmannThomas	n200w120.003.txt	880	868.60	336	983	65.82	336	880	61.82
OhlmannThomas	n200w120.004.txt	777	775.80	291	887	67.19	291	777	62.55
OhlmannThomas	n200w120.005.txt	841	833.20	258	960	73.12	371	841	55.89
OhlmannThomas	n200w140.001.txt	834	826.20	177	1053	83.19	177	834	78.78
OhlmannThomas	n200w140.002.txt	760	756.20	180	895	79.89	180	760	76.32
OhlmannThomas	n200w140.003.txt	758	756.00	241	897	73.13	252	758	66.75
OhlmannThomas	n200w140.004.txt	816	807.10	284	932	69.53	284	816	65.20
OhlmannThomas	n200w140.005.txt	822	819.60	148	927	84.03	148	822	82.00
SolomonPesant	rc203.0	727.45	726.22	608.28	734.00	17.13	608.24	727.45	16.39
SolomonPesant	rc204.2	778.40	774.77	667.18	905.93	26.35	685.22	778.40	11.97



Table A.5 (**Part 2 of 2**) TSP-TW Results of PnB at  $\omega = 2048$  on Open Problems: Seeded and Unseeded

LB = Lower Bound, UB = Upper Bound, OG = Optimality Gap.

Set	Problem Information			Baldacci et. al. (2012) LB	Single Thread: 2048			Seeded Single Thread: 2048		
	Name	Best Known Solution			LB	UB	OG	LB	UB	OG
AFG	rbg049a.tw	10018		10006.10	9877	10034	1.56	9877	10018	1.41
AFG	rbg050b.tw	9863		9855.80	9711	9876	1.67	9713	9863	1.52
AFG	rbg050c.tw	10024		10020.40	9745	10074	3.27	9746	10024	2.77
AFG	rbg132.2.tw	8192		8188.20	6701	8316	19.42	6695	8192	18.27
AFG	rbg152.3.tw	9788		9785.20	7240	9982	27.47	7240	9788	26.03
AFG	rbg193.2.tw	12142		12136.30	9755	12441	21.59	9755	12142	19.66
GendreauDumas	n100w100.004.txt	684		680.58	484	738	34.42	485	684	29.09
GendreauDumas	n100w120.002.txt	540		535.68	279	581	51.98	279	540	48.33
GendreauDumas	n100w120.004.txt	663		660.01	383	713	46.28	383	663	42.23
GendreauDumas	n100w140.005.txt	509		504.42	346	571	39.40	346	509	32.02
GendreauDumas	n100w160.002.txt	532		528.94	287	601	52.25	286	532	46.24
GendreauDumas	n100w160.005.txt	586		585.41	327	642	49.07	327	586	44.20
GendreauDumas	n100w80.002.txt	668		666.00	542	681	20.41	549	668	17.81
GendreauDumas	n40w180.004.txt	354		352.58	280	359	22.01	283	354	20.06
GendreauDumas	n40w200.003.txt	339		322.05	305	340	10.29	305	339	10.03
GendreauDumas	n40w200.004.txt	301		300.10	232	301	22.92	229	301	23.92
GendreauDumas	n60w140.001.txt	423		421.31	331	425	22.12	332	423	21.51
GendreauDumas	n60w140.002.txt	462		461.08	394	469	15.99	400	462	13.42
GendreauDumas	n60w140.005.txt	460		455.40	334	462	27.71	335	460	27.17
GendreauDumas	n60w160.001.txt	560		556.08	457	572	20.10	457	560	18.39
GendreauDumas	n60w160.003.txt	434		432.70	212	464	54.31	212	434	51.15
GendreauDumas	n60w180.001.txt	411		407.30	234	460	49.13	234	411	43.07
GendreauDumas	n60w180.003.txt	445		440.00	314	458	31.44	314	445	29.44
GendreauDumas	n60w180.004.txt	456		455.54	272	490	44.49	272	456	40.35
GendreauDumas	n60w180.005.txt	395		388.68	274	409	33.01	274	395	30.63
GendreauDumas	n60w200.003.txt	455		444.08	319	471	32.27	321	455	29.45
GendreauDumas	n60w200.004.txt	431		429.71	343	441	22.22	347	431	19.49
GendreauDumas	n60w200.005.txt	427		425.29	292	449	34.97	292	427	31.62
GendreauDumas	n80w120.001.txt	498		497.50	362	514	29.57	362	498	27.31
GendreauDumas	n80w120.002.txt	577		576.42	415	587	29.30	416	577	27.90
GendreauDumas	n80w120.004.txt	501		491.48	301	509	40.86	301	501	39.92
GendreauDumas	n80w120.005.txt	591		586.86	415	594	30.13	415	591	29.78
GendreauDumas	n80w140.002.txt	470		469.06	346	528	34.47	347	470	26.17
GendreauDumas	n80w140.003.txt	580		574.78	297	636	53.30	295	580	49.14
GendreauDumas	n80w140.004.txt	423		420.31	262	449	41.65	262	423	38.06
GendreauDumas	n80w160.002.txt	549		547.45	197	638	69.12	197	549	64.12
GendreauDumas	n80w180.001.txt	551		550.45	373	560	33.39	373	551	32.30
GendreauDumas	n80w200.001.txt	490		486.57	150	555	72.97	150	490	69.39
GendreauDumas	n80w200.002.txt	488		487.51	270	572	52.80	271	488	44.47
GendreauDumas	n80w200.003.txt	464		462.14	194	495	60.81	194	464	58.19
GendreauDumas	n80w200.004.txt	526		521.27	291	565	48.50	292	526	44.49
GendreauDumas	n80w200.005.txt	439		438.12	235	484	51.45	235	439	46.47
SolomonPotvinBengio	rc_203.2.txt	784.16		781.64	659.63	784.16	15.88	658.22	784.16	16.06
SolomonPotvinBengio	rc_203.3.txt	817.53		810.46	735.91	823.44	10.63	756.34	817.53	7.48
SolomonPotvinBengio	rc_204.1.txt	878.64		872.62	791.45	889.53	11.03	794.15	878.64	9.62
SolomonPotvinBengio	rc_204.2.txt	662.16		650.94	525.69	664.52	20.89	526.72	662.16	20.45
SolomonPotvinBengio	rc_208.3.txt	634.44		622.48	549.17	670.56	18.10	550.15	634.44	13.29

Table A.6 (**Part 1 of 6**) TSP-TW Results of PnB at  $\omega = 2048$  on Closed Problems: Seeded and Unseeded

LB = Lower Bound, UB = Upper Bound, OG = Optimality Gap, O = Optimality Proved.

Problem Information			Baldacci et. al. (2012)	Single Thread: 2048				Seeded Single Thread: 2048			
Set	Name	Best Known Solution		LB	UB	OG	Time	LB	UB	OG	Time
AFG	rbg010a.tw	671	O	671	671	0	0.08	671	671	0	0.26
AFG	rbg016a.tw	938	O	938	938	0	0.22	938	938	0	0.28
AFG	rbg016b.tw	1304	O	1304	1304	0	5.62	1304	1304	0	4.43
AFG	rbg017.2.tw	852	O	852	852	0	9.08	852	852	0	4.71
AFG	rbg017.tw	893	O	893	893	0	3.92	893	893	0	2.96
AFG	rbg017a.tw	4296	O	4296	4296	0	0.28	4296	4296	0	0.22
AFG	rbg019a.tw	1262	O	1262	1262	0	0.21	1262	1262	0	0.19
AFG	rbg019b.tw	1866	O	1866	1866	0	12.96	1866	1866	0	5.67
AFG	rbg019c.tw	4536	O	4536	4536	0	2.91	4536	4536	0	3.64
AFG	rbg019d.tw	1356	O	1356	1356	0	3.66	1356	1356	0	0.37
AFG	rbg020a.tw	4689	O	4689	4689	0	0.22	4689	4689	0	0.23
AFG	rbg021.2.tw	4528	O	4528	4528	0	10.08	4528	4528	0	5.45
AFG	rbg021.3.tw	4528	O	4528	4528	0	15.74	4528	4528	0	7.80
AFG	rbg021.4.tw	4525	O	4525	4525	0	25.67	4525	4525	0	9.71
AFG	rbg021.5.tw	4515	O	4515	4515	0	25.13	4515	4515	0	12.88
AFG	rbg021.6.tw	4480	O	4480	4480	0	2458.01	4480	4480	0	2501.88
AFG	rbg021.7.tw	4479	O	1050	4480	76.56	-	1050	4479	76.56	-
AFG	rbg021.8.tw	4478	O	1042	4480	76.74	-	1041	4478	76.75	-
AFG	rbg021.9.tw	4478	O	1037	4480	76.85	-	1039	4478	76.8	-
AFG	rbg021.tw	4536	O	4536	4536	0	3.11	4536	4536	0	2.81
AFG	rbg027a.tw	5091	O	5091	5091	0	10.69	5091	5091	0	8.34
AFG	rbg031a.tw	1863	O	1863	1863	0	28.16	1863	1863	0	13.51
AFG	rbg033a.tw	2069	O	2069	2069	0	33.44	2069	2069	0	18.26
AFG	rbg034a.tw	2222	O	2222	2222	0	83.72	2222	2222	0	34.02
AFG	rbg035a.2.tw	2056	O	1837	2088	12.02	-	1838	2056	10.6	-
AFG	rbg035a.tw	2144	O	2144	2144	0	44.80	2144	2144	0	28.45
AFG	rbg038a.tw	2480	O	2480	2480	0	75.79	2480	2480	0	17.41
AFG	rbg040a.tw	2378	2376.9	2378	2378	0	57.1	2378	2378	0	36.71
AFG	rbg041a.tw	2598	O	2598	2598	0	109.23	2598	2598	0	81.37
AFG	rbg042a.tw	2772	O	2772	2772	0	154.46	2772	2772	0	88.88
AFG	rbg048a.tw	9383	O	9011	9419	4.33	-	9013	9383	3.94	-
AFG	rbg050a.tw	2953	O	2361	2990	21.04	-	2362	2953	20.01	-
AFG	rbg055a.tw	3761	O	3761	3761	0	405.98	3761	3761	0	121.22
AFG	rbg067a.tw	4625	O	4625	4625	0	634.69	4625	4625	0	158.26
AFG	rbg125a.tw	7936	O	7918	8051	1.65	-	7936	7936	0	2479.97
AFG	rbg132.tw	8468	O	8449	8530	0.95	-	8468	8468	0	549.16
AFG	rbg152.tw	10032	O	10024	10086	0.61	-	10032	10032	0	1011.24
AFG	rbg172a.tw	10950	O	10082	11172	9.76	-	10082	10950	7.93	-
AFG	rbg193.tw	12535	O	11994	12781	6.16	-	11958	12535	4.6	-
AFG	rbg201a.tw	12948	O	11528	13240	12.93	-	11753	12948	9.23	-
AFG	rbg233.2.tw	14495	O	11634	14827	21.54	-	11634	14495	19.74	-
AFG	rbg233.tw	14992	O	13355	15330	12.88	-	12988	14992	13.37	-
Dumas	n100w20.001.txt	738	NA	738	738	0	32.87	738	738	0	5.06
Dumas	n100w20.002.txt	715	NA	715	715	0	64.56	715	715	0	30.67
Dumas	n100w20.003.txt	762	NA	762	762	0	49.73	762	762	0	42.79
Dumas	n100w20.004.txt	799	NA	799	799	0	51.93	799	799	0	28.00
Dumas	n100w20.005.txt	774	NA	774	774	0	45.66	774	774	0	35.66
Dumas	n100w40.001.txt	770	NA	770	770	0	178.71	770	770	0	129.09
Dumas	n100w40.002.txt	653	NA	653	653	0	573.14	653	653	0	455.42
Dumas	n100w40.003.txt	736	NA	736	736	0	390.53	736	736	0	341.37
Dumas	n100w40.004.txt	651	NA	651	651	0	500.63	651	651	0	406.68
Dumas	n100w40.005.txt	699	NA	699	699	0	518.71	699	699	0	455.31
Dumas	n100w60.002.txt	659	NA	659	659	0	1348.85	659	659	0	1503.28
Dumas	n100w60.005.txt	661	NA	661	661	0	1027.05	661	661	0	1054.34
Dumas	n150w20.001.txt	925	NA	925	925	0	380.14	925	925	0	248.27
Dumas	n150w20.002.txt	864	NA	864	864	0	234.78	864	864	0	126.45
Dumas	n150w20.003.txt	834	NA	834	834	0	410.38	834	834	0	305.72
Dumas	n150w20.004.txt	873	NA	873	873	0	343.26	873	873	0	304.33
Dumas	n150w20.005.txt	846	NA	846	846	0	423.04	846	846	0	452.06
Dumas	n200w20.001.txt	1019	NA	1019	1019	0	2739.07	1019	1019	0	2898.58
Dumas	n200w20.002.txt	972	NA	971	1000	2.9	-	972	972	0	614.20
Dumas	n200w20.003.txt	1050	NA	1050	1050	0	1330.33	1050	1050	0	1523.48
Dumas	n200w20.004.txt	984	NA	984	984	0	2360.97	984	984	0	3345.67
Dumas	n200w20.005.txt	1020	NA	1020	1020	0	445.04	1020	1020	0	1059.73
Dumas	n20w100.001.txt	237	NA	237	237	0	7.62	237	237	0	4.32
Dumas	n20w100.002.txt	222	NA	222	222	0	10.21	222	222	0	4.97
Dumas	n20w100.003.txt	310	NA	310	310	0	8.85	310	310	0	4.35
Dumas	n20w100.004.txt	349	NA	349	349	0	2.56	349	349	0	0.70
Dumas	n20w100.005.txt	258	NA	258	258	0	10.40	258	258	0	6.06

Table A.7 (**Part 2 of 6**) TSP-TW Results of PnB at  $\omega = 2048$  on Closed Problems: Seeded and Unseeded

LB = Lower Bound, UB = Upper Bound, OG = Optimality Gap, O = Optimality Proved.

Set	Problem Information		Baldacci et. al. (2012)	Single Thread: 2048				Seeded Single Thread: 2048			
	Name	Best Known Solution	LB	LB	UB	OG	Time	LB	UB	OG	Time
Dumas	n20w20.001.txt	378	NA	378	378	0	0.06	378	378	0	0.18
Dumas	n20w20.002.txt	286	NA	286	286	0	0.08	286	286	0	0.18
Dumas	n20w20.003.txt	394	NA	394	394	0	0.07	394	394	0	0.18
Dumas	n20w20.004.txt	396	NA	396	396	0	0.06	396	396	0	0.18
Dumas	n20w20.005.txt	352	NA	352	352	0	0.07	352	352	0	0.26
Dumas	n20w40.001.txt	254	NA	254	254	0	0.12	254	254	0	0.31
Dumas	n20w40.002.txt	333	NA	333	333	0	0.07	333	333	0	0.24
Dumas	n20w40.003.txt	317	NA	317	317	0	0.07	317	317	0	0.19
Dumas	n20w40.004.txt	388	NA	388	388	0	0.07	388	388	0	0.20
Dumas	n20w40.005.txt	288	NA	288	288	0	0.08	288	288	0	0.20
Dumas	n20w60.001.txt	335	NA	335	335	0	2.22	335	335	0	1.48
Dumas	n20w60.002.txt	244	NA	244	244	0	0.56	244	244	0	0.21
Dumas	n20w60.003.txt	352	NA	352	352	0	0.10	352	352	0	0.22
Dumas	n20w60.004.txt	280	NA	280	280	0	7.50	280	280	0	3.67
Dumas	n20w60.005.txt	338	NA	338	338	0	0.62	338	338	0	0.22
Dumas	n20w80.001.txt	329	NA	329	329	0	2.00	329	329	0	1.37
Dumas	n20w80.002.txt	338	NA	338	338	0	1.27	338	338	0	1.34
Dumas	n20w80.003.txt	320	NA	320	320	0	0.74	320	320	0	0.41
Dumas	n20w80.004.txt	304	NA	304	304	0	1.99	304	304	0	0.90
Dumas	n20w80.005.txt	264	NA	264	264	0	8.19	264	264	0	4.24
Dumas	n40w100.001.txt	429	NA	429	429	0	67.70	429	429	0	68.84
Dumas	n40w100.002.txt	358	NA	358	358	0	84.97	358	358	0	73.49
Dumas	n40w100.003.txt	364	NA	364	364	0	71.79	364	364	0	50.69
Dumas	n40w100.004.txt	357	NA	357	357	0	66.54	357	357	0	58.87
Dumas	n40w100.005.txt	377	NA	377	377	0	82.67	377	377	0	52.66
Dumas	n40w20.001.txt	500	NA	500	500	0	0.51	500	500	0	0.41
Dumas	n40w20.002.txt	552	NA	552	552	0	2.08	552	552	0	0.54
Dumas	n40w20.003.txt	478	NA	478	478	0	1.11	478	478	0	0.34
Dumas	n40w20.004.txt	404	NA	404	404	0	1.19	404	404	0	0.34
Dumas	n40w20.005.txt	499	NA	499	499	0	0.09	499	499	0	0.29
Dumas	n40w40.001.txt	465	NA	465	465	0	2.41	465	465	0	0.61
Dumas	n40w40.002.txt	461	NA	461	461	0	18.31	461	461	0	10.02
Dumas	n40w40.003.txt	474	NA	474	474	0	4.11	474	474	0	1.86
Dumas	n40w40.004.txt	452	NA	452	452	0	13.72	452	452	0	9.67
Dumas	n40w40.005.txt	453	NA	453	453	0	20.16	453	453	0	8.88
Dumas	n40w60.001.txt	494	NA	494	494	0	15.85	494	494	0	16.40
Dumas	n40w60.002.txt	470	NA	470	470	0	24.24	470	470	0	15.00
Dumas	n40w60.003.txt	408	NA	408	408	0	21.72	408	408	0	13.27
Dumas	n40w60.004.txt	382	NA	382	382	0	64.78	382	382	0	51.30
Dumas	n40w60.005.txt	328	NA	328	328	0	32.39	328	328	0	11.37
Dumas	n40w80.001.txt	395	NA	395	395	0	31.40	395	395	0	21.92
Dumas	n40w80.002.txt	431	NA	431	431	0	59.08	431	431	0	52.11
Dumas	n40w80.003.txt	412	NA	412	412	0	28.76	412	412	0	20.84
Dumas	n40w80.004.txt	417	NA	417	417	0	33.35	417	417	0	28.22
Dumas	n40w80.005.txt	344	NA	344	344	0	58.02	344	344	0	47.72
Dumas	n60w100.001.txt	515	NA	515	515	0	1063.65	515	515	0	970.51
Dumas	n60w100.002.txt	538	NA	538	538	0	456.89	538	538	0	453.45
Dumas	n60w100.003.txt	560	NA	560	560	0	361.90	560	560	0	397.47
Dumas	n60w100.004.txt	510	NA	510	510	0	264.85	510	510	0	313.67
Dumas	n60w20.001.txt	551	NA	551	551	0	6.33	551	551	0	2.46
Dumas	n60w20.002.txt	605	NA	605	605	0	4.83	605	605	0	1.78
Dumas	n60w20.003.txt	533	NA	533	533	0	6.45	533	533	0	1.23
Dumas	n60w20.004.txt	616	NA	616	616	0	7.34	616	616	0	1.77
Dumas	n60w20.005.txt	603	NA	603	603	0	4.11	603	603	0	0.64
Dumas	n60w40.001.txt	591	NA	591	591	0	45.45	591	591	0	35.15
Dumas	n60w40.002.txt	621	NA	621	621	0	72.70	621	621	0	63.07
Dumas	n60w40.003.txt	603	NA	603	603	0	51.21	603	603	0	41.32
Dumas	n60w40.004.txt	597	NA	597	597	0	22.87	597	597	0	11.06
Dumas	n60w40.005.txt	539	NA	539	539	0	28.22	539	539	0	15.14
Dumas	n60w60.001.txt	609	NA	609	609	0	77.16	609	609	0	71.60
Dumas	n60w60.002.txt	566	NA	566	566	0	96.04	566	566	0	91.61
Dumas	n60w60.003.txt	485	NA	485	485	0	130.48	485	485	0	124.08
Dumas	n60w60.004.txt	571	NA	571	571	0	106.17	571	571	0	84.95
Dumas	n60w60.005.txt	569	NA	569	569	0	76.37	569	569	0	71.14
Dumas	n60w80.001.txt	458	NA	458	458	0	546.14	458	458	0	444.94
Dumas	n60w80.002.txt	498	NA	498	498	0	147.40	498	498	0	146.75
Dumas	n60w80.003.txt	550	NA	550	550	0	133.79	550	550	0	129.27
Dumas	n60w80.004.txt	566	NA	566	566	0	207.43	566	566	0	178.19

Table A.8 (**Part 3 of 6**) TSP-TW Results of PnB at  $\omega = 2048$  on Closed Problems: Seeded and Unseeded

LB = Lower Bound, UB = Upper Bound, OG = Optimality Gap, O = Optimality Proved.

Problem Information			Baldacci et. al. (2012)	Single Thread: 2048				Seeded Single Thread: 2048			
Set	Name	Best Known Solution		LB	UB	OG	Time	LB	UB	OG	Time
Dumas	n60w80.005.txt	468	NA	468	468	0	363.20	468	468	0	363.03
Dumas	n80w20.001.txt	616	NA	616	616	0	58.66	616	616	0	31.87
Dumas	n80w20.002.txt	737	NA	737	737	0	30.47	737	737	0	20.49
Dumas	n80w20.003.txt	667	NA	667	667	0	14.48	667	667	0	5.52
Dumas	n80w20.004.txt	615	NA	615	615	0	41.62	615	615	0	32.25
Dumas	n80w20.005.txt	748	NA	748	748	0	23.05	748	748	0	10.52
Dumas	n80w40.001.txt	606	NA	606	606	0	313.45	606	606	0	245.90
Dumas	n80w40.002.txt	618	NA	618	618	0	135.46	618	618	0	132.07
Dumas	n80w40.003.txt	674	NA	674	674	0	109.76	674	674	0	131.28
Dumas	n80w40.004.txt	557	NA	557	557	0	115.15	557	557	0	112.15
Dumas	n80w40.005.txt	695	NA	695	695	0	130.12	695	695	0	126.17
Dumas	n80w60.001.txt	554	NA	554	554	0	237.93	554	554	0	239.36
Dumas	n80w60.002.txt	633	NA	633	633	0	488.91	633	633	0	562.44
Dumas	n80w60.003.txt	651	NA	651	651	0	421.10	651	651	0	404.65
Dumas	n80w60.004.txt	619	NA	619	619	0	319.16	619	619	0	220.47
Dumas	n80w60.005.txt	575	NA	575	575	0	765.78	575	575	0	918.69
Dumas	n80w80.001.txt	624	NA	624	624	0	995.03	624	624	0	919.20
Dumas	n80w80.005.txt	570	NA	570	570	0	285.37	570	570	0	348.52
GendreauDumas	n100w100.001.txt	643	O	467	670	30.3	-	467	643	27.37	-
GendreauDumas	n100w100.002.txt	619	O	480	629	23.69	-	477	619	22.94	-
GendreauDumas	n100w100.003.txt	685	O	474	703	32.57	-	474	685	30.8	-
GendreauDumas	n100w100.005.txt	572	O	365	617	40.84	-	367	572	35.84	-
GendreauDumas	n100w120.001.txt	629	O	405	639	36.62	-	405	629	35.61	-
GendreauDumas	n100w120.003.txt	617	O	451	634	28.86	-	451	617	26.9	-
GendreauDumas	n100w120.005.txt	537	O	306	592	48.31	-	306	537	43.02	-
GendreauDumas	n100w140.001.txt	604	O	393	704	44.18	-	393	604	34.93	-
GendreauDumas	n100w140.002.txt	615	O	403	650	38	-	403	615	34.47	-
GendreauDumas	n100w140.003.txt	481	O	229	494	53.64	-	229	481	52.39	-
GendreauDumas	n100w140.004.txt	533	O	342	570	40	-	343	533	35.65	-
GendreauDumas	n100w160.001.txt	582	O	415	614	32.41	-	416	582	28.52	-
GendreauDumas	n100w160.003.txt	495	O	301	551	45.37	-	302	495	38.99	-
GendreauDumas	n100w160.004.txt	580	O	391	627	37.64	-	392	580	32.41	-
GendreauDumas	n100w80.001.txt	670	O	591	683	13.47	-	591	670	11.79	-
GendreauDumas	n100w80.003.txt	691	O	480	691	30.54	-	480	691	30.54	-
GendreauDumas	n100w80.004.txt	700	O	574	-	100	-	573	700	18.14	-
GendreauDumas	n100w80.005.txt	603	O	416	648	35.8	-	427	603	29.19	-
GendreauDumas	n20w120.001.txt	267	O	267	267	0	14.18	267	267	0	7.09
GendreauDumas	n20w120.002.txt	218	O	218	218	0	13.27	218	218	0	8.66
GendreauDumas	n20w120.003.txt	303	O	303	303	0	11.09	303	303	0	5.49
GendreauDumas	n20w120.004.txt	300	O	300	300	0	9.41	300	300	0	5.67
GendreauDumas	n20w120.005.txt	240	O	240	240	0	12.67	240	240	0	8.52
GendreauDumas	n20w140.001.txt	176	O	176	176	0	11.91	176	176	0	6.26
GendreauDumas	n20w140.002.txt	272	O	272	272	0	11.35	272	272	0	7.76
GendreauDumas	n20w140.003.txt	236	O	236	236	0	13.22	236	236	0	6.43
GendreauDumas	n20w140.004.txt	255	O	255	255	0	12.99	255	255	0	6.01
GendreauDumas	n20w140.005.txt	225	O	225	225	0	10.73	225	225	0	6.15
GendreauDumas	n20w160.001.txt	241	O	241	241	0	14.81	241	241	0	5.58
GendreauDumas	n20w160.002.txt	201	O	201	201	0	11.25	201	201	0	7.81
GendreauDumas	n20w160.003.txt	201	O	201	201	0	10.86	201	201	0	5.00
GendreauDumas	n20w160.004.txt	203	O	203	203	0	19.55	203	203	0	8.33
GendreauDumas	n20w160.005.txt	245	O	245	245	0	13.27	245	245	0	8.35
GendreauDumas	n20w180.001.txt	253	O	253	253	0	11.97	253	253	0	5.78
GendreauDumas	n20w180.002.txt	265	O	265	265	0	16.99	265	265	0	7.46
GendreauDumas	n20w180.003.txt	271	O	271	271	0	18.30	271	271	0	9.70
GendreauDumas	n20w180.004.txt	201	O	201	201	0	17.58	201	201	0	7.11
GendreauDumas	n20w180.005.txt	193	O	193	193	0	32.53	193	193	0	8.59
GendreauDumas	n20w200.001.txt	233	O	233	233	0	20.03	233	233	0	10.08
GendreauDumas	n20w200.002.txt	203	O	203	203	0	25.27	203	203	0	9.63
GendreauDumas	n20w200.003.txt	249	O	249	249	0	21.51	249	249	0	9.40
GendreauDumas	n20w200.005.txt	227	O	227	227	0	21.91	227	227	0	7.87
GendreauDumas	n40w120.001.txt	434	O	434	434	0	91.00	434	434	0	83.12
GendreauDumas	n40w120.002.txt	445	O	445	445	0	141.74	445	445	0	97.37
GendreauDumas	n40w120.003.txt	357	O	357	357	0	881.45	357	357	0	1338.21
GendreauDumas	n40w120.004.txt	303	O	303	303	0	105.54	303	303	0	96.98
GendreauDumas	n40w120.005.txt	350	O	350	350	0	80.86	350	350	0	62.93
GendreauDumas	n40w140.001.txt	328	O	328	328	0	196.16	328	328	0	148.20
GendreauDumas	n40w140.002.txt	383	O	383	383	0	2768.09	383	383	0	2677.04
GendreauDumas	n40w140.003.txt	398	O	398	398	0	125.13	398	398	0	72.96

Table A.9 (**Part 4 of 6**) TSP-TW Results of PnB at  $\omega = 2048$  on Closed Problems: Seeded and Unseeded

LB = Lower Bound, UB = Upper Bound, OG = Optimality Gap, O = Optimality Proved.

Set	Problem Information		Baldacci et. al. (2012)	Single Thread: 2048				Seeded Single Thread: 2048			
	Name	Best Known Solution		LB	UB	OG	Time	LB	UB	OG	Time
GendreauDumas	n40w140.004.txt	342	O	246	342	28.07	-	246	342	28.07	-
GendreauDumas	n40w140.005.txt	371	O	371	371	0	471.26	371	371	0	289.28
GendreauDumas	n40w160.001.txt	348	O	348	348	0	179.81	348	348	0	159.90
GendreauDumas	n40w160.002.txt	337	O	337	337	0	127.84	337	337	0	115.75
GendreauDumas	n40w160.003.txt	346	O	346	346	0	301.12	346	346	0	156.85
GendreauDumas	n40w160.004.txt	288	O	288	288	0	313.23	288	288	0	292.72
GendreauDumas	n40w160.005.txt	315	O	315	315	0	1389.91	315	315	0	704.93
GendreauDumas	n40w180.001.txt	337	O	274	352	22.16	-	337	337	0	2852.42
GendreauDumas	n40w180.002.txt	347	O	347	347	0	689.55	347	347	0	652.55
GendreauDumas	n40w180.003.txt	279	O	196	279	29.75	-	192	279	31.18	-
GendreauDumas	n40w180.005.txt	335	O	237	340	30.29	-	238	335	28.96	-
GendreauDumas	n40w200.001.txt	330	O	208	343	39.36	-	208	330	36.97	-
GendreauDumas	n40w200.005.txt	296	O	239	302	20.86	-	251	296	15.2	-
GendreauDumas	n60w120.001.txt	384	O	274	396	30.81	-	274	384	28.65	-
GendreauDumas	n60w120.002.txt	427	O	371	435	14.71	-	370	427	13.35	-
GendreauDumas	n60w120.003.txt	407	O	304	438	30.59	-	306	407	24.82	-
GendreauDumas	n60w120.004.txt	490	O	425	490	13.27	-	425	490	13.27	-
GendreauDumas	n60w120.005.txt	547	O	498	549	9.29	-	501	547	8.41	-
GendreauDumas	n60w140.003.txt	427	O	364	448	18.75	-	369	427	13.58	-
GendreauDumas	n60w140.004.txt	488	O	394	498	20.88	-	394	488	19.26	-
GendreauDumas	n60w160.002.txt	423	O	325	431	24.59	-	326	423	22.93	-
GendreauDumas	n60w160.004.txt	401	O	288	409	29.58	-	289	401	27.93	-
GendreauDumas	n60w160.005.txt	502	O	368	512	28.12	-	368	502	26.69	-
GendreauDumas	n60w180.002.txt	399	O	305	412	25.97	-	305	399	23.56	-
GendreauDumas	n60w200.001.txt	410	O	297	437	32.04	-	298	410	27.32	-
GendreauDumas	n60w200.002.txt	414	O	251	431	41.76	-	253	414	38.89	-
GendreauDumas	n80w100.001.txt	565	O	433	580	25.34	-	432	565	23.54	-
GendreauDumas	n80w100.002.txt	567	O	439	-	100	-	436	567	23.1	-
GendreauDumas	n80w100.003.txt	580	O	422	605	30.25	-	423	580	27.07	-
GendreauDumas	n80w100.005.txt	532	O	396	560	29.29	-	396	532	25.56	-
GendreauDumas	n80w120.003.txt	540	O	361	558	35.3	-	368	540	31.85	-
GendreauDumas	n80w140.001.txt	512	O	346	560	38.21	-	347	512	32.23	-
GendreauDumas	n80w140.005.txt	545	O	332	555	40.18	-	332	545	39.08	-
GendreauDumas	n80w160.001.txt	506	O	340	530	35.85	-	340	506	32.81	-
GendreauDumas	n80w160.003.txt	521	O	305	574	46.86	-	307	521	41.07	-
GendreauDumas	n80w160.004.txt	509	O	252	575	56.17	-	252	509	50.49	-
GendreauDumas	n80w160.005.txt	439	O	246	519	52.6	-	246	439	43.96	-
GendreauDumas	n80w180.002.txt	479	O	239	532	55.08	-	239	479	50.1	-
GendreauDumas	n80w180.003.txt	524	O	368	575	36	-	368	524	29.77	-
GendreauDumas	n80w180.004.txt	479	O	277	504	45.04	-	277	479	42.17	-
GendreauDumas	n80w180.005.txt	470	O	263	486	45.88	-	264	470	43.83	-
Langevin	N20ft301.dat	661.6	O	661.6	661.6	0	0.06	661.6	661.6	0	0.18
Langevin	N20ft302.dat	684.2	O	684.2	684.2	0	0.06	684.2	684.2	0	0.18
Langevin	N20ft303.dat	746.4	O	746.4	746.4	0	0.06	746.4	746.4	0	0.18
Langevin	N20ft304.dat	817	O	817	817	0	0.06	817	817	0	0.18
Langevin	N20ft305.dat	716.5	O	716.5	716.5	0	0.06	716.5	716.5	0	0.20
Langevin	N20ft306.dat	727.8	O	727.8	727.8	0	0.06	727.8	727.8	0	0.18
Langevin	N20ft307.dat	691.8	O	691.8	691.8	0	0.08	691.8	691.8	0	0.20
Langevin	N20ft308.dat	788.2	O	788.2	788.2	0	0.06	788.2	788.2	0	0.20
Langevin	N20ft309.dat	730.7	O	730.7	730.7	0	0.06	730.7	730.7	0	0.18
Langevin	N20ft310.dat	683	O	683	683	0	0.06	683	683	0	0.18
Langevin	N20ft401.dat	660.8	O	660.8	660.8	0	0.06	660.8	660.8	0	0.20
Langevin	N20ft402.dat	684.2	O	684.2	684.2	0	0.07	684.2	684.2	0	0.20
Langevin	N20ft403.dat	746.4	O	746.4	746.4	0	0.07	746.4	746.4	0	0.18
Langevin	N20ft404.dat	817	O	817	817	0	0.07	817	817	0	0.18
Langevin	N20ft405.dat	716.5	O	716.5	716.5	0	0.07	716.5	716.5	0	0.20
Langevin	N20ft406.dat	727.8	O	727.8	727.8	0	0.06	727.8	727.8	0	0.19
Langevin	N20ft407.dat	691.8	O	691.8	691.8	0	0.08	691.8	691.8	0	0.19
Langevin	N20ft408.dat	757.3	O	757.3	757.3	0	0.09	757.3	757.3	0	0.19
Langevin	N20ft409.dat	730.7	O	730.7	730.7	0	0.07	730.7	730.7	0	0.18
Langevin	N20ft410.dat	683	O	683	683	0	0.07	683	683	0	0.19
Langevin	N40ft201.dat	1100.6	O	1100.6	1100.6	0	0.07	1100.6	1100.6	0	0.31
Langevin	N40ft202.dat	1010.4	O	1010.4	1010.4	0	0.08	1010.4	1010.4	0	0.24
Langevin	N40ft203.dat	876.8	O	876.8	876.8	0	0.07	876.8	876.8	0	0.24
Langevin	N40ft204.dat	885.8	O	885.8	885.8	0	0.07	885.8	885.8	0	0.27
Langevin	N40ft205.dat	940.9	O	940.9	940.9	0	0.08	940.9	940.9	0	0.28
Langevin	N40ft206.dat	1054.2	O	1054.2	1054.2	0	0.07	1054.2	1054.2	0	0.27
Langevin	N40ft207.dat	867.5	O	867.5	867.5	0	0.09	867.5	867.5	0	0.27

Table A.10 (**Part 5 of 6**) TSP-TW Results of PnB at  $\omega = 2048$  on Closed Problems: Seeded and Unseeded

LB = Lower Bound, UB = Upper Bound, OG = Optimality Gap, O = Optimality Proved.

Problem Information			Baldacci et. al. (2012)	Single Thread: 2048				Seeded Single Thread: 2048			
Set	Name	Best Known Solution		LB	UB	OG	Time	LB	UB	OG	Time
Langevin	N40ft208.dat	1050.7	O	1050.7	1050.7	0	0.08	1050.7	1050.7	0	0.24
Langevin	N40ft209.dat	1013.9	O	1013.9	1013.9	0	0.07	1013.9	1013.9	0	0.29
Langevin	N40ft210.dat	1026.3	O	1026.3	1026.3	0	0.08	1026.3	1026.3	0	0.28
Langevin	N40ft401.dat	1085	O	1085	1085	0	0.11	1085	1085	0	0.79
Langevin	N40ft402.dat	995.6	O	995.6	995.6	0	0.07	995.6	995.6	0	0.34
Langevin	N40ft403.dat	845.8	O	845.8	845.8	0	2.03	845.8	845.8	0	0.41
Langevin	N40ft404.dat	868	O	868	868	0	0.22	868	868	0	0.32
Langevin	N40ft405.dat	936.5	O	936.5	936.5	0	1.50	936.5	936.5	0	0.38
Langevin	N40ft406.dat	969.1	O	969.1	969.1	0	0.86	969.1	969.1	0	0.32
Langevin	N40ft407.dat	831.2	O	831.2	831.2	0	0.54	831.2	831.2	0	1.57
Langevin	N40ft408.dat	1002.7	O	1002.7	1002.7	0	0.32	1002.7	1002.7	0	0.36
Langevin	N40ft409.dat	1000.5	O	1000.5	1000.5	0	0.10	1000.5	1000.5	0	0.32
Langevin	N40ft410.dat	983.8	O	983.8	983.8	0	0.08	983.8	983.8	0	0.29
Langevin	N60ft201.dat	1353.5	O	1353.5	1353.5	0	0.11	1353.5	1353.5	0	0.48
Langevin	N60ft202.dat	1161.6	O	1161.6	1161.6	0	0.08	1161.6	1161.6	0	1.13
Langevin	N60ft203.dat	1182.9	O	1182.9	1182.9	0	0.10	1182.9	1182.9	0	0.49
Langevin	N60ft204.dat	1257.5	O	1257.5	1257.5	0	0.08	1257.5	1257.5	0	0.52
Langevin	N60ft205.dat	1184.1	O	1184.1	1184.1	0	0.07	1184.1	1184.1	0	0.50
Langevin	N60ft206.dat	1199.6	O	1199.6	1199.6	0	0.34	1199.6	1199.6	0	0.54
Langevin	N60ft207.dat	1299	O	1299	1299	0	0.07	1299	1299	0	0.43
Langevin	N60ft208.dat	1113	O	1113	1113	0	1.19	1113	1113	0	0.54
Langevin	N60ft209.dat	1171.3	O	1171.3	1171.3	0	0.18	1171.3	1171.3	0	0.56
Langevin	N60ft210.dat	1234.3	O	1234.3	1234.3	0	2.35	1234.3	1234.3	0	0.62
Langevin	N60ft301.dat	1337	O	1337	1337	0	0.67	1337	1337	0	1.46
Langevin	N60ft302.dat	1089.5	O	1089.5	1089.5	0	4.40	1089.5	1089.5	0	1.03
Langevin	N60ft303.dat	1179	O	1179	1179	0	0.20	1179	1179	0	0.59
Langevin	N60ft304.dat	1230	O	1230	1230	0	6.64	1230	1230	0	3.07
Langevin	N60ft305.dat	1151.6	O	1151.6	1151.6	0	2.32	1151.6	1151.6	0	0.73
Langevin	N60ft306.dat	1167.9	O	1167.9	1167.9	0	4.87	1167.9	1167.9	0	0.76
Langevin	N60ft307.dat	1220.1	O	1220.1	1220.1	0	0.28	1220.1	1220.1	0	1.24
Langevin	N60ft308.dat	1097.6	O	1097.6	1097.6	0	4.88	1097.6	1097.6	0	0.72
Langevin	N60ft309.dat	1140.6	O	1140.6	1140.6	0	3.22	1140.6	1140.6	0	0.74
Langevin	N60ft310.dat	1219.2	O	1219.2	1219.2	0	12.93	1219.2	1219.2	0	5.55
Langevin	N60ft401.dat	1335	O	1335	1335	0	10.41	1335	1335	0	3.18
Langevin	N60ft402.dat	1088.1	O	1088.1	1088.1	0	9.70	1088.1	1088.1	0	3.29
Langevin	N60ft403.dat	1173.7	O	1173.7	1173.7	0	4.23	1173.7	1173.7	0	1.71
Langevin	N60ft404.dat	1184.7	O	1184.7	1184.7	0	4.59	1184.7	1184.7	0	0.80
Langevin	N60ft405.dat	1146.2	O	1146.2	1146.2	0	3.30	1146.2	1146.2	0	0.90
Langevin	N60ft406.dat	1140.2	O	1140.2	1140.2	0	17.09	1140.2	1140.2	0	7.23
Langevin	N60ft407.dat	1198.9	O	1198.9	1198.9	0	9.75	1198.9	1198.9	0	3.11
Langevin	N60ft408.dat	1029.4	O	1029.4	1029.4	0	7.47	1029.4	1029.4	0	1.23
Langevin	N60ft409.dat	1121.4	O	1121.4	1121.4	0	8.25	1121.4	1121.4	0	1.07
Langevin	N60ft410.dat	1189.6	O	1189.6	1189.6	0	19.56	1189.6	1189.6	0	7.72
OhlmannThomas	n150w120.001.txt	734	725.5	324	809	59.95	-	324	734	55.86	-
OhlmannThomas	n150w120.002.txt	677	668.4	209	717	70.85	-	209	677	69.13	-
OhlmannThomas	n150w120.003.txt	747	746.4	416	807	48.45	-	416	747	44.31	-
OhlmannThomas	n150w120.004.txt	763	761.6	381	835	54.37	-	381	763	50.07	-
OhlmannThomas	n150w120.005.txt	689	684.7	266	754	64.72	-	274	689	60.23	-
OhlmannThomas	n150w140.001.txt	762	754	394	893	55.88	-	394	762	48.29	-
OhlmannThomas	n150w140.002.txt	755	752	407	855	52.4	-	416	755	44.9	-
OhlmannThomas	n150w140.003.txt	613	608.5	215	738	70.87	-	215	613	64.93	-
OhlmannThomas	n150w140.004.txt	676	O	391	834	53.12	-	391	676	42.16	-
OhlmannThomas	n150w140.005.txt	663	662	197	750	73.73	-	197	663	70.29	-
OhlmannThomas	n150w160.001.txt	706	701.4	293	777	62.29	-	293	706	58.5	-
OhlmannThomas	n150w160.002.txt	711	709.7	286	826	65.38	-	286	711	59.77	-
OhlmannThomas	n150w160.003.txt	608	603.2	170	772	77.98	-	170	608	72.04	-
OhlmannThomas	n150w160.004.txt	672	672	336	749	55.14	-	344	672	48.81	-
OhlmannThomas	n150w160.005.txt	658	655	320	736	56.52	-	320	658	51.37	-
OhlmannThomas	n200w120.001.txt	799	793.3	312	910	65.71	-	312	799	60.95	-
OhlmannThomas	n200w120.002.txt	721	713.9	184	822	77.62	-	184	721	74.48	-
OhlmannThomas	n200w120.003.txt	880	868.6	336	983	65.82	-	336	880	61.82	-
OhlmannThomas	n200w120.004.txt	777	775.8	291	887	67.19	-	291	777	62.55	-
OhlmannThomas	n200w120.005.txt	841	833.2	258	960	73.12	-	371	841	55.89	-
OhlmannThomas	n200w140.001.txt	834	826.2	177	1053	83.19	-	177	834	78.78	-
OhlmannThomas	n200w140.002.txt	760	756.2	180	895	79.89	-	180	760	76.32	-
OhlmannThomas	n200w140.003.txt	758	756	241	897	73.13	-	252	758	66.75	-
OhlmannThomas	n200w140.004.txt	816	807.1	284	932	69.53	-	284	816	65.2	-
OhlmannThomas	n200w140.005.txt	822	819.6	148	927	84.03	-	148	822	82	-

Table A.11 (**Part 6 of 6**) TSP-TW Results of PnB at  $\omega = 2048$  on Closed Problems: Seeded and Unseeded

LB = Lower Bound, UB = Upper Bound, OG = Optimality Gap, O = Optimality Proved.

Problem Information			Baldacci et. al. (2012)	Single Thread: 2048				Seeded Single Thread: 2048			
Set	Name	Best Known Solution		LB	UB	OG	Time	LB	UB	OG	Time
SolomonPesant	rc201.0	628.62	O	628.62	628.62	0	3.36	628.62	628.62	0	2.13
SolomonPesant	rc201.1	654.7	O	654.70	654.70	0	3.97	654.70	654.70	0	2.56
SolomonPesant	rc201.2	707.65	O	707.65	707.65	0	3.34	707.65	707.65	0	1.63
SolomonPesant	rc201.3	422.54	O	422.54	422.54	0	0.52	422.54	422.54	0	0.73
SolomonPesant	rc202.0	496.22	O	496.22	496.22	0	19.91	496.22	496.22	0	12.48
SolomonPesant	rc202.1	426.53	O	426.53	426.53	0	14.76	426.53	426.53	0	10.34
SolomonPesant	rc202.2	611.77	O	611.77	611.77	0	19.53	611.77	611.77	0	9.32
SolomonPesant	rc202.3	627.85	O	627.85	627.85	0	27.07	627.85	627.85	0	13.18
SolomonPesant	rc203.2	617.46	O	617.46	617.46	0	42.21	617.46	617.46	0	20.88
SolomonPesant	rc204.0	541.45	O	495.85	551.56	10.1	-	541.45	541.45	0	1560.34
SolomonPesant	rc204.1	485.37	O	485.37	485.37	0	184.27	485.37	485.37	0	101.42
SolomonPesant	rc205.0	511.65	O	511.65	511.65	0	17.52	511.65	511.65	0	9.86
SolomonPesant	rc205.1	491.22	O	491.22	491.22	0	13.18	491.22	491.22	0	9.71
SolomonPesant	rc205.2	714.69	O	714.69	714.69	0	17.80	714.69	714.69	0	11.57
SolomonPesant	rc205.3	601.24	O	601.24	601.24	0	10.30	601.24	601.24	0	10.06
SolomonPesant	rc206.0	835.23	O	835.23	835.23	0	49.27	835.23	835.23	0	30.81
SolomonPesant	rc206.1	664.73	O	664.73	664.73	0	54.26	664.73	664.73	0	25.98
SolomonPesant	rc206.2	655.37	O	655.37	655.37	0	89.91	655.37	655.37	0	29.53
SolomonPesant	rc207.0	806.69	O	806.69	806.69	0	116.56	806.69	806.69	0	62.17
SolomonPesant	rc207.1	726.36	O	726.36	726.36	0	93.50	726.36	726.36	0	61.84
SolomonPesant	rc207.2	546.41	O	546.41	546.41	0	159.83	546.41	546.41	0	33.82
SolomonPesant	rc208.0	820.56	O	728.35	837.66	13.05	-	757.44	820.56	7.69	-
SolomonPesant	rc208.1	509.04	O	509.04	509.04	0	309.35	509.04	509.04	0	50.02
SolomonPesant	rc208.2	503.92	O	503.92	503.92	0	280.49	503.92	503.92	0	19.23
SolomonPotvinBengio	rc_201.1.txt	444.54	O	444.54	444.54	0	0.74	444.54	444.54	0	1.31
SolomonPotvinBengio	rc_201.2.txt	711.54	O	711.54	711.54	0	0.63	711.54	711.54	0	1.07
SolomonPotvinBengio	rc_201.3.txt	790.61	O	790.61	790.61	0	7.28	790.61	790.61	0	3.12
SolomonPotvinBengio	rc_201.4.txt	793.64	O	793.64	793.64	0	0.27	793.64	793.64	0	0.27
SolomonPotvinBengio	rc_202.1.txt	771.78	O	771.78	771.78	0	577.89	771.78	771.78	0	556.81
SolomonPotvinBengio	rc_202.2.txt	304.14	O	304.14	304.14	0	1.57	304.14	304.14	0	0.61
SolomonPotvinBengio	rc_203.1.txt	453.48	O	453.48	453.48	0	10.84	453.48	453.48	0	6.52
SolomonPotvinBengio	rc_203.4.txt	314.29	O	314.29	314.29	0	9.24	314.29	314.29	0	1.69
SolomonPotvinBengio	rc_204.3.txt	455.03	O	455.03	455.03	0	682.03	455.03	455.03	0	139.37
SolomonPotvinBengio	rc_205.1.txt	343.21	O	343.21	343.21	0	0.35	343.21	343.21	0	0.32
SolomonPotvinBengio	rc_205.2.txt	755.93	O	755.93	755.93	0	9.24	755.93	755.93	0	8.16
SolomonPotvinBengio	rc_205.3.txt	825.06	O	825.06	825.06	0	59.00	825.06	825.06	0	36.06
SolomonPotvinBengio	rc_206.1.txt	117.85	O	117.85	117.85	0	0.06	117.85	117.85	0	0.20
SolomonPotvinBengio	rc_206.3.txt	574.42	O	574.42	574.42	0	21.92	574.42	574.42	0	12.96
SolomonPotvinBengio	rc_207.4.txt	119.64	O	119.64	119.64	0	0.06	119.64	119.64	0	0.20
SolomonPotvinBengio	rc_208.2.txt	533.78	O	533.78	533.78	0	835.29	533.78	533.78	0	37.47

Table A.12 (**Part 1 of 7**) Makespan Results of PnB at  $\omega = 2048$  on Closed Problems: Seeded and Unseeded

LB = Lower Bound, UB = Upper Bound.

Set	Problem Information		Single Thread: 2048			Seeded Single Thread: 2048		
	Name	Best Known Solution	LB	UB	Time	LB	UB	Time
AFG	rbg010a.tw	3840	3840	3840	0.10	3840	3840	0.27
AFG	rbg016a.tw	2596	2596	2596	0.59	2596	2596	0.23
AFG	rbg016b.tw	2094	2094	2094	1.79	2094	2094	0.74
AFG	rbg017.2.tw	2351	2351	2351	4.68	2351	2351	2.93
AFG	rbg017.tw	2351	2351	2351	1.55	2351	2351	0.97
AFG	rbg017a.tw	4296	4296	4296	0.29	4296	4296	0.25
AFG	rbg019a.tw	2694	2694	2694	0.55	2694	2694	0.20
AFG	rbg019b.tw	3840	3840	3840	5.79	3840	3840	4.11
AFG	rbg019c.tw	4536	4536	4536	2.56	4536	4536	2.72
AFG	rbg019d.tw	3479	3479	3479	1.71	3479	3479	0.25
AFG	rbg020a.tw	4689	4689	4689	0.28	4689	4689	0.24
AFG	rbg021.2.tw	4528	4528	4528	9.20	4528	4528	4.82
AFG	rbg021.3.tw	4528	4528	4528	14.33	4528	4528	6.57
AFG	rbg021.4.tw	4525	4525	4525	19.95	4525	4525	6.64
AFG	rbg021.5.tw	4516	4516	4516	18.36	4516	4516	6.19
AFG	rbg021.6.tw	4492	4492	4492	116.12	4492	4492	15.35
AFG	rbg021.7.tw	4481	4481	4481	330.28	4481	4481	89.87
AFG	rbg021.8.tw	4481	4481	4481	397.88	4481	4481	136.81
AFG	rbg021.9.tw	4481	4481	4481	387.29	4481	4481	104.94
AFG	rbg021.tw	4536	4536	4536	2.56	4536	4536	2.91
AFG	rbg027a.tw	5093	5093	5093	9.35	5093	5093	8.87
AFG	rbg031a.tw	3498	3498	3498	12.87	3498	3498	5.94
AFG	rbg033a.tw	3757	3757	3757	26.84	3757	3757	11.43
AFG	rbg034a.tw	3314	3314	3314	18.78	3314	3314	10.52
AFG	rbg035a.2.tw	3325	3325	3325	31.96	3325	3325	18.98
AFG	rbg035a.tw	3388	3388	3388	17.85	3388	3388	9.03
AFG	rbg038a.tw	5699	5699	5699	22.98	5699	5699	9.84
AFG	rbg040a.tw	5679	5679	5679	20.86	5679	5679	10.07
AFG	rbg041a.tw	3793	3793	3793	23.72	3793	3793	11.82
AFG	rbg042a.tw	3260	3260	3260	133.16	3260	3260	55.62
AFG	rbg048a.tw	9799	9799	9799	47.50	9799	9799	32.45
AFG	rbg049a.tw	13257	13257	13257	43.08	13257	13257	24.53
AFG	rbg050a.tw	12050	12050	12050	76.87	12050	12050	42.58
AFG	rbg050c.tw	10985	10985	10985	608.13	10985	10985	41.37
AFG	rbg055a.tw	6929	6929	6929	37.86	6929	6929	19.52
AFG	rbg067a.tw	10331	10331	10331	50.47	10331	10331	26.74
AFG	rbg086a.tw	16899	16899	16899	55.93	16899	16899	23.52
AFG	rbg092a.tw	12501	12501	12501	105.84	12501	12501	57.06
AFG	rbg125a.tw	14214	14214	14214	240.51	14214	14214	159.05
AFG	rbg132.2.tw	18524	18524	18524	383.13	18524	18524	256.79
AFG	rbg132.tw	18524	18524	18524	166.49	18524	18524	109.79
AFG	rbg152.3.tw	17455	17455	17455	708.13	17455	17455	449.59
AFG	rbg152.tw	17455	17455	17455	293.16	17455	17455	214.97
AFG	rbg193.2.tw	21401	21401	21401	1072.79	21401	21401	890.64
AFG	rbg193.tw	21401	21401	21401	408.75	21401	21401	397.71
AFG	rbg201a.tw	21380	21380	21380	690.22	21380	21380	516.17
AFG	rbg233.2.tw	26143	26143	26143	1386.83	26143	26143	880.14
AFG	rbg233.tw	26143	26143	26143	310.61	26143	26143	543.33
Dumas	n100w20.001.txt	827	827	827	22.60	827	827	4.66
Dumas	n100w20.002.txt	801	801	801	27.68	801	801	20.23
Dumas	n100w20.003.txt	834	834	834	29.63	834	834	29.75
Dumas	n100w20.004.txt	828	828	828	40.49	828	828	19.34
Dumas	n100w20.005.txt	825	825	825	43.67	825	825	29.66
Dumas	n100w40.001.txt	841	841	841	81.21	841	841	49.35
Dumas	n100w40.002.txt	786	786	786	233.56	786	786	179.21
Dumas	n100w40.003.txt	835	835	835	164.55	835	835	162.40
Dumas	n100w40.004.txt	809	809	809	110.67	809	809	48.29
Dumas	n100w40.005.txt	834	834	834	242.46	834	834	117.29
Dumas	n100w60.001.txt	824	824	824	128.02	824	824	162.88
Dumas	n100w60.002.txt	782	782	782	148.01	782	782	104.97
Dumas	n100w60.003.txt	856	856	856	393.23	856	856	124.66
Dumas	n100w60.004.txt	834	834	834	86.16	834	834	95.44
Dumas	n100w60.005.txt	790	790	790	330.56	790	790	87.49
Dumas	n150w20.001.txt	1034	1034	1034	95.90	1034	1034	82.38
Dumas	n150w20.002.txt	967	967	967	95.51	967	967	58.82
Dumas	n150w20.003.txt	959	959	959	184.01	959	959	101.22
Dumas	n150w20.004.txt	975	975	975	113.70	975	975	83.60
Dumas	n150w20.005.txt	957	957	957	172.32	957	957	174.31



Table A.13 (**Part 2 of 7**) Makespan Results of PnB at  $\omega = 2048$  on Closed Problems: Seeded and Unseeded

LB = Lower Bound, UB = Upper Bound.

Set	Problem Information		Single Thread: 2048			Seeded Single Thread: 2048		
	Name	Best Known Solution	LB	UB	Time	LB	UB	Time
Dumas	n150w40.001.txt	1058	1058	1058	1189.38	1058	1058	183.42
Dumas	n150w40.002.txt	1072	1072	1072	302.80	1072	1072	227.35
Dumas	n150w40.003.txt	894	894	894	202.91	894	894	495.32
Dumas	n150w40.004.txt	948	948	948	261.44	948	948	162.54
Dumas	n150w40.005.txt	980	980	980	3533.27	980	980	194.03
Dumas	n150w60.001.txt	1024	1024	1024	1306.06	1024	1024	1361.14
Dumas	n150w60.003.txt	984	984	984	1541.55	984	984	493.46
Dumas	n150w60.004.txt	998	998	998	604.28	998	998	316.13
Dumas	n150w60.005.txt	997	997	997	537.80	997	997	506.47
Dumas	n200w20.001.txt	1139	1139	1139	236.02	1139	1139	591.72
Dumas	n200w20.002.txt	1124	1124	1124	477.22	1124	1124	178.85
Dumas	n200w20.003.txt	1178	1178	1178	382.11	1178	1178	252.84
Dumas	n200w20.004.txt	1125	1125	1125	889.15	1125	1125	815.18
Dumas	n200w20.005.txt	1123	1123	1123	569.79	1123	1123	403.14
Dumas	n200w40.001.txt	1188	1188	1188	272.29	1188	1188	561.61
Dumas	n200w40.003.txt	1134	1134	1134	383.87	1134	1134	1001.46
Dumas	n200w40.004.txt	1150	1150	1150	1004.33	1150	1150	854.40
Dumas	n200w40.005.txt	1171	1171	1171	494.15	1171	1171	675.02
Dumas	n20w100.001.txt	309	309	309	4.80	309	309	2.83
Dumas	n20w100.002.txt	285	285	285	7.64	285	285	3.10
Dumas	n20w100.003.txt	358	358	358	9.12	358	358	4.41
Dumas	n20w100.004.txt	379	379	379	2.14	379	379	1.05
Dumas	n20w100.005.txt	327	327	327	5.79	327	327	2.96
Dumas	n20w20.001.txt	387	387	387	0.08	387	387	0.19
Dumas	n20w20.002.txt	296	296	296	0.09	296	296	0.19
Dumas	n20w20.003.txt	403	403	403	0.11	403	403	0.19
Dumas	n20w20.004.txt	401	401	401	0.07	401	401	0.19
Dumas	n20w20.005.txt	365	365	365	0.07	365	365	0.19
Dumas	n20w40.001.txt	280	280	280	0.12	280	280	0.24
Dumas	n20w40.002.txt	357	357	357	0.07	357	357	0.19
Dumas	n20w40.003.txt	355	355	355	0.08	355	355	0.21
Dumas	n20w40.004.txt	397	397	397	0.10	397	397	0.21
Dumas	n20w40.005.txt	325	325	325	0.09	325	325	0.24
Dumas	n20w60.001.txt	400	400	400	2.10	400	400	1.03
Dumas	n20w60.002.txt	300	300	300	0.68	300	300	0.22
Dumas	n20w60.003.txt	381	381	381	0.08	381	381	0.23
Dumas	n20w60.004.txt	337	337	337	3.11	337	337	2.13
Dumas	n20w60.005.txt	392	392	392	0.59	392	392	0.24
Dumas	n20w80.001.txt	403	403	403	2.12	403	403	1.70
Dumas	n20w80.002.txt	397	397	397	1.24	397	397	0.88
Dumas	n20w80.003.txt	365	365	365	0.75	365	365	0.31
Dumas	n20w80.004.txt	345	345	345	1.84	345	345	1.00
Dumas	n20w80.005.txt	307	307	307	5.82	307	307	2.75
Dumas	n40w100.001.txt	471	471	471	78.64	471	471	65.33
Dumas	n40w100.002.txt	452	452	452	55.80	452	452	25.75
Dumas	n40w100.003.txt	458	458	458	44.81	458	458	29.04
Dumas	n40w100.004.txt	476	476	476	25.18	476	476	16.11
Dumas	n40w100.005.txt	458	458	458	26.47	458	458	13.81
Dumas	n40w20.001.txt	523	523	523	0.54	523	523	0.36
Dumas	n40w20.002.txt	607	607	607	2.21	607	607	0.56
Dumas	n40w20.003.txt	514	514	514	1.37	514	514	0.35
Dumas	n40w20.004.txt	442	442	442	1.04	442	442	0.30
Dumas	n40w20.005.txt	520	520	520	0.11	520	520	0.31
Dumas	n40w40.001.txt	510	510	510	2.45	510	510	0.52
Dumas	n40w40.002.txt	519	519	519	16.34	519	519	9.47
Dumas	n40w40.003.txt	536	536	536	4.86	536	536	1.98
Dumas	n40w40.004.txt	508	508	508	9.24	508	508	5.51
Dumas	n40w40.005.txt	488	488	488	17.23	488	488	6.18
Dumas	n40w60.001.txt	535	535	535	13.88	535	535	11.12
Dumas	n40w60.002.txt	509	509	509	13.42	509	509	10.98
Dumas	n40w60.003.txt	465	465	465	23.80	465	465	14.94
Dumas	n40w60.004.txt	475	475	475	20.68	475	475	12.92
Dumas	n40w60.005.txt	423	423	423	25.15	423	423	8.20
Dumas	n40w80.001.txt	497	497	497	31.58	497	497	20.59
Dumas	n40w80.002.txt	498	498	498	37.16	498	498	19.14
Dumas	n40w80.003.txt	488	488	488	24.64	488	488	8.43
Dumas	n40w80.004.txt	462	462	462	31.94	462	462	21.09
Dumas	n40w80.005.txt	488	488	488	50.14	488	488	25.29

Table A.14 (**Part 3 of 7**) Makespan Results of PnB at  $\omega = 2048$  on Closed Problems: Seeded and Unseeded

LB = Lower Bound, UB = Upper Bound.

Set	Problem Information			Single Thread: 2048			Seeded Single Thread: 2048		
	Name	Best Known Solution		LB	UB	Time	LB	UB	Time
Dumas	n60w100.001.txt	576		576	576	47.90	576	576	41.17
Dumas	n60w100.002.txt	622		622	622	49.11	622	622	31.61
Dumas	n60w100.003.txt	610		610	610	51.91	610	610	28.48
Dumas	n60w100.004.txt	625		625	625	56.84	625	625	55.01
Dumas	n60w100.005.txt	668		668	668	85.20	668	668	42.20
Dumas	n60w20.001.txt	586		586	586	6.05	586	586	2.00
Dumas	n60w20.002.txt	656		656	656	4.56	656	656	1.72
Dumas	n60w20.003.txt	593		593	593	6.96	593	593	1.26
Dumas	n60w20.004.txt	670		670	670	7.91	670	670	1.76
Dumas	n60w20.005.txt	629		629	629	4.24	629	629	0.60
Dumas	n60w40.001.txt	664		664	664	22.66	664	664	14.66
Dumas	n60w40.002.txt	697		697	697	32.01	697	697	26.93
Dumas	n60w40.003.txt	675		675	675	29.48	675	675	19.28
Dumas	n60w40.004.txt	628		628	628	15.52	628	628	7.76
Dumas	n60w40.005.txt	608		608	608	27.57	608	608	12.15
Dumas	n60w60.001.txt	722		722	722	24.45	722	722	30.56
Dumas	n60w60.002.txt	715		715	715	36.76	715	715	23.87
Dumas	n60w60.003.txt	619		619	619	123.67	619	619	49.79
Dumas	n60w60.004.txt	639		639	639	101.05	639	639	97.53
Dumas	n60w60.005.txt	669		669	669	39.90	669	669	20.10
Dumas	n60w80.001.txt	606		606	606	70.71	606	606	48.23
Dumas	n60w80.002.txt	611		611	611	45.52	611	611	34.00
Dumas	n60w80.003.txt	656		656	656	116.60	656	656	33.02
Dumas	n60w80.004.txt	679		679	679	59.47	679	679	42.62
Dumas	n60w80.005.txt	589		589	589	86.38	589	589	44.11
Dumas	n80w20.001.txt	729		729	729	40.45	729	729	29.10
Dumas	n80w20.002.txt	798		798	798	23.83	798	798	13.71
Dumas	n80w20.003.txt	727		727	727	16.35	727	727	5.47
Dumas	n80w20.004.txt	694		694	694	29.54	694	694	13.28
Dumas	n80w20.005.txt	793		793	793	16.04	793	793	7.97
Dumas	n80w40.001.txt	743		743	743	76.57	743	743	64.60
Dumas	n80w40.002.txt	701		701	701	34.24	701	701	32.70
Dumas	n80w40.003.txt	731		731	731	49.56	731	731	59.81
Dumas	n80w40.004.txt	662		662	662	38.17	662	662	47.81
Dumas	n80w40.005.txt	791		791	791	69.94	791	791	28.40
Dumas	n80w60.001.txt	674		674	674	123.69	674	674	46.05
Dumas	n80w60.002.txt	733		733	733	141.75	733	733	88.40
Dumas	n80w60.003.txt	743		743	743	127.93	743	743	165.29
Dumas	n80w60.004.txt	718		718	718	98.69	718	718	64.72
Dumas	n80w60.005.txt	695		695	695	71.97	695	695	48.39
Dumas	n80w80.001.txt	728		728	728	85.54	728	728	61.20
Dumas	n80w80.002.txt	690		690	690	314.50	690	690	290.70
Dumas	n80w80.003.txt	730		730	730	110.76	730	730	68.74
Dumas	n80w80.004.txt	743		743	743	206.81	743	743	108.65
Dumas	n80w80.005.txt	682		682	682	166.66	682	682	127.21
GendreauDumas	n100w100.001.txt	787		787	787	182.66	787	787	211.74
GendreauDumas	n100w100.002.txt	780		780	780	123.94	780	780	80.68
GendreauDumas	n100w100.004.txt	844		844	844	159.35	844	844	138.09
GendreauDumas	n100w100.005.txt	749		749	749	429.20	749	749	235.80
GendreauDumas	n100w120.001.txt	882		882	882	304.59	882	882	251.81
GendreauDumas	n100w120.002.txt	893		893	893	214.04	893	893	133.62
GendreauDumas	n100w120.003.txt	909		909	909	193.51	909	909	115.35
GendreauDumas	n100w120.004.txt	923		923	923	262.32	923	923	189.78
GendreauDumas	n100w120.005.txt	870		870	870	342.28	870	870	294.69
GendreauDumas	n100w140.001.txt	1008		1008	1008	275.39	1008	1008	200.57
GendreauDumas	n100w140.002.txt	1021		1021	1021	249.58	1021	1021	158.90
GendreauDumas	n100w140.003.txt	844		844	844	245.12	844	844	177.10
GendreauDumas	n100w140.004.txt	854		854	854	264.06	854	854	157.21
GendreauDumas	n100w140.005.txt	805		805	805	477.20	805	805	177.25
GendreauDumas	n100w160.001.txt	868		868	868	1258.25	868	868	887.56
GendreauDumas	n100w160.002.txt	791		791	791	861.44	791	791	513.75
GendreauDumas	n100w160.003.txt	935		935	935	232.75	935	935	129.39
GendreauDumas	n100w160.004.txt	814		814	814	569.60	814	814	307.28
GendreauDumas	n100w160.005.txt	917		917	917	622.96	917	917	331.31
GendreauDumas	n100w80.001.txt	797		797	797	2557.77	797	797	167.87
GendreauDumas	n100w80.002.txt	790		790	790	83.34	790	790	74.09
GendreauDumas	n100w80.004.txt	854		854	854	119.96	854	854	131.93
GendreauDumas	n100w80.005.txt	759		759	759	262.46	759	759	252.80

Table A.15 (**Part 4 of 7**) Makespan Results of PnB at  $\omega = 2048$  on Closed Problems: Seeded and Unseeded

LB = Lower Bound, UB = Upper Bound.

Set	Problem Information		Single Thread: 2048			Seeded Single Thread: 2048		
	Name	Best Known Solution	LB	UB	Time	LB	UB	Time
GendreauDumas	n20w120.001.txt	337	337	337	5.13	337	337	4.10
GendreauDumas	n20w120.002.txt	246	246	246	6.13	246	246	5.79
GendreauDumas	n20w120.003.txt	347	347	347	8.12	347	347	3.19
GendreauDumas	n20w120.004.txt	353	353	353	9.36	353	353	5.69
GendreauDumas	n20w120.005.txt	315	315	315	11.88	315	315	7.86
GendreauDumas	n20w140.001.txt	230	230	230	7.33	230	230	5.63
GendreauDumas	n20w140.002.txt	307	307	307	5.99	307	307	3.57
GendreauDumas	n20w140.003.txt	301	301	301	12.81	301	301	7.32
GendreauDumas	n20w140.004.txt	318	318	318	6.35	318	318	5.07
GendreauDumas	n20w140.005.txt	275	275	275	8.33	275	275	3.07
GendreauDumas	n20w160.001.txt	347	347	347	16.31	347	347	9.73
GendreauDumas	n20w160.002.txt	250	250	250	12.65	250	250	9.72
GendreauDumas	n20w160.003.txt	331	331	331	5.24	331	331	3.20
GendreauDumas	n20w160.004.txt	287	287	287	7.85	287	287	4.51
GendreauDumas	n20w160.005.txt	342	342	342	14.35	342	342	8.68
GendreauDumas	n20w180.001.txt	353	353	353	6.90	353	353	5.81
GendreauDumas	n20w180.002.txt	347	347	347	7.44	347	347	5.66
GendreauDumas	n20w180.003.txt	315	315	315	17.43	315	315	4.43
GendreauDumas	n20w180.004.txt	284	284	284	30.96	284	284	16.48
GendreauDumas	n20w180.005.txt	257	257	257	9.46	257	257	7.35
GendreauDumas	n20w200.001.txt	259	259	259	9.02	259	259	8.05
GendreauDumas	n20w200.002.txt	242	242	242	32.44	242	242	10.46
GendreauDumas	n20w200.003.txt	305	305	305	8.32	305	305	4.56
GendreauDumas	n20w200.004.txt	326	326	326	26.12	326	326	14.22
GendreauDumas	n20w200.005.txt	277	277	277	9.29	277	277	6.64
GendreauDumas	n40w120.001.txt	470	470	470	42.95	470	470	24.57
GendreauDumas	n40w120.002.txt	557	557	557	26.93	557	557	16.01
GendreauDumas	n40w120.003.txt	464	464	464	37.06	464	464	13.26
GendreauDumas	n40w120.004.txt	392	392	392	59.60	392	392	30.90
GendreauDumas	n40w120.005.txt	470	470	470	24.70	470	470	16.39
GendreauDumas	n40w140.001.txt	460	460	460	77.30	460	460	53.39
GendreauDumas	n40w140.002.txt	459	459	459	30.01	459	459	20.75
GendreauDumas	n40w140.003.txt	476	476	476	23.65	476	476	11.08
GendreauDumas	n40w140.004.txt	458	458	458	103.45	458	458	41.30
GendreauDumas	n40w140.005.txt	438	438	438	79.34	438	438	35.90
GendreauDumas	n40w160.001.txt	470	470	470	93.24	470	470	61.27
GendreauDumas	n40w160.002.txt	451	451	451	131.65	451	451	50.20
GendreauDumas	n40w160.003.txt	415	415	415	73.22	415	415	36.75
GendreauDumas	n40w160.004.txt	425	425	425	34.02	425	425	19.69
GendreauDumas	n40w160.005.txt	373	373	373	77.57	373	373	36.81
GendreauDumas	n40w180.001.txt	444	444	444	134.27	444	444	99.12
GendreauDumas	n40w180.002.txt	448	448	448	137.02	448	448	83.30
GendreauDumas	n40w180.003.txt	398	398	398	69.47	398	398	26.74
GendreauDumas	n40w180.004.txt	409	409	409	59.34	409	409	22.53
GendreauDumas	n40w180.005.txt	438	438	438	168.31	438	438	89.39
GendreauDumas	n40w200.001.txt	416	416	416	44.65	416	416	32.94
GendreauDumas	n40w200.002.txt	402	402	402	90.46	402	402	51.80
GendreauDumas	n40w200.003.txt	408	408	408	107.36	408	408	48.11
GendreauDumas	n40w200.004.txt	426	426	426	37.55	426	426	24.35
GendreauDumas	n40w200.005.txt	408	408	408	182.99	408	408	50.26
GendreauDumas	n60w120.001.txt	536	536	536	101.97	536	536	68.94
GendreauDumas	n60w120.002.txt	606	606	606	160.15	606	606	96.68
GendreauDumas	n60w120.003.txt	541	541	541	215.21	541	541	40.41
GendreauDumas	n60w120.004.txt	607	607	607	72.17	607	607	29.59
GendreauDumas	n60w120.005.txt	579	579	579	167.99	579	579	70.92
GendreauDumas	n60w140.001.txt	596	596	596	136.02	596	596	54.80
GendreauDumas	n60w140.002.txt	647	647	647	69.59	647	647	40.44
GendreauDumas	n60w140.003.txt	625	625	625	89.37	625	625	63.29
GendreauDumas	n60w140.004.txt	578	578	578	72.23	578	578	48.45
GendreauDumas	n60w140.005.txt	554	554	554	96.83	554	554	39.83
GendreauDumas	n60w160.001.txt	672	672	672	148.81	672	672	57.99
GendreauDumas	n60w160.002.txt	665	665	665	65.43	665	665	36.21
GendreauDumas	n60w160.003.txt	569	569	569	115.57	569	569	78.76
GendreauDumas	n60w160.004.txt	573	573	573	280.64	573	573	214.91
GendreauDumas	n60w160.005.txt	619	619	619	162.69	619	619	72.59
GendreauDumas	n60w180.001.txt	556	556	556	120.08	556	556	85.01
GendreauDumas	n60w180.002.txt	561	561	561	206.83	561	561	59.32
GendreauDumas	n60w180.003.txt	606	606	606	100.03	606	606	64.04

Table A.16 (**Part 5 of 7**) Makespan Results of PnB at  $\omega = 2048$  on Closed Problems: Seeded and Unseeded

LB = Lower Bound, UB = Upper Bound.

Set	Problem Information		Single Thread: 2048			Seeded Single Thread: 2048		
	Name	Best Known Solution	LB	UB	Time	LB	UB	Time
GendreauDumas	n60w180.004.txt	629	629	629	264.17	629	629	110.83
GendreauDumas	n60w180.005.txt	528	528	528	219.24	528	528	102.20
GendreauDumas	n60w200.001.txt	526	526	526	846.26	526	526	645.07
GendreauDumas	n60w200.002.txt	572	572	572	235.71	572	572	68.66
GendreauDumas	n60w200.004.txt	575	575	575	286.27	575	575	53.71
GendreauDumas	n60w200.005.txt	618	618	618	292.92	618	618	159.49
GendreauDumas	n80w100.001.txt	676	676	676	966.52	676	676	43.50
GendreauDumas	n80w100.002.txt	721	721	721	1071.28	721	721	232.78
GendreauDumas	n80w100.003.txt	729	729	729	99.03	729	729	110.61
GendreauDumas	n80w100.004.txt	759	759	759	435.13	759	759	351.61
GendreauDumas	n80w100.005.txt	671	671	671	423.50	671	671	244.70
GendreauDumas	n80w120.001.txt	675	675	675	260.17	675	675	136.39
GendreauDumas	n80w120.002.txt	748	748	748	102.10	748	748	50.59
GendreauDumas	n80w120.003.txt	677	677	677	681.72	677	677	280.42
GendreauDumas	n80w120.004.txt	644	644	644	247.44	644	644	108.18
GendreauDumas	n80w120.005.txt	743	743	743	315.00	743	743	140.92
GendreauDumas	n80w140.001.txt	689	689	689	185.95	689	689	123.68
GendreauDumas	n80w140.002.txt	651	651	651	155.80	651	651	101.24
GendreauDumas	n80w140.003.txt	673	673	673	1408.64	673	673	303.24
GendreauDumas	n80w140.004.txt	612	612	612	326.50	612	612	122.16
GendreauDumas	n80w160.001.txt	624	624	624	217.47	624	624	91.49
GendreauDumas	n80w160.003.txt	690	690	690	2930.56	690	690	1014.22
GendreauDumas	n80w160.004.txt	655	655	655	944.97	655	655	389.99
GendreauDumas	n80w160.005.txt	645	645	645	488.20	645	645	325.87
GendreauDumas	n80w180.001.txt	678	678	678	349.91	678	678	98.66
GendreauDumas	n80w180.003.txt	680	680	680	148.69	680	680	98.86
GendreauDumas	n80w180.004.txt	659	659	659	689.71	659	659	554.48
GendreauDumas	n80w200.001.txt	626	626	626	271.52	626	626	79.82
GendreauDumas	n80w200.002.txt	638	638	638	916.82	638	638	637.81
GendreauDumas	n80w200.003.txt	679	679	679	548.18	679	679	164.24
GendreauDumas	n80w200.005.txt	621	621	621	2143.38	621	621	1627.26
Langevin	N20ft301.dat	661.6	661.6	661.6	0.06	661.6	661.6	0.19
Langevin	N20ft302.dat	703	703	703	0.06	703	703	0.19
Langevin	N20ft303.dat	746.4	746.4	746.4	0.06	746.4	746.4	0.19
Langevin	N20ft304.dat	817	817	817	0.06	817	817	0.19
Langevin	N20ft305.dat	724.7	724.7	724.7	0.06	724.7	724.7	0.19
Langevin	N20ft306.dat	729.5	729.5	729.5	0.06	729.5	729.5	0.19
Langevin	N20ft307.dat	691.8	691.8	691.8	0.06	691.8	691.8	0.20
Langevin	N20ft308.dat	788.2	788.2	788.2	0.06	788.2	788.2	0.24
Langevin	N20ft309.dat	751.8	751.8	751.8	0.06	751.8	751.8	0.19
Langevin	N20ft310.dat	693.8	693.8	693.8	0.06	693.8	693.8	0.21
Langevin	N20ft401.dat	660.9	660.9	660.9	0.07	660.9	660.9	0.20
Langevin	N20ft402.dat	701	701	701	0.07	701	701	0.23
Langevin	N20ft403.dat	746.4	746.4	746.4	0.08	746.4	746.4	0.19
Langevin	N20ft404.dat	817	817	817	0.08	817	817	0.21
Langevin	N20ft405.dat	724.7	724.7	724.7	0.08	724.7	724.7	0.20
Langevin	N20ft406.dat	728.5	728.5	728.5	0.06	728.5	728.5	0.22
Langevin	N20ft407.dat	691.8	691.8	691.8	0.06	691.8	691.8	0.20
Langevin	N20ft408.dat	786.1	786.1	786.1	0.06	786.1	786.1	0.19
Langevin	N20ft409.dat	749.8	749.8	749.8	0.06	749.8	749.8	0.21
Langevin	N20ft410.dat	693.8	693.8	693.8	0.06	693.8	693.8	0.21
Langevin	N40ft201.dat	1109.3	1109.3	1109.3	0.06	1109.3	1109.3	0.26
Langevin	N40ft202.dat	1017.4	1017.4	1017.4	0.06	1017.4	1017.4	0.25
Langevin	N40ft203.dat	903.1	903.1	903.1	0.09	903.1	903.1	0.25
Langevin	N40ft204.dat	897.4	897.4	897.4	0.06	897.4	897.4	0.27
Langevin	N40ft205.dat	983.6	983.6	983.6	0.07	983.6	983.6	0.27
Langevin	N40ft206.dat	1081.9	1081.9	1081.9	0.06	1081.9	1081.9	0.26
Langevin	N40ft207.dat	884.9	884.9	884.9	0.07	884.9	884.9	0.27
Langevin	N40ft208.dat	1051.6	1051.6	1051.6	0.08	1051.6	1051.6	0.24
Langevin	N40ft209.dat	1027.5	1027.5	1027.5	0.06	1027.5	1027.5	0.29
Langevin	N40ft210.dat	1035.3	1035.3	1035.3	0.06	1035.3	1035.3	0.27
Langevin	N40ft401.dat	1105.2	1105.2	1105.2	0.11	1105.2	1105.2	0.29
Langevin	N40ft402.dat	1016.4	1016.4	1016.4	0.06	1016.4	1016.4	0.29
Langevin	N40ft403.dat	903.1	903.1	903.1	2.57	903.1	903.1	0.41
Langevin	N40ft404.dat	897.4	897.4	897.4	0.16	897.4	897.4	0.34
Langevin	N40ft405.dat	982.6	982.6	982.6	1.10	982.6	982.6	0.37
Langevin	N40ft406.dat	1081.9	1081.9	1081.9	0.88	1081.9	1081.9	0.30
Langevin	N40ft407.dat	872.2	872.2	872.2	0.60	872.2	872.2	0.34

Table A.17 (**Part 6 of 7**) Makespan Results of PnB at  $\omega = 2048$  on Closed Problems: Seeded and Unseeded

LB = Lower Bound, UB = Upper Bound.

Set	Problem Information		Single Thread: 2048			Seeded Single Thread: 2048		
	Name	Best Known Solution	LB	UB	Time	LB	UB	Time
Langevin	N40ft408.dat	1043.5	1043.5	1043.5	0.25	1043.5	1043.5	0.33
Langevin	N40ft409.dat	1025.5	1025.5	1025.5	0.12	1025.5	1025.5	0.35
Langevin	N40ft410.dat	1034.3	1034.3	1034.3	0.07	1034.3	1034.3	0.30
Langevin	N60ft201.dat	1375.4	1375.4	1375.4	0.13	1375.4	1375.4	0.46
Langevin	N60ft202.dat	1186.4	1186.4	1186.4	0.09	1186.4	1186.4	0.44
Langevin	N60ft203.dat	1194.2	1194.2	1194.2	0.08	1194.2	1194.2	0.45
Langevin	N60ft204.dat	1283.6	1283.6	1283.6	0.09	1283.6	1283.6	0.53
Langevin	N60ft205.dat	1215.5	1215.5	1215.5	0.07	1215.5	1215.5	0.47
Langevin	N60ft206.dat	1238.8	1238.8	1238.8	0.34	1238.8	1238.8	0.57
Langevin	N60ft207.dat	1305.3	1305.3	1305.3	0.07	1305.3	1305.3	0.42
Langevin	N60ft208.dat	1172.6	1172.6	1172.6	1.17	1172.6	1172.6	0.47
Langevin	N60ft209.dat	1243.8	1243.8	1243.8	0.23	1243.8	1243.8	0.60
Langevin	N60ft210.dat	1273.2	1273.2	1273.2	2.22	1273.2	1273.2	0.51
Langevin	N60ft301.dat	1375.4	1375.4	1375.4	0.72	1375.4	1375.4	0.53
Langevin	N60ft302.dat	1184.4	1184.4	1184.4	2.10	1184.4	1184.4	0.57
Langevin	N60ft303.dat	1194.2	1194.2	1194.2	0.23	1194.2	1194.2	0.53
Langevin	N60ft304.dat	1283.6	1283.6	1283.6	3.33	1283.6	1283.6	0.63
Langevin	N60ft305.dat	1214.5	1214.5	1214.5	2.24	1214.5	1214.5	0.59
Langevin	N60ft306.dat	1237.8	1237.8	1237.8	4.87	1237.8	1237.8	0.65
Langevin	N60ft307.dat	1298.4	1298.4	1298.4	0.30	1298.4	1298.4	0.55
Langevin	N60ft308.dat	1168.8	1168.8	1168.8	3.93	1168.8	1168.8	0.56
Langevin	N60ft309.dat	1242.8	1242.8	1242.8	3.51	1242.8	1242.8	0.68
Langevin	N60ft310.dat	1273.2	1273.2	1273.2	2.91	1273.2	1273.2	1.98
Langevin	N60ft401.dat	1375.4	1375.4	1375.4	6.22	1375.4	1375.4	0.74
Langevin	N60ft402.dat	1183.4	1183.4	1183.4	4.65	1183.4	1183.4	1.46
Langevin	N60ft403.dat	1194.2	1194.2	1194.2	3.74	1194.2	1194.2	1.81
Langevin	N60ft404.dat	1283.6	1283.6	1283.6	5.46	1283.6	1283.6	0.77
Langevin	N60ft405.dat	1212.5	1212.5	1212.5	4.10	1212.5	1212.5	0.85
Langevin	N60ft406.dat	1236.8	1236.8	1236.8	10.36	1236.8	1236.8	4.25
Langevin	N60ft407.dat	1296.4	1296.4	1296.4	4.46	1296.4	1296.4	0.66
Langevin	N60ft408.dat	1150	1150	1150	5.79	1150	1150	1.41
Langevin	N60ft409.dat	1241.8	1241.8	1241.8	6.75	1241.8	1241.8	1.18
Langevin	N60ft410.dat	1273.2	1273.2	1273.2	10.13	1273.2	1273.2	3.92
OhlmannThomas	n150w120.001.txt	972	972	1164.11		972	972	825.73
OhlmannThomas	n150w120.002.txt	917	917	654.04		917	917	561.33
OhlmannThomas	n150w120.003.txt	909	909	1189.99		909	909	1001.17
OhlmannThomas	n150w120.005.txt	907	907	1591.25		907	907	876.81
OhlmannThomas	n150w140.001.txt	1008	1008	462.98		1008	1008	342.39
OhlmannThomas	n150w140.003.txt	844	844	3630.57		844	844	519.89
OhlmannThomas	n150w160.001.txt	959	959	1907.07		959	959	553.85
OhlmannThomas	n150w160.003.txt	934	934	2623.29		934	934	581.80
OhlmannThomas	n150w160.005.txt	920	920	2762.04		920	920	1329.53
OhlmannThomas	n200w120.003.txt	1128	1128	1855.39		1128	1128	696.30
OhlmannThomas	n200w120.004.txt	1072	1072	1564.66		1072	1072	1445.56
OhlmannThomas	n200w120.005.txt	1073	1073	1665.78		1073	1073	599.20
OhlmannThomas	n200w140.002.txt	1087	1087	3097.33		1087	1087	1927.46
OhlmannThomas	n200w140.004.txt	1100	1100	2265.02		1100	1100	791.76
SolomonPesant	rc201.0	853.71	853.71	853.71	3.17	853.71	853.71	1.84
SolomonPesant	rc201.1	850.48	850.48	850.48	5.02	850.48	850.48	2.08
SolomonPesant	rc201.2	883.97	883.97	883.97	4.70	883.97	883.97	1.31
SolomonPesant	rc201.3	722.43	722.43	722.43	1.62	722.43	722.43	0.31
SolomonPesant	rc202.0	850.48	850.48	850.48	10.65	850.48	850.48	7.31
SolomonPesant	rc202.1	702.28	702.28	702.28	19.59	702.28	702.28	7.96
SolomonPesant	rc202.2	853.71	853.71	853.71	11.40	853.71	853.71	7.55
SolomonPesant	rc202.3	883.97	883.97	883.97	42.31	883.97	883.97	27.06
SolomonPesant	rc203.0	870.52	870.52	870.52	247.92	870.52	870.52	48.78
SolomonPesant	rc203.1	850.48	850.48	850.48	41.22	850.48	850.48	24.01
SolomonPesant	rc203.2	853.71	853.71	853.71	53.03	853.71	853.71	9.52
SolomonPesant	rc204.0	839.24	839.24	839.24	36.25	839.24	839.24	24.46
SolomonPesant	rc204.1	492.60	492.60	492.60	254.83	492.60	492.60	62.26
SolomonPesant	rc205.0	834.62	834.62	834.62	11.58	834.62	834.62	9.49
SolomonPesant	rc205.1	899.24	899.24	899.24	6.65	899.24	899.24	6.72
SolomonPesant	rc205.2	908.79	908.79	908.79	12.72	908.79	908.79	10.47
SolomonPesant	rc205.3	684.21	684.21	684.21	10.51	684.21	684.21	8.72
SolomonPesant	rc206.0	893.21	893.21	893.21	40.68	893.21	893.21	22.23
SolomonPesant	rc206.1	756.45	756.45	756.45	61.77	756.45	756.45	19.49
SolomonPesant	rc206.2	776.19	776.19	776.19	115.81	776.19	776.19	32.61
SolomonPesant	rc207.0	847.63	847.63	847.63	224.89	847.63	847.63	102.04

Table A.18 (**Part 7 of 7**) Makespan Results of PnB at  $\omega = 2048$  on Closed Problems: Seeded and Unseeded

LB = Lower Bound, UB = Upper Bound.

Problem Information			Single Thread: 2048			Seeded Single Thread: 2048		
Set	Name	Best Known Solution	LB	UB	Time	LB	UB	Time
SolomonPesant	rc207.1	785.37	785.37	785.37	132.40	785.37	785.37	39.23
SolomonPesant	rc207.2	650.8	650.80	650.80	252.09	650.80	650.80	35.39
SolomonPesant	rc208.0	836.04	836.04	836.04	2256.13	836.04	836.04	1484.62
SolomonPesant	rc208.1	615.51	615.51	615.51	417.09	615.51	615.51	160.41
SolomonPesant	rc208.2	596.21	596.21	596.21	276.60	596.21	596.21	46.52
SolomonPotvinBengio	rc_201.1.txt	592.06	592.06	592.06	1.75	592.06	592.06	0.53
SolomonPotvinBengio	rc_201.2.txt	860.17	860.17	860.17	1.06	860.17	860.17	0.49
SolomonPotvinBengio	rc_201.3.txt	853.71	853.71	853.71	4.57	853.71	853.71	1.32
SolomonPotvinBengio	rc_201.4.txt	889.18	889.18	889.18	0.32	889.18	889.18	0.28
SolomonPotvinBengio	rc_202.1.txt	850.48	850.48	850.48	29.56	850.48	850.48	18.33
SolomonPotvinBengio	rc_202.2.txt	338.52	338.52	338.52	1.54	338.52	338.52	0.56
SolomonPotvinBengio	rc_202.3.txt	894.1	894.10	894.10	9.24	894.10	894.10	6.64
SolomonPotvinBengio	rc_202.4.txt	853.71	853.71	853.71	11.17	853.71	853.71	10.08
SolomonPotvinBengio	rc_203.1.txt	488.42	488.42	488.42	5.00	488.42	488.42	3.19
SolomonPotvinBengio	rc_203.2.txt	853.71	853.71	853.71	24.89	853.71	853.71	19.70
SolomonPotvinBengio	rc_203.3.txt	921.44	921.44	921.44	229.75	921.44	921.44	144.62
SolomonPotvinBengio	rc_203.4.txt	338.52	338.52	338.52	2.70	338.52	338.52	1.18
SolomonPotvinBengio	rc_204.2.txt	690.06	690.06	690.06	445.91	690.06	690.06	21.80
SolomonPotvinBengio	rc_204.3.txt	455.03	455.03	455.03	571.24	455.03	455.03	20.02
SolomonPotvinBengio	rc_205.1.txt	417.81	417.81	417.81	0.26	417.81	417.81	0.24
SolomonPotvinBengio	rc_205.2.txt	820.19	820.19	820.19	11.47	820.19	820.19	9.12
SolomonPotvinBengio	rc_205.3.txt	950.05	950.05	950.05	20.15	950.05	950.05	12.70
SolomonPotvinBengio	rc_205.4.txt	837.71	837.71	837.71	13.12	837.71	837.71	7.29
SolomonPotvinBengio	rc_206.1.txt	117.85	117.85	117.85	0.06	117.85	117.85	0.20
SolomonPotvinBengio	rc_206.2.txt	870.49	870.49	870.49	95.97	870.49	870.49	74.81
SolomonPotvinBengio	rc_206.3.txt	650.59	650.59	650.59	22.40	650.59	650.59	12.57
SolomonPotvinBengio	rc_206.4.txt	911.98	911.98	911.98	249.85	911.98	911.98	107.78
SolomonPotvinBengio	rc_207.1.txt	804.67	804.67	804.67	289.99	804.67	804.67	171.01
SolomonPotvinBengio	rc_207.2.txt	713.9	713.90	713.90	110.75	713.90	713.90	43.02
SolomonPotvinBengio	rc_207.3.txt	745.77	745.77	745.77	664.62	745.77	745.77	499.02
SolomonPotvinBengio	rc_207.4.txt	133.14	133.14	133.14	0.09	133.14	133.14	0.22
SolomonPotvinBengio	rc_208.1.txt	810.7	810.70	810.70	1870.15	810.70	810.70	560.76
SolomonPotvinBengio	rc_208.2.txt	579.51	579.51	579.51	496.96	579.51	579.51	91.32