

Carlos Eduardo de Andrade

**“Evolutionary Algorithms for some Problems in  
Telecommunications”**

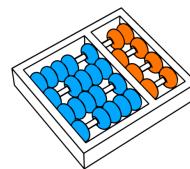
*“Algoritmos Evolutivos para alguns Problemas em  
Telecomunicações”*

CAMPINAS  
2015





University of Campinas  
Institute of Computing



*Universidade Estadual de Campinas  
Instituto de Computação*

Carlos Eduardo de Andrade

## “Evolutionary Algorithms for some Problems in Telecommunications”

Supervisor/*Orientador*: Prof. Dr. Flávio Keidi Miyazawa

Co-Supervisor/*Co-orientador*: Dr. Mauricio Guilherme de Carvalho Resende

## “Algoritmos Evolutivos para alguns Problemas em Telecomunicações”

PhD Thesis presented to the Post Graduate Program of the Institute of Computing of the University of Campinas to obtain a PhD degree in Computer Science.

*Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Computação da Universidade Estadual de Campinas para obtenção do título de Doutor em Ciência da Computação.*

THIS VOLUME CORRESPONDS TO THE FINAL VERSION OF THE THESIS DEFENDED BY CARLOS EDUARDO DE ANDRADE, UNDER THE SUPERVISION OF PROF. DR. FLÁVIO KEIDI MIYAZAWA.

ESTE EXEMPLAR CORRESPONDE À VERSÃO FINAL DA TESE DEFENDIDA POR CARLOS EDUARDO DE ANDRADE, SOB ORIENTAÇÃO DE PROF. DR. FLÁVIO KEIDI MIYAZAWA.

A handwritten signature in blue ink.

Supervisor's signature / Assinatura do Orientador

CAMPINAS  
2015

Ficha catalográfica  
Universidade Estadual de Campinas  
Biblioteca do Instituto de Matemática, Estatística e Computação Científica  
Ana Regina Machado - CRB 8/5467

An24e Andrade, Carlos Eduardo de, 1981-  
Andrade, Carlos Eduardo de. Evolutionary algorithms for some problems in telecommunications / Carlos Eduardo de Andrade. – Campinas, SP : [s.n.], 2015.  
  
Orientador: Flávio Keidi Miyazawa.  
Coorientador: Mauricio Guilherme de Carvalho Resende.  
Tese (doutorado) – Universidade Estadual de Campinas, Instituto de Computação.  
  
1. Otimização combinatória. 2. Redes de computadores - Projetos e construção. 3. Algoritmos genéticos. I. Miyazawa, Flávio Keidi, 1970-. II. Resende, Mauricio Guilherme de Carvalho. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Informações para Biblioteca Digital

**Título em outro idioma:** Algoritmos evolutivos para alguns problemas em telecomunicações

**Palavras-chave em inglês:**

Combinatorial optimization

Computer networks - Design and construction

Genetic algorithms

**Área de concentração:** Ciência da Computação

**Titulação:** Doutor em Ciência da Computação

**Banca examinadora:**

Mauricio Guilherme de Carvalho Resende [Coorientador]

Thiago Ferreira de Noronha

Luciana Salete Buriol

Cid Carvalho de Souza

Kelly Cristina Poldi

**Data de defesa:** 05-03-2015

**Programa de Pós-Graduação:** Ciência da Computação

## T E R M O   D E   A P R O V A Ç Ã O

Defesa de Tese de Doutorado em Ciência da Computação, apresentada pelo(a)  
Doutorando(a) **Carlos Eduardo de Andrade**, aprovado(a) em **05 de março de 2015**, pela  
Banca examinadora composta pelos Professores Doutores:

  
Prof(º). Dr(º). Thiago Ferreira de Noronha

Titular

  
Prof(º). Dr(º). Luciana Salete Buriol

Titular

  
Prof(º). Dr(º). Cid Carvalho de Souza

Titular

  
Prof(º). Dr(º). Kelly Cristina Poldi

Titular

  
Prof(º). Dr(º). Mauricio Guilherme de Carvalho Resende  
Presidente(a)



---

---

Institute of Computing / *Instituto de Computação*  
University of Campinas / *Universidade Estadual de Campinas*

---

# Evolutionary Algorithms for some Problems in Telecommunications

**Carlos Eduardo de Andrade<sup>1</sup>**

March 05, 2015

## Examiner Board/*Banca Examinadora:*

- Dr. Mauricio Guilherme de Carvalho Resende (Co-supervisor/*Co-orientador*)  
Amazon.com
- Prof. Dr. Thiago Ferreira de Noronha  
Department of Computer Science – Federal University of Minas Gerais
- Prof. Dr. Luciana Salete Buriol  
Institute of Informatics – Federal University of Rio Grande do Sul
- Prof. Dr. Cid Carvalho de Souza  
Institute of Computing – University of Campinas
- Prof. Dr. Kelly Cristina Poldi  
Institute of Mathematics, Statistics, and Scientific Computing – University of Campinas
- Prof. Dr. Débora Pretti Ronconi  
Production Engineering Department – University of São Paulo (Substitute/*Suplente*)
- Prof. Dr. Eduardo Cândido Xavier  
Institute of Computing – University of Campinas (Substitute/*Suplente*)
- Prof. Dr. Luis Augusto Angelotti Meira  
School of Technology – University of Campinas (Substitute/*Suplente*)

---

<sup>1</sup>Financial support: CAPES 2010; FAPESP grant 2010/05233-5 (2010–2015) and grant 2012/08222-0 (2012–2014)



# Abstract

In last twenty years, telecommunication networks have experienced a huge increase in data utilization. From massive on-demand video to uncountable mobile devices exchanging text and video, traffic reached scales that overcame the network capacities. Therefore, telecommunication companies around the world have been forced to increase their capacity to serve this increasing demand. As the cost to deploy network infrastructure is usually very large, the design of a network heavily uses optimization tools to keep costs as low as possible. In this thesis, we analyze several aspects of the design and deployment of communication networks. First, we present a new network design problem used to serve wireless demands from mobile devices and route the traffic to the core network. Such access networks are based on modern wireless access technologies such as Wi-Fi, LTE, and HSPA. This problem has several real world constraints and it is hard to solve. We study real cases of the vicinity of a large city in the United States. Following, we present a variation of the hub-location problem used to model these core networks. Such problem is also suitable to model transportation networks. We also study the overlapping correlation clustering problem used to model the user's behavior when using their mobile devices. In such problem, one can label an object with multiple labels and analyzes the connections between them. Although this problem is very generic, it is suitable to analyze device mobility which can be used to estimate traffic in geographical regions. Finally, we analyze spectrum licensing from a governmental perspective. In these cases, a governmental agency wants to sell rights for telecommunication companies to operate over a given spectrum range. This process usually is conducted using combinatorial auctions. For all problems we propose biased random-key genetic algorithms and mixed integer linear programming models (except in the case of the overlapping correlation clustering problem due its non-linear nature). Our algorithms were able to overcome the state of the art algorithms for all problems.

x

# Resumo

Nos últimos anos, as redes de telecomunicação tem experienciado um grande aumento no fluxo de dados. Desde a utilização massiva de vídeo sob demanda até o incontável número de dispositivos móveis trocando texto e vídeo, o tráfego alcançou uma escala capaz de superar a capacidade das redes atuais. Portanto, as companhias de telecomunicação ao redor do mundo tem sido forçadas a aumentar a capacidade de suas redes para servir esta crescente demanda. Como o custo de instalar uma infraestrutura de rede é geralmente muito grande, o projeto de redes usa fortemente ferramentas de otimização para manter os custos tão baixos quanto possível. Nesta tese, nós analisamos vários aspectos do projeto e implementação de redes de telecomunicação. Primeiramente, nós apresentamos um novo problema de projeto de redes usado para servir demandas sem fio de dispositivos móveis e rotear tal tráfego para a rede principal. Tais redes de acesso são baseadas em tecnologias sem fio modernas como Wi-Fi, LTE e HSPA. Este problema considera várias restrições reais e é difícil de ser resolvido. Nós estudamos casos reais nas vizinhanças de uma grande cidade nos Estados Unidos. Em seguida, nós apresentamos uma variação do problema de localização de hubs usado para modelar as redes principais (backbones ou laços centrais). Este problema também pode ser utilizado para modelar redes de transporte de cargas e passageiros. Nós também estudamos o problema de clusterização correlacionada com sobreposições usado para modelar o comportamento dos usuários quando utilizam seus equipamentos móveis. Neste problema, nós podemos rotular um objeto usando múltiplos rótulos e analisar a conexão entre eles. Este problema é adequado para análise de mobilidade de equipamentos que pode ser usada para estimar o tráfego em uma dada região. E finalmente, nós analisamos o licenciamento de espectro sobre uma perspectiva governamental. Nestes casos, uma agência do governo deseja vender licenças para companhias de telecomunicação para que operem em uma dada faixa de espectro. Este processo usualmente é conduzido usando leilões combinatoriais. Para todos problemas, nós propomos algoritmos genéticos de chaves aleatórias viciadas e modelos de programação linear inteira mista para resolvê-los (exceto para o problema de clusterização correlacionada com sobreposição, devido sua natureza não-linear). Os algoritmos que propusemos foram capazes de superar algoritmos do estado da arte para todos problemas.



*A minha esposa Andreia  
e minha avó Judith.*

*To my wife Andreia  
and my grandmother Judith.*



# Agradecimentos

Gostaria de agradecer meu orientador Prof. Flávio Keidi Miyazawa por seu constante suporte, orientação e, especialmente, por acreditar no meu trabalho. Também gostaria de agradecer meu coorientador Mauricio G. C. Resende por seu suporte, amizade, e por ter me introduzido em um distinto grupo de pesquisadores. Eu me considero muito afortunado por ter sido supervisionado por tão grandes pessoas e certamente não tenho palavras para expressar minha gratidão.

Eu agradeço meus colegas do Laboratório de Otimização e Combinatória da Unicamp e meus colegas do AT&T Labs Research (Florham Park and Middletown, NJ, EUA), especialmente meus co-autores Mauro Lopes, Howard Karloff, Weiyi (Max) Zhang, Rakesh Sinha, Kenneth Reichmann, and Robert Doverspike. Também deixo minha gratidão para o *staff* do IC/Unicamp por sua ajuda e cortesia. Agradeço a CAPES e a FAPESP pelo suporte financeiro.

Eu gostaria de agradecer Lúcia Resende por sua grande hospitalidade e por ter me recebido tantas vezes em sua casa. Eu agradeço Rodrigo Toso e Daniela Vianna por sua amizade, por ter abrigado minha esposa e eu em nosso primeiro mês nos EUA, e por todo suporte provido. Eu agradeço os amigos do departamento de engenharia química e ambiental da Universidade de Yale, em especial o grupo do prof. Menachem Elimelech, onde minha esposa tem feito seu pós-doutoramento, por momentos alegres e, algumas vezes, comida grátis. Eu agradeço a amizade oferecida por Carlos e Janaina Oliveira, Frank e Dora Bacchus, e Fernando Stefanello. Os últimos anos tem sido inestimáveis.

Eu agradeço toda minha família especialmente meus pais José Carlos e Nilza, e meus sogros Djair e Helena (*in memoriam*). Eu tenho um gratidão especial por minha avó Judith (*in memoriam*) por seus incondicionais apoio e amor.

Finalmente e mais importante, eu gostaria de agradecer muito minha esposa Andreia por seu amor e dedicação durante todos esses anos. Sem seu suporte e companheirismo, não teria sido possível realizar esta jornada. Seu trabalho duro, cortesia e delicadeza tem, e sempre o fará, inspirar e confortar-me. Eu sou muito grato a Deus por ter me permitido compartilhar minha vida com tão boa alma. Eu gostaria tem apenas uma fração da bondade dela. Existem muitas poucas mulheres como Andreia neste mundo e eu sou, certamente, muito sortudo de tê-la ao meu lado.



*Combati o bom combate, completei a corrida e guardei a Fé.*  
Segunda Carta de São Paulo a Timóteo, Capítulo 4, Versículo 7

*I have fought a good fight, I have finished my course, I have kept the faith.*  
The Second Epistle of St. Paul to Timothy, Chapter 4, Verse 7



# Contents

<b>Abstract</b>	<b>ix</b>
<b>Resumo</b>	<b>xi</b>
<b>Dedication</b>	<b>xiii</b>
<b>Agradecimentos</b>	<b>xv</b>
<b>Epigraph</b>	<b>xvii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Definitions and main techniques . . . . .	2
1.2 Biased random-key genetic algorithms . . . . .	5
1.2.1 Basic framework . . . . .	5
1.2.2 Decoding . . . . .	7
1.2.3 Improvements . . . . .	9
1.2.4 Parameters . . . . .	10
1.2.5 Successful use cases . . . . .	10
1.3 Results and thesis organization . . . . .	11
<b>2 The Wireless Backhaul Network Design Problem</b>	<b>15</b>
2.1 Introduction . . . . .	16
2.2 Problem description . . . . .	17
2.2.1 Demand splitting and routing trees . . . . .	18
2.2.2 Capacity of access equipment . . . . .	18
2.2.3 Capacity of retransmitter equipment . . . . .	18
2.2.4 Sight and minimum distance . . . . .	19
2.2.5 Number of hops . . . . .	20
2.3 Related literature . . . . .	22
2.4 Formal Definition . . . . .	24

2.4.1	Mixed integer linear programming model . . . . .	27
2.5	Solution procedure using BRKGA . . . . .	30
2.5.1	Representation . . . . .	31
2.5.2	Decoder . . . . .	32
2.6	Maximum Backhaul Flow Problem . . . . .	39
2.6.1	Bounds . . . . .	39
2.6.2	Solution approach . . . . .	42
2.7	Experimental Setup . . . . .	46
2.7.1	Instances and scenario descriptions . . . . .	46
2.7.2	Instance preprocessing . . . . .	47
2.7.3	Post-optimization flow recomputation . . . . .	49
2.7.4	Computational environment and parameters . . . . .	49
2.8	Experimental Results and Discussion . . . . .	50
2.8.1	Instance preprocessing . . . . .	50
2.8.2	Computing flow during the optimization . . . . .	51
2.8.3	Comparing the profit generated by the algorithms . . . . .	53
2.8.4	Analyzing a solution . . . . .	57
2.9	Final considerations . . . . .	59
<b>3</b>	<b>The <math>k</math>-Interconnected Multi-Depot Multi-Travelling Salesmen Problem for backbone network design</b>	<b>63</b>
3.1	Introduction . . . . .	64
3.2	Related Work . . . . .	64
3.3	Definitions . . . . .	66
3.4	BRKGA for the $k$ -IMDMTSP . . . . .	69
3.4.1	Representation . . . . .	69
3.4.2	Decoding a solution . . . . .	70
3.4.3	Initial Population . . . . .	74
3.5	Experimental Results . . . . .	74
3.5.1	Computational environment . . . . .	75
3.5.2	Algorithm settings . . . . .	75
3.5.3	Instances . . . . .	76
3.5.4	Results and Discussion . . . . .	76
3.6	Concluding Remarks . . . . .	79
<b>4</b>	<b>The Overlapping Correlation Clustering Problem</b>	<b>81</b>
4.1	Introduction . . . . .	82
4.2	Definitions . . . . .	83
4.3	Biased random-key genetic algorithms and local search . . . . .	84

4.3.1	Representation . . . . .	85
4.3.2	Decoding a solution . . . . .	86
4.3.3	Error Reduction Local Search . . . . .	88
4.3.4	Bonchi et al. Local Search . . . . .	90
4.4	Experimental results . . . . .	91
4.4.1	Instances . . . . .	91
4.4.2	Evaluated algorithms . . . . .	93
4.4.3	Computational environment and parameters . . . . .	94
4.4.4	Defining maximum running times for BRKGA . . . . .	94
4.4.5	Evaluating the quality of the algorithms on ground-truth instances	96
4.4.6	Evaluating the algorithms for instances with unknown multi-labeling	99
4.5	Concluding remarks . . . . .	102
<b>5</b>	<b>The Winner Determination Problem in Combinatorial Auctions</b>	<b>103</b>
5.1	Introduction . . . . .	104
5.2	General combinatorial auctions and their formulations . . . . .	106
5.3	Biased random-key genetic algorithms for the winner determination problem	108
5.4	Initializing the population of BRKGA . . . . .	112
5.5	Experimental Setup . . . . .	114
5.5.1	Instances . . . . .	115
5.5.2	Algorithms . . . . .	117
5.5.3	Computational environment and algorithm settings . . . . .	122
5.6	Experimental Results and Discussion . . . . .	123
5.6.1	Comparing revenue . . . . .	123
5.6.2	Iterations and runtime analyses . . . . .	128
5.6.3	Comparing the heuristics on hard instances . . . . .	131
5.6.4	Comparing heuristics on small number of generations . . . . .	131
5.6.5	Effect of LP-based initialization . . . . .	134
5.7	Final Remarks . . . . .	136
<b>6</b>	<b>Concluding remarks</b>	<b>137</b>
<b>Bibliography</b>		<b>141</b>
<b>A Additional results for Chapter 2</b>		<b>161</b>
A.1	Instance details . . . . .	161
A.2	Experimental results . . . . .	162

<b>B Additional results for Chapter 3</b>	<b>167</b>
B.1 Detailed results . . . . .	167
<b>C Additional results for Chapter 4</b>	<b>173</b>
C.1 Additional plots of Section 4.4.4 . . . . .	173
C.2 Statistical test tables of Section 4.4.6 . . . . .	176
<b>D Additional results for Chapter 5</b>	<b>179</b>
D.1 Statistical tests . . . . .	179
D.2 Additional running time results . . . . .	184
D.3 Best results for each instance . . . . .	186

# List of Tables

2.1	Summary of characteristics for WBNDP instances . . . . .	47
2.2	Description of parameters for WBNDP instances . . . . .	48
2.3	Difference in median location for profit distributions . . . . .	55
2.4	Algorithm performance . . . . .	56
3.1	Median test for cost distributions ( $k$ -IMDMTSP) . . . . .	78
3.2	BRKGA means of best results . . . . .	78
4.1	Algorithms that computed the best results (OCC) . . . . .	102
5.1	Instance classes and sizes of the WDP . . . . .	116
5.2	Average of instances tightness . . . . .	118
5.3	Algorithm performance . . . . .	127
5.4	Average of iterations in finding the best solution . . . . .	129
5.5	Ratio between the revenue of LP-based and the best chromosomes . . . . .	134
5.6	Ratio between the revenue of LP-based and the best chromosomes by instance class . . . . .	135
A.1	Detailed characteristics of the instances for WBNDP . . . . .	161
A.2	Instance characteristics after preprocessing – restricted scenario . . . . .	162
A.3	Instance characteristics after preprocessing – unrestricted scenario . . . . .	163
A.4	Best results – 5hr, restricted scenario . . . . .	164
A.5	Best results – unrestricted scenario . . . . .	165
B.1	Best results for ST scenario ( $k$ -IMDMTSP) . . . . .	168
B.2	Best results for SL scenario . . . . .	169
B.3	Best results for LT scenario . . . . .	170
B.4	Best results for LL scenario . . . . .	171
B.5	Best results for SQ scenario . . . . .	172
C.1	Difference in median location for cost distributions for Starkey dataset (OCC) . . . . .	176

C.2	Difference in median location for cost distributions for SCOP datasets . . .	177
C.3	Difference in median location for cost distributions for newsgroup messages	178
D.1	Difference in median location for revenue distributions for all instances (WDP) . . . . .	180
D.2	Difference in median location for revenue distributions for instances with 400 bids or less . . . . .	180
D.3	Difference in median location for revenue distributions for instances with 1000 bids or more . . . . .	181
D.4	Difference in median location for revenue distributions for LG 1500/1500 instances . . . . .	181
D.5	Difference in median location of revenue distributions for all instances, considering the best solutions until 100 generations . . . . .	182
D.6	Difference in median location of revenue distributions for instances with 400 bids or less, considering the best solutions until 100 generations . . . . .	182
D.7	Difference in median location of revenue distributions for instances with more than 400 bids, considering the best solutions until 100 generations . . . . .	183
D.8	Difference in median location of revenue distributions for for LG 1500/1500 instances, considering the best solutions until 100 generations . . . . .	183
D.9	Running time comparison among the algorithms . . . . .	185
D.10	Best results for CATS instances with less than 400 bids . . . . .	186
D.11	Best results for CATS instances more than 400 bids . . . . .	188
D.12	Best results for LG 1000/500 instances . . . . .	191
D.13	Best results for LG 1000/1000 instances . . . . .	193
D.14	Best results for LG 1500/1500 instances . . . . .	195

# List of Figures

1.1	Illustration of an iteration of BRKGA . . . . .	7
1.2	Example of two decoders for the knapsack problem . . . . .	8
2.1	Example of instance of the WBNDP . . . . .	21
2.2	Evolution of the profit using different flow algorithms . . . . .	52
2.3	Dispersion of profit for each algorithm . . . . .	54
2.4	Evolution of revenue, cost, profit, and coverage . . . . .	57
2.5	Example of region . . . . .	60
2.6	Example of coverage . . . . .	61
2.7	Example of backhaul network . . . . .	62
3.1	Examples of a simple and degenerate cases of $k$ -IMDMTSP . . . . .	68
3.2	Example of solution extraction . . . . .	70
3.3	Local improvement steps . . . . .	72
3.4	Boxplot of the medians . . . . .	77
3.5	Time-to-target plots . . . . .	80
4.1	Compact representation of a solution by a chromosome (OCC) . . . . .	85
4.2	Evolution of the cost for the EMOTIONS dataset . . . . .	95
4.3	Comparison of costs, precision, and recall for the EMOTIONS dataset . . . . .	97
4.4	Comparison of costs, precision, and recall for the YEAST dataset . . . . .	98
4.5	Boxplot of median and quartiles in Starkey dataset . . . . .	100
4.6	Boxplot of median and quartiles for each algorithm in SCOP dataset . . . . .	101
4.7	Boxplot of median and quartiles for each algorithm in newsgroup messages	101
5.1	Example of decoding for the WDP . . . . .	110
5.2	Dispersion of revenue for each algorithm . . . . .	125
5.3	Running time distributions for each algorithm . . . . .	130
5.4	Dispersion of revenue LG 1500/1500 instances . . . . .	132
5.5	Dispersion of revenue using 100 generations at most . . . . .	133

C.1	Evolution of the cost for the YEAST dataset (OCC) . . . . .	173
C.2	Evolution of the cost for the Starkey project dataset . . . . .	174
C.3	Evolution of the cost for the protein alignment dataset 1 . . . . .	174
C.4	Evolution of the cost for the newsgroup messages . . . . .	175

# List of Algorithms

1.1	BRKGA scheme . . . . .	6
2.1	Decoder for the WBNDP . . . . .	33
2.2	Equipment activation and installation . . . . .	34
2.3	Building of backhaul forest (level 0) . . . . .	35
2.4	Building of backhaul forest (level $\geq 1$ ) . . . . .	37
2.5	Equipment and poles pruning . . . . .	38
-	Procedure $\text{pruneSubtree}(\cdot)$ . . . . .	38
2.6	Pumping Root . . . . .	44
2.7	Pump pole level 0 . . . . .	44
2.8	Pump pole level $\geq 1$ . . . . .	45
3.1	Adjust cycle sizes ( $k$ -IMDMTSP) . . . . .	71
3.2	Decoder for $k$ -IMDMTSP . . . . .	73
4.1	Decoder for OCC – Phase 1 . . . . .	87
4.2	Error Reduction Local Search – OLS . . . . .	89
4.3	Bonchi et al. Local Search – BSL . . . . .	91
5.1	Decoder for the WDP . . . . .	111
5.2	Initialization by LP relaxations . . . . .	113



# Chapter 1

## Introduction

SINCE the first deployment of a copper wire between two telegraphs in separate rooms in 1832, telecommunication networks have expanded healthily. These ancient “digital-like” telegraph networks of the 19<sup>th</sup> century brought to humankind a powerful tool to satisfy the necessity of communication of our modern society. Several technologies were developed in the past 180 years, and other thousand were imagined. Indeed, some prophesied tools became reality. In the Star Trek TV series of the early 60’s, the members of the star fleet usually make use of a device called a “communicator”. The communicator was used for direct communication between individuals or via the ship’s command center. This device can be considered the fictional father of the modern mobile phones. In the same TV series, video conferencing was also usually used, foreseeing the video conference calls of current days. One can find several examples of such devices in the popular media, again showing the necessity for communication.

Although great part of population have gained access to such communication devices, the history of such devices brings us back to the beginning of 20<sup>th</sup> century. At that time, the telephone was invented and the first intercontinental submarine cables deployed. Wireless communications also dates back to that time, although the concept of cell phones was only proposed in 1947. However, only at the end of 20<sup>th</sup> century, did phone calls, wired or wireless, become accessible to most people due to the steep decrease in cost.

Concurrent with the evolution of the communication between people, the increasing of utilization of computing machines also brought with it the necessity for communication between equipment. In 1963, the idea of an “intergalactic computer network” emerged in the United States Department of Defense. This network, called ARPANET, was first deployed in 1969 connecting the University of California in Los Angeles (UCLA), the University of California in Santa Barbara, the Stanford Research Institute’s Augmentation Research Center, and the University of Utah. The first message on the ARPANET was sent from UCLA in the same year. The ARPANET was a packet switching network and

was the foundation of the modern Internet. The commercial Internet was launched in the 1990's and its widespread adoption occurred in the first decade of the 2000's. Primary connecting computers, such as desktops and servers, most links were wired using copper wires and optical fiber.

Currently, we have seen a huge increase in the utilization of voice and data. In private communication, text, voice, photos, and videos are exchanged in an incredible rate. The entertainment industry has also pushed terabytes of content into the internet: almost all TV show can be watched on the Internet, and some shows are specifically produced to be streamed over the Internet. Streaming of meetings, sport events, on-line classes, on-line gaming also contribute to the congestion of these networks. More and more, this content is being consumed on mobile devices using wireless networks. For example, in the United States, mobile devices now account for more than 50% of Internet usage (O'Toole [160]). Cisco VNI Global IP Traffic Forecast predicts a 61% annual growth rate for mobile data, resulting in an 11-fold increase from 2013 to 2018 (Cisco [37]). Therefore, wired and wireless networks have suffered large congestions and telecommunication companies have been forced to expand their networks. This is a trillionaire market and fine optimizations must be carried out to achieve economical efficiency.

This thesis investigates several optimization problems that appear in the planning and deployment of telecommunication networks and related activities. Problems in several levels are considered: from the market of electromagnetic spectrum for wireless communications to the planning of core and access networks. This thesis also studies a clustering problem used in analysis of network utilization. This work is oriented from a practical point of view, and the objective is to obtain solutions that can be used in real scenarios. Several algorithms are proposed and a novel family of problems are devised. The work presented here makes heavy use of biased random-key genetic algorithms and local search procedures. We also use branch-and-bound algorithms provided by commercial solvers. Our techniques overcome the state of the art algorithms for the known problems and present very good results for the novel problems.

## 1.1 Definitions and main techniques

According to Pedrosa [164], an optimization problem is the task to find the best solution among a well-defined set of feasible solutions. There are several classes that an optimization problem can lie. These classes are usually linked to the space of the set of solutions and the constraints that define the problem. In a linear continuous problem, the set of feasible solutions is defined in the space of the rational numbers and the constraints and objective function can be described using linear inequalities, i.e., the feasible solutions are in a convex space limited by the polytope defined by the constraints.

Such problems can be solved in polynomial time (in the size of the input) by the ellipsoid algorithm (Khachiyan [110]) or Karmarkar's interior point method (Karmarkar [108]). In practice, most linear programming problems are solved using the simplex algorithm (Dantzig [44]). Although simplex is an exponential-time algorithm, it is often faster than the other algorithms in practice, and offers duality information while solving the problem. Another class of optimization problems is the class of the non-linear problems where the objective function and/or constraints cannot be described by linear inequalities. Such problems are in general referred as global optimization problems. For more details see Horst et al. [99].

An important class of optimization problems are the combinatorial optimization problems. In such problems, the feasible solutions lie in a discrete space. For example, suppose we must determine the best schedule for crews of an airline company. Picking half of a pilot or 5.25 flight attendants is impossible. The solution for this problem must have a positive integer number of pilots and flight attendants. Several practical problems in real life are combinatorial optimization problems. Some of them can be solved in deterministic polynomial time with respect to the input size, e.g., the maximum flow problem (Edmonds and Karp [53], Ford Jr. and Fulkerson [62], Goldberg and Tarjan [71]), the shortest path problem (Bellman [17], Dijkstra [48]), and the minimum spanning tree problem (Boruvka [24], Kruskal [115], Prim [169]). However, for most practical combinatorial optimization problems such as routing (Applegate et al. [9], Lahyani et al. [118], Savelsbergh and Sol [187]), packing (Kantorovic [106], Martello and Toth [141], Wäscher et al. [205]), location (Melo et al. [146], Pedrosa [164], Prodhon and Prins [171]), scheduling (Allahverdi et al. [3], Graham [88], Vance et al. [198]), and network design problems (Pathak and Dutta [163], Prömel and Steger [172], Winter [208]), a deterministic polynomial time algorithm is unknown. Such problems belong to the  $\mathcal{NP}$ -hard class (Garey and Johnson [65]) and most researchers believe that there are no deterministic polynomial time algorithms to solve them to the optimality. Currently, only enumeration algorithms and pseudo-polynomial time algorithms are able to solve these problems to optimality. Unfortunately, the worst case running-times for enumeration algorithms for these problems are exponential. Some weakly  $\mathcal{NP}$ -hard problems, such as the 0-1 knapsack problem (Martello and Toth [141]), admits a pseudo-polynomial time algorithm. Although these algorithms are fast in practice, their running-times are polynomial in the size of the problem and in the magnitudes of the data involved (provided these are given as integers).

Several techniques have been proposed to solve combinatorial optimization problems, namely exact algorithms, approximation algorithms, probabilistic algorithms, and heuristics. Exact algorithms are able to guarantee that the best solution will be found in the end of the optimization when carried out completely. Generally, this is done observing the gap between the lower and upper bound values for feasible solutions. Exact algorithms

usually follow some enumeration procedure and use some techniques to prune the search space. Branch-and-bound algorithms (Land and Doig [119]) are the most used exact algorithms and several variants of branch-and-bound have been proposed in the literature. This thesis uses some branch-and-bound algorithms although indirectly through commercial solvers. One may refer to Nemhauser and Wolsey [155] and Wolsey [209] for more details.

Approximation algorithms are able to produce, in polynomial time, a solution with value guaranteed to be close to the value of an optimal solution. Such algorithms define an approximation factor that can be a constant or a ratio. Although approximation algorithms are not commonly used in practice, they are invaluable to better understand the underlying structure of the treated problem. This thesis does not make use of approximation algorithms. The interested reader can consult Vazirani [200] or Williamson and Shmoys [207].

Probabilistic algorithms follow the same idea of approximation algorithms but present their results in a given probability. Such algorithms use a random signal to make decisions about the optimization process. In general, probabilistic algorithms are simple albeit their analyses are complex. We do not make use of probabilistic algorithms in this thesis. The reader may refer to Mitzenmacher and Upfal [148] and Motwani and Raghavan [150] for more details.

A heuristic is a method to obtain a good solution for a given problem, however, without offering a guarantee in the quality of this solution. In general, a heuristic takes into account the problem structure and improvement steps are done considering this structure. Constructive heuristics generally start from an infeasible solution and, in each iteration, a move is done trying to achieve the feasibility. Local search heuristics start from a feasible solution and, in each iteration, explore a certain neighborhood of this solution trying to improve it. These neighborhoods are defined on the problem structure and can be complex. For example, in the traveling salesman problem, the 2-opt neighborhood consists in exchanging two edges in the cycle for two other edges that are not in the cycle, trying to improve the cost (Applegate et al. [9]). While heuristics are designed for a specific problem, metaheuristics are more general frameworks that guide the search in a well-defined way. Some metaheuristics make allusions to natural processes such as simulated annealing (Černý [32], Kirkpatrick et al. [111]), genetic algorithms (Goldberg [72], Mitchell [147]), ant colony optimization (Dorigo et al. [51], Dorigo and Stützle [52]), particle swarm optimization (Kennedy and Eberhart [109], Olsson [159]), and variants. Other metaheuristics only describe the search procedure such as tabu search (Glover [67], Glover and Laguna [68]), greedy randomized adaptive search procedure (Feo and Resende [59], Resende and Ribeiro [179]), and variable neighborhood search (Hansen et al. [93], Mladenović and Hansen [149]). For other metaheuristics and variations, refer to Gendreau and Potvin [66].

This thesis makes heavy use of a particular variant of genetic algorithms called biased random-key genetic algorithms which is described in detail in the next section.

## 1.2 Biased random-key genetic algorithms

### 1.2.1 Basic framework

Before we discuss the origins and main characteristics of the Biased Random-Key Genetic Algorithms (BRKGAs), let us recall some terminology of genetic algorithms: the *chromosome*  $\mathbf{c}$  is a vector in a certain space  $\mathcal{H}^n$  such that  $n$  is the number of components. Each component  $\mathbf{c}_i$ , for  $i = 1, \dots, n$ , is called *gene* and its value is called *allele*. The genetic algorithm keeps a pool of vectors called *population* and for each chromosome in this population, a *fitness function* is applied to evaluate these chromosomes. In general, the fitness function constructs a solution from the chromosome and calculates its value. These values are used as *fitness* measures of each individual. The evolutionary step consists in building a new population by combining individuals of the current population, in general, selecting alleles from them to create offspring. An additional step, called *mutation*, is applied with low probability when an allele is chosen and modified randomly. The great advantage of genetic algorithms is to combine two or more different solutions. This is easier to achieve by keeping a pool of solutions in a different space of solutions (chromosome space) from the original space of solutions of the problem to be solved. Using the concept of genetic operators and inheritance, genetic algorithms are able to combine portions of good solutions and keep them “alive”. There is a large number of variations of this process and a comprehensive list can be found in Goldberg [72].

One of the main issues in the design of a genetic algorithm for solving a problem is how to represent or encode a solution, i.e., how to define the chromosome space. In 1994, Bean [14] proposed a standardized chromosome representation using a unit hypercube. To obtain a solution, it is necessary to apply a function  $f : [0, 1]^n \rightarrow \mathcal{S}$  that maps a vector in the  $n$ -dimensional space  $[0, 1]^n$  to a solution in the space  $\mathcal{S}$  which is the space of solutions of the problem to be solved. Such function is called *decoder*. Bean named his algorithm *Random-Key Genetic Algorithm* (RKGA) and applied this algorithm to machine scheduling problems where one seeks a sequence of machines to perform operations such that the completion time of the last operation is minimized.

The *Biased Random-Key Genetic Algorithm* (BRKGA) was introduced in Gonçalves and de Almeida [80] and Ericsson et al. [55] and follows the same structure of RKGAs with respect to chromosome representation. Three key features distinguish BRKGAs from traditional genetic algorithms:

1. A standardized chromosome encoding that uses a vector with  $n$  uniformly drawn random keys (*alleles*) over the interval  $[0, 1]$  as proposed by Bean [14];
2. A well-defined evolutionary process which uses parameterized uniform crossover for exploitation (Spears and DeJong [194]);
3. The substitution of the application of the mutation operator on existing chromosomes with newly introduced *mutants* — defined as  $n$ -long vectors of (uniformly drawn) random keys — for exploration.

Algorithm 1.1 summarizes a typical BRKGA framework. Basically, we generate  $p$  chromosomes as initial individuals using vectors with  $n$  uniformly drawn random keys over the interval  $[0, 1]$ . At each iteration, a problem-specific *decoder* extracts a solution from each chromosome whose value is used as the fitness of that chromosome. To build a new population, we copy the best  $p_e$  individuals (called the *elite set*), add  $p_\mu$  random chromosomes (the *mutants*), and generate  $p - p_e - p_\mu$  offspring by applying the crossover operator. The crossover operator is the component of genetic algorithms responsible to combine two or more chromosomes generating new ones. Crossover is done between a random individual from the elite set and an individual from the remainder of the population: an offspring is generated by *mating*, where we take each allele from the elite parent with probability  $\rho_e$  or from the other parent with probability  $1 - \rho_e$ . Figure 1.1 exemplifies this procedure. With  $\rho_e = 0.5$ , the standard uniform crossover occurs. With  $\rho_e > 0.5$  by definition, intensification happens at two levels: when parents are selected, because one is drawn from the elite set, and when offspring are conceived, because their alleles

---

**Algorithm 1.1:** BRKGA scheme.

---

- 1 Generate the initial population  $P$ ;
  - 2 **while** a stopping criteria is not reached **do**
  - 3   **Decode** each chromosome of  $P$  and extract their solutions and fitness;
  - 4   Sort the population  $P$  in non-increasing order of fitness. Consider the top  $p_e$  individuals as the elite group  $E$ ;
  - 5   Copy  $E$  to the next generation  $Q$ , unaltered;
  - 6   Add  $p_\mu$  randomly-generated new chromosomes (*mutants*) to  $Q$ ;
  - 7   Generate  $p - p_e - p_\mu$  chromosomes (*offspring*) by parameterized crossover, selecting a random parent from  $E$  and another from  $P \setminus E$ . Add them to  $Q$ ;
  - 8    $P \leftarrow Q$ ;
  - 9 **return** best individual found.
-

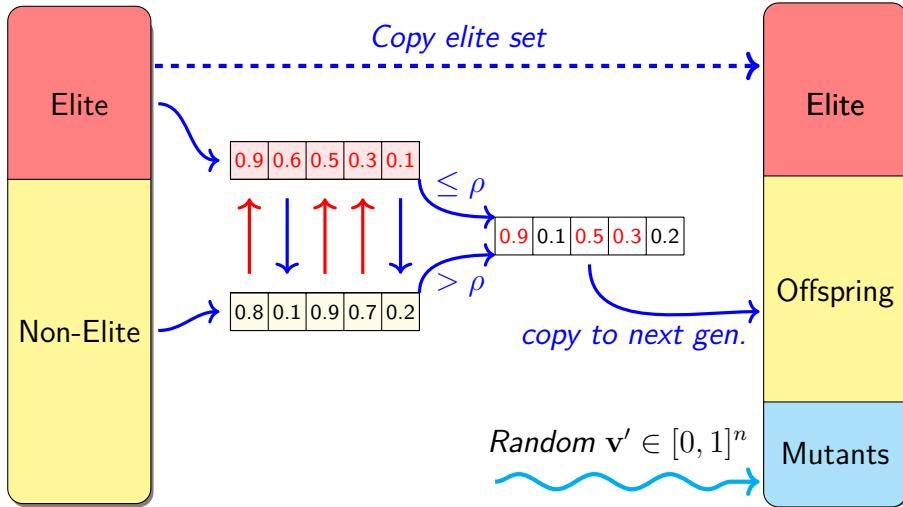


Figure 1.1: Illustration of an iteration of BRKGA. Note that the mating is done in favor to the elite individual (in red) when  $\rho > 0.5$ .

are inherited from the elite parent with greater probability. Diversification occurs with the introduction of mutants at each generation, since they are vectors with  $n$  uniformly drawn random keys. The usual mutation operators on individual genes are not employed by the BRKGA. Observe that the above scheme prevents infeasibility with respect to the chromosome space since, by definition, the resulting chromosomes — both offspring and mutants — are always vectors of random keys over  $[0, 1]$ .

### 1.2.2 Decoding

The decoding process makes use of a function  $f : [0, 1]^n \rightarrow \mathcal{S}$ , where  $n$  is a given number and  $\mathcal{S}$  is the space of solutions of the problem to be solved. Function  $f$  is usually called a *decoder*. To use the BRKGA, one must define the size of the chromosome  $n$  and build the decoder  $f$ . Such task is problem dependent and can be done in several ways. The alleles can be used as indicators values, induce permutations, or yet, be combined to produce complex structures.

To illustrate some decoding alternatives, consider the knapsack problem (Martello and Toth [141]). This problem consists in packing a set of items  $\mathcal{I} = \{1, \dots, n\}$  into a knapsack of capacity  $C$ . Each item  $i \in \mathcal{I}$  has a weight  $w_i$  and a value  $v_i$ , and the objective is to maximize the sum of the values of packed items respecting the knapsack capacity. This problem is well-known in the literature and, in fact, it does not require a sophisticated genetic algorithm to solve. We use the knapsack problem only for illustrative purposes. First, one must determine the size of the chromosome. In this case, let  $n = |\mathcal{I}|$ . Each gene of the chromosome, represented by the vector  $x \in [0, 1]^n$ , will be tied to an item. We call

$x_i$  the *key* of item  $i$ . Two simple decoders can be built from this representation. The first decoder consists in picking, in order, the items that have their keys greater than 0.5. If the current item cannot be packed due to the capacity constraint, it is just ignored. Decoders of this type are usually referred as indicator decoders. In the second decoder, the items are sorted according to the value of their keys generating a permutation. Next, one packs the items according to this permutation, ignoring those that cannot be packed due to the capacity constraint. Decoders of this type are usually referred as permutation decoders. Figure 1.2 depicts an example of these two decoders. Consider  $\mathcal{I} = \{1, 2, 3, 4\}$  with weights  $\{5, 1, 7, 4\}$  and values  $\{5, 3, 20, 10\}$ , respectively, and a knapsack of capacity  $C = 10$ . The indicator decoder first picks item 1, skips item 2 due to  $v_2 < 0.5$ , also skips item 3 since it does not fit in the knapsack together item 1, and finally picks item 4 resulting in a solution with value 15. In the permutation decoder, the items are sorted according to their keys resulting in the permutation  $(3, 1, 4, 2)$ . Then, the decoder proceeds picking item 3, skipping items 1 and 4 due to their size, and finally picking item 2. The solution in this case has value 25.

The decoders presented above are simple but very effective. In Chapter 5, simple decoders were used and were able to outperform state of the art algorithms for the winner

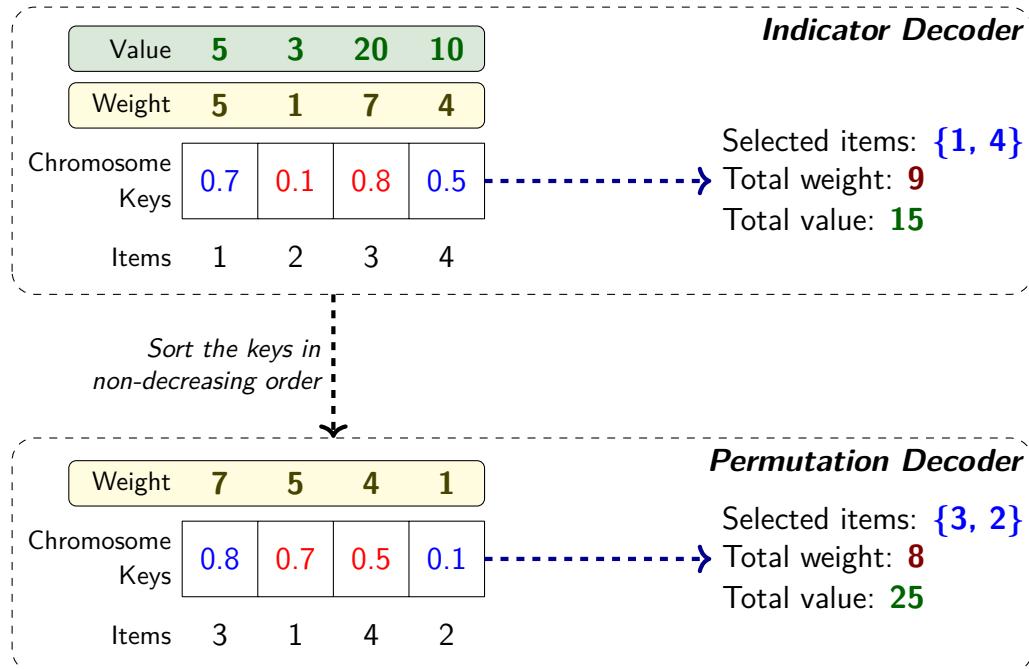


Figure 1.2: Example of two decoders for the knapsack problem. In the indicator decoder, the items are picked in the order that they appear. In the permutation decoder, the items are sorted by their keys and, then, picked using the resulting ordering.

determination problem. For some problems, it is convenient to use the chromosome to represent structural parts of a solution or actions to build a solution. In Chapter 2, a complex representation is used where the chromosome is divided in five parts. Some parts are used as indicator parameters and others are used to induce some type of ordering. The representation used in Chapter 3 also has several parts but is simpler than the representation of Chapter 2.

One may note that, since the chromosome space is a unitary continuous hypercube, theoretically we have an infinity number of chromosomes (in the practice, this number is limited to the float point precision of the machine used). In turn, an infinity number of chromosomes maybe be decoded to the same solution. For example, the chromosomes  $x_1 = (0.1, 0.2, 0.3, 0.4)$  and  $x_2 = (0.5, 0.6, 0.7, 0.8)$  may result in the same solution if we use them to induce permutations by ordering the keys solely. In fact, the basic BRKGA framework does not tackle issue and the population can be made from different chromosomes that result in the same solution. This can slow down the convergence of the algorithm leading to useless decoding iterations. However, the crossover between two different chromosomes that result in the same solution, can generate a completely different solution. For example, the offspring  $x_3 = (0.1, 0.5, 0.6, 0.4)$  resulting from the crossover of  $x_1$  and  $x_2$ , may be a completely different solution if we consider the induced permutation by ordering the keys solely.

During the decoding phase, local search procedures can be applied to the decoded solution. Often, the solution is improved offering faster convergence. In such cases, a good idea is to rewrite the chromosome to reflect these changes. Note that it is not necessary in fact, if both the decoding process and the local search are deterministic (i.e. the multiple decoding of same chromosome always results in the same solution). However, a chromosome that reflects exactly the improved solution can offer good subregions to be combined with another chromosome in the mating phase. In some cases, rewriting a chromosome is straightforward (Chapters 4 and 5), while in others one must be careful due to the magnitude of the keys (Chapter 3). In some cases it is impossible due to the complexity of decoding.

### 1.2.3 Improvements

A common approach used in genetic algorithms is the island model (Whitley et al. [206]), where several populations are evolved independently and exchange their best individuals every given number of generations. This improves the variability of individuals, usually speeding up convergence and reduces the risk that the algorithm will get stuck in local optima. Note that it is not necessary that this process be done in parallel in the sense of using several parallel machines or CPUs. It is straightforward to adapt the BRKGA

framework:  $\pi$  separate populations are created such that they are evolved simultaneously applying the evolutionary process in lines 3–8 of Algorithm 1.1 to each population. In this case, we will have  $P_1, \dots, P_\pi$  populations,  $E_1, \dots, E_\pi$  elite sets, and  $Q_1, \dots, Q_\pi$  “next generation” pools. The individual exchanges occur when a given threshold is reached, for instance, at every  $\delta$  generations. For each population  $P_i$ , the  $\eta$  best individuals are copied from other populations  $P_{j \neq i}$  and replace the  $\eta(\pi - 1)$  worst individuals in  $P_i$ .

Another common approach is to restart the algorithm following a given criterion. This is particularly important for evolutionary algorithms since most of them are considered Las Vegas algorithms, i.e., they always produce a correct solution when they stop, however their running time is a random variable. There is a rich literature with respect restart to strategies and the reader may refer to Jansen [103], Luby et al. [136], Shylo et al. [192] and Shylo et al. [191]. For illustration, consider an evolutionary algorithm  $\mathcal{A}$  that solves a given problem. Consider an instance of this problem such that the best solution is  $x$ . Suppose that several independent runs of this algorithm on the given instance are performed. In some of these runs, say 75% of total runs,  $\mathcal{A}$  is able to reach  $x$  using no more than 400 iterations. This means that the (empirical) probability of  $\mathcal{A}$  will be running after 400 iterations is 25%. Some runs, say 5%,  $\mathcal{A}$  takes more than 2,000 iterations to reach  $x$ . Now, suppose that  $\mathcal{A}$  is restarted after each 400 iterations. The probability of  $\mathcal{A}$  will still be running after 2,000 iterations is the probability of  $\mathcal{A}$  will still be running after the first 400 iterations, and the probability of  $\mathcal{A}$  will still be running after the first, the second, the third, the forth, and the fifth restarts (totalizing 2,000 iterations). Therefore, this probability is  $0.25^5 \simeq 0.0009 = 0.09\%$  which is much lower than 5% of probability given by runs without restart.

### 1.2.4 Parameters

The parameters that must be specified beforehand are the size of the chromosomes  $n$ , the size of the population  $p$ , the size of elite set  $p_e$ , the number of mutants  $p_\mu$  introduced at each generation, and the inheritance probability  $\rho_e$ . If using parallel populations, we must set the number of populations  $\pi$  and the generation threshold  $\delta$  to exchange the  $\eta$  best individuals. If using restart, we must choose a strategy and respective parameters. Advice for the parameter setup can be found in Gonçalves and Resende [81].

### 1.2.5 Successful use cases

Biased random-key genetic algorithms have been used with success both in several classical hard combinatorial optimization problems as well as on real-world problems. For example, Resende [178] surveys applications of BRKGA in optimization problems arising in telecommunications. Gonçalves and Resende [82, 83, 84] present BRKGAs for 2D and

3D packing and bin packing. BRKGAs have been applied to a number of scheduling problems, e.g. job-shop scheduling (Gonçalves et al. [75], Gonçalves and Resende [78]), and resource constrained project scheduling (Gonçalves et al. [76, 77, 79]). In Lucena et al. [138], some variations of BRKGA are proposed. The first variation is a gender-defining BRKGA where each chromosome has a gender. Two chromosome are chosen for mating if and only if their gender are different. The second variation consists in performing the mating with several parents at once. Experimental results show that the gender-defining BRKGA does not present improvement over the original BRKGA. However, the multi-parent version was able to overcome both the original and gender-defining algorithms.

## 1.3 Results and thesis organization

This thesis presents optimization algorithms for a list of problems that appears in network planning. Chapters 2 and 3 consider the project of access and core networks respectively. Chapter 4 studies a clustering problem that can be used to estimate the usage of a network. Finally, Chapter 5 deals with spectrum auctions from the point of view of government regulatory agencies. Each chapter is self-contained with exception of references to biased random-key algorithms. Such content is concentrated in Section 1.2 of this chapter. In each chapter, the formal definition of the problem treated there as well as a brief literature review of similar problems and techniques are presented.

Chapter 2 presents a new problem called *Wireless Backhaul Network Design Problem* (WBNDP). The objective of this problem is to build backhaul networks responsible for collecting access traffic from mobile devices such as smart phones and tablets. Such networks are constituted of routing trees that use wireless links. Each routing tree must have a limited depth and breadth respecting the geographical characteristics of the region where the network will be deployed. The access traffic is collected by technologies such as Wi-Fi and LTE. Several real-world constraints such as radio interference between pieces of equipment, capacities, and geographical location are imposed. The WBNDP is closely related to facility location and the Steiner tree problem. However, WBNDP differs from other problems in the literature mainly for its objective function which depends on the total traffic that the network can route. Moreover, the underlying maximum flow problem is not common and presents some peculiarities which differs it from the classical maximum flow problems. A mixed integer linear programming model (MIP) and a biased random-key genetic algorithm (BRKGA) are proposed to solve the WBNDP. Although the model is not complex, it is lengthy due to the complexity of this problem. Several real-world instances were considered and two scenarios analyzed. Using IBM ILOG CPLEX Optimizer [101] to solve the MIP, only one instance was solved to optimality. BRKGA

presented solid results, mainly on large instances, and exposed itself as a valuable tool to solve the WBNDP.

Once built the network responsible to collect and route access traffic, it is necessary to build the core infrastructure. In general, such core is characterized by main loops or backbones with high speed and capacity. Such links are usually deployed using fiber or high frequency microwaves. Chapter 3 presents the *k-Interconnected Multi-Depot Multi-Travelling Salesmen Problem* (*k*-IMDMTSP) that can be used to model core networks. In the *k*-IMDMTSP, one must select  $k$  vertices from a graph and link them in a cycle. For each chosen location, a cycle of size at most  $C$  must span the remaining vertices of the graph. One can use the cycle connecting the  $k$  chosen vertices to model a satellite loop, and the spanning cycles to model fiber loops, for example. A BRKGA and a multi-start heuristic were proposed to solve the *k*-IMDMTSP. Both algorithms use local search procedures based on heuristics for the traveling salesman problem in the decoding procedure. Five scenarios were proposed varying the sizes of the inner cycle and the outer cycles. For each scenario, 63 instances were tested. BRKGA outperformed the multi-start heuristic in all cases with significant difference.

Chapter 4 addresses the *Overlapping Correlation Clustering* (OCC). In such problem, we want to assign labels to objects such that the labeling reflects the similarities between the objects. In OCC, each object may be labeled with one or more labels, differing from traditional clustering problems where usually only one label is allowed per object. In other words, while in the traditional correlation clustering we create a partition of objects, in the OCC the “partitions” may overlap (in fact, there is no partition in the strict sense). The OCC can be used to analyze several kinds of scenarios in different fields. Particularly in telecommunications, we may use the OCC to analyze patterns of mobility from mobile devices, consumption paths, and make demand predictions. Note that such analysis are very important to telecommunication companies in offering improved and new services. We propose a BRKGA with two representations and two decoders. We also improved the algorithm considered the state of the art for OCC. The algorithms were applied to three scenarios with distinctive characteristics and similarity functions: animal trajectories, protein analysis, and text analysis. BRKGA was able to overcome the state of the art algorithm in most and relevant cases although requiring large running-time to do it.

In Chapter 5, we take the side of governmental agencies selling electromagnetic spectrum rights to telecommunication companies. The large increase of mobile devices has pushed the utilization of new spectra, which are licensed by governmental regulatory agencies in most countries. Such market is billionaire and represents a large revenue for governments. In general, such spectrum licenses are auctioned in so called combinatorial auctions. In Chapter 5, we address the *Winner Determination Problem* (WDP), which is used to determine the winners of these auctions. In the WDP for combinatorial auctions,

the bidders submit bids for bundles of objects that can represent complementarity or substitutability among these objects. For instance, suppose that we want to sell two adjacent regions A and B. A telecommunication company may value US\$ 10 million the rights to operate in region A, and US\$ 15 million the rights in region B, if the company can only secure one region. But the company knows that operating in both regions at same time is more profitable. Therefore, it can offer US\$ 40 million for the rights of both regions A and B. Such combinatorial auctions are able to generate efficiency in the market. The task in the WDP is to determine the mutual exclusive bundles for which the sum of the bid values is maximized. We proposed six variants of BRKGA using simple decoders. We compared our approaches with: an integer linear programming model solved by IBM ILOG CPLEX Optimizer [101], a state of the art exact algorithm, and two state of the art genetic algorithms for the WDP. These ten algorithms were applied to 537 instances with several sizes and difficult levels. Again, the BRKGAs were able to produce the best results in the most cases, specially for large and hard instances.

Finally, Chapter 6 summarizes our main contributions.



# Chapter 2

## The Wireless Backhaul Network Design Problem

With the increasing utilization of mobile devices, telecommunication companies have been pushed to expand their network infrastructure offering better coverage to their clients. Wireless devices generate a huge amount of data traffic when compared with traffic due to voice, and to manage these data networks is substantially more expensive and complex than to manage voice networks. This is partially because coverage range of data traffic equipment is much smaller than that of voice equipment and because data traffic is more susceptible to external interference such as weather conditions. To be able to offer a quality service, the provider must build a network that maximizes coverage and be capable of routing the traffic to the core network. This kind of network is known as *Backhaul Network*. To cover all demand by fiber may be very costly (this is referred as *the last mile problem*) and, recently, some providers have not honored network expansion contracts because of their high costs (Brodkin [28]). Instead, providers have tried to build backhaul networks using high capacity wireless links.

This chapter describes a biased random-key genetic algorithm for a real-world wireless backhaul network design problem. This is a novel problem, closely related to variants of the Steiner tree problem and the facility location problem. Given a parameter  $H$ , we want to build a forest where each tree has at most  $H$  hops. Each tree is rooted at specific nodes, called root nodes, and has leaves at demand nodes, where traffic originates. Candidate Steiner nodes do not have any demand but represent locations where we can install cellsites to cover the traffic and equipment to backhaul the traffic to the cellular core network. Each Steiner node can cover demand nodes within a given distance, subject to a capacity constraint. The aggregate set of constraints may make it impossible to cover or backhaul all demands. A revenue function computes the revenue associated with the total amount of traffic covered and backhauled to the root nodes. The objective of the

problem is to build a forest that maximizes the difference between the total revenue and the cost associated with the installed equipment. Although we will have a forest when we consider only the backhaul links and root nodes, the addition of demand vertices can induce undirected cycles resulting in a directed acyclic graph. We consider real instances of this problem with several additional constraints that are motivated by the requirements of real telecommunication networks. This chapter is based on Andrade et al. [6].

## 2.1 Introduction

There has been a surge in the popularity of mobile devices (smart-phones and tablets). For example, in United States, mobile devices now account for more than 50% of Internet usage (O'Toole [160]). This, coupled with the popularity of high bandwidth services such as video, has pushed mobile data usage. Cisco VNI Global IP Traffic Forecast (Cisco [37]) predicts a 61% annual growth rate for mobile data, resulting in an 11-fold increase from 2013 to 2018. Service providers need to keep up with this growth in mobile data usage by providing better coverage and higher rates to their customers.

The associated network design problem needs to decide on the optimal concentration of cellular and Wi-Fi equipment to provide good service to users. We also need to find the right backhaul strategy to route this traffic to the core network. A naive solution may be to run fiber to all sites, but it may be prohibitively expensive. Instead we judiciously use the existing fiber infra-structure to pick the right backhaul. First of all, for installing Wi-Fi or cellular equipment, we may prefer a site that already has fiber. For other sites, we may be able to use a wireless backhaul to aggregate their traffic to a fibered site. Finally there are sites where running new fiber may be the best option.

We propose the *Wireless Backhaul Network Design Problem* (WBNDP) with practical constraints, and present a model to deal with real networks. The motivation of the proposed problem is to model wireless backhaul networks that operate over technologies such as Wi-Fi, LTE (4G technology), and HSPA+ (3G technology). In such networks, we must collect data traffic in a given geographic region and route it to the core network. Their structure is determined by equipment and service quality constraints. Usually, one wants to build tree at high-capacity nodes over a sparse graph with capacity constraints. Although the WBNDP resembles variants of the Steiner tree and facility location problems, its revenue and cost structures distinguish it from these two problems.

To solve the WBNDP, we propose a biased random-key genetic algorithm (BRKGA) with a sophisticated decoding procedure. The algorithm has several phases, such as equipment deployment, construction of routing trees, flow computation, and pruning of unused equipment. The choice of BRKGA is grounded on its recent success in solving large combinatorial optimization problems (Gonçalves and Resende [81]). In order to

evaluate the quality of results obtained by the BRKGA, we also formulate and solve a mixed integer linear programming model for computing the optimal solution.

The structure of the chapter is as follows. Section 2.2 presents the WBNDP in detail and Section 2.3 reviews the related literature. In Section 2.4, a formal definition as well as a mixed integer linear programming model are presented. Section 2.5 describes a biased random-key genetic algorithm to solve the WBNDP and Section 2.6 discusses the maximum flow problem that arises in WBNDP. Section 2.7 describes some instances derived from real-world problems and pre- and post-processing phases. Section 2.8 presents experimental results, and we make some final considerations and conclusions in Section 2.9.

## 2.2 Problem description

Consider a geographical region with each point in the region identified by its coordinates (latitude and longitude). Consider the graph  $G = (V, A)$ , where  $V$  and  $A$  are, respectively, the sets of vertices and arcs of  $G$ . The set  $V^d \subset V$  is the subset of vertices that correspond to demand points or city blocks in this region. These demand points consist of mobile devices, such as mobile phones, tablets, laptops, and other devices that make use of wireless communication. The set  $V^s \subset V$  is the set of vertices that correspond to equipment used to collect and route traffic from the demand points. This equipment is installed on utility poles distributed across the streets and highways of the region. We consider *small cells* network design, meaning that cellsites have relatively small coverage radii. Small cells have the advantage of using spectrum more efficiently. We consider three types of equipment: Wi-Fi and LTE for access traffic (demand collection), and retransmitter for routing/backhauling. The set  $V^r \subset V$  represents the *Fibered Access Points* (FAPs) where we have existing fiber connected to the core network. VRADs (Video-Ready Access Devices) are one example of FAPs. Installing equipment close to a VRAD will enable us to route the traffic on this fiber with no or little additional cost. We also have many existing *macro cellsites* where we can add additional radio equipment to carry traffic. If this macro has a fiber backhaul, we can technically consider it a FAP but because macros also cover access traffic directly, we make a distinction between macro cells and other FAPs for notational convenience. Finally certain macro cellsites are connected to the core with very high speed wireless links. In our formulation, they serve the same purpose as macros with a fiber backhaul and we treat them identically.

The objective is to create routing trees such that the tradeoff between the revenue derived from the routed traffic and the network cost be the best possible. We call this problem the *Wireless Backhaul Network Design Problem* (WBNDP) and in the following we describe each one of these particularities so that later on we can present a formal definition.

### 2.2.1 Demand splitting and routing trees

In general, it is difficult to estimate the demand of each user due to the user's mobility in the region where the network will be built. One way to approximately model this scenario is to concentrate, for each city block, its total demand at the center of the block. Although this appears to be oversimplified, the errors are diluted by the mobility of the users. Therefore, it is reasonable to assume that a certain block may be served by several cellsites and its demand split among them. Although the demand may be split among several pieces of access equipment, this equipment and the routing equipment must be connected through trees rooted at the FAPs or the macrocells.

Thus, one can note that a solution  $S$  for a backhaul network may contain an undirected cycle originated at a demand. Therefore, although we have a forest when we consider only the Steiner vertices and root nodes of  $S$ , the addition of demand vertices can induce these undirected cycles, resulting in a directed acyclic graph (DAG).

### 2.2.2 Capacity of access equipment

Each access equipment is limited in its handling of traffic. These constraints are access radius and access capacity. Values of these parameters vary across vendors and also depend on geographic terrain. We have used the following representative values in our simulations. For access radius, we have Wi-Fi: 100 meters; LTE small cell: 400 meters; LTE macro cell and HSPA: 3,000 meters. For access capacity, we have Wi-Fi: 100 Mbps; LTE small cell: 20 Mbps single band, 40 Mbps dual band; LTE macro cell and HSPA: 25 Mbps. Although, for sake of simplification, we considered LTE macro cell and HSPA with the same access radius and capacity, these two technologies can present different values. See more details about these limitations, see Rayal [175].

### 2.2.3 Capacity of retransmitter equipment

The capacity of retransmitter equipment is one of the most complex aspects of this type of network. In addition to having a maximum access radius (representative value of 1,000 meters), retransmitter equipment also have a physical restriction that limits the sum of the flow that it receives and sends to the other retransmitters. This quantity is limited to a value  $U_{bh}$  (e.g., 100 Mbps). Though this equipment also deals with access traffic from other equipment (Wi-Fi/LTE) that share the same utility pole, this limit is not applied to that traffic. Therefore, the limit  $U_{bh}$  does not account for all incoming traffic but shapes the outgoing traffic.

More precisely, let  $v \in V^s$  be a retransmitter. Take  $A_v^{+s} = \{(w, v) \in A : w \in V^s\}$  as the set of arcs outgoing from neighbors of  $v$  that send to it backhaul traffic. Let  $a_v^-$  be

the outgoing arc of  $v$  in the solution (recall that we are looking for a backhaul forest). Let  $A_v^{+d} = \{(w, v) \in A : w \in V^d\}$  be the set of arcs from demand vertices to  $v$ . Let  $f : A \rightarrow \mathbb{R}^+$  be a function that defines the flow on the arcs. Therefore, we have

$$\sum_{e \in A_v^{+s}} f_e + f_{a_v^-} \leq U_{bh}, \quad \forall v \in V^s \quad (2.1)$$

such that

$$\sum_{e \in A_v^{+s}} f_e + \sum_{e \in A_v^{+d}} f_e = f_{a_v^-}, \quad \forall v \in V^s. \quad (2.2)$$

While Inequality (2.1) only restricts the backhaul flow to at most the capacity  $U_{bh}$ , Equality (2.2) is the classical flow conservation equation which ensures that  $v$  has no excess flow. Although these restrictions do not influence the structure of the backhaul forest, they have a direct impact on the maximum flow in the forest, used to compute the revenue. In this case, we cannot use a classical algorithm for maximum flow such as Edmonds-Karp or Goldberg-Tarjan directly (see Goldberg and Tarjan [71] for a comprehensive list).

Another physical restriction on the retransmitters is that they support only a limited number of neighbors sending backhaul traffic to it (known as *fan-in*, equal to 5 in our simulation results). Thus, this restriction limits the incoming degree of each retransmitter vertex.

#### 2.2.4 Sight and minimum distance

One can note that the subgraph induced by  $V^s$  may be sparse. The main reason for this is the existence of physical barriers, such as buildings and hills, between pairs of utility poles. Moreover, the distance between them can be so large that the signal loses strength and impairs communication. Therefore, one must consider that the links  $(u, v)$  and  $(v, u)$  only exists when poles  $u$  and  $v$  are close enough (1,000 meters in our simulations) and they are in each other's *line of sight*, i.e. we can trace a straight line between them without obstruction. In this problem, we do not treat omnidirectional communication in the backhaul.

Another important aspect is the interference among cellsites. Both LTE and HSPA use licensed spectrum and can interfere with each other. Therefore, a pair of LTE equipment cannot be installed too close to each other. In our simulations, we restrict a minimum distance of 300 meters between two LTE small cellsites and a minimum distance of 500 meters between LTE small cell and macro cellsites. Note that this concern does not exist with Wi-Fi equipment which, although can exhibit interference, occupies a non-licensed spectrum. Such spectrum is very hard to control due to external interferences.

### 2.2.5 Number of hops

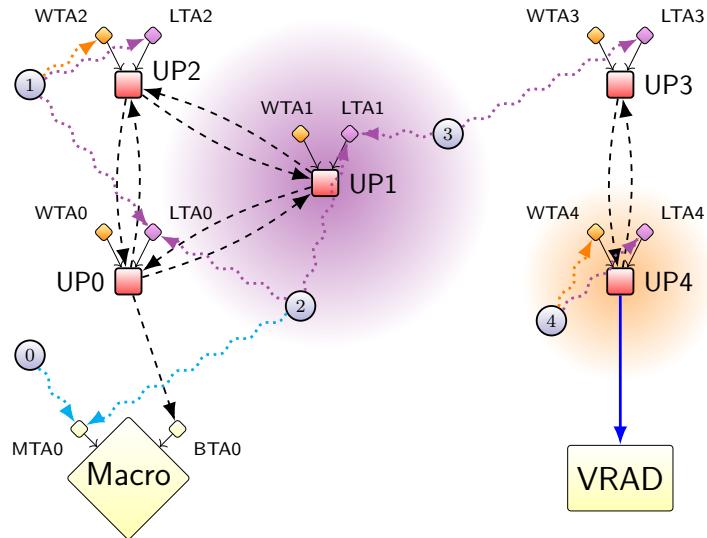
The first hops are defined to be arcs that link root nodes to poles. In practice, these hops may be built using either wireless links or fiber links. The links between utility poles and FAPs must use fiber. Links between poles and macrocells may use fiber or be wireless. The other hops are built over wireless links. The restriction is that the number of wireless hops must be limited to an upper bound (in practice, two or three hops). This restriction aims to reduce the latency of the wireless network and is commonly considered in network planning according to Dahl et al. [43]. Note that only backhaul links are considered.

### Example

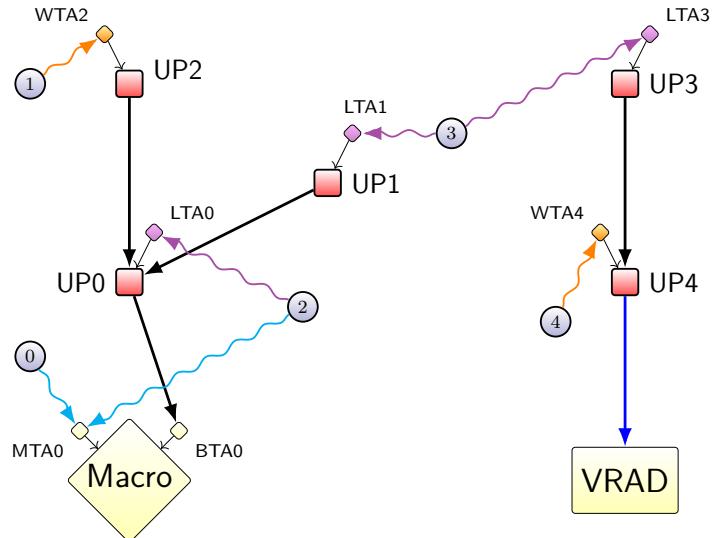
Figure 2.1a depicts an example of a base graph where the dashed arcs represent possible wireless links and the solid blue arc represents an optical fiber link. The dotted sinuous arcs represent the possible links between a demand block and an access equipment. This graph is based on geographic information of the backhaul region and takes into consideration maximum coverage radii as well as the lines of sight of pairs of equipment. Note that the macrocell (represented by a large yellow rhombus/diamond) contains two attached vertices. These are the backhaul traffic aggregator (BTA) and the macrocell traffic aggregator (MTA) (represented by small yellow diamonds). Note that these vertices are virtual and the links between them are internal to the devices. They are used to better model the traffic (more detail is given in Section 2.4). As with the macrocells, each utility pole is represented by a red square and has attached to it two virtual vertices representing Wi-Fi and LTE equipment and their respective aggregators (small orange diamonds for WTAs and purple diamonds for LTAs). The demands are represented by light blue circles. Note that each demand may be satisfied in three distinct ways: Wi-Fi, LTE, and HSPA. In this example, only demands 0 and 2 can be served by the macro cell because of their proximity to the macrocell. Demand 2 can split its traffic between the macrocell and utility poles UP0 and UP1 using LTE. The same situation occurs with demand 3. Demand 1 can use Wi-Fi or LTE on pole UP2, and LTE on pole UP1. Demand 4 is only close to pole UP4 and can use both Wi-fi and LTE. Note that utility poles UP0, UP1, and UP2 are close enough to each other and in each other's lines of sight, enabling the installation of retransmitters on them. Utility poles UP3 and UP4 are in another region and do not have communication links with UP0, UP1, or UP2.

Figure 2.1b shows a valid solution. The solid arcs represent the links among the pieces of equipment. In this example we ignore capacities. Note that several pieces of equipment are not installed since they do not serve any demand. For example, in utility pole UP2 it suffices to install Wi-Fi. Note that if we consider only the black and blue arcs (straight lines), we have a forest. Note also that demand 2 splits its traffic between

utility pole UP0 using LTE and the macrocell. Demand 3 is more interesting: it is split between two distinct subtrees. In this case, demand 3 uses LTE on UP3 and UP1 but nothing prevents it from using the Wi-Fi on UP1 and LTE on UP3.



(a) Base graph.



(b) Valid solution.

Figure 2.1: The figures above represent a base graph from where a valid solution is extracted. We omitted equipment capacities for legibility.

## 2.3 Related literature

The wireless backhaul network design problem (WBNDP) is closely related to variants of the Steiner tree and the facility location problems. Steiner tree problems have been widely studied and among their many applications, network design may be the most expressive. Given a graph with a set of *terminal nodes*, a set of intermediate nodes called *Steiner nodes*, and edge costs, the Steiner Tree Problem (STP) consists in creating a minimum cost tree connecting all terminal nodes. The general version is  $\mathcal{NP}$ -hard (Garey and Johnson [65]) and a survey of the STP is presented in Voß [202].

A variant with many applications in industry is the Prize-Collecting Steiner Tree Problem (PCSP). In this problem, each vertex has a penalty value and each edge has a cost. The objective is to build a tree minimizing the sum of the costs of used edges plus the sum of the penalties of the vertices not spanned by the tree. This is a well-studied problem originated in Goemans and Williamson [69, 70]. Goemans and Williamson [69] presented a 2-approximation algorithm for the PCSP for which practical results were discussed in Canuto et al. [30] and Johnson et al. [104]. Lucena and Resende [137] presented lower bounds using linear programming and cutting planes and their results were improved by Ljubić et al. [131]. Klau et al. [112] proposed a hybrid heuristic where the final population from a memetic algorithm is used to build reduced instances to be solved by an exact algorithm. In da Cunha et al. [42] a Lagrangian non-delayed relax-and-cut algorithm is proposed to generate primal and dual bounds for the problem.

The first problem to deal with limited number of hops was proposed by Gouveia [85] and is called Hop-Constrained Minimum Spanning Tree Problem (HMSTP). In this problem, we want to obtain a minimum spanning tree such the path between the root node and a leaf node has no more than  $H$  hops (edges). In Gouveia [85], a formulation based on subcycle elimination inequalities was presented as well as several lifting procedures and bounds based on Lagrangian relaxation. This approach was refined in several papers described in Dahl et al. [43]. Recently, Gouveia et al. [86] presented several local search neighborhoods resulting in good solutions. In Gouveia et al. [87], HMSTP was presented as a directed Steiner tree model over layered graphs. Several cutting planes from other Steiner problems were applied. A branch-and-cut algorithm was also proposed and it was able to obtain the best results so far described in the literature. Furthermore, Gouveia et al. [87] reduced the Diameter-Constrained Minimum Spanning Tree Problem (DMSTP) to a Steiner tree problem using the same technique. In the DMSTP, the hop constraint is applied to a path between any pair of vertices in the tree.

Another related problem is the Steiner Tree Problem with Revenues, Budget and Hop Constraints (STPRBH). In this problem, the objective is to build a tree that maximizes the collected profit respecting an upper bound on the network costs and the maximum

number of hops from the root node to other any node in the network. Differing from the HMSTP, in the STPRBH it is not mandatory to include all vertices and one has a limited budget to spend building the network. This problem was proposed by Costa et al. [39] in which a two-phase greedy algorithm was developed: one phase consists in a simple local search that destroys part of a solution and rebuilds it greedily, and a tabu search using two simple search neighborhoods. Later, Costa et al. [40] presented several integer linear programming models for the STPRBH and branch-and-cut algorithms were developed for each formulation. Layeb et al. [121] presented a compact formulation based on Miller-Tucker-Zemlim constraints which resulted in similar solutions to those found previously in the literature. Recently, Fu and Hao [64] proposed a new heuristic for the STPRBH which was able to find optimum solutions for instance with known optima and improved solutions for instances with unknown optima.

A similar problem is the Connected Facility Location Problem (CONFL) introduced in Karger and Minkoff [107]. This problem consists in assigning each client to exactly one opened facility and connecting the opened facilities using a Steiner tree. Ljubić and Gollowitzer [130] introduced a modification in the CONFL limiting the number of hops between the facilities and a root node. This problem is known as the Hop Constrained Connected Facility Location Problem (HCCONFL). Ljubić and Gollowitzer [130] used the same technique presented in Gouveia et al. [87] where the problem is modeled on a layered graph. Branch-and-cut algorithms were developed.

One can note that the problem addressed in here (WBNDP) has characteristics that are similar to those of the problems reviewed above. This is especially true with respect to the maximum number of hops. However, there are two main characteristics that distinguish the WBNDP from other problems. The first is the possibility that each demand node be served by more than one Steiner node or root node. In this sense, we do not have a tree like in the other problems. The second and most important characteristic is how the revenue is computed. In the previous works in the literature, the revenue considers the “full value” of a vertex if it is in the solution, i.e., the backhaul network is capable of routing all the traffic. In the WBNDP, due to natural constraints, this is not always true. In fact, we consider that the network has a limited routing capacity and the revenue is a function of the maximum flow in this network. This way, there is a strict relation between the network structure and the revenue, once both topology and link capacity directly influence the flow. Another small difference is that the degree constraints are applied only to Steiner nodes and macrocells (only wireless links and not fiber links).

## 2.4 Formal Definition

Let  $V^d$  be the set of demand points or *demand nodes*. Let  $V^s$  be the set of utility poles where the access and retransmission equipment are installed. We refer to  $V^s$  as *Steiner nodes*. Let  $V^r$  be the set of points which have fiber or a high capacity link which we call *root nodes*. Consider  $V^m \subseteq V^r$  as the set of macrocells and  $V^v \subseteq V^r$  as the set of FAPs. Let  $V = V^d \cup V^s \cup V^r$  and note that  $V^m \cup V^v = V^r$ , and the sets  $V^d$ ,  $V^s$ ,  $V^m$ , and  $V^v$  are pairwise disjoint. Consider  $V^{ld} \subseteq V^r$  to be the set of FAPs whose through traffic is leased.

To model the traffic in the different technologies, consider the following sets:

- *WTA*: set of vertices that represents units of Wi-Fi equipment to be installed on the poles. We call each  $w \in WTA$  a *Wi-Fi Traffic Aggregator*. There is a one-to-one correspondence between Wi-Fi traffic aggregators and utility poles;
- *LTA*: set of *LTE Traffic Aggregators* whose description is similar to *WTA*, except that they are for LTE equipment;
- *MTA*: set of *Macrocell Traffic Aggregators* which have a one-to-one correspondence with each macrocell;
- *BTA*: set of *Backhaul Traffic Aggregators* which also have a one-to-one correspondence with each macrocell. The BTAs are restricted to wireless backhaul traffic.

Let  $v \in V^s$ ,  $u \in WTA$ . If  $u$  is assigned to pole  $v$ , then  $wta(v) = u$  and  $wta(u) = v$ . Consider the same for *LTA*, *MTA*, and *BTA*. As in previous sections, consider a directed graph  $G = (W, A)$  such that  $W = V \cup WTA \cup LTA \cup MTA \cup BTA$ . We will define the set of arcs  $A$  later. Let  $d : V^d \rightarrow \mathbb{R}^+$  be the function that maps the maximum traffic (in Mbps) that originates at each demand point.

Consider the following constants:

- Maximum number of wireless hops:  $H$ ;
- Access Radii (in meters):
  - Wi-Fi access radius:  $R_{wifi}$ ;
  - LTE access radius:  $R_{lte}$ ;
  - Macrocell access radius:  $R_{mc}$ ;
  - Retransmitter radius:  $R_{bh}$ ;
- Minimum distance (in meters):

- LTE to LTE (pole to pole):  $\delta_{lte}$ ;
- LTE (pole) to macrocell:  $\delta_{macro}$ ;
- Capacities (in Mbps):
  - Wi-Fi:  $U_{wifi}$ ;
  - LTE:  $U_{lte}$ ;
  - Macrocell (HSPA and LTE access):  $U_{mc}$ ;
  - Retransmitter:  $U_{bh}$ ;
- Revenue factor (in some monetary unit):  $\varrho$ ;
- Cost (in some monetary unit):
  - Equipment deployment on a pole:  $C_p$ ;
  - Wi-Fi equipment:  $C_{wifi}$ ;
  - LTE equipment:  $C_{lte}$ ;
  - Retransmitter with one-element antenna (fan-in = 1):  $C_{fan1}$ ;
  - Retransmitter with two-or-more-element antenna (fan-in  $\geq 2$ ):  $C_{fan2}$ ;
  - Maintenance of equipment on a pole (per year):  $C_{man}$ . The maintenance cost is the sum of the pole leasing costs and small cell maintenance costs, per year. Note that, if the pole has no access equipment but has a retransmitter, the maintenance cost shall be paid;
  - Macrocell annual cost:  $C_{mc}$ . This costs is compounded by the annual site leasing cost and the annual maintenance cost;
  - Meter of deployed fiber:  $C_{fiber}$ ;
  - Leased traffic (in \$/Mbps):  $C_{ld}$ ;
- Maximum number of incoming backhaul neighbors:  $\delta_{bh}^+$ ;
- Length of fibered hop:  $\ell_{uv}$  for arc  $(u, v)$ ;
- Maximum length of fibered link:  $R_{fiber}$ . Theoretically, this limit does not exist since we may spread fiber over the entire network. In practice, the cost to do this is very high and therefore we impose this limit. However, this is a weak restriction which may be violated if necessary.

Let  $dist : V \times V \rightarrow \mathbb{R}^+$  be the (geodesic) distance between two points. For  $v \in V^s \cup V^m$ , consider  $LOS_v = \{w \in V^s \cup V^m : dist(v, w) \leq R_{bh}$  and there is direct line of sight between  $v$  and  $w\}$ . The set of arcs  $A$  is defined as the union of the following sets:

- $A^{dm} = \{(u, v) : u \in V^d, v \in MTA \text{ and } dist(u, mta(v)) \leq R_{mc}\}$ : Set of arcs from demand blocks to macrocells (MTAs) whose blocks are inside the radius of action of a macrocell;
- $A^{dw} = \{(u, v) : u \in V^d, v \in WTA \text{ and } dist(u, wta(v)) \leq R_{wifi}\}$ : Set of arcs from demand blocks to utility poles (WTAs) inside the radius of action of Wi-Fi equipment;
- $A^{dl} = \{(u, v) : u \in V^d, v \in LTA \text{ and } dist(u, lta(v)) \leq R_{lte}\}$ : Set of arcs from demand blocks to utility poles (LTAs) inside the radius of action of LTE equipment;
- $A^{ss} = \{(u, v) : u, v \in V^s \text{ and } v \in LOS_u\}$ : Set of arcs between utility poles. Note that both  $(u, v)$  and  $(v, u)$  belongs to  $A^{ss}$ ;
- $A^{smw} = \{(u, v) : u \in V^s, v \in BTA \text{ and } u \in LOS_{bta(v)}\}$ : Set of arcs from utility poles to macrocells (BTAs). This set is restricted to wireless links;
- $A^{smf} = \{(u, v) : u \in V^s, v \in V^m \text{ and } dist(u, v) \leq R_{fiber} + \epsilon\}$ : Set of arcs from utility poles to macrocells using fibered links (recall that this is a weak restriction which can be relaxed by adjusting the value of  $\epsilon$ );
- $A^{sv} = \{(u, v) : u \in V^s, v \in V^v \text{ and } dist(u, v) \leq R_{fiber} + \epsilon\}$ : Set of arcs from utility poles to FAPs (same as before);
- $A^{wta} = \{(wta(v), v) : v \in V^s\}$ : Set of arcs from WTAs to utility poles;
- $A^{lta} = \{(lta(v), v) : v \in V^s\}$ : Set of arcs from LTAs to utility poles;
- $A^{mta} = \{(mta(v), v) : v \in V^m\}$ : Set of arcs from MTAs to macrocells;
- $A^{bta} = \{(bta(v), v) : v \in V^m\}$ : Set of arcs from BTAs to macrocells.

Note that arc sets  $A^{dm}$ ,  $A^{dw}$ ,  $A^{dl}$ ,  $A^{ss}$ , and  $A^{smw}$  correspond to wireless links. Arc sets  $A^{sv}$  and  $A^{smf}$  correspond to fibered links. Arc sets  $A^{wta}$ ,  $A^{lta}$ ,  $A^{mta}$ , and  $A^{bta}$  represent connections between several pieces of equipment in the same small cell or macrocell. We use these arcs to model equipment capacities.

### 2.4.1 Mixed integer linear programming model

To better describe the objective function and constraints of the WBNDP, we next model it as a mixed integer linear program (MIP). The constraints are based on the observations of Section 2.2 and on topological restrictions of a DAG, as well as maximum capacity and flow conservation limitations. One important definition to model the WBNDP, it is the notion of level. We define *level* as the number of wireless hops between a utility pole and a root node. Consider the following decision variables:

- $x_{uv}^p \in \{0, 1\}$  for  $(u, v) \in A$ ,  $u \in V^s$ ,  $v \in V^s \cup V^r \cup MTA$ :  $x_{uv}^p = 1$  indicates that arc  $(u, v)$  is in level  $p$  of some tree, for  $p = 0, \dots, H$ ;  $x_{uv}^p = 0$ , otherwise. For  $v \in V^r$ , only variables  $x_{uv}^0$  are defined. For  $v \in BTA$ , only variables  $x_{uv}^1$  are defined;
- $y_v \in \{0, 1\}$  for  $v \in V^s$ :  $y_v = 1$  indicates that a vertex/pole  $v$  is in the solution;  $y_v = 0$ , otherwise;
- $f_{uv} \in \mathbb{R}^+$  for  $(u, v) \in A$ : is the flow through arc  $(u, v)$ ;
- $t_v \in \{0, 1\}$  for  $u \in V^s$ :  $t_v = 1$  indicates that pole  $v$  has installed Wi-Fi equipment;  $t_v = 0$ , otherwise;
- $z_v \in \{0, 1\}$  for  $u \in V^s$ :  $z_v = 1$  indicates that pole  $v$  has installed LTE equipment;  $z_v = 0$ , otherwise;
- $a_v^1 \in \{0, 1\}$  for  $v \in V^s \cup BTA$ :  $a_v^1 = 1$  indicates that the pole or the macrocell has a retransmitter with a one-element antenna (fan-in = 1);  $a_v^1 = 0$ , otherwise;
- $a_v^2 \in \{0, 1\}$  for  $v \in V^s \cup BTA$ :  $a_v^2 = 1$  indicates that the pole or the macrocell has a retransmitter with a two-or-more-element antenna (fan-in  $\geq 2$ );  $a_v^2 = 0$ , otherwise.

Variables  $x$  are used to model the backhaul trees in levels. If an arc  $(u, v)$  is in level 0 (i.e.,  $x_{uv}^0 = 1$ ), then the arc is a fibered link of high capacity. If the same arc is in level 1 (i.e.,  $x_{uv}^1 = 1$ ), then it is a wireless link. Note that the variables  $x$  may be not defined for all arcs  $(u, v)$  and all levels  $p$  while some arcs only appear in deep levels of some tree. For instance, arc (UP1, UP2) from the example of Figure 2.1a can only appear in level 3. Such cases can be identified in a preprocessing phase, as we will describe in Section 2.7.2.

The following MIP models the WBNDP:

$$\max \quad \varrho \sum_{(u,v):v \in V^r} f_{uv} \quad (2.3a)$$

$$- \sum_{v \in V^s} (C_p + Y C_{man}) y_v \quad (2.3b)$$

$$- \sum_{v \in V^s} C_{wifit}_v - \sum_{v \in V^s} C_{lte} z_v \quad (2.3c)$$

$$- \sum_{v \in V^s \cup V^m} (C_{fan1} a_v^1 + (C_{fan2} - C_{fan1}) a_v^2) \quad (2.3d)$$

$$- \sum_{(u,v) \in A^{sv} \cup A^{smf}} C_{fiber} \ell_{uv} x_{uv}^0 \quad (2.3e)$$

$$- \sum_{(u,v) \in A: v \in V^{ld}} C_{ld} f_{uv} + C_{mc} \quad (2.3f)$$

$$\text{s.t.} \quad \sum_{p=0}^H x_{uv}^p \leq 1 \quad \forall (u,v) \in A : u \in V^s \quad (2.3g)$$

$$x_{uv}^{p+1} \leq \sum_{(v,w) \in A: w \neq u} x_{vw}^p \quad \forall (u,v) \in A : u \in V^s, p = 0, \dots, H-1 \quad (2.3h)$$

$$\sum_{(u,v) \in A} \sum_{p=1}^H x_{uv}^p \leq \delta_{bh}^+ y_v \quad \forall v \in V^s \cup BTA \quad (2.3i)$$

$$\sum_{(v,w) \in A} \sum_{p=0}^H x_{vw}^p \leq y_v \quad \forall v \in V^s \quad (2.3j)$$

$$a_v^1 \geq \frac{1}{\delta_{bh}^+} \left( \sum_{(u,v) \in A} \sum_{p=1}^H x_{uv}^p \right) \quad \forall v \in V^s \cup BTA \quad (2.3k)$$

$$a_v^2 \geq \frac{1}{\delta_{bh}^+} \left( \sum_{(u,v) \in A} \sum_{p=1}^H x_{uv}^p - 1 \right) \quad \forall v \in V^s \cup BTA \quad (2.3l)$$

$$a_v^1 \geq \sum_{(v,w) \in A} \sum_{p=1}^H x_{vw}^p \quad \forall v \in V^s \quad (2.3m)$$

$$t_v \leq y_v \quad \forall v \in V^s \quad (2.3n)$$

$$z_v \leq y_v \quad \forall v \in V^s \quad (2.3o)$$

$$z_u + z_v \leq 1 \quad \forall u, v \in V^s : dist(u, v) \leq \delta_{lte} \quad (2.3p)$$

$$z_u = 0 \quad \forall u \in V^s, v \in V^m : dist(u, v) \leq \delta_{macro} \quad (2.3q)$$

$$\sum_{(v,w) \in A} f_{vw} \leq d_v \quad \forall v \in V^d \quad (2.3r)$$

$$f_{uv} \leq U_{wifi} t_v \quad \begin{aligned} \forall (u, v) \in A : \\ u \in WTA \end{aligned} \quad (2.3s)$$

$$f_{uv} \leq U_{lte} z_v \quad \begin{aligned} \forall (u, v) \in A : \\ u \in LTA \end{aligned} \quad (2.3t)$$

$$f_{uv} \leq \sum_{p=1}^H U_{bh} x_{uv}^p \quad \begin{aligned} \forall (u, v) \in A : \\ u, v \in V^s \cup BTA \end{aligned} \quad (2.3u)$$

$$f_{uv} \leq M x_{uv}^0 \quad \begin{aligned} \forall (u, v) \in A : \\ u \in V^s, v \in V^r \end{aligned} \quad (2.3v)$$

$$\sum_{(u,v) \in A: u \in V^s} f_{uv} + \sum_{(v,w) \in A} f_{vw} \leq U_{bh} \quad \forall v \in V^s \quad (2.3w)$$

$$\sum_{(u,v) \in A} f_{uv} - \sum_{(v,w) \in A} f_{vw} = 0 \quad \forall v \in V \setminus (V^r \cup V^d) \quad (2.3x)$$

$$f_{vw} \leq U_{bh} \quad \forall v \in BTA \quad (2.3y)$$

$$f_{vw} \leq U_{mc} \quad \forall v \in MTA. \quad (2.3z)$$

Terms (2.3a–2.3f) constitute the objective function and computes the net profit that the network generates in a time windows of  $Y \in \mathbb{R}^+$  years. In practice, the revenue is computed by a complex function based on service packages offered to customers and, in general, is an estimate based on the experience of operators and on market fluctuations. For the particular scenario considered in here, the revenue is a function of the total traffic routed through the FAPs. We consider a linear function using the revenue factor  $\varrho$  as shown by Term (2.3a). The remaining terms add up to the total cost: Term (2.3b) is the cost of deployment and maintenance of poles over  $Y$  years; Term (2.3c) is the cost of Wi-Fi and LTE equipment; Term (2.3d) is the cost of retransmitters; Term (2.3e) is the cost of trenching for fiber; and Term (2.3f) is the cost of leased traffic. The constant  $C_{mc}$  represents the macrocell cost as described in the beginning of this section.

The constraints can be partitioned into three blocks. The first block (2.3g–2.3j) models the backhaul trees: Constraint (2.3g) forbids an arc to be in more than one level; Constraint (2.3h) requires that if an incoming arc into node  $v$  is in level  $p+1$ ,  $v$  should have an outgoing arc in level  $p$ ; Constraint (2.3i) limits the incoming degree for poles and BTAs according Section 2.2.3 (note that the constraint only consider levels greater than or equal to one, since fibered arcs of level 0 are only allowed to be incoming arcs at FAPs and macrocells); and Constraint (2.3j) guarantees that each pole has at most one outgoing arc.

The second block of constraints (2.3k–2.3p) is tied to equipment deployment. Constraints (2.3k) and (2.3l) indicate, respectively, the presence of a retransmitter with a one-element antenna or a multiple-element antenna. Note that if, for some node  $v$ ,  $a_v^2 = 1$ , then necessarily  $a_v^1 = 1$ . In this case, we do not consider a one-element antenna sub-

tracting its cost from the objective function as shown in Term (2.3d). Constraint (2.3m) guarantees placement of a retransmitter in node  $v$  if there exists an arc outgoing from  $v$ . Constraints (2.3n) and (2.3o) permit the deployment of Wi-Fi and LTE equipment on pole  $v$ , respectively, only if pole  $v$  is used. Constraints (2.3p) and (2.3q) prohibit the deployment of two pieces of LTE equipment near each other or close to a macrocell, respectively. Note that Constraint (2.3q) may be redundant since closeby pole/macrocell pairs are discarded in a preprocessing phase.

The last block of constraints (2.3r–2.3z) is related to flow. Constraint (2.3r) limits the flow outgoing from demand blocks. Constraints (2.3s) and (2.3t) ensure that Wi-Fi and LTE capacities are respected. Constraint (2.3u) limits the capacity of wireless arcs to the retransmitter capacity. Constraint (2.3v) enables unlimited flow on fibered links when  $M \geq \sum_{v \in V^d} d_v$ . Constraint (2.3w) limits the retransmitter capacity as discussed in Section 2.2.3 (note that only backhaul flow is considered in this constraint). Constraint (2.3x) is the classical flow conservation constraint. Finally, Constraint (2.3y) applies the retransmitter capacity to BTAs, while Constraint (2.3z) limits macro cellsite traffic. All integrality requirements are omitted since they are described in the beginning of this section.

## 2.5 Solution procedure using BRKGA

Because of the large-scale nature of practical instances of MIP (2.3), it can be difficult to solve the WBNDP using an exact approach, such as a branch-and-cut algorithm. Typically, these instances are situated in regions of approximately 80 km<sup>2</sup> with about 15 macrocells, 130 VRADs, 3,200 utility poles, and 16,000 demand blocks. Although the underlying graph is relatively sparse, the number of valid solutions can be huge. To deal with this situation, we propose a Biased Random-Key Genetic Algorithm (BRKGA) to solve the WBNDP. See Section 1.2 for more details about the BRKGA.

From an implementation point of view, most of the effort in building a BRKGA focuses on devising a decoder, which takes as input a vector of random keys and outputs a valid solution for the problem. The decoding phase for the WBNDP consists in iteratively building acyclic directed graphs and the computation of maximum flows, costs, and revenues. This procedure builds the solution in a *bottom-up* fashion, starting at the root nodes and ending at the demand nodes. Since the WBNDP has a large number of peculiarities, the design of a decoder is not trivial and consists of a number of intermediate steps. In the next subsections, we describe each of these steps.

### 2.5.1 Representation

Given an instance of the problem, without loss of generality we assume that the utility poles are listed in an arbitrary but fixed order  $V^s = (\bar{p}_1, \dots, \bar{p}_n)$  where  $n$  is the number of poles, i.e.,  $n = |V^s|$ . We also assume that the root nodes are listed in an arbitrary but fixed order beginning with macrocells and ending with the FAPs, i.e.,  $V^r = (\bar{m}_1, \dots, \bar{m}_\alpha, \bar{f}_1, \dots, \bar{f}_\beta)$  where  $\alpha$  is the number of macrocells and  $\beta$  is the number of FAPs.

A chromosome is a vector of real numbers  $\mathbf{v} \in [0, 1]^{5n}$ . This vector is partitioned into five sections whose values are used to build the backhaul network. Each value is associated with a utility pole through an index given by  $V^s$ . As we explain below, each pole is associated with five values, or keys, of the chromosome, one in each partition.

The first section consists of values  $\mathbf{v}_1, \dots, \mathbf{v}_n$  that define which poles are present in the solution and their deployment order. The deployment order defines the sequence of LTE equipment installation. We refer to this first group as  $\kappa^A = (\mathbf{v}_1, \dots, \mathbf{v}_n)$ .

The second section consists of the values  $\mathbf{v}_{n+1}, \dots, \mathbf{v}_{2n}$  which are used as activation parameters. They determine whether:

- an LTE equipment is deployed on the utility pole (L or NL);
- the utility pole is connected via fiber (F or NF);
- the utility pole connects directly to a FAP or a macrocell (D or ND).

Using this notation we have:

- $v_i \in [0.000, 0.125)$ , then: NL, NF, ND;
- $v_i \in [0.125, 0.250)$ , then: NL, NF, D;
- $v_i \in [0.250, 0.375)$ , then: NL, F, ND;
- $v_i \in [0.375, 0.500)$ , then: NL, F, D;
- $v_i \in [0.500, 0.625)$ , then: L, NF, ND;
- $v_i \in [0.625, 0.750)$ , then: L, NF, D;
- $v_i \in [0.750, 0.875)$ , then: L, F, ND;
- $v_i \in [0.875, 1.000)$ , then: L, F, D.

The first parameter controls how to distribute LTE equipment and, as a consequence, how the demands are distributed amongst them. Furthermore, it may be valuable not to install LTE on a given utility pole even if there is nearby demand, since the cost/benefit ratio

may be too low or negative. The second parameter dictates if a pole can be connected by fiber. It is used in cases where the utility pole is connected to a macrocell by fiber or by a wireless link. Again, this parameter controls the cost/benefit ratio. Lastly, the third parameter dictates the minimum tree level in which a pole appears in the backhaul network. We refer to these keys as  $\kappa^P = (\mathbf{v}_{n+1}, \dots, \mathbf{v}_{2n})$ .

The third section  $\kappa^O = (\mathbf{v}_{2n+1}, \dots, \mathbf{v}_{3n})$  defines the evaluation order of the utility poles as the network is grown. This order is used to build the next level of nodes in the forest. Suppose, for example, that a utility pole is already connected to the network and that it can only support one additional backhaul connection (i.e., Constraint (2.3i) is nearly saturated). If there are two other poles to be connected to this one, the order induced by  $\kappa^O$  will define which pole will be connected. Another order induced by  $\kappa^{O'}$  can potentially generate a different network.

The fourth section  $\kappa^N = (\mathbf{v}_{3n+1}, \dots, \mathbf{v}_{4n})$  defines the evaluation order of a pole neighborhood. Suppose that a pole is about to be included in the network and that it has two or more previously deployed neighbor nodes with utility poles to which it can connect to. The order induced by  $\kappa^N$  will determine to which neighbor the connection will be made.

Lastly, the fifth section  $\kappa^L = (\mathbf{v}_{4n+1}, \dots, \mathbf{v}_{5n})$  defines the minimum tree level on which a utility pole can be placed. The root nodes and fibered utility poles are considered to be at level zero. Thus, the remaining poles are distributed among levels 1 through  $H$ . The minimum level of utility pole  $i$  is given by  $\lfloor (H + 1)\kappa_i^L \rfloor$ . If the minimum level of pole  $i$  is zero, it can be placed in any tree level. If it is one, pole  $i$  can only be placed in level 1 or above. In this case, the pole cannot be connected by fiber.

Although at first glance, there is a superposition of functionalities among the several keys, we see below that each key plays a very particular role in the process of network construction.

### 2.5.2 Decoder

Algorithm 2.1 shows the basic steps to decode a chromosome into a valid solution. It consists of several procedures described in Algorithms 2.2–2.5. In addition to the above discussion, the algorithms also make use of the following definitions:

- $parent(p)$ : indicates which utility pole, macrocell, or FAP that pole  $p$  is linked to, i.e., arc  $(p, parent(p))$  exists in the backhaul network;
- $children(p)$ : set of poles linked to  $p$ , i.e., there exist links  $(w, p)$  for  $w \in children(p)$ ;
- $level(p)$ : indicates the level in which pole  $p$  is placed in the tree;
- $fibered(p)$ : indicates if pole  $p$  is linked by fiber to a root node;

---

**Algorithm 2.1:** Decoder.

- 
- 1 Define the activation order and install (LTE) equipment in the poles;
  - 2 Build the backhaul graph;
  - 3 Remove non-used pieces of equipment and poles (1<sup>st</sup> phase);
  - 4 Compute the maximum flow. Let  $f_{max}$  be the value of maximum flow;
  - 5 Remove non-used pieces of equipment and poles (2<sup>nd</sup> phase);
  - 6 Compute the cost over the time window of  $Y$  years. Let  $C_{total}$  be the total cost;
  - 7 Compute the revenue over the time window of  $Y$  years. Let  $R_{total}$  be the revenue obtained;
  - 8 **return**  $R_{total} - C_{total}$
- 

- $\deg^+(p)$ : number of arcs leaving  $p$  (i.e., outgoing degree of  $p$ );
- $CP$ : set of the most external poles in the current stage of network construction, i.e., the leaves of current forest;
- $TK$ : set of all utility poles in current forest. Note that  $CP \subseteq TK$ ;
- $NL$ : set of utility poles to be considered for connection in the next forest levels.

The first step is to choose which poles can be added to the solution. Line 1 of Algorithm 2.2 chooses the poles whose keys have a value greater than or equal to 0.5, and activates these poles in an order defined by their key values. Therefore, for each activated pole, we install Wi-Fi and LTE equipment in the given order when there is demand in the access radii of the pole. In the case of LTE, we need to check whether the pole is sufficiently far from any macrocell or other poles with previously installed LTE equipment. Line 1 has run-time complexity of  $O(|V^s| \log |V^s|)$ , since it is a sorting of the keys  $\kappa^A$ . For each pole in the loop of line 2, in the worst case, we must test all possible neighbors in line 5, leading to run-time complexity of  $O(|V^s|^2)$  which is the dominant run-time complexity of Algorithm 2.2.

The construction of the backhaul forest is an iterative bottom-up process. First, we create the first level connecting poles to the root nodes using Algorithm 2.3. Sets  $CP$ ,  $TK$ , and  $NL$  are initially empty. We initialize pole levels to indicate that level assignment has not yet been made. Likewise, we initialize no connectivity by fiber to all poles (lines 1 and 2). We add to  $CP$ , the set of current poles, all active neighbors of each root node (lines 3 and 5) sorted in non-increasing order of their corresponding keys in the chromosome (line 6). Using this permutation, we try to connect each pole to a FAP or a macrocell obeying the minimum level and activation parameters (note that the function `extractparameters()` returns a triple according to the description of Section 2.5.1).

---

**Algorithm 2.2:** Equipment activation and installation.

---

```

1 Let  $L$  be a pole list in non-increasing order of keys  $\kappa^A$  such that  $p \in L$  iff  $\kappa_p^A \geq 0.5$ .
  Each  $p \in L$  is said active and each  $p' \notin L$  is said inactive;
2 foreach  $p \in L$  in the given order do
3   if  $\exists v \in V^d : d(p, v) \leq R_{wif}$  then
4     Install a Wi-Fi equipment on  $p$ ;
5   if ( $\kappa_p^P \geq 0.5$ ) and ( $\exists v \in V^d : d(p, v) \leq R_{lte}$ ) and ( $\nexists v \in V^m : d(p, v) \leq \delta_{macro}$ )
and ( $\nexists v \in V^s : lte(v) = 1$  and  $d(p, v) \leq \delta_{lte}$ ) then
6     Install a LTE equipment on  $p$ ;

```

---

If a connection by fiber is allowed, we choose the closest FAP or macrocell under the maximum distance constraint (line 12) and connect the pole to the chosen root node, setting it as “parent” of this pole, and placing the pole in the set of children of the root node. As it is a fibered connection, we consider that the pole is in level zero. If it is not possible to connect by fiber, we try to create a wireless link. To do this, we create a list of neighbor root nodes of the pole which will be visited circularly from the point  $i$  determined by the corresponding key in the chromosome. This is done until we obtain a connection (lines 19–31). Note that, in this case, the pole will be in level 1 and its parent will have its incoming degree incremented by one (line 26). In the last case, when it is impossible to make a connection, we remove the pole from the set  $CP$  of current poles and place it in the set  $NL$  of poles for consideration in the next level. The run-time complexity of lines 1 and 2 are  $O(|V^s|)$ . Again, the keys are sorted in line 6 which leads to  $O(|V^s| \log |V^s|)$ . Although the loop of line 7 is relatively long, the most operations are  $O(1)$ . The exception if the inner loop beginning in line 22, which test all root nodes in the worst case. Therefore, the outer loop leads to  $O(|V^s| |V^r|)$ . Since the number of root nodes is usually smaller than the number of utility poles, i.e.,  $|V^r| < |V^s|$ , we can consider the run-time complexity of Algorithm 2.3 to be  $O(|V^s|^2)$ .

After the first level of the backhaul forest is created, set  $CP$  consists in the poles that are forest leaves. From them, Algorithm 2.4 tries to augment the reach of the network. First, we add to list  $NL$  the neighbor poles of  $CP$  that are active, in the line of sight with some pole in  $CP$ , and are not yet part of the forest. We only consider neighbors from poles whose incoming degree is not saturated (line 3). These operations are described in lines 2–5. After constructing these lists, we consider the poles in  $NL$  as current poles and take a permutation according to the order induced by the chromosome keys. The remainder of the algorithm (lines 12–28) is similar to Algorithm 2.3 with respect to wireless

---

**Algorithm 2.3:** Building of backhaul forest (level 0).

---

```

1  $CP \leftarrow \emptyset; TK \leftarrow \emptyset; NL \leftarrow \emptyset;$ 
2 foreach  $p \in V^s$  do  $level(p) \leftarrow -1; fibered(p) \leftarrow 0;$ 
3 foreach  $r \in V^r$  do
4   Let  $N_r^- = \{p \in V^s : (p, r) \in E \text{ and } p \text{ is active}\};$ 
5    $CP \leftarrow CP \cup N_r^-;$ 

6 Let  $\Pi$  to be a permutation of  $CP$  induced by the non-increasing order of
correspondent keys in  $\kappa^O$ ;

7 foreach  $p \in \Pi$  in given order do
8    $minimum\_level \leftarrow \lfloor (H + 1)\kappa_p^L \rfloor;$ 
9    $(\hat{\ell}, \hat{f}, \hat{d}) \leftarrow extractParameters(\kappa_p^P);$ 
10  if  $\hat{d} = 'D'$  and  $minimum\_level \leq 1$  then
11    if  $\hat{f} = 'F'$  then
12      Choose  $r = \operatorname{argmin}_{r' \in V^r} (dist(p, r) - R_{fiber}) \leq 0;$ 
13      if  $r$  exists then
14         $parent(p) \leftarrow r;$ 
15         $children(r) \leftarrow children(r) \cup \{p\};$ 
16         $fibered(p) \leftarrow 1; level(p) \leftarrow 0;$ 
17         $TK \leftarrow TK \cup \{p\};$ 

18    if  $level(p) = -1$  then
19      Let  $N_p^+ = \{m \in V^m : (p, bta(m)) \in A^{smw}\}$  and consider that  $N_p^+$  is in
the order induced by  $V^r$ ;
20       $i \leftarrow \lfloor |N_p^+| \kappa_p^N \rfloor;$ 
21       $begin \leftarrow i; repeated \leftarrow \text{False};$ 
22      while  $level(p) = -1$  and not  $repeated$  do
23        if  $\deg^+(N_p^+[i]) < \delta_{bh}^+$  then
24           $parent(p) \leftarrow N_p^+[i];$ 
25           $children(N_p^+[i]) \leftarrow children(N_p^+[i]) \cup \{p\};$ 
26           $\deg^+(N_p^+[i]) \leftarrow \deg^+(N_p^+[i]) + 1; level(p) \leftarrow 1;$ 
27           $TK \leftarrow TK \cup \{p\};$ 
28        else
29           $i++;$ 
30          if  $i = |N_p^+|$  then  $i \leftarrow 1;$ 
31           $repeated \leftarrow (i = begin);$ 

32    if  $level(p) = -1$  then
33       $CP \leftarrow CP \setminus \{p\}; NL \leftarrow NL \cup \{p\};$ 

```

---

connections. Note that at the end of this procedure, set  $NL$  may have poles that are not connected to any tree. These poles will be eliminated from the solution in a pruning phase. The run-time complexity of Algorithm 2.4 is trick to compute. First, note that in each iteration, the number of connections is a function of the size of current leaves  $CP$  and maximum number of incoming backhaul neighbors  $\delta_{bh}^+$ . Note also that the size of  $NL$  depends on the size of  $CP$  and  $\delta_{bh}^+$  but also depends on the number of poles not connected in the previous iteration and postponed to be connected in the current iteration. Let  $n = |V^s|$ ,  $n_i = |CP|$  and  $m_i = |NL|$  in the iteration  $i$ , and suppose that the algorithm iterates over  $k$  iterations. Note that  $\sum_{i=1}^k n_i = n$  but  $\sum_{i=1}^k m_i \geq n$  due to the carry over of poles from previous iterations. In line 8, for each pole to be connected, we sort the possible neighbors in  $CP$ . Since, in each iteration, there are performed  $m_i$  sortings, the complexity of this piece of code is

$$\begin{aligned} \sum_{i=1}^k (m_i n_i \log n_i) &\leq \sum_{i=1}^k (m_i n_i \log n) \\ &\leq \log n \sum_{i=1}^k m_i n_i \\ &\leq \log n \sum_{i=1}^k nn_i \\ &= n \log n \sum_{i=1}^k n_i \\ &= n^2 \log n, \end{aligned}$$

since  $n_i \leq n$  and  $m_i \leq m$ , for all iterations  $i$ . Therefore, the complexity of these sortings is  $O(|V^s|^2 \log |V^s|)$ . Note that the loop of line 12 iterates over the same frontier nodes used in the previous sortings. Therefore, the run-time complexity is

$$\sum_{i=1}^k m_i n_i \leq n^2 \in O(|V^s|^2).$$

Therefore, the run-time complexity of Algorithm 2.4 belongs to  $O(|V^s|^2 \log |V^s|)$ .

After the complete construction of the backhaul forest, it is possible that there exist active poles not used in the forest, or yet poles present in the forest but not serving any demand. Algorithm 2.5 removes these poles. We have two pruning phases. The first occurs after the construction of the backhaul forest and the second after the maximum flow computation. In the first phase, we generate a “virtual” flow in each arc only present for the convenience of the algorithm. Thus, a recursive pruning procedure is applied in each root node (`pruneSubtree()`). This procedure traverses each tree using the depth-first strategy until it reaches a leaf node. This way it verifies if demand is served by this

---

**Algorithm 2.4:** Building of backhaul forest (level  $\geq 1$ ).

---

```

1 while  $CP \neq \emptyset$  do
2   foreach  $p \in CP$  do
3     if  $\deg^+(p) < \delta_{bh}^+$  then
4       Let  $N_p^- = \{p' \in V^s : (p', p) \in A^{ss}, p' \text{ is active and } p' \notin TK\}$ ;
5        $NL \leftarrow NL \cup N_p^-$ ;
6   foreach  $p \in NL$  do
7      $N_p^+ \leftarrow \{p' \in CP : (p, p') \in A^{ss}\}$ ;
8      $\forall p \in V^s$ , sort  $N_p^+$  in a non-increasing order of  $dist(p, q)$  such that  $q \in N_p^+$ ;
9      $CP \leftarrow NL$ ;
10     $NL \leftarrow \emptyset$ ;
11    Let  $\Pi$  to be a permutation of  $CP$  induced by the non-increasing order of
correspondent keys in  $\kappa^O$ ;
12    foreach  $p \in \Pi$  do
13       $minimum\_level \leftarrow \lfloor (H + 1)\kappa_p^L \rfloor$ ;
14       $i \leftarrow \lfloor |N_p^+| \kappa_p^N \rfloor$ ;
15       $begin \leftarrow i$ ;  $repeated \leftarrow \text{False}$ ;
16      while  $level(p) = -1$  and not  $repeated$  do
17        if  $\deg^+(N_p^+[i]) < \delta_{bh}^+$  then
18           $parent(p) \leftarrow N_p^+[i]$ ;
19           $level(p) \leftarrow level(N_p^+[i]) + 1$ ;
20           $\deg^+(N_p^+[i]) \leftarrow \deg^+(N_p^+[i]) + 1$ ;
21           $TK \leftarrow TK \cup \{p\}$ ;
22        else
23           $i++$ ;
24          if  $i = |N_p^+|$  then  $i \leftarrow 1$ ;
25           $repeated \leftarrow (i = begin)$ ;
26        if  $level(p) = -1$  then
27           $CP \leftarrow CP \setminus \{p\}$ ;
28           $NL \leftarrow NL \cup \{p\}$ ;

```

---

leaf node using either Wi-Fi or LTE (lines 5–8). In case demand is present, the pole is kept. If there is no demand but the pole has children nodes, then we can deduce that the pole is being used only as a retransmitter and we keep the pole in the forest. Otherwise, the pole is marked for later removal from set  $TK$ . Note that when the recursion returns, line 3 removes all marked children. Since this procedure visits just once each pole, its run-time complexity is  $O(|V^s|)$ .

After the forest construction and first pruning phases, it is necessary to create a graph induced by this forest to compute the maximum flow from the demand nodes to the root nodes. As pointed out in Section 2.2.3, the maximum flow problem to be solved is neither classical nor straightforward. To solve this problem, we proposed two solutions which are described in Section 2.6. The run-time complexity of this phase depends on the chosen algorithm to compute the maximum flow. As pointed out in Section 2.2.3, we chose

---

**Algorithm 2.5:** Equipment and poles pruning.

---

```

1 if first phase then
2   foreach  $e \in E$  do
3      $f_e \leftarrow 1;$ 
4   foreach  $r \in V^r$  do
5      $\text{pruneSubtree}(r);$ 
6   Mark as inactive all  $p \notin TK$  that is marked as active;

```

---

**Procedure**  $\text{pruneSubtree}(r)$ .

---

```

1 foreach  $p \in \text{children}(r)$  do
2    $\text{pruneSubtree}(p);$ 
3 Remove from  $\text{children}(r)$ , all  $p$  marked to remotion;
4 if  $r \in V^s$  then
5   if  $\deg^+(wta(r)) = 0$  or  $f_{(wta(r),r)} = 0$  then
6      $wifi(r) \leftarrow 0;$ 
7   if  $\deg^+(lta(r)) = 0$  or  $f_{(lta(r),r)} = 0$  then
8      $lte(r) \leftarrow 0;$ 
9   if  $|\text{children}(r)| = 0$  and  $wifi(r) = lte(r) = 0$  then
10    Mark  $r$  to remotion;
11     $TK \leftarrow TK \setminus \{r\};$ 

```

---

to use the Goldberg and Tarjan [71] push-relabel algorithm whose run-time complexity is  $O(|V|^2\sqrt{|A|})$ . Note that we must consider all nodes  $V$  and all arcs  $A$  as defined in Section 2.4. The pumping algorithm proposed in Section 2.6.2 to augment the flow has run-time complexity  $O(|V^s|)$ .

Computed the maximum flow, a second pruning phase is applied to the forest and all non-used pieces of equipment are removed as described earlier (Algorithm 2.5). Lastly, the revenue and the costs are computed. As aforementioned, these calculations may use different approaches depending on the objective of the study. Here, the revenue is derived from the maximum flow directly as shown in Term (2.3a) of the objective function of MIP (2.3). The cost is computed using the remaining Terms (2.3b–2.3f). The total cost depends on the deployed equipment, deployed fiber, deployment and maintenance costs, and leased traffic. Computed the revenue and costs, the profit is returned as the solution value and fitness of the current decoded chromosome. This phase is pretty simple and has run-time complexity  $O(|V|)$ .

The run-time complexity of the decoder is the sum of the run-time complexities from each component. The dominant ones are from Algorithm 2.4 and the algorithm used to compute the maximum flow. Therefore, the run-time complexity of the decoder is  $O(|V^s|^2 \log |V^s| + |V|^2\sqrt{|A|})$ .

## 2.6 Maximum Backhaul Flow Problem

### 2.6.1 Bounds

As pointed out in Section 2.2.3, wireless backhaul equipment has very specific constraints with respect to the reception and retransmission of backhaul traffic. These constraints are mainly related to the physical proprieties of the wave spectrum used. This way, the total capacity of reception and retransmission is limited to a certain constant  $U_{bh}$ . At the same time, this equipment also collects local traffic sent by other equipment, such as Wi-Fi and LTE receptors.

For a given retransmitter  $v$ , we can assume three distinct components. The first component is the *incoming backhaul traffic*, denoted by  $F_b^i$ , such that  $F_b^i = \sum_{u \in V^s : (u,v) \in A^{ss}} f_{uv}$ , i.e., the sum of backhaul traffic sent to  $v$  from neighbors. The second component is the *access traffic*, denoted by  $F_a$ , which is the sum of the Wi-Fi and LTE traffic in  $v$ , i.e.,  $F_a = f_{wta(v),v} + f_{lta(v),v}$ .<sup>1</sup> The third component is the *outgoing backhaul traffic*, denoted by  $F_b^o$ , such that  $F_b^o = f_{v,parent(v)} = F_b^i + F_a$ . The relationship among these components is given by Inequality (2.1), that restricts the backhaul flow capacity, and by Equation (2.2) that ensures flow conservation. The difficulty is that classical maximum flow algorithms

---

<sup>1</sup>Notation detail:  $f_{u,v} = f_{uv}$

do not deal with these restrictions at the same time and, as far as we know, there is no reduction from the maximum backhaul flow problem to any classical flow problem.

One way to bypass this problem is to consider the access traffic as incoming backhaul traffic. Thus, note that

$$F_b^i + F_a + F_b^o \leq U_{bh}. \quad (2.4)$$

However  $F_b^i + F_a = F_b^o$ , which leads us to

$$F_b^o \leq \frac{U_{bh}}{2} \quad \forall v \in V^s. \quad (2.5)$$

In this case, as both incoming flows are of the same kind, it suffices to bound either the incoming or outgoing flow to half of the original capacity. The major drawback of this approach is the large flow loss that may result. Suppose, for example, a backhaul capacity of  $U_{bh} = 100$ , backhaul incoming flow of  $F_b^i = 30$ , and an access flow of  $F_a = 40$ . Using the previous technique, we will have the nominal capacity of  $U'_{bh} = 50$  in the outgoing arc, which limits the maximum flow to the same value. In this case, 20 units of traffic, either demand or backhaul traffic, cannot be backhauled. But note that using the original constraints, all traffic can be routed since the outgoing traffic would  $F_b^o = F_a + F_b^i = 70$  that respects the capacity constraint ( $F_b^i + F_b^o = 30 + 70 = 100 = U_{bh}$ ). In fact, we can provide a bound on this loss using simple algebra. Note that the most constraining factor is the incoming backhaul traffic. As it tends to zero, it enables the increase of the capacity of the outgoing traffic, thus allowing more access traffic be routed (see Lemma 1 below). Consider Inequality (2.1) in terms of access traffic:

$$\begin{aligned} F_b^i + F_b^o &\leq U_{bh} \\ F_b^i + F_b^i + F_a &\leq U_{bh} \\ F_a &\leq U_{bh} - 2F_b^i. \end{aligned} \quad (2.6)$$

Now consider Inequality (2.4) in terms of access traffic:

$$\begin{aligned} F_b^i + F_a + F_b^o &\leq U_{bh} \\ 2F_b^i + 2F_a &\leq U_{bh} \\ F_a &\leq \frac{U_{bh} - 2F_b^i}{2}. \end{aligned} \quad (2.7)$$

Taking the limit of the proportion between Inequalities (2.6) and (2.7) when the incoming backhaul traffic tends to zero, we have:

$$\lim_{F_b^i \rightarrow 0} \frac{\frac{U_{bh} - 2F_b^i}{2}}{U_{bh} - 2F_b^i} = \frac{1}{2}. \quad (2.8)$$

Therefore, the proposed simplification may cause a loss of up to 50% in access traffic (and, consequently, total traffic) that the network can transport. Although the theoretical bound is not very good, this approach leads to reasonable results in practice, as shown in Section 2.8.2. The following lemmata also give us bounds of the flows in the forest.

**Lemma 1:** Consider a vertex  $v \in V^s$  with backhaul capacity  $U_{bh}$ ,  $F_b^o$  be the outgoing backhaul traffic from  $v$ , and  $F_a$  be the value of the access traffic incoming in  $v$ . Then,  $F_b^o$  is maximum only if  $F_a$  is maximum.

*Proof.* The proof is simple by inspection of the maximality. First, note that we want to maximize  $F_b^o = F_a + F_b^i$ . But, by constraint capacity (2.1), we have that  $F_b^i + F_b^o \leq U_{bh}$  which means that  $F_a + 2F_b^i \leq U_{bh}$ . Let  $0 \leq \hat{F}_a < F_a$  and  $0 \leq F_b^i < \hat{F}_b^i$  and suppose that  $\hat{F}_a$  and  $\hat{F}_b^i$  yield the maximum flow  $\hat{F}_b^o$ . Suppose that  $\hat{F}_b^i$  is maximum which means that  $\hat{F}_b^i = U_{bh}/2$  enforcing  $\hat{F}_a = 0$ . Therefore  $\hat{F}_b^o = U_{bh}/2$ . But choosing  $F_b^i = \hat{F}_b^i - \varepsilon$ , we have that  $F_a + 2(U_{bh}/2 - \varepsilon) \leq U_{bh}$  which is  $F_a \leq 2\varepsilon$ . Therefore  $\max F_b^o = 2\varepsilon + U_{bh}/2 - \varepsilon = \varepsilon + U_{bh}/2 > \hat{F}_b^o$  contradicting the maximality of  $\hat{F}_b^o$ .  $\square$

**Lemma 2:** Let  $v \in V^s$  such that  $v$  is in level 1 or greater in the backhaul forest. Let  $T(v)$  be a subtree of Steiner vertices rooted at  $v$ . For all vertices  $x, y \in T(v)$  such that arc  $(x, y) \in A$ ,  $f_{xy} \leq U_{bh}/2$ .

*Proof.* Let  $N^b(v)$  be the neighbor vertices of  $v$  that send to it backhaul traffic. Also consider  $F_a$  and  $F_b^o$  as defined before.

Let  $x \in T(v)$  such that  $(x, v) \in A$ . Therefore:

$$\begin{aligned} \sum_{u \in N^b(v)} f_{uv} + F_b^o &\leq U_{bh} \\ f_{xv} + \sum_{u \neq x \in N^b(v)} f_{uv} + F_b^o &\leq U_{bh} \\ f_{xv} &\leq U_{bh} - F_b^o - \sum_{u \neq x \in N^b(v)} f_{uv} \\ &= U_{bh} - \left( \sum_{u \in N^b(v)} f_{uv} + F_a \right) - \sum_{u \neq x \in N^b(v)} f_{uv} \\ &= U_{bh} - f_{xv} - 2 \sum_{u \neq x \in N^b(v)} f_{uv} - F_a \\ &= \frac{U_{bh} - 2 \sum_{u \neq x \in N^b(v)} f_{uv} - F_a}{2} \\ &\leq U_{bh}/2. \end{aligned}$$

As  $T(v)$  is a tree, all descendent arcs of  $v$  have their capacities bounded by  $U_{bh}/2$  since  $v$  is the unique output vertex in  $T(v)$ .  $\square$

Lemma 1 shows that it is worthwhile to route the maximum access traffic available at a pole. Although this lemma is valid for all poles, it may be enforced in nodes at level 1 of the forest, since nodes at level 0 have a fiber connection allowing us to route all access traffic and backhaul traffic subject to the processing constraint. For poles at level 2 or above, we may drop Lemma 1 due to Lemma 2. Note that by Lemma 2, the capacity Constraint (2.1) does not play a role at poles at levels 2 or above since all traffic in those poles will respect this constraint.

### 2.6.2 Solution approach

The maximum backhaul flow problem can be solved to optimality using a linear programming formulation derived from Constraints (2.1) and (2.2). The major problem with this approach is that, computationally, it is too slow to be used within the decoder. In Section 2.8.2, experimental results illustrate this problem. Another approach is to map this flow problem into a classical maximum flow problem [71]. One way to implement backhaul capacity constraint (2.1) in a classical maximum flow problem, is to set capacities on arcs instead of node equipment. To guarantee feasibility, one sets the capacities of all arcs connecting pairs of poles and arcs connecting pairs of poles/BTAs to half of the backhaul capacity, i.e.,  $U_{bh}/2$ . Restricting capacity this way enables the utilization of classical flow algorithms at the expense, however, of potentially producing suboptimal flows.

Consider a forest generated with Algorithms 2.2–2.5. In particular, consider the set  $TK$  of poles determined to be in the backhaul network. The maximum flow is computed over the graph induced by  $TK$ . For this, we take all vertices in  $TK$  and create subsets  $WTA'$  and  $LTA'$  restricted to poles in  $TK$ . This means that  $WTA' \subseteq WTA$  and  $LTA' \subseteq LTA$  since not all poles are in the forest and, for some poles, LTE equipment are forbidden. We also create the set  $BTA'$  with vertices that aggregate wireless backhaul traffic in the macrocells. Note that a BTA exists in a macrocell only if it has children connected to it by wireless links. If all children are connected by fiber, then neither backhaul equipment nor a BTA are needed. Vertices in  $V^d$  and  $V^r$  also are considered when they are part of the backhaul forest. We add the vertex  $s$  to be the source node and vertex  $t$  to be the sink. We consider all arcs induced by the chosen vertices and create arcs from  $s$  to all demands and from all root nodes to  $t$ . In the following, define  $cap : E \rightarrow \mathbb{R}^+$  be the capacity of an arc:

- For arc  $a$  incident to  $v \in V^d$ , let  $cap(a) = d_v$ ;
- For arc  $a$ , outgoing from vertex:
  - $v \in WTA$ , let  $cap(a) = U_{wifi}$ ;

- $v \in LTA$ , let  $\text{cap}(a) = U_{lte}$ ;
- $v \in MTA$ , let  $\text{cap}(a) = U_{mc}$ ;
- $v \in BTA$ , let  $\text{cap}(a) = U_{bh}$ ;
- $v \in V^r$ , let  $\text{cap}(a) = \infty$ ;
- For each arc  $a \in A^{ss} \cup A^{smw}$ , let  $\text{cap}(a) = U_{bh}/2$ ;
- For each arc  $a \in A^{sv} \cup A^{smf}$ , let  $\text{cap}(a) = U_{bh}$ .

Note that, although the fibered links in set  $A^{sv} \cup A^{smf}$  are considered to have unlimited capacity, we set their capacities to the capacity of retransmitter, modeling the incoming wireless backhaul traffic. In such case, we may lose access traffic if the pole with the fibered link has Wi-Fi and/or LTE traffic. To overcome this, we do the following. Let  $v$  be a pole with a fibered link to some root node  $w$ . We remove the arcs  $(wta(v), v)$  and  $(lta(v), v)$  and add the arcs  $(wta(v), w)$  and  $(lta(v), w)$  with the same respective capacities. Such change allows the maximum access traffic to by-pass pole  $v$  and only limits the incoming backhaul traffic. Since we remove capacity Constraint (2.1), we may use any classical maximum flow algorithm to solve the maximum backhaul flow problem.

As noted above, our approach may generate a suboptimal flow. To improve this, we propose a *pumping algorithm* to augment the generated flow. This algorithm is inspired on push-relabel algorithm of Goldberg and Tarjan [71] using Lemmas 1 and 2. The general idea is to push residual flow from the root vertices to the demand vertices observing Lemmas 1 and 2 and the capacity constraints. For each vertex  $v$ , let  $\text{excess}(v)$  be the excess flow in  $v$  that must be pushed away. Algorithm 2.6 considers each root vertex and pumps flow through its subtrees. Lines 3–7 treat the fibered connections. For each child pole, the maximum flow increment is computed and passed to it as excess traffic. Then a procedure applied only to poles in level zero is called, and upon return, the flow is accumulated. In lines 10–19, the wireless connections are considered. In this case, the maximum flow of  $U_{bh}$  must be shared with all wirelessly connected children. This is done by computing the maximum flow from the remaining capacity and excess. Since wireless children are considered be in level two or greater, a special procedure is called to treat this case. Again, the totals are accumulated.

Algorithm 2.7 deals with poles at level zero. Since the above described by-pass guarantees that access traffic is maximum, one can limit their attention to only the incoming backhaul traffic. The algorithm just accumulated the access traffic (line 1) and pumped the maximum allowed flow to the children poles. In the end, if the pole has excess flow, it is pumped back to the parent vertex (line 9).

Algorithm 2.8 deals with poles at level 1 or greater. The basic idea is the same of previous algorithms except that one must pay attention to the backhaul capacity and

---

**Algorithm 2.6:** Pumping Root.

---

```

1 Let  $r$  be a FAP or macrocell;
2  $f_{rt} \leftarrow 0$ ;
3 foreach  $p \in \text{children}(m)$  such that  $\text{fibered}(p) = 1$  do
4    $excess(p) \leftarrow C_{bh} - f_{pr}$ ;
5    $f_{pr} \leftarrow C_{bh}$ ;
6    $\text{pumpPoleLevelZero}(p)$ ;
7    $f_{rt} \leftarrow f_{rt} + f_{pr}$ ;
8 if  $r$  is not a macrocell then
9   return;
10  $excess(r) \leftarrow C_{bh} - f_{bta(r),r}$ ;
11  $f_{bta(r),r} \leftarrow 0$ ;
12 foreach  $p \in \text{children}(r)$  such that  $\text{fibered}(p) = 0$  do
13    $maxflow \leftarrow \min(excess(r), C_{bh} - f_{p,bta(r)})$ ;
14    $excess(r) \leftarrow excess(r) - maxflow$ ;
15    $excess(p) \leftarrow excess(p) + maxflow$ ;
16    $f_{p,bta(r)} \leftarrow f_{p,bta(r)} + maxflow$ ;
17    $\text{pumpPoleLevelOneorMore}(p)$ ;
18    $f_{bta(r),r} \leftarrow f_{bta(r),r} + f_{p,bta(r)}$ ;
19    $f_{rt} \leftarrow f_{rt} + f_{p,bta(r)}$ ;

```

---



---

**Algorithm 2.7:** pumpPoleLevelZero( $p$ ).

---

```

1  $f_{p,parent(p)} \leftarrow f_{wta(p),p} + f_{lta(p),p}$ ;
2 foreach  $c \in \text{children}(p)$  do
3    $maxflow \leftarrow \min(excess(p), C_{bh} - f_{cp})$ ;
4    $excess(p) \leftarrow excess(p) - maxflow$ ;
5    $excess(c) \leftarrow excess(c) + maxflow$ ;
6    $f_{cp} \leftarrow f_{cp} + maxflow$ ;
7    $\text{pumpPoleLevelOneorMore}(c)$ ;
8    $f_{p,parent(p)} \leftarrow f_{p,parent(p)} + f_{cp}$ ;
9    $excess(\text{parent}(p)) \leftarrow excess(\text{parent}(p)) + excess(p)$ ;
10   $excess(p) \leftarrow 0$ ;

```

---

---

**Algorithm 2.8:** pumpPoleLevelOneorMore( $p$ ).

---

```

1  $bf \leftarrow \sum_{c \in children(p)} f_{cp};$ 
2 Let  $\omega$  be  $wta(p)$ ,  $\phi$  be  $f_{wta(p),p}$ ,  $\Phi$  be  $f_{lta(p),p}$ , and  $\Gamma$  be  $C_{wifi}$ ;
3  $maxflow \leftarrow \min(excess(p), \Gamma - f_{\omega p});$ 
4 if  $level(p) = 1$  then
5    $maxflow \leftarrow \min(maxflow, C_{bh} - 2bf - \Phi);$ 
6 else
7    $maxflow \leftarrow \min(maxflow, C_{bh}/2 - (bf + \phi + \Phi));$ 
8 foreach  $b \in V^d$  such that  $(b, \omega) \in E$  do
9    $maxinc \leftarrow \min(maxflow, f_{sb} - f_{b\omega});$ 
10   $f_{b\omega} \leftarrow f_{b\omega} + maxinc; \phi \leftarrow \phi + maxinc;$ 
11   $maxflow \leftarrow maxflow - maxinc; excess(p) \leftarrow excess(p) - maxinc;$ 
12 if  $\omega = wta(p)$  then
13   Let  $\omega$  be  $lta(p)$ ,  $\phi$  be  $f_{lta(p),p}$ ,  $\Phi$  be  $f_{wta(p),p}$ , and  $\Gamma$  be  $C_{lte}$ ;
14   Go to line 3;
15  $f_{p,parent(p)} \leftarrow f_{wta(p),p} + f_{lta(p),p};$ 
16 if  $level(p) = 1$  then
17    $residue \leftarrow (C_{bh} - f_{p,parent(p)})/2;$ 
18 else
19    $residue \leftarrow (C_{bh}/2) - f_{p,parent(p)};$ 
20  $pushable \leftarrow \min(residue - bf, excess(p));$ 
21 foreach  $c \in children(p)$  do
22    $maxflow \leftarrow \min(pushable, C_{bh} - f_{cp});$ 
23   if  $maxflow \leq 0$  then
24      $f_{p,parent(p)} \leftarrow f_{p,parent(p)} + f_{cp};$ 
25     Go to line 21;
26    $t \leftarrow excess(p);$ 
27    $excess(p) \leftarrow excess(p) - maxflow; excess(c) \leftarrow excess(c) + maxflow;$ 
28    $f_{cp} \leftarrow f_{cp} + maxflow;$ 
29   pumpPoleLevelOneorMore( $c$ );
30    $f_{p,parent(p)} \leftarrow f_{p,parent(p)} + f_{cp}; bf \leftarrow bf - excess(p) + t;$ 
31    $pushable \leftarrow \min(residue - bf, excess(p));$ 
32  $excess(parent(p)) \leftarrow excess(parent(p)) + excess(p);$ 
33  $excess(p) \leftarrow 0;$ 

```

---

Lemmas 1 and 2. Considering Lemma 1, lines 2–14 aim to first maximize the access traffic. This block is considered twice: once for Wi-Fi and once for LTE. Because of this, we rename some terms to reduce the algorithm (lines 2 and 13). After the maximization of the access traffic, lines 20–31 try to push the remaining flow to the children nodes using a recursive call. As in other pumping algorithms, in the last two lines the excess is pumped back to the parent vertex. The key of this algorithm are lines 4–7 and 16–19. If the pole is in level 1, the access flow is given by Equation (2.7) using simple substitution (the same occurs for the backhaul traffic in line 16). If the pole is in level two or greater, Lemma 2 comes into scene limiting the traffic to at most  $C_{bh}/2$ . In this case, we can consider that all traffic flows are of the same type and calculate the maximum local flow from the residue flow of all types. This ensures that the flow through that pole will respect the capacity constraint.

Note that the proposed pumping heuristic has no relabel phase as in the push-relabel algorithm. The pumping algorithm ends after no more pushing is possible in the recursive calls and, therefore, its run-time complexity is  $O(|V|)$ . Furthermore, the resulting flow is sensitive to the order that the poles are visited. Although the optimum flow is not guaranteed to be found, the pumping heuristic can improve the flow considerably (see Section 2.8.2 for more details).

## 2.7 Experimental Setup

### 2.7.1 Instances and scenario descriptions

In this section, we describe the setup of the computational experiments performed to analyze the algorithms. The experiments were conducted using 30 instances derived from real-world scenarios. These instances are taken from neighborhoods of a large city in the United States. Each instance consists of a set of macrocells, VRADs, utility poles, and demand blocks. For each location, longitude and latitude coordinates are given. For each macrocell and utility pole, a list of street segments is given. We assume that if two locations share a segment, they are in the line of sight of each other. For each block, a traffic demand is given. For each macrocell and VRAD, there is an indication of whether traffic through them is leased or not. We classify the instances as small, medium, and large according the number of poles. Each class has ten instances. Table 2.1 shows a summary and Table A.1 (in Appendix A.1) brings a complete description. The areas of the regions were computed for illustrative purposes only. The calculation of each area was based on the convex hull considering all locations in the region and their geodesic characteristics. Instance **re01** is the smallest in terms of number of poles with 454 poles

while instance `re30` is the largest with 8740 poles. In terms of area, the smallest instance is `re19` with  $4.82 \text{ km}^2$  and the largest is `re30` with  $411.71 \text{ km}^2$ .

While all locations are real, the demand values are based on estimates of the actual demand and are scaled in an arbitrary range. The access radii, minimum distances, capacities, and backhaul constraints are real life constraints and are displayed in Table 2.2. We also consider a scenario where the backhaul trees have restrictions neither in depth nor in breadth, and therefore the maximum number of hops  $H$  and the maximum number of incoming backhaul links  $\delta_{bh}^+$  are unlimited (in practice, they are the number of poles in the instance). We call this scenario *unrestricted* in opposition to the *restricted* real life scenario. The revenue factor and the costs are based on actual values but are also scaled in an arbitrary range. It is worthwhile to mention that the revenue factor, costs, and demands were scaled similarly so as to mimic real world values. The size of the fibered hop,  $\ell_{uv}$ , is defined by the geodesic distance, in meters, between locations  $u$  and  $v$ . We consider a 3-year planning horizon, i.e.  $Y = 3$ .

### 2.7.2 Instance preprocessing

The instance preprocessing aims to reduce the size of the instance and build the base graph that represents the potential wireless and fibered links, and the arcs representing the links between the demands points and access equipment. This graph is built using the definitions of Section 2.4. Note that, due to the minimum distance constraint between a macrocell and a LTE equipment, a pole  $u$  has a LTA associated with if and only if for each macrocell  $v$ ,  $\text{dist}(u, v) \geq \delta_{macro}$ .

The first step is to prune poles that will never be used in feasible solutions. To do this, we calculate the shortest path from each root node to each utility pole annotating the size of the shortest path from any root node to that utility pole. We consider that wireless arcs have weight one and fibered arcs have weight zero. Such paths represents the minimum level that a pole can have in the forest. Let  $q$  be the length of the shortest path from pole  $u$  to its closest root node. All poles for which  $q$  is greater than the maximum

Table 2.1: Summary of instance characteristics. The presented values are averages of the numbers of respective locations and are rounded to the next integer (except the demand and area).

Type	Poles	VRADs	Macros	Blocks	Demand (Mbps)	Area ( $\text{km}^2$ )
Small	718	63	10	3907	8210.70	35.92
Medium	2281	86	14	17306	36348.00	72.14
Large	6396	243	22	25566	53601.00	132.87

number of hops allowed (i.e.,  $q > H$ ) are eliminated since they cannot be used in any valid solution. Note that the corresponding WTA and LTA vertices are also deleted.

The distances are also used to create the  $x$  variables of MIP (2.3). We only define the variables  $x_{uv}^p$ , for  $p = q, \dots, H$ , and  $(u, v) \in A$ . Note that as  $u$  can be in level  $q$  or greater, the outgoing backhaul link  $(u, v)$  can only be in level  $q$  or greater. This preprocessing significantly reduces the size of the MIP and, consequently, the computational time needed to solve it.

Another important observation is that several demand blocks may be served by the same group of poles and macrocells. This is particularly true for residential buildings

Table 2.2: Description of equipment capacities, design constraints, revenue factor, and costs. The values reflect usual assumptions made in the practice.

Short Description	Symbol	Value	Unit
Wi-Fi radius	$R_{wifi}$	100	Meters
LTE radius	$R_{lte}$	400	
Macrocell radius	$R_{mc}$	3000	
Retransmitter radius	$R_{bh}$	1000	
Wi-Fi capacity	$U_{wifi}$	100	Mbps
LTE capacity	$U_{lte}$	20	
HSPA capacity	$U_{mc}$	25	
Retransmitter capacity	$U_{bh}$	100	
LTE to LTE min. distance	$\delta_{lte}$	300	Meters
LTE to macro min. distance	$\delta_{macro}$	500	
Max. fiber size	$R_{fiber}$	300	
Max. # of incoming links	$\delta_{bh}^+$	5 / $\infty$	Units
Max. # of wireless hops	$H$	2 / $\infty$	
Revenue factor	$\varrho$	150.00	Monetary units
Equipment deployment	$C_p$	90.00	
Wi-Fi	$C_{wifi}$	12.00	
LTE	$C_{lte}$	70.00	
Backhaul equip. (fan-in = 1)	$C_{fan1}$	40.00	
Backhaul equip. (fan-in $\geq 2$ )	$C_{fan2}$	70.00	
Maintenance (annual)	$C_{man}$	120.00	
Macrocell (annual)	$C_{mc}$	1050.00	
Meter of deployed fiber	$C_{fiber}$	12.00	
Leased traffic (in \$/Mbps)	$C_{ld}$	10.00	

and commercial areas. In such cases, we group these blocks making a super block whose demand is the sum of the demands of the original blocks. However, at the conclusion of the optimization, it will be necessary to “ungroup” these super blocks and redistribute the access flow to the original blocks.

### 2.7.3 Post-optimization flow recomputation

At the conclusion of the BRKGA iterations, we obtain an optimal or near-optimal solution using the strategy described in Section 2.6.2. One may note that if we compute the exact flow using the forest structure of the best solution found so far, we may be able to improve its objective function value. Note that this is true since this best solution was obtained using the heuristic maximum flow algorithm, a lower bound of the actual maximum flow. In view of this fact, at the end of the BRKGA iterations, we recompute the maximum flow for the best solution found using the linear programming model with Constraints (2.1) and (2.2). As we compute the exact flow just once, the execution time of entire algorithm is not compromised.

The new linear programming based flow may traverse new paths. In some cases, some devices will no longer serve demands and can be disregarded. We can apply Algorithm 2.5 again to prune such unused equipment. Note that this post-processing can potentially further reduce the costs and improve the overall solution.

### 2.7.4 Computational environment and parameters

The experiments were conducted on identical machines with four-core Intel Xeon 2.4 GHz CPUs (two threads per core) and 50 GB of RAM running GNU/Linux. Running times reported are UNIX real wall-clock times in seconds, excluding the effort to read the instance. The algorithms are implemented in C++ and we use the GNU `g++` compiler version 4.8. Random numbers were generated by an implementation of the Mersenne Twister (Matsumoto and Nishimura [143]). We used the Lemon library (Dezső et al. [47]) to implement the graph structures and compute the maximum flow using its push-relabel algorithm implementation.

To tune the BRKGA parameters, we use the *iterated racing* procedure (Birattari et al. [21]). This method consists in sampling configurations from a particular distribution, evaluating them using either the Friedman test or the *t*-test, and refining the sampling distribution with repeated applications of F-Race. We use the `irace` package (López-Ibáñez et al. [135]), implemented in R, for parameter tuning. For each heuristic, we use a budget of 1,000 experiments in the tuning procedure, where each experiment was limited to one hour. To tune the BRKGA parameters, we used the following ranges: population size  $\in [300, 2000]$ , elite percentage  $\in [0.15, 0.30]$ ; percentage of mutants introduced at each

generation  $\in [0.10, 0.20]$ ; probability of inheriting each allele from elite parent  $\in [0.5, 0.8]$ ; number of independent populations  $\in \{1, 2\}$ ; exchange interval  $\in [50, 200]$ ; number of elite individuals in an exchange  $\in [1, 2]$ ; and reset population  $\in [300, 700]$ .

The following values were recommended by `irace`. The population size was set to  $p = 500$ , the elite size to  $p_e = \lceil 0.30p \rceil$ , and the number of mutants to  $p_m = \lfloor 0.15p \rfloor$ . The probability of inheriting each allele from the elite parent was  $\rho_e = 0.70$ . We used the island model (Whitley et al. [206]) with three independent and concurrent populations where every 100 generations each population exports its best solution to the other populations. After 300 generations without improvement, all populations are reset to vectors of random keys. We use four simultaneous cores for decoding.

To solve the MIP, we used IBM ILOG CPLEX Optimizer version 12.6.0.0. We set CPLEX to use a maximum of 40 GB of memory, using at most 40 GB of disk memory when necessary. We allowed CPLEX use four threads in parallel. All other parameters were kept at their default values. We use a short run of the BRKGA to generate an incumbent solution for CPLEX. For this, we use BRKGA with the same parameters as above but limit its run to 100 iterations or 10% of maximum time, whichever comes first.

We also tested a multi-start algorithm that uses the decoder of Section 2.5.2. In each iteration, the multi-start algorithm generates a random vector and uses the decoding function to obtain a solution. It also keeps the best solution over all iterations. In the end, the post-processing is applied to the best solution.

Thirty independent runs were performed for the BRKGA and the multi-start algorithm. Since CPLEX is an implementation of an exact algorithm<sup>2</sup>, a single run for each instance was performed. We carried out two types of experiments, one limiting the running time of each algorithm to one hour and another to five hours. The limit of one hour enables network designers to work with several models in a manageable time while the limit of five hours enables a more thorough search. For both the BRKGA and the multi-start algorithm, we use an additional stopping criterion: 1,000 generations (or iterations) without improvement of the best solution.

## 2.8 Experimental Results and Discussion

### 2.8.1 Instance preprocessing

As discussed in Section 2.7.2, it is important to preprocess the instance in order to reduce its size before optimization. With respect to the number of poles, the preprocessing phase in the restricted scenario achieved an average reduction of  $13.00 \pm 16.39\%$  ( $\min =$

---

<sup>2</sup>According to its documentation, the IBM ILOG CPLEX Optimizer, version 12.6.0.0 is fully deterministic when used with its default parameters (as done in our experiments).

0.59, 1<sup>st</sup> Qu. = 2.31, median = 6.06, 3<sup>rd</sup> Qu. = 17.97, max = 67.61). For the unrestricted scenarios, the reduction was  $4.04 \pm 7.68\%$  (min = 0.15, 1<sup>st</sup> Qu. = 0.58, median = 1.30, 3<sup>rd</sup> Qu. = 3.82, max = 39.40), since there is no restriction imposed on depth and breadth of the forest. As expected, this reduction has more impact on the MIP than it does on the BRKGA.

The reduction in the number of demand blocks was huge in both scenarios:  $95.96 \pm 1.88\%$  of the original number of blocks for the restricted scenarios (min = 88.26, 1<sup>st</sup> Qu. = 95.49, median = 96.25, 3<sup>rd</sup> Qu. = 97.04, max = 98.05), and  $95.65 \pm 2.69\%$  of the original number of blocks for the unrestricted scenarios (min = 85.67, 1<sup>st</sup> Qu. = 95.45, median = 96.17, 3<sup>rd</sup> Qu. = 96.98, max = 98.05). This fact is mainly due to the concentration of demand blocks in residential buildings and commercial areas. On average, each super block corresponds to  $30.97 \pm 11.43$  original blocks.

The instance graphs that result from preprocessing are sparse. In the restricted instances, the number of vertices varies from 717 to 25,690 and the number of arcs from 6,917 to 314,229. The graph density, given by  $2|A|/(|V|(|V| - 1))$ , has an average of  $0.0056 \pm 0.0063$ . For unrestricted instances, the number of vertices varies from 1,290 to 27,322 and the number of arcs from 10,224 to 315,330. The graph density has an average of  $0.0043 \pm 0.0035$ . Detailed results can be found in Tables A.2 and A.3 in Appendix A.

## 2.8.2 Computing flow during the optimization

In Section 2.6, one can see that the maximum flow problem embedded in the WBNDP is not trivial and, as far as we know, a fast combinatorial algorithm to solve it does not exist in the literature. As commented in Section 2.6.2, one can solve this flow problem with a linear programming formulation using Constraints (2.1) and (2.2), but this approach can be too slow to be used in the decoding procedure. Therefore, in the same section, we presented a fast heuristic, coupled with *pumping*, to compute the maximum flow. Given a forest, this section studies the effects of choosing one or the other strategy to compute the maximum flow.

The first experiment consists of 1,200 independent runs of the decoder. For each run a random chromosome was generated and decoding was done twice: once using the heuristic flow algorithm and once using the exact flow algorithm. Then, the proportional difference in the flow values and solution times were computed. The heuristic flow algorithm was able to compute an average of  $95.19 \pm 2.68\%$  of the maximum flow computed by the exact flow algorithm (min = 79.66, 1<sup>st</sup> Qu. = 93.58, 3<sup>rd</sup> Qu. = 96.47, max = 100.00). We consider this performance very good even in light of Lemma 1. However, the computing times were extremely different. Since the decoding process is very fast, we chose to compare the number of CPU ticks used by each algorithm. The exact flow algorithm used an average

of  $2027 \pm 2080.42\%$  more CPU ticks than did the heuristic flow algorithm ( $\min = 200$ , 1<sup>st</sup> Qu. = 800, 3<sup>rd</sup> Qu. = 2450, max = 15100). This means that the exact flow algorithm is three orders of magnitude slower than the heuristic flow algorithm.

Figure 2.2 shows the evolution of the profit as a function of number of iterations and CPU wall-clock time. For this, we used instance **re30** and let BRKGA evolve for 100 iterations. In Figure 2.2a the X axis represents the number of iterations and, in Figure 2.2b, the wall-clock time in seconds in log scale. The Y-axis represents the scaled profit in both figures (see the definition of scaled profit in the beginning of Section 2.8.3). The shaded area represents the standard deviation. The red line and dots represent the algorithm using exact flow computation, while the blue squares and line represent the algorithm using the heuristic flow. The green triangles and line represent the utilization of heuristic flow and post-processing. In the experiment with the heuristic flow computation and post-processing, the exact flow was recomputed (followed by pruning) in each iteration. One may note in Figure 2.2a that both heuristic and exact flow are able to generate almost the same profit in a given iteration. The post-processing approach performs better than the others since it has a second phase pruning as described in Section 2.7.3. In terms of running time, one can note in Figure 2.2b that the exact flow algorithm obtains about the same profit as the heuristic but using much more time.

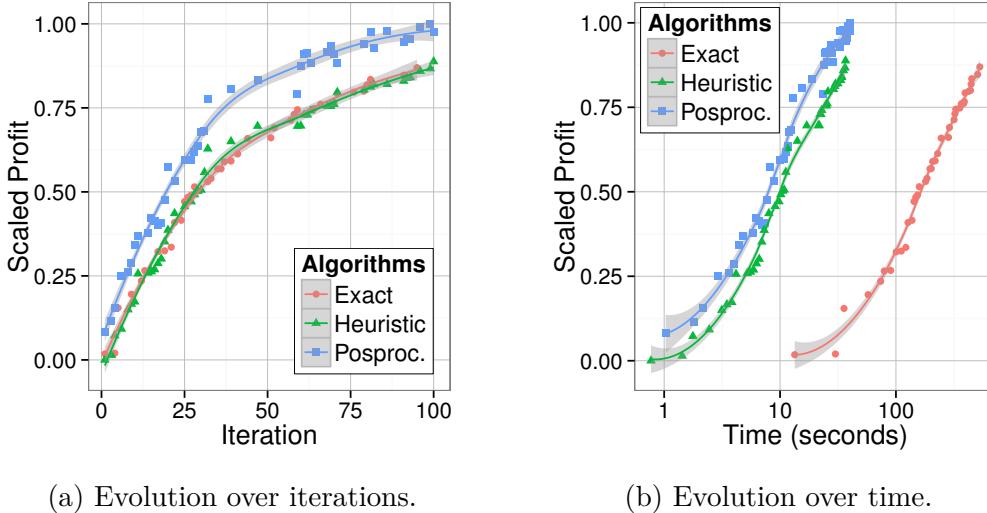


Figure 2.2: Evolution of the profit using different flow algorithms. The shadow around the curves represents the standard deviation.

### 2.8.3 Comparing the profit generated by the algorithms

To compare the algorithms with respect to profit, it is necessary to scale the results since instances can have very different profit values. For each instance  $\mathcal{I}$ , let  $\chi_{\mathcal{I}}$  be the set of values of the solutions found for  $\mathcal{I}$ , and  $D_{\mathcal{I}} = \max(\chi_{\mathcal{I}}) - \min(\chi_{\mathcal{I}})$ . The scaling is done by the simple transformation

$$\chi'_{\mathcal{I}} = \begin{cases} (x - \min(\chi_{\mathcal{I}}))/D_{\mathcal{I}} & \forall x \in \chi_{\mathcal{I}} \text{ and } D_{\mathcal{I}} > 0, \\ 1 & \text{otherwise,} \end{cases}$$

where  $\chi'_{\mathcal{I}}$  is the set of scaled values. Note that all values are scaled to the range  $[0, 1]$ .

Using this scaling process, Figure 2.3 shows the distribution of profits for each algorithm. The box plots show the location of the first quartile, the profit median, and the third quartile. The whiskers extend to the most extreme revenue no more than 1.5 times the length of the box. The dots are the outliers. In the bar labels, **MS** stands for the multi-start algorithm, **MIP** stands for the exact algorithm using CPLEX and Formulation (2.3), and **BRKGA** is the biased random-key genetic algorithm. In the restricted scenario, **MIP** was able to overcome **BRKGA** for most small instances on the one-hour experiments and on most small and medium instances on the five-hour experiments. **BRKGA** presented a small variation in its results, although close to those of the **MIP** in both cases. **MS** presented a large variation and its results were the worst in all cases. This is due to, mainly, the decoder has no local search procedures (unless the pumping algorithm). This fact just reinforces the importance of the learning mechanism of **BRKGA**. For medium size instances, **BRKGA** presented more solid results while **MIP** showed more variation for one-hour experiments. In the five-hour runs, the results were similar to those for the small instances. For large instances, **BRKGA** was able to produce very good results when compared to **MS** and **MIP** in both cases. In fact, **MIP** does not produce any results besides the incumbent value generated by the **BRKGA** short run in large instances considering the limit of one hour. For five hours, only in two instances did **MIP** improve the incumbent. In the unrestricted scenario, **BRKGA** presented the best results. In fact, for all medium and large instances, **MIP** was not able to produce any solution (because of memory issues in building the model).

To confirm the results presented in Figure 2.3, we tested the normality of these distributions using the Shapiro-Wilk test and applied the Mann-Whitney-Wilcoxon U test, considered more effective than the *t*-test for distributions sufficiently far from normal and for sufficiently large sample sizes (Conover [38], Fay and Proschan [57]). For all tests, we assume a confidence interval of 99%. For small, medium, and large instances, the Shapiro-Wilk tests revealed that no profit distribution fits a normal distribution since the *p*-values for all tests are less than 0.01. Therefore, we applied the U test which assumes

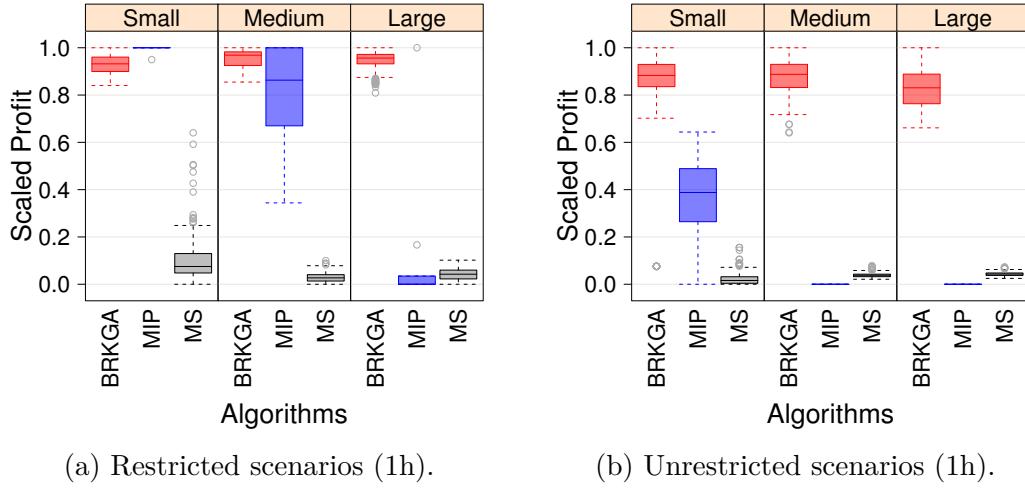


Figure 2.3: Dispersion of profit for each algorithm. Each section corresponds to a instance class.

as null hypothesis that the location statistics are equal in both distributions. As several statistical tests were performed, we used a  $p$ -value correction procedure based on false discovery rate (FDR) to minimize the number of false positives (Type I error) as indicated by Benjamini and Hochberg [18].

Table 2.3 shows U test results for each pair of algorithms and different instance sizes of the restricted scenario, at a 99% confidence level. The structure of this table is as follows: Each row and column is indexed by one algorithm. Each element in the diagonal (bold) is the median of the scaled profit of the corresponding algorithm. The upper-right diagonal elements are the differences in location statistics for each pair of algorithms. A positive difference indicates that the “row algorithm” has its location statistics higher

(better) than the “column algorithm,” and the negative difference is the opposite. The bottom-left diagonal elements are the  $p$ -values of each test. We omitted all  $p < 0.01$  values, that indicate that the difference is statistically significant for those pairs. We also omitted confidence intervals since for all tests the values lie in these intervals and are very narrow. One can notice that almost all comparisons are statistically significant, confirming the box plot results. The exception is BRKGA and MIP for medium instances using one-hour runs for which the test was inconclusive since  $p > 0.01$ . MS is significantly worse than the other algorithms except for the one-hour MIP experiments on large instances. Summarizing, BRKGA was better than MIP on large instances and the opposite was true for the small instances. For medium-size instances, both apparently performed in a similar way although we cannot affirm this since  $p > 0.01$ . For the unrestricted scenario, the statistical test only makes sense for the small instances since for the medium and large instances, MIP did not produce any feasible solution. In this case, BRKGA presented the median of 0.89, MS presented 0.01, and MIP presented 0.40. The tests indicate that all differences are significant, confirming the box plot.

Table 2.4 reports the performance of the algorithms. The first column indicates the instance class and the second column is the name of the algorithm. The two large blocks consider experiments limited to one hour and five hours, respectively. Each block has four columns. Column “% Best” represents the percentage of the number of instances for which

Table 2.3: Difference in median location for profit distributions for the restricted scenario using a confidence interval of 99%. The omitted  $p$ -values are less than 0.008. The diagonal elements represents the medians, the upper-right elements represent the median diffence, and the bottom-left elements represent the  $p$ -values.

Class		1h experiment			5h experiment		
		BRKGA	MIP	MS	BRKGA	MIP	MS
Small	BRKGA	<b>0.92</b>	-0.07	0.84	<b>0.91</b>	-0.08	0.83
	MIP		<b>1.00</b>	-0.92		<b>1.00</b>	0.92
	MS			<b>0.08</b>			<b>0.07</b>
Medium	BRKGA	<b>0.96</b>	0.07	0.92	<b>0.92</b>	-0.08	0.89
	MIP	$p > 0.33$	<b>0.88</b>	-0.85		<b>1.00</b>	0.97
	MS			<b>0.03</b>			<b>0.03</b>
Large	BRKGA	<b>0.94</b>	0.92	0.89	<b>0.97</b>	0.80	0.95
	MIP		<b>0.00</b>	0.03		<b>0.15</b>	0.13
	MS			<b>0.05</b>			<b>0.02</b>

the algorithm found a best solution; column “% Run” shows a percentage of the number of runs on which the algorithm found a best solution. The two columns under label “Prod. diff.” show, respectively, the average of the proportional difference between the value of the best solution found and the achieved value (%), and its corresponding standard deviation ( $\sigma$ ). First, note that MIP presents the same values for % Best and % Run since only one experiment is done per instance. Considering the restricted scenario, MIP found 9 of 10 best solutions, although only one was proved to be optimal in the five-hour run. BRKGA found 20% of best solutions in about 10% of the runs. However, the BRKGA results are stable and its solutions are within about 5% of the best. MS did not find any good solution on any instance. For the medium and large instances, the roles of BRKGA and MIP were exchanged on the one-hour runs. However, MIP obtained all best solutions on the medium-size instances when it was given five hours to run. In general, BRKGA did not find best solutions in several runs but presented very good alternative solutions. MIP found many best solutions but its results varied considerably. In the unrestricted scenario, BRKGA dominates MS and MIP since the latter could not solve any instance. BRKGA found the best solutions in 4.16% of the runs. The average proportional difference between the BRKGA results and the best solutions found was  $13.94 \pm 9.67$ . For detailed results, please refer to Tables A.4 and A.5 in Appendix A.

Table 2.4: Algorithm performance considering the best results found in restricted scenario.

Class	Alg.	1h experiment				5h experiment			
		Best solutions		Prop. diff.		Best solutions		Prop. diff.	
		% Best	% Run	%	$\sigma$	% Best	% Run	%	$\sigma$
Small	BRKGA	20.00	10.42	4.55	2.17	20.00	10.41	5.06	2.34
	MIP	90.00	90.00	3.30	—	90.00	90.00	2.36	—
	MS	0.00	0.00	94.14	124.78	0.00	0.00	94.37	124.71
Med.	BRKGA	60.00	2.50	3.27	2.60	0.00	0.00	6.06	2.91
	MIP	40.00	40.00	21.33	15.17	100.00	100.00	—	—
	MS	0.00	0.00	64.33	5.03	0.00	0.00	66.16	4.71
Large	BRKGA	90.00	3.75	3.63	2.94	90.00	3.75	3.29	4.68
	MIP	10.00	10.00	57.49	7.38	10.00	10.00	50.54	18.44
	MS	0.00	0.00	58.11	8.77	0.00	0.00	65.37	8.01
All	BRKGA	56.67	5.56	3.80	2.65	36.67	4.72	4.81	3.66
	MIP	46.67	46.67	40.54	22.76	66.67	66.67	45.72	23.12
	MS	0.00	0.00	72.20	73.88	0.00	0.00	75.57	74.37

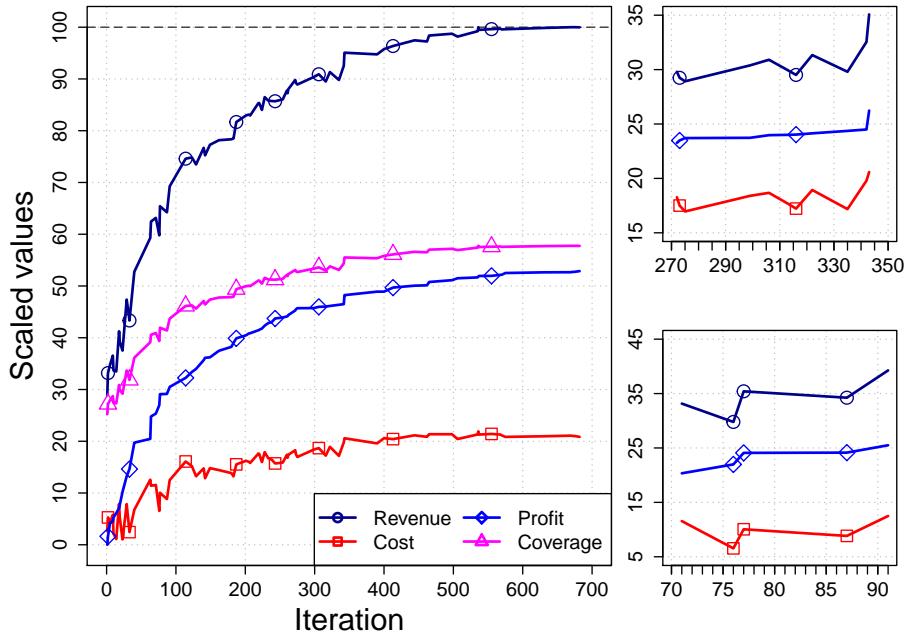


Figure 2.4: Evolution of revenue, cost, profit, and coverage over the first iterations of the algorithm. All values are scaled in the range [0, 100].

#### 2.8.4 Analyzing a solution

One can note that the WBNDP is a problem rich in structure from the point of view of network engineering. Similar to the number of input parameters, a typical output has more than 50 parameters, such as number of pieces of equipment of each type, flows, coverage, costs, revenue, and other metrics besides the network structure itself. In this section, we briefly analyze some of these output parameters considering solutions for the restricted scenario since it is based on real constraints.

Figure 2.4 shows the evolution of revenue (dark blue line with dots), cost (red line with squares), profit (light blue with rhombus), and coverage (magenta dashed line with triangles) for instance `re17` for a given run. Revenue, cost, and profit are scaled to the range [0, 100]. Coverage is already represented in this range. One can note as the coverage increases, so does the revenue. This is expected since revenue is a function of traffic volume. Note that while the profit is monotone increasing, the revenue and cost display some bumps. In the close-up figure showed on the top right, one can see that both revenue and cost vary up and down while profit always non-decreasing. Such cases show a phase transition. When both revenue and cost curves are sloped downwards, the algorithm has found a solution that is less expensive using less equipment. In bottom

right close-up, note that the profit is constant between iterations 77 and 87, but both revenue and cost have slightly negative slopes.

Each tree in the backhaul forest has an average depth of  $2.49 \pm 0.50$ . Each node has an average of  $1.381 \pm 0.76$  incoming neighbors which shows that the fan-in limit is rarely reached. Indeed, the average of the maximum fan-in is  $3.73 \pm 0.64$ , and for some instances, no pole has more than two incoming neighbors. On average,  $48.88 \pm 14.32\%$  of the used poles have only Wi-Fi,  $19.51 \pm 15.42\%$  have only LTE, and  $9.22 \pm 5.98\%$  have both technologies. In  $22.38 \pm 6.93\%$  of the used poles, only retransmitters are installed. In only  $0.34 \pm 0.51\%$  of used poles, can one find a small cell without a retransmitter. Such poles have no neighbors and are linked directly to a FAP or macrocell using fiber.

With respect to traffic, Wi-Fi was responsible for an average of  $77.28 \pm 16.50\%$  of the total covered access traffic while LTE served only  $17.14 \pm 12.59\%$  of the traffic. The macrocells served only  $4.02 \pm 6.70\%$  of the traffic. Wi-Fi has shown itself as an important resource to serve the demand due to its large capacity and low cost when compared to LTE and HSPA equipment. The backhaul networks were able to serve, on average,  $55.78 \pm 16.82\%$  of total demand.

For the 3-year scenario, the deployment cost reached an average of  $5.46 \pm 1.22\%$  of the total cost and the pole annual maintenance was about  $21.86 \pm 4.88\%$ . The average proportional cost of the equipment was the following: Wi-Fi  $0.43 \pm 0.13\%$ ; LTE  $1.26 \pm 0.86\%$ ; retransmitter  $2.78 \pm 0.62\%$ . The cost for fiber trenching was  $57.00 \pm 13.22\%$  and represents the most expensive component of the network. The leased traffic was about  $1.87 \pm 2.60\%$ . However, even with a high cost infrastructure, these backhaul networks can potentially generate a profit margin of  $213.54 \pm 88.32\%$ .

In Figure 2.5, we show an example which is a small portion of a given region where a backhaul network is to be built. This region has three macrocells represented by small antenna figures. Each light green square represents a set of demand points in a specific block. The average number of demand points is 30 per block, but can reach hundreds of residential and commercial buildings. The dark red squares represent the VRAD/FAPs. The very small dark blue squares represent the utility poles. Figure 2.6 shows the equipment deployment and the coverage radii. The symbols with small equipment and two antennas represent Wi-Fi equipment while the blue stars represent LTE equipment. The poles with only retransmitters (backhaul equipment) are represented by a parabolic antenna. The brown and blue circles are, respectively, the Wi-Fi and LTE access radii. One can note that some blocks are not served. In particular, some blocks in the upper right-hand side of the figure are not covered. Figure 2.7 shows the backhaul network. The dashed purple arrows are wireless links, the black solid arrows are fibered links, and the arrows indicate the direction of a root nodes.

## 2.9 Final considerations

We proposed a new problem called the Wireless Backhaul Network Design Problem (WBNDP) which resembles variants of the Steiner tree and the facility location problems. The objective is to build a forest to collect and route wireless traffic. Differing from other problems in the literature, WBNDP uses routed traffic to compute the profit. This traffic is constrained to the network infrastructure with several real-world constraints. We proposed a biased random-key genetic algorithm (BRKGA) to solve the WBNDP. Its decoder relies on building the forest in a bottom-up fashion. We also proposed a mixed integer linear programming model to solve the WBNDP.

BRKGA presented solid results with little variation. It was able to overcome the IBM ILOG CPLEX 12.6 using MIP (2.3), in several medium and large instances of one-hour runs. For longer experiments, BRKGA excelled on large instances. Such results enable BRKGA to be used as an important tool in the planning phase of a wireless backhaul network where, usually fast iterations are required. However, BRKGA also showed itself valuable for longer optimizations, mainly on large instances. The stable results produced by the BRKGA give network engineers a better understanding of the characteristics of an optimal network and makes it easier to modify some assumptions if needed.

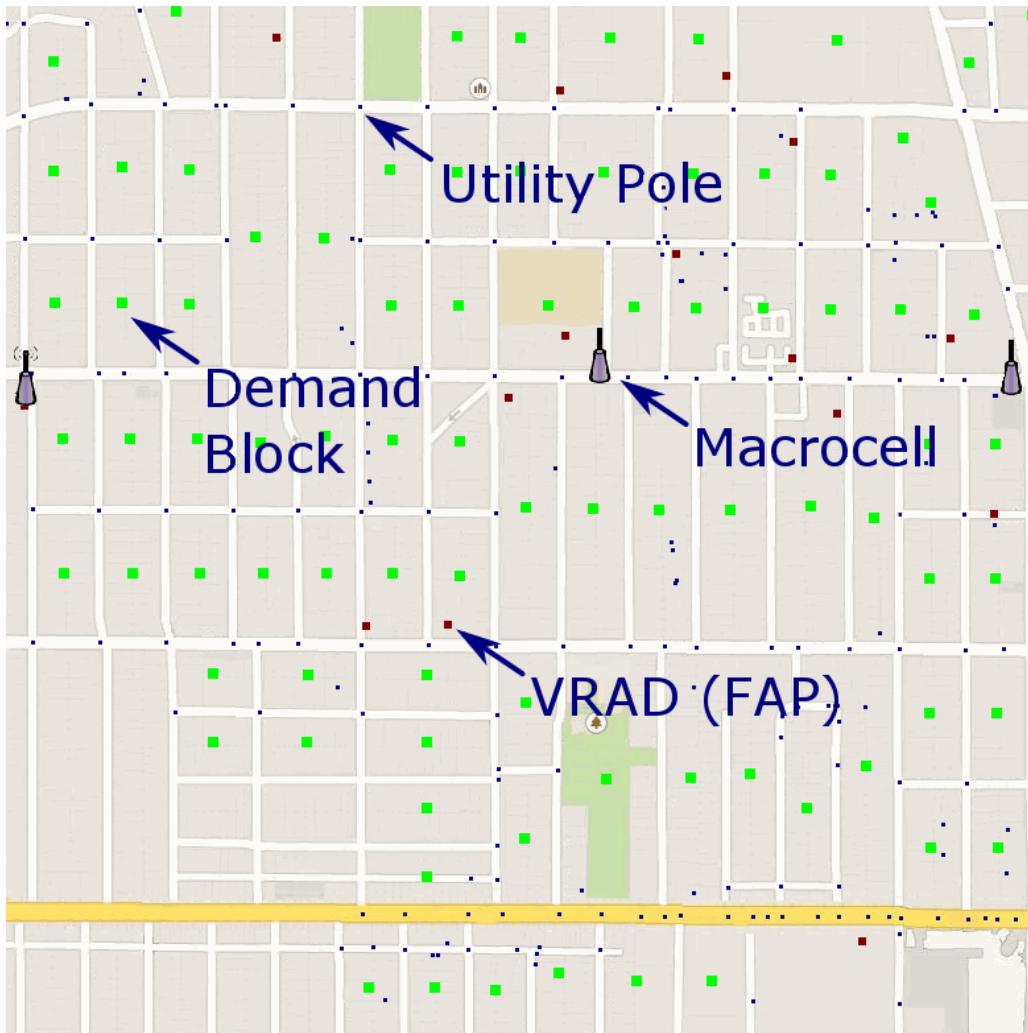


Figure 2.5: Example of region.

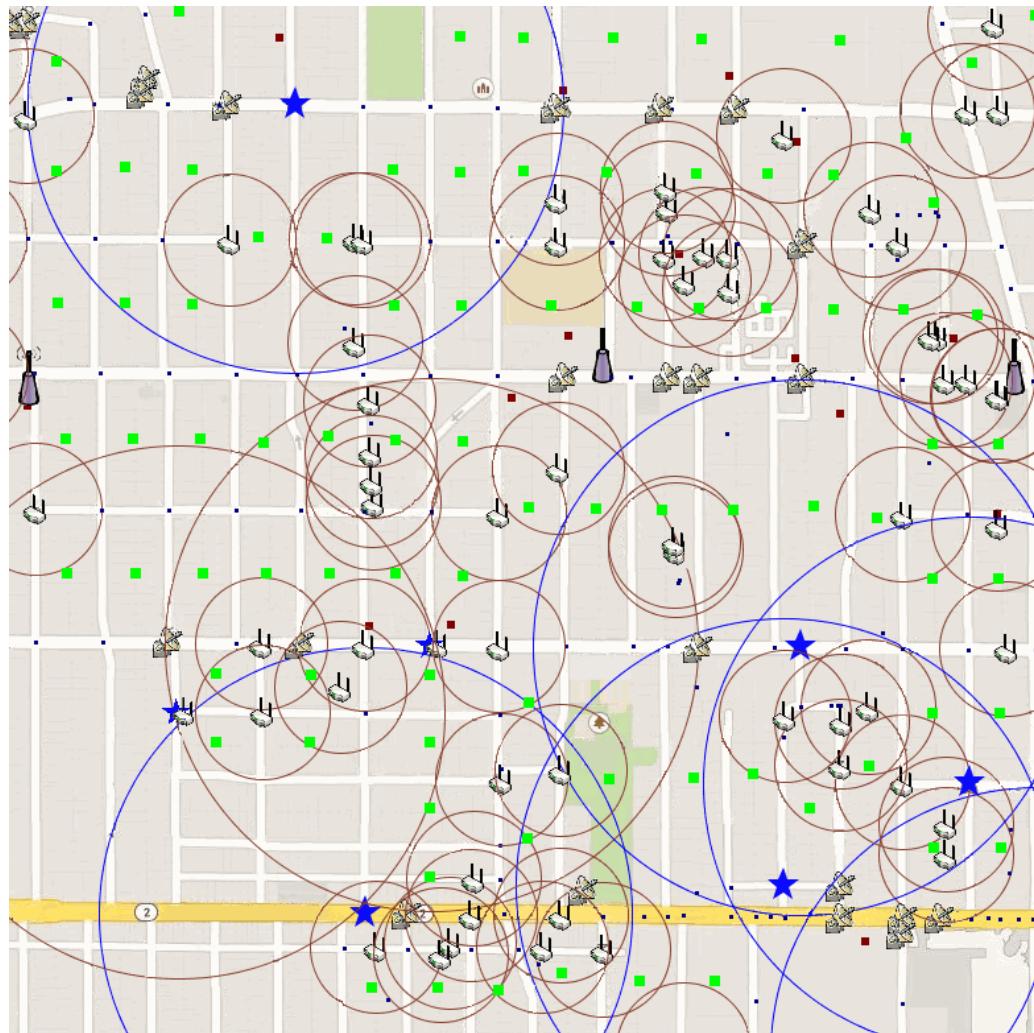


Figure 2.6: Example of coverage.

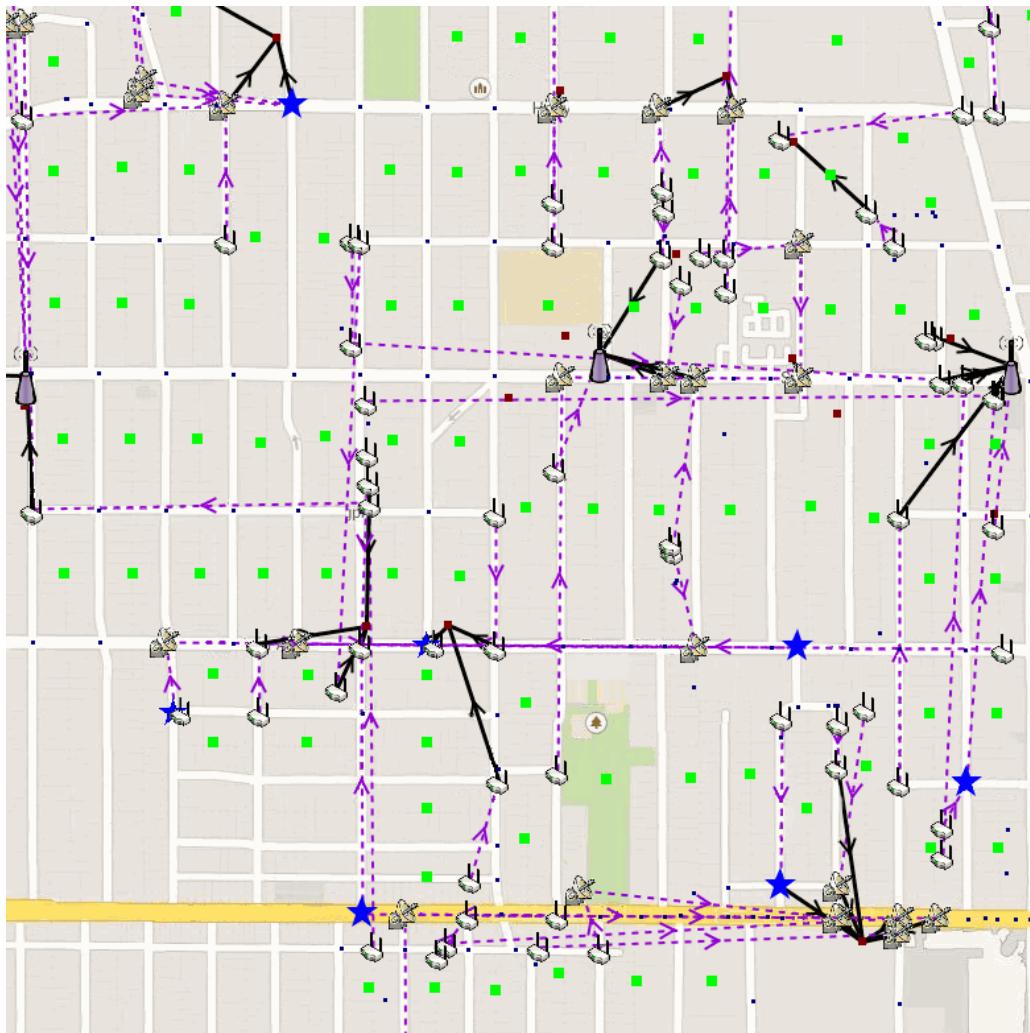


Figure 2.7: Example of backhaul network.

# Chapter 3

## The $k$ -Interconnected Multi-Depot Multi-Travelling Salesmen Problem for backbone network design

**I**N recent years, the huge growth of data utilization has pushed the current backhaul networks to their limits. Among many factors, the on-demand streaming of voice and video are the major consumers of bandwidth. Even using techniques to reduce the bandwidth usage over entire network, such as caching and content delivery prediction, traffic has been very large. On other hand, large band services are able to generate large profits for several tiers in the market: content generators, paid streaming services, and advertisement in free services. All these tiers have pushed telecommunication companies to improve the quality of their services.

In this chapter, we introduce the  $k$ -Interconnected Multi-Depot Multi-Travelling Salesmen Problem that can be used to plan backhaul networks. This problem resembles some network design and location routing problems but carries the inherent difficulty of not having a fixed set of depots or terminals. We propose a heuristic based on a biased random-key genetic algorithm to solve it. This heuristic uses local search procedures to best choose the terminal vertices and improve the tours of a given solution. We compare our heuristic with a multi-start procedure using the same local improvements and we show that the proposed algorithm is competitive. This chapter is based on Andrade et al. [4] and Lopes et al. [133].

## 3.1 Introduction

Many practical routing and scheduling problems seek to determine sequences of actions subject to one or more constraints. Most of these problems can be modeled as variants of the Traveling Salesman Problem (TSP). Given an undirected weighted graph, the TSP consists in finding a Hamiltonian cycle of minimum weight. Other practical applications extend this concept, requiring the salesman to visit a given subset of the vertices. Problems such as the vehicle routing problem and the location routing problem are examples of these extensions. Such problems are also usually used to model telecommunication networks since the structures of a freight network and a data network are very similar.

We propose a variant of this family of problems called the  *$k$ -Interconnected Multi-Depot Multi-Traveling Salesmen Problem* ( $k$ -IMDMTSP) where we choose a subset of *terminal vertices* of size  $k$  from a base graph and connect these vertices with a cycle (called the *inner cycle*). Furthermore, for each terminal vertex, we build a cycle that begins and ends at the terminal and covers a subset of vertices from the remaining graph (called the *outer cycles*). All outer cycles must be pairwise disjoint, span all vertices, and have at most  $C$  vertices ( $C$  is the *capacity* of the outer cycle). The objective is to minimize the sum of edge weights of the inner and outer cycles. This problem occurs in the design of ring networks used to prevent communication failures in presence of single link-fault events. Another application is in transportation networks that use several echelons with different external constraints, such as vehicle types, transportation rules, and service-level agreements.

This chapter is organized as follows. In Section 3.2 we present related work. In Section 3.3 we formalize the  $k$ -IMDMTSP. We then present a description of our heuristic in Section 3.4. Experimental results are provided and discussed in Section 3.5. Finally, concluding remarks are made in Section 3.6.

## 3.2 Related Work

The  $k$ -IMDMTSP resembles some network design and location routing problems, widely study in the literature. A closely related problem is the Multi-Depot Multi-Traveling Salesmen Problem (MDMTSP) introduced by Oberlin et al. [158]. In the MDMTSP, an unlimited number of salesmen have to visit a set of customers using routes that can be based on a subset of available depots. Malik et al. [139] presented a 2-approximation algorithm for the Generalized MDMTSP, a variant where each depot has  $m$  salesmen available but at most  $p$  can be used. Yadlapalli et al. [210] proposed a Lagrangian-based algorithm to deal with a variation of GMDMTSP where each tour must have at least three vertices.

Another family of problems related to the  $k$ -IMDMTSP are the Location Routing Problems (LRP). In LRPs, we seek to find a set of routes based at a given set of depots to serve customer demands with vehicles of limited capacity. Depots have opening costs and limited capacity to serve its routes. There is an abundant literature with respect to these problems and their variations and a recent survey can be found in Nagy and Salhi [154]. In Prins et al. [170], a hybrid heuristic based on Lagrangean relaxation and tabu search was developed for the LRP. This heuristic is considered to be the most effective to date for the LRP. The best results of an exact algorithm are from Belenguer et al. [16] where a branch-and-cut procedure was developed with a series of new inequalities tailored for the LRP.

A close variant of  $k$ -IMDMTSP is the Many-to-Many Hub Location-Routing Problem (MMLRP) that was proposed by Nagy and Salhi [153] and which has many applications, e.g. freight industry, transportation of parcels, including passengers, mail, and beverages (Kuby and Gray [116], Lin et al. [127]), as well as in telecommunications (Wang et al. [204]). On the other hand, if there is no inner cycle, the problem reduces to a variant of the Plant-Cycle Location Problem (Labbé et al. [117]) in which any node is a candidate to be a plant. This problem arises in the design of Global System for Mobile (GSM) networks. Camargo et al. [45] proposed a large-scale integer programming formulation for the MMLRP which combines models of the single allocation hub location and traveling salesman problems. Their formulation uses four-index integer variables combined with five-index continuous variables, but once integer variables are fixed, two simple subproblems result. This characteristic allowed Benders decomposition to be applied so that instances with up to 100 nodes were solved. Recently, Rodríguez-Martín et al. [181] proposed a branch-and-cut algorithm for a variant of the MMLRP in which there is a fixed number of hubs to locate, at most one local route per hub, and each of these routes is constrained to have at most a given number of non-hub nodes. Their integer programming model considers five types of variables related to: hub location; hubs without local routes; local routes with one, two, or more non-hub nodes; flow between nodes; and routes with three or more edges. Families of valid inequalities and separation algorithms for connectivity, subtour elimination, and capacity constraints were proposed so that their branch-and-cut algorithm was capable of solving to optimality instances with up to 50 nodes within a time limit of two hours. Lopes et al. [134] presented an integer linear programming formulation for a variant of MMLRP called Many-to-Many  $p$ -Location-Hamiltonian Cycle Problem and Lopes et al. [133] proposed several heuristics to the same problem.

Another interesting problem related to the  $k$ -IMDMTSP is the 2-Echelon Vehicle Routing Problem (2E-TSP), introduced by Perboli et al. [165]. In this problem, freight is dispatched from a distribution center on long-haul vehicles to small satellite depots. From

the satellite depots, the freight is transported in small vehicles to their final destinations. In the first echelon, routes are set between the distribution center and the satellites. In the second, routing occurs between the satellites and customers. Each route must start at the distribution center (respectively satellite) and visit a subset of satellites (respectively customers), to supply the required demand. Heuristics for the 2E-TSP were developed in Mehrjerdi and Nadizadeh [145], Nguyen et al. [156], Yu et al. [212] and Zarandi et al. [213].

The main difference between the  $k$ -IMDMTSP and these related problems is that the latter have a specific set of vertices where the depots are located, while in the former, each vertex is a potential depot/terminal. Furthermore, with respect to the MDMTSP and LRP, the  $k$ -IMDMTSP requires an extra “inner cycle” connecting the terminals. With respect to LRP and 2E-TSP, the  $k$ -IMDMTSP permits only a single cycle connecting the terminals and a single outer cycle (possibly empty) for each terminal. The  $k$ -IMDMTSP plays an important role in networking planning specially because it enables flexibility in the optimization of the internal gateway locations and leads to cheaper networks with link-failure tolerance.

### 3.3 Definitions

Let  $G = (V, E)$  be an undirected complete simple graph with vertex set  $V$  and edge set  $E$ , where each  $e \in E$  has a weight or cost  $c_e \in \mathbb{R}^+$ . Without loss of generality, consider  $V = \{1, 2, \dots, n\}$  where  $n = |V|$ . Consider also  $V[A]$  and  $E[A]$ , respectively, as subsets of vertices and edges induced by  $A$ . A cycle  $\mathcal{D}$  of size  $t$  is a subgraph of  $G$  such that vertices  $V[\mathcal{D}]$  form a sequence of distinct vertices  $v_1, \dots, v_t$  and each edge  $(v_i, v_{(i \bmod t)+1}) \in E[\mathcal{D}]$  for all  $1 \leq i \leq t$ . Note that, by this definition, a cycle of size two is a simple edge. We also consider a cycle of size 1 (without edges) and call it *empty cycle*. In this way, we consider that all vertices belong to a cycle, even an empty cycle.

In the  $k$ -IMDMTSP, we seek a subset of vertices  $T \subseteq V$ , called *terminals*, such that the size of  $T$  is exactly a constant  $k$ . A cycle  $I$  formed by the terminals  $T$  is called the *inner cycle*. For each terminal  $t \in T$ , an *outer cycle*  $O_t$  is built such that for all  $t' \in T$  and  $t \neq t'$ ,  $V[O_t] \cap V[O_{t'}] = \emptyset$ , i.e., the outer cycles are pairwise disjoint. We assume that the terminal  $t \in V[O_t]$  is accounted for in the size of  $O_t$ . Therefore,  $|V[O_t]| \geq 1$ , for all  $t \in T$ . Each external cycle must have no more than  $C$  vertices. A valid solution  $\mathcal{S}$  for the  $k$ -*Interconnected Multi-Depot Multi-Travelling Salesmen Problem* is a set of terminals  $T$ , of size  $k$ , connected by an inner cycle  $I$  and a collection of spanning outer cycles  $O_t$  such that  $|V[O_t]| \leq C$ , for all  $t \in T$ . The cost of a solution  $\mathcal{S}$  is defined as

$$\sum_{e \in E[\mathcal{S} \setminus \mathcal{R}]} c_e + 2 \sum_{e \in E[\mathcal{R}]} c_e \quad (3.1)$$

where  $\mathcal{R} = \{O \in \{O_{\forall t \in T}, I\} : |E[O]| = 1\}$ , i.e., the set of cycles with one edge only. The second term of the sum corresponds to the edges that belong to a cycle of size two. In this case, we consider a round trip and the cost of these edges are doubled. The objective of the  $k$ -IMDMTSP is to minimize this cost.

We can address degenerate cases for the  $k$ -IMDMTSP. With respect to the external cycles, we have two cases. In the first, the outer cycle has size one indicating that only the terminal is in the cycle. We call this case an *empty cycle*. In the second case, the outer cycle has size two, indicating that it has a terminal and a non-terminal vertex connected by an edge. Such case is called *return trip*. Figure 3.1 shows examples of a simple case and the two degenerate cases using the TSP instance `bayg29` from TSPLIB [176] and six terminals. The edges of inner cycle are the thick double red lines while the edges of outer cycles are thin blue lines. In Figure 3.1a, all terminals are attached by an outer cycle. In Figure 3.1b, vertices 1, 9, and 19 are not attached to any cycle and we therefore consider they lead empty cycles. In Figure 3.1c, each terminal is connected to a single vertex outside the inner cycle (having the appearance of spikes of the inner cycle).

With respect to the inner cycle, one can address three special cases all related to parameter  $k$ . For  $k = 2$ , the inner cycle is formed by one edge as a “bridge” between two outer cycles (Figure 3.1d). For  $k = 1$  or  $k = n$ , we have a single cycle (either an outer cycle or an inner cycle, respectively). If the capacity  $C \geq n$ , the problem reduces to the Traveling Salesman Problem. In fact, using this reduction, one can easily argue that  $k$ -IMDMTSP is  $\mathcal{NP}$ -hard.

**Corollary 3.3.1:** *Given an instance  $\mathcal{I}$ , let  $\text{TSP}(\mathcal{I})$  be an optimal solution value for the TSP, and  $k$ -IMDMTSP( $\mathcal{I}$ ) be an optimal solution value for  $k$ -IMDMTSP such that  $k \geq 1$  and  $C \geq \lceil |V[\mathcal{I}]|/k \rceil$ . Then  $\text{TSP}(\mathcal{I}) \leq k\text{-IMDMTSP}(\mathcal{I})$ .*

Note that the maximum capacity of the outer cycles and the number of terminals are closely related. A low capacity requires a greater number of short cycles to cover all the vertices in a valid solution. Therefore, feasible instances have the number of terminals and the capacity of the outer cycles related according to  $C \geq \left\lceil \frac{n}{k} \right\rceil$ .

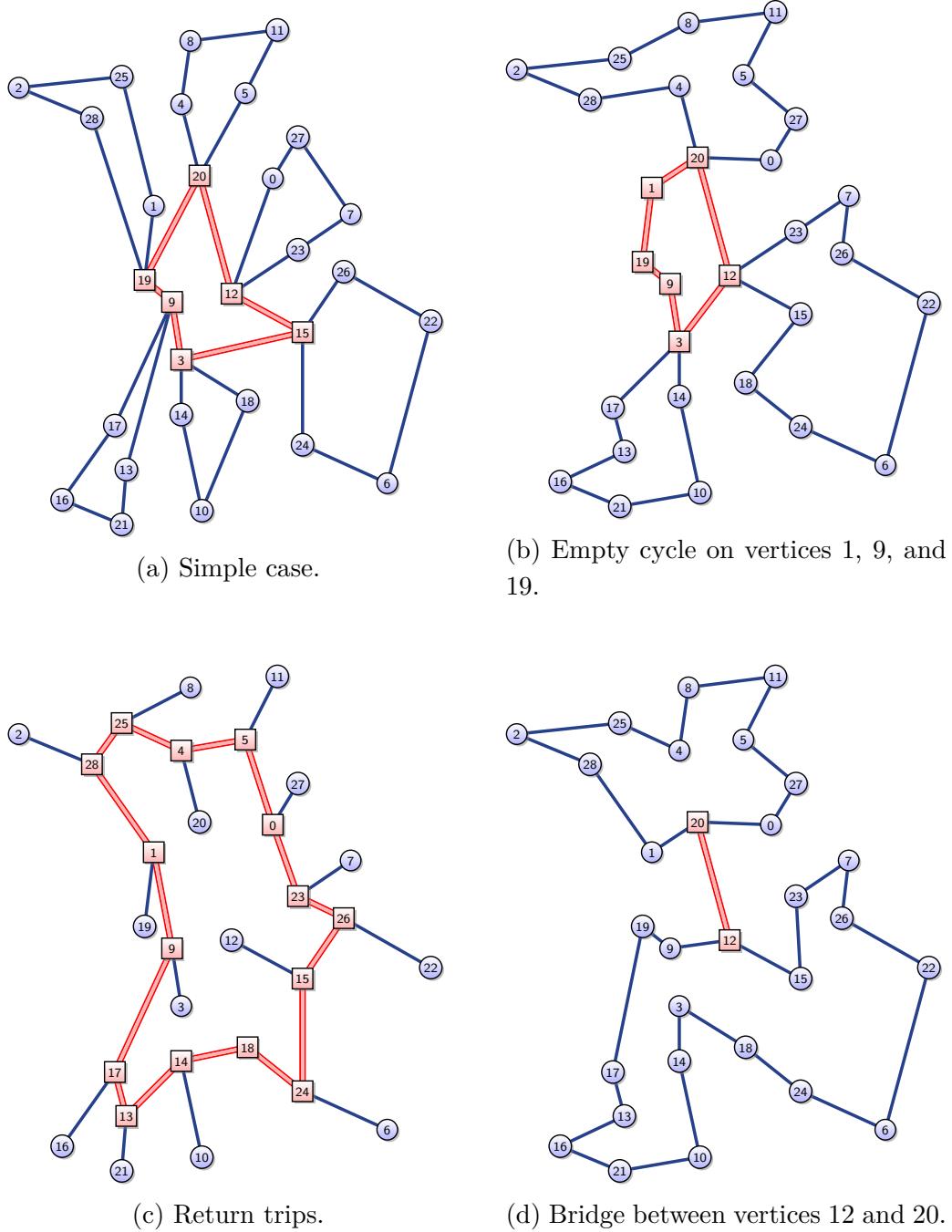


Figure 3.1: Examples of a simple and degenerate cases of  $k$ -IMDMTSP on bayg29 TSP instance.

## 3.4 Biased random-key genetic algorithm for the $k$ -IMDMTSP

To use a biased random-key genetic algorithm (BRKGA), we need to represent a solution with a chromosome and implement a decoder using this representation. The decoding phase is the only connection between the BRKGA and the problem itself. The implementation of the decoder needs to maintain the compatibility of the solution extraction and the genetic operators which means to choose an adequate representation of a solution by a chromosome. In this phase, it is also possible to intensify the search applying a local search procedure on the decoded solution. Note that to find a better solution in the neighborhood implies in changing the allele values of the chromosome to reflect this new solution. The fitness of a decoded chromosome is the cost of the cycles found. In the following sections, we describe these particularities.

### 3.4.1 Representation

In most BRKGA implementations, a chromosome is a real vector  $\mathbf{v}' \in [0, 1]^m$  following the procedure given in Bean [14]. Here, we use an integer vector  $\mathbf{v} \in \mathbb{N}^m$ . The projection of  $\mathbf{v}' \rightarrow \mathbf{v}$  is done with the function  $f(\mathbf{v}') = \lceil \alpha \mathbf{v}' \rceil$  with  $\alpha > 1$  sufficiently large. Recall that  $k$  is the number of terminals,  $n$  is the number of vertices, and  $C$  is the maximum number of vertices in an outer cycle.

A chromosome is a  $(k+n)$ -vector  $\mathbf{v}$  of integers. The first alleles  $\mathbf{v}_1, \dots, \mathbf{v}_k \in \{1, \dots, C\}^k$  represent the size of the outer cycles. The remaining alleles  $\mathbf{v}_{k+1}, \dots, \mathbf{v}_{k+n} \in \mathbb{N}^n$  represent the position of a vertex in an outer cycle.

To extract a solution from this chromosome, we sort the alleles  $\mathbf{v}_{k+1}, \dots, \mathbf{v}_{k+n}$  in a non-increasing order, generating a permutation  $\pi$  of the vertices. Vertices  $\pi_1, \dots, \pi_k$  represent the terminals  $T$ . Each terminal  $\pi_i$ , for  $i = 1, \dots, k$ , has an outer cycle formed by vertices  $\pi_{k+o(i)+j}$ , for  $j = 1, \dots, \mathbf{v}_i$  and  $\mathbf{v}_i \geq 2$ , such that

$$o(i) = \begin{cases} o(i-1) + \mathbf{v}_{i-1} - 1 & \text{if } i > 1, \\ 0 & \text{otherwise.} \end{cases}$$

The function  $o(\cdot)$  indicates the offset in the alignment of cycles in the permutation  $\pi$ . Note that  $\mathbf{v}_i = 1$  indicates terminal  $i$  is alone in an empty cycle.

For instance, consider the chromosome of Figure 3.2, where  $n = C = 6$  and  $k = 3$ . To extract a solution, we sort the indices representing the vertices, using their corresponding keys and generate the permutation  $\pi$ . The first terminal is the vertex 3 and it leads the cycle  $O_3$ :  $3 \rightarrow 6 \rightarrow 4$ . The second cycle  $O_5$  is an empty cycle formed only by terminal 5. Cycle  $O_1$  is a degenerate cycle formed by terminal 1 and vertex 2.

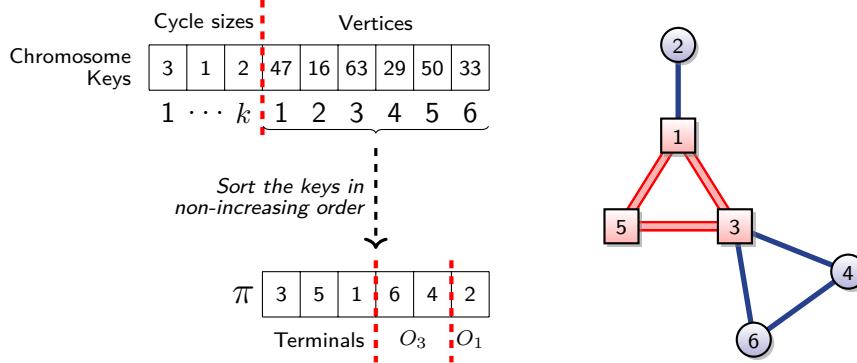


Figure 3.2: Example of solution extraction. Note that vertex 5 has an empty cycle  $O_5$  that is not shown in the chromosome.

The advantage of this representation is that any modification in the values of keys  $v_{k+1}, \dots, v_{k+n}$  will not destroy the feasibility of the solution extracted from the chromosome. This approach was used in several papers, such as Samanlioglu et al. [184] and Snyder and Daskin [193]. On the other hand, arbitrary modifications of keys  $v_1, \dots, v_k$  can result in invalid cycle sizes. In this case, it is necessary to use the repair procedure described in Section 3.4.2.

Note that distinct chromosomes may decode to the same solution either by rotation of the position of the alleles or by different allele values in the same order. However, this is unlikely to occur because of the range of the integers used in practice.

### 3.4.2 Decoding a solution

In BRKGAs, the decoder extracts a solution from the chromosomes as well as tries to improve the solution with some local search procedure. Using the representation of the previous section requires that we guarantee that the sizes of the outer cycles are correct, i.e., that the total sum of the keys equals the number of vertices. Algorithm 3.1 increments or decrements the sizes of each cycle based on the difference between the sum of the key values and the number of vertices. Note that the modifications are done among randomly selected cycles with the objective of preserving the “structures” inherited from the parents. Therefore, we avoid deep changes in a particular cycle. The complexity of this repair procedure is  $O(|kC - n|)$ .

After the repair phase, the decoder proceeds with solution extraction as described in Section 3.4.1. At this point, it is possible (although rare) that two or more keys have identical values. This could generate different solutions with different values from the same chromosome. To avoid this situation, the decoder modifies the keys to have different values and, therefore, guarantee a unique solution. However, note that this

modification must be done only on the keys corresponding to the node ordering. The keys corresponding to the cycle size are kept untouched. Algorithm 3.2 summarizes the entire process, including the local search procedures described in next section.

---

**Algorithm 3.1:** `adjustCycleSizes( $\mathbf{v}, k, n, C$ )`


---

**Input:** a vector  $\mathbf{v} \in \mathbb{N}^{k+n}$ , integers  $k, n, C \in \mathbb{N}$

**Output:** modified  $\mathbf{v}$

```

1 Let  $A = \{1, \dots, k\}$ ;
2  $diff \leftarrow \sum_{i \in A} \mathbf{v}_i - n$ ;
3 if  $diff > 0$  then
4    $sign \leftarrow -1$ ;
5 else
6    $sign \leftarrow 1$ ;
7 while  $diff \neq 0$  do
8   Choose  $i$  from  $A$  uniformly;
9   if ( $\mathbf{v}_i \geq 2$  and  $sign < 0$ ) or ( $\mathbf{v}_i < C$  and  $sign > 0$ ) then
10     $\mathbf{v}_i \leftarrow \mathbf{v}_i + sign$ ;
11     $diff \leftarrow diff + sign$ ;
12   else
13     $A \leftarrow A \setminus \{i\}$ ;

```

---

## Local Improvements

Local improvement procedures have been shown to help convergence of genetic algorithms. Potentially, each chromosome can be in the neighborhood of a different local minimum. In such a case, local search would be able to reach these minima (Levine [124]).

To perform local improvements, we explore two different neighborhoods. The first is the well-known “edge exchange” neighborhood obtained from 2-opt and 3-opt movements (it is also known as the “ $\lambda$ -opt” neighborhood). Let  $(u, v)$  and  $(u', v')$  be two edges of the cycle (appearing in the orientation). If  $c_{(u,v')} + c_{(u',v)} < c_{(u,v)} + c_{(u',v')}$ , we remove  $(u, v)$  and  $(u', v')$ , and insert  $(u, v')$  and  $(u', v)$ , performing a 2-opt move. Note that the path  $v, \dots, v'$  is reversed in this process becoming  $v', \dots, v$ . A  $\lambda$ -opt move is done in a similar way but involves the exchange of  $\lambda n$  edges. Lin and Kernighan [128] generalize these methods resulting in one of best heuristics for the traveling salesman problem. Their procedure is adaptive and at each step decides dynamically what type of  $\lambda$ -opt should

be applied. In Martin et al. [142], the Lin-Kernighan heuristic was improved using, at the start of each iteration, a “guess” or “kick” tour instead of only doing restarts. The *double-bridge move*, cheaper than a 4-exchange move, was also proposed. This heuristic is known as the *Chained Lin-Kernighan procedure*. In Helsgaun [95] several refinements of Lin-Kernighan are described. A sequential 5-opt step, restricted to a neighborhood of five  $\alpha$ -nearness members, is used. This is computed using 1-trees on a modified weight distance matrix.

To explore the “edge exchange” neighborhood, we apply the Chained Lin-Kernighan heuristic to each outer cycle as well as the inner cycle but with a limited number of iterations. One can note that an exchange in positions of vertices in a cycle implies that the allele values of the vertices must be also swapped. In particular, the modification in the positions of terminals in the inner cycle must be done carefully so that the remaining structure of the chromosome is not destroyed. Lines 5 and 6 of Algorithm 3.2 summarize this procedure.

The second neighborhood “rotates” the outer cycles looking for new terminal vertices capable of reducing the cost of the solution. Let  $t$  be the terminal of outer cycle  $O$ ,  $t_s$ , and  $t_p$  be, respectively, the successor and predecessor terminals of  $t$  in the inner cycle  $I$ . Rotating cycle  $O$  means finding a vertex  $t' = \operatorname{argmin}_{t'' \in O} c_{(t_p, t'')} + c_{(t'', t_s)}$ , i.e., finding a vertex to be the new terminal that minimizes the cost of the inner cycle. Note that the outer cycles are kept intact but the terminal and, consequently, the cost of inner cycle is changed. In fact, the vertex exchange can result in a new inner cycle to which is applied the Chained Lin-Kernighan procedure. Lines 7–11 of Algorithm 3.2 summarize this procedure. One can note that these operations can be done efficiently, but we prefer to describe it this way for notational convenience. Figure 3.3 illustrates these improvements.

It is important to note that all these modifications on the solution implies that the keys of the corresponding chromosome must be modified to reflect the new solution. Lines 14–19 copy back the key values from ordered vector  $s$  to reflect these modifications.

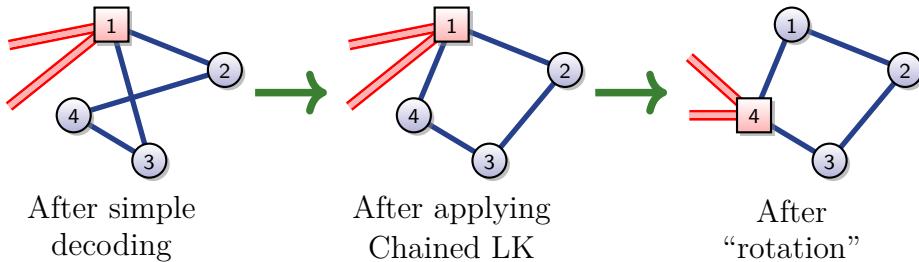


Figure 3.3: Example of local improvement steps. First, we apply the Lin-Kernighan heuristic and then the “rotate” neighborhood.

---

**Algorithm 3.2:** Decoder

---

**Input:** a vector  $\mathbf{v} \in \mathbb{N}^{k+n}$ , integers  $k, n, C \in \mathbb{N}$

**Output:** modified  $\mathbf{v}$ , the *fitness* of  $\mathbf{v}$

```

1 adjustCycleSizes( $\mathbf{v}, k, n, C$ );
2 Sort the alleles  $\mathbf{v}_{k+1}, \dots, \mathbf{v}_{k+n}$  in non-decreasing order, generating a vector  $\mathbf{s}$  of
   sorted keys and a permutation  $\pi$  of the vertices;
3 Keeping the order, modify the values of  $\mathbf{s}$  to ensure all different;
4 Extract the inner cycle  $I$  and the outer cycles  $\mathcal{O}$  using  $\pi$  and  $\mathbf{s}$ ;
5 foreach  $O \in (\mathcal{O} \cup \{I\})$  do
6    $\quad$  ChainedLinKernighan( $O$ );
7   foreach  $O_t \in \mathcal{O}$  do
8      $t' \leftarrow \text{findBestTerminal}(O_t);$ 
9      $O_{t'} \leftarrow O_t;$ 
10     $I \leftarrow (I \setminus \{t\}) \cup \{t'\};$  // Keeping the order
11     $\mathcal{O} \leftarrow (\mathcal{O} \setminus \{O_t\}) \cup \{O_{t'}\};$ 
12 ChainedLinKernighan( $I$ );
13  $i \leftarrow 1;$ 
14 foreach  $O_t \in \mathcal{O}$  do
15    $\mathbf{v}_{k+t} \leftarrow \mathbf{s}_i;$ 
16    $i++;$ 
17   foreach  $v \in O_t$  do
18      $\mathbf{v}_{k+v} \leftarrow \mathbf{s}_i;$ 
19      $i++;$ 
20 return the solution value according to Equation (3.1)

```

---

The complexity of Algorithm 3.2 is hard to determine due to the chained Lin-Kernighan multiple calls. The chained Lin-Kernighan is a complex procedure that uses computation of minimum spanning trees, subgradient optimizations, and other features that are very sensible to the input. The  $\lambda$ -opt neighborhoods can be large since consider the combination of  $\lambda$  vertices (which can be exponential).

### 3.4.3 Initial Population

The common approach to initialize the population of a BRKGA is to generate the chromosomes with uniformly drawn random keys over the interval  $[0, 1]$ . This results in highly heterogeneous individuals that may slow down the convergence of the algorithm. On the other hand, these individuals may lie in different local minima neighborhoods and help avoid premature convergence of the algorithm. In an attempt to speed up the search, we use a fast heuristic to generate some good initial solutions and then, complete the initial population with random chromosomes. An advantage of using such individuals in the starting population is that they are probably closer to a good solution than most random individuals.

The heuristic is based on  $k$ -means clustering algorithm that partitions the vertices in  $k$  clusters such that each vertex belongs to the cluster with the nearest mean (Lloyd [132]). This is a well-known algorithm, used in several fields, such as machine learning and computational geometry. Each cluster generated by  $k$ -means is considered as an outer cycle and the terminal is the vertex closest to the mean of the cluster. The vertices are taken in arbitrary order since the cycles will be optimized in the decoding procedure. Although the cost of this solution is usually not very good, it carries some geometric information about the distribution of the vertices and it is likely that some parts of these clusters will be part of the final solution. One can note that this approach can be used only in geometric instances. In Barreto et al. [12], a similar approach is used to solve the capacitated location-routing problem.

The encoding of the chromosome from this solution is done in the following way: a vector  $\mathbf{s}'$  is generated with  $n$  random keys, sorted in non-increasing order. The first  $k$  positions of the chromosome are associated with the size of the clusters taken in some arbitrary but fixed order, say  $t_1, \dots, t_k$ . Using this same order, we associate each terminal  $t_i$  with a key of  $\mathbf{s}'$ . For each cluster  $O_{t_i}$ , we associate the vertices of this cluster with the remaining keys of  $\mathbf{s}'$  in the order they appear. This ensures the compatibility and uniformity with the BRKGA framework.

## 3.5 Experimental Results

We conducted several experiments with the goal of evaluating the effectiveness of the BRKGA to find good solutions on instances of varying sizes. By adjusting the BRKGA framework, we also implemented a simple Multi-Start Heuristic (MSH) using the same local search procedure as used by the BRKGA and compare the results obtained by both heuristics.

### 3.5.1 Computational environment

We implemented the algorithms in the C++ programming language using the GNU g++ compiler version 4.4.2. Random numbers were generated by an implementation of the Mersenne-Twister [143]. We use the Chained Lin-Kernighan implementation of the Concorde TSP Solver [8] to do the local search. The experiments were conducted on a quad-core Intel Xeon 2.4 GHz CPU with 8 GBytes of RAM running GNU/Linux. Running times reported are UNIX real wall-clock times in seconds, excluding the time to read the instance. Each run was limited to 3,600 seconds.

### 3.5.2 Algorithm settings

The proposed algorithm was implemented on top of the BRKGA API [196], which implements all of the problem-independent components described in Section 1.2. We set the population size to  $p = \min(10n, 100)$ , where  $n$  is the number of vertices, the size of elite set of individuals to  $p_e = \lceil 0.30p \rceil$ , and the number of mutants introduced at each generation to  $p_m = \lfloor 0.15p \rfloor$ . The probability of inheriting each allele from the elite parent was set to  $\rho_e = 0.70$ . We evolved  $\Pi = 3$  populations simultaneously and once every 100 generations each population exchanged its two best solutions with the other populations. After 500 iterations without improvement, all three populations are reset to randomly generated individuals. This setup followed suggestions of Gonçalves and Resende [81].

To create the Multi-Start Heuristic (MSH), we set the size of the elite set to  $p_e = 1$  and the number of mutants in each generation to  $p_m = p - 1$ . This setting disables any type of offspring generation and only keeps the best solution found throughout the optimization. We also disabled the initial heuristic leaving just the random chromosome generation.

At each iteration of the decoder, the Chained Lin-Kernighan heuristic was executed once for each cycle, using no time or tour length bounds. The stall counter was set to at most 1000 iterations, limiting the number of 4-swaps without progress. We use a random walk as the kicking strategy. This perturbs the Lin-Kernighan solution creating three independent random walks of a given length from a vertex  $v$  in the neighborhood of this solution. According to Fischer and Merz [61], this type of kicking strategy presents better results than other strategies suggested in Applegate et al. [9].

In addition to stopping after 3,600 seconds, we also stop if 1,000 generations go by without improvement of the best solution. The algorithm decodes the chromosomes in parallel, using four cores. It was necessary to make slight modifications in the Concorde code to support multi-threading.

### 3.5.3 Instances

Our experiments use TSP instances from the TSPLIB repository [176] as base graphs. We build five scenarios based on different numbers of terminals  $k$  and cycle capacity  $C$ . We consider the 63 TSP instances with fewer than 700 vertices. We exclude instance `linhp318` due to problems in the loading routine of Concorde.

The first scenario varies the size of the inner cycle. We argued earlier that the extremal values  $k = 1$  and  $k = n$  reduce the  $k$ -IMDMTSP to the TSP (with capacity  $C \geq n$ ). Therefore, we use other values of  $k$  in the experiments. We first set  $k = \lceil 0.2n \rceil$  to build a small inner cycle. This is common, for instance, in transportation problems that deliver heavy truckloads to one or more depots. We then set  $k = \lceil 0.5n \rceil$ , promoting larger inner cycles. This occurs, for instance, when large optical fiber ring is used to maximize the throughput of a network. Our third variation is described below.

We also vary the capacity of the outer cycles looking for interesting instances. First, we set  $C = \lceil n/k \rceil$ , forcing each terminal to have a non-empty outer cycle. One can easily see that the sizes of these cycles differ by at most one. Second, we set  $C = 2\lceil n/k \rceil$ , allowing for the possibility of some empty or larger cycles. Note that  $C \geq n$  leads to a solution with one outer cycle and one inner cycle.

Using the above settings, we built the following five scenarios for each of the 63 TSP graphs, generating 315 instances:

**ST** – small inner cycle:  $k = \lceil 0.2n \rceil$  and tight outer cycles:  $C = \lceil n/k \rceil$ ;

**SL** – small inner cycle and loose outer cycles:  $C = 2\lceil n/k \rceil$ ;

**LT** – large inner cycle:  $k = \lceil 0.5n \rceil$  and tight outer cycles;

**LL** – large inner cycle and loose outer cycles;

**SQ** – Inner and outer cycles of same size:  $k = C = \sqrt{n}$ .

It is important to note that, for the instance classes ST and SL, the capacities converge, respectively, to 5 and 10 for all  $n \geq 17$ , since the limits for those functions converge to these values. Similarly, for the instances in LT and LL, the capacities converge to 2 and 4, respectively. Therefore, we observe that the instance class LT corresponds to the “return trip” instances exemplified in Figure 3.1c.

### 3.5.4 Results and Discussion

For each instance, 30 independent runs of each algorithm were performed. This resulted in 1890 experiments per scenario, and 9450 experiments per algorithm. To compare

the algorithms, we needed to scale the resulting costs since their magnitudes vary greatly from instance to instance. For each instance, we scaled the cost in the interval  $[0, 1]$  using the minimum and maximum costs from both algorithms.

Figure 3.4 shows the distributions of the results for each scenario. The box plots show the smallest cost, the first quartile, the median cost, the third quartile and the largest cost. First, we can note that all distributions have fat tails, indicating that they are not symmetric. A skewness statistic shows a skew of, at least, 0.46 for BRKGA indicating that these distributions have the most points in the left-side that means lower costs. To the MSH, the skewness is, at most, -0.17, resulting a long left-side tail and the points concentrated in right-side, therefore with higher costs. We can note that the medians and the quartiles of BRKGA results are systematically lower than the those of MSH, indicating better performance of BRKGA over MSH.

To confirm these results, we tested the normality of these distributions using the Shapiro-Wilk test and we applied the Wilcoxon-Mann-Whitney test, considered more efficient than the  $t$ -test for distributions sufficiently far from normal and for sufficiently large sample sizes (Conover [38], Fay and Proschan [57]). For all tests, we assume a confidence interval of 95%. The Shapiro-Wilk tests revealed that no cost distribution fit normal distributions since the  $p$ -values for all tests are less than  $2.5 \times 10^{-28}$ . Therefore, we applied the Wilcoxon-Mann-Whitney which we show in Table 3.1. This test assumes as null hypothesis that the location statistics are equal in both distributions. The second column shows a difference in medians of the BRKGA and the MSH such that the medians

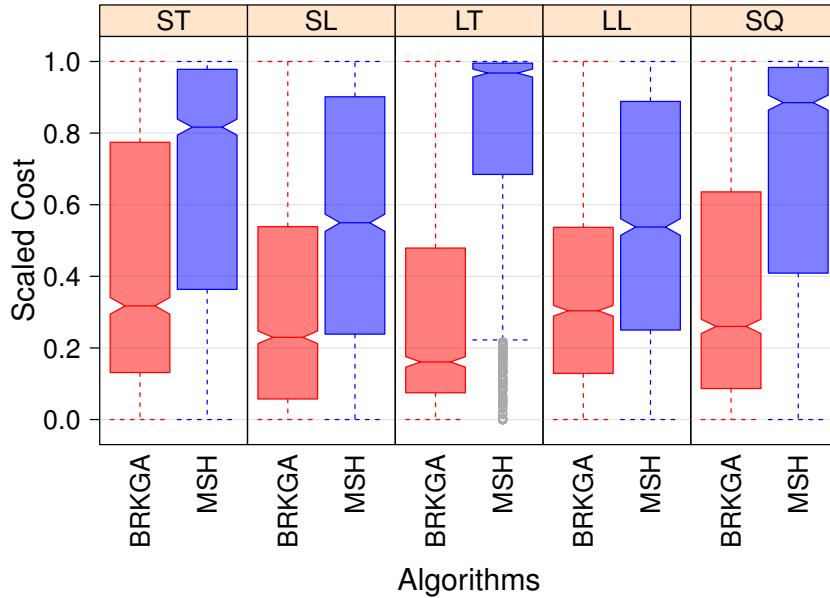


Figure 3.4: Boxplot of the medians and quartiles for each algorithm in each scenario.

of the BRKGA are less than those of the MSH. As all differences are in the respective confidence intervals and all  $p$ -values are much less than 0.05, we can conclude that the BRKGA presents significantly better results than does the MSH.

Table 3.2 shows, for BRKGA over all instances, the average number of iterations, the time taken by the initial heuristic, and the total time to reach the best solution. As suggested by McGeoch [144], we use the bootstrapping method with resample size of 1000 and show the standard errors and 95% confidence intervals. Note the BRKGA uses an average of 2900 iterations to reach the best solution, showing that the genetic operations contribute to the convergence. We can also note that the minimum number of iterations is below 1,000, indicating that the maximum running time was used to stop the algorithm in certain large instances. In general, the initialization heuristic is very fast and does not add much in total running time. With respect to the last, we computed good means although in some extreme cases we use about 3,600 seconds. These cases are related to the size of the instance and, in general, do not perform many iterations. Detailed results can be found in Appendix B.

Figure 3.5 shows the cumulative probability to reach a target cost over time. The target costs were set using the values of the best solution found for each instance. For each algorithm, we performed 30 runs for each instance (9,600 experiments per algorithm), limiting the maximum runtime to 1,800 seconds. The experiments with running time

Table 3.1: Median test for cost distributions.

Scen.	Diff.	Loc.	Conf. Interv.	U	$p$ -value
ST	-0.24		[-0.27, -0.21]	$1.1 \times 10^6$	$8.0 \times 10^{-86}$
SL	-0.23		[-0.26, -0.21]	$1.1 \times 10^6$	$2.9 \times 10^{-82}$
LT	-0.63		[-0.66, -0.60]	$5.3 \times 10^5$	$8.5 \times 10^{-305}$
LL	-0.19		[-0.22, -0.17]	$1.2 \times 10^6$	$1.1 \times 10^{-67}$
SQ	-0.36		[-0.39, -0.33]	$9.4 \times 10^5$	$6.1 \times 10^{-140}$
Overall	-0.32		[-0.33, -0.31]	$2.4 \times 10^7$	0.00

Table 3.2: BRKGA means of best results using bootstrapping resampling (1000 replicates).

Dimension	Mean	Error	Conf. Interval	Min	Max
Iterations	2901.33	165.58	[2580, 3244]	199	17989
Init. Time	0.57	0.11	[0.34, 0.78]	0.00	23.34
Total Time	573.00	46.84	[480.50, 664.1]	1.30	3604.78

greater than or equal to 1,800 were pruned from the plot. Note that the cumulative probability curves for BRKGA are to the left and above those of MSH, indicating that it has a higher probability than MSH of reaching good solutions in a given time. Overall, BRKGA reached 53.19% of the target costs while MSH reach only 14.56%, with 1,000 seconds. Following the methodology proposed in Ribeiro et al. [180], BRKGA has a 79.83% probability (with error  $e = 0.001$ ) of finding a target solution in less time than MSH.

In both the LT and LL scenarios, both algorithms found few good solutions. We think that the main reason for this are the larger internal cycles required by these instances. In the other scenarios where the size of the cycles are more balanced, BRKGA reached more than 70% of the best solutions and outperformed MSH by a large margin. Another interesting observation is that on the SL and LL scenarios, the target values are found faster than on the other scenarios, indicating that the former are apparently easier than the latter.

## 3.6 Concluding Remarks

We introduced a new problem called the  $k$ -Interconnected Multi-Depot Multi-Travelling Salesmen Problem. The  $k$ -IMDMTSP is closely related to the Location Routing Problem and the 2-Echelon Vehicle Routing Problem. Unlike these problems, the  $k$ -IMDMTSP has the inherent difficulty of not having a fixed set of depots. To solve the  $k$ -IMDMTSP, we propose a biased random-key genetic algorithm (BRKGA) with a local search procedure in the decoding phase, using adaptations of classical procedures for the traveling salesman problem. We compare our algorithm with a multi-start heuristic (MSH) using the same local search algorithm, on five different scenarios with 63 instances each. We found that the BRKGA is significantly better than the MSH in our experiments, presenting a good performance that trades off solution quality with running time. Future research directions include extending our approach to address constraints like demands and flow limits and to compare it with other possible approaches like exact algorithms and their hybridizations.

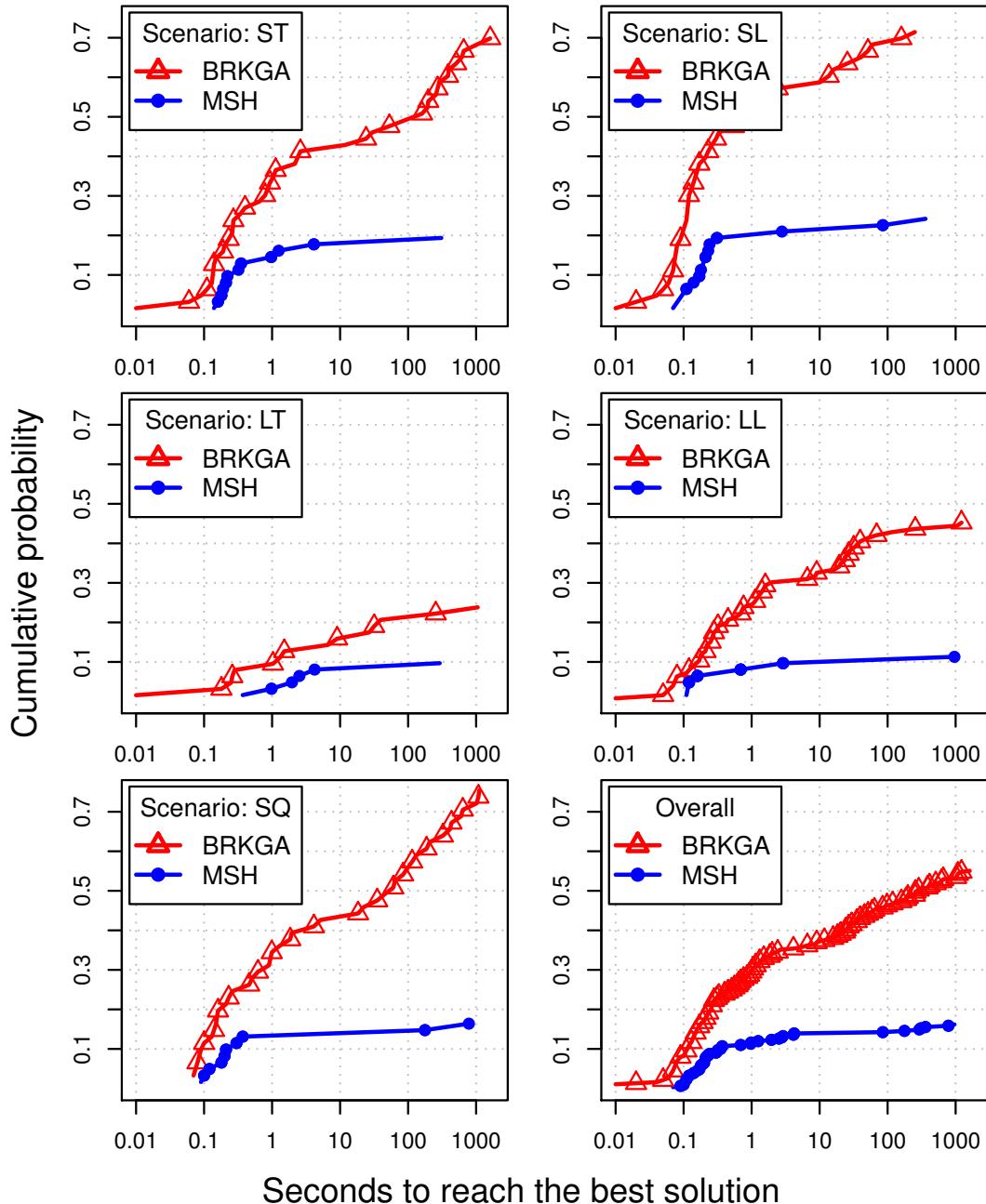


Figure 3.5: Time-to-target plots showing the cumulative probability to reach a determined cost in 1,800 seconds. The points corresponds to 0.2% of runs.

# Chapter 4

## The Overlapping Correlation Clustering Problem

MODERN wireless networks have been capable to route a huge mass of data. In addition of this capability, these networks have also collected information about the mobility and usage of wireless devices. Such information may consist of geographical location, data consumption, calls, and texting over a period of time. This behavioral information is invaluable in several aspects. First, such information is crucial for planning, delivery, and operating old and new networks. Second, the analysis of this data may reveal behaviors of the users and help to plan for new products and/or sales promotions. It is also possible to use this information for security reasons such as finding threats, estimate travel routings, carbon emissions, traffic volumes, and others (Becker et al. [15]).

In this chapter, we address the Overlapping Correlation Clustering Problem (OCC) which is suitable for such analyses. In this problem, a number of objects are assigned to clusters such that two objects in the same cluster have correlated characteristics. As opposed to traditional clustering where objects are assigned to a single cluster, in OCC objects may be assigned to one or more clusters. Such multi-classification can be used in contexts where we do not know clearly the boundaries between the characteristics of the objects. We propose biased random-key algorithms (BRKGAs) using four different decoders and compare the results with those obtained from state of the art algorithms for the OCC. We use databases from several different contexts such as mobility patterns, bioinformatics, and text analysis. The BRKGAs presented very good results outperforming in almost all types of instances other algorithms. This chapter is based on Andrade et al. [5].

## 4.1 Introduction

One of the most fundamental tasks in data analysis is to categorize objects in different sets such that two objects in the same set have certain characteristics that are correlated. This correlation is usually measured by a similarity value. The standard way of clustering is the creation of a partition of these objects. However, for some applications, it is natural that an object belongs to two or more clusters since it can share characteristics with objects in more than one cluster. In this case, to partition the ground set does not make sense. Instead, it is more appropriate to assign the objects to clusters with possible overlapping. Several scenarios with these properties can be addressed: in social networks, a user belongs to several communities; in mobility analysis, individuals share trajectories with respect to time and space; in biology, a protein belongs to several protein complexes having similar expressions. We can model scenarios like these using *Overlapping Correlation Clustering* (OCC), introduced in Bonchi et al. [23]. This problem is closely related to Correlation Clustering Problem (Bansal et al. [11]) but allows overlapping of the clusters. Another major difference is in the relation among the objects. In OCC, this relation is expressed as a value in the range  $[0, 1]$  while in the correlation clustering it takes on one of two discrete values in the set  $\{0, 1\}$ . This enables the utilization of different types of similarity functions leading to a more sophisticated analysis.

As commented, the OCC has similarities with the correlation clustering problem. In this problem, a complete graph is given and each edge is labeled as positive or negative. The objective is to find a partition in the vertices such that it minimizes the number of positive edges between the partitions and the number of negative edges within the partitions. This problem has received a lot of attention in the literature (Ailon and Liberty [1], Charikar et al. [34], Demaine et al. [46], Swamy [195]). The best-known approximation algorithm for this problem was proposed by Charikar et al. [34] and is a linear programming based method with a factor of 4. Demaine et al. [46] presented a  $O(\log n)$ -approximation for edges with arbitrary weights. Recently, Wang and Li [203] proposed a neighborhood-based heuristic which considers that if two vertices belong to the same cluster, they should have the same neighborhood. Following this observation, the algorithm iteratively chooses two connected vertices and restores their neighborhood. Lingas et al. [129] proposed an iterative heuristic that starts from singleton clusters. Whenever merging two clusters improves the current quality score, the heuristic merges them into a single cluster.

In general, one may use the overlapping clustering to label objects as members of one or more clusters and two or more objects may share a single cluster. As far as we know, the origin of the overlapping clustering problem is not known. However, there is an abundant literature with respect to this problem and a comprehensive list can be found in

Pérez-Suárez et al. [166]. One of the first algorithms that deal with overlapping clustering was proposed by Shepard and Arabie [190]. This algorithm considers that the similarity of any two objects is a simple additive function of weights associated with properties that are shared by both objects. Banerjee et al. [10] proposed a model-based overlapping clustering using probabilistic relational models that works with any regular exponential family distribution making the model applicable for a wide variety of distance functions. Goldberg et al. [73] analyzes the overlapping clustering from a different perspective: instead to worry about finding the clusters, they are concerned in how to compare two overlapping clusters. The paper describes three new definitions of distance between the clusters and presented algorithms to compute them. Recently, Pérez-Suárez et al. [166] presented a graph-based algorithm for building overlapping clusters. The method is based on graph covering given a relevance metric for each vertex. This method was able to overcome other algorithms previously proposed in the literature.

This chapter presents biased random-key genetic algorithms associated with local search procedures aiming to solve the overlapping correlation clustering problem under an optimization point of view. This chapter is organized as follows. Section 4.2 formalizes the problem. Section 4.3 presents high level algorithms to deal with OCC. Section 4.4 depicts the experimental results and Section 4.5 presents the concluding remarks.

## 4.2 Definitions

Let  $V = \{v_1, \dots, v_n\}$  be a set of  $n$  objects such that there exists a symmetric function  $s : V \times V \rightarrow [0, 1]$  which gives the similarity  $s(u, v)$  between two objects  $u$  and  $v$  in  $V$ . Let  $L = \{1, \dots, k\}$  be the set of  $k$  available labels (clusters). Since we allow an object to be in several clusters, we define  $\ell : V \rightarrow 2^L \setminus \emptyset$  with  $\ell(v)$  being the set of labels assigned to object  $v$ . Note that we require that each object have at least one label. If the set  $\ell(v) = \{c_1, \dots, c_s\}$  of labels is given to object  $v$ , then we assume that  $v$  is in clusters  $c_1, \dots, c_s$ . Consider also  $\mathcal{H} : 2^L \times 2^L \rightarrow [0, 1]$ , a symmetric function which gives the similarity between two sets of labels. Higher “similarity” of  $E$  and  $F$  should correspond to higher  $\mathcal{H}(E, F)$ , and  $\mathcal{H}(E, E)$  should be 1 for all  $E$ . In *Overlapping Correlation Clustering* (OCC), the task is to find a multi-labeling function  $\ell$  that minimizes the cost function

$$\frac{1}{2} \sum_{(u,v) \in V \times V, u \neq v} |\mathcal{H}(\ell(u), \ell(v)) - s(u, v)|. \quad (4.1)$$

Note that Objective Function (4.1) represents the absolute error between the labeling and the similarity measure. In this sense, the objective is to find a labeling as close as possible to the given similarities.

This formulation enables the application in different contexts. However, we first need to measure the similarities that are context dependent. As the problem is general enough, we can use any type of comparison measure as long as it scales to the real interval  $[0, 1]$ . Another challenging task is to find the appropriate  $\mathcal{H}$  function. As suggested in Bonchi et al. [23], we use two functions: The *Jaccard Similarity Coefficient* and the *Set-Intersection Indicator*. The Jaccard similarity coefficient, also known as the Jaccard index, was proposed by Jaccard [102] and is a well-known measure of similarity between two sample sets, widely used in biology and machine learning (e.g. Kosman and Leonard [113], Leskovec et al. [123], Shamshurin [189], Vorontsov et al. [201]). Let  $E$  and  $F$  be two sets, not both empty. The Jaccard index of  $E$  and  $F$  is defined as

$$\mathcal{J}(E, F) = \begin{cases} \frac{|E \cap F|}{|E \cup F|} & \text{if } |E| + |F| > 0, \\ 1 & \text{otherwise.} \end{cases} \quad (4.2)$$

The set-intersection indicator function is a simple function defined as

$$\mathcal{I}(E, F) = \begin{cases} 1 & \text{if } E \cap F \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases} \quad (4.3)$$

Some scenarios require objects to be assigned to a restricted number of labels among those available. A good example is in the analysis of mobility patterns, where in spite of the fact that trajectories can have a large number of characteristics, we restrict ourselves to only a few of the most meaningful ones. In these cases, we introduce a parameter  $p$  such that  $|\ell(v)| \leq p$  for all  $v \in V$ .

It is easy to see that, if we consider  $\mathcal{H} = \mathcal{I}$  and  $p = 1$ , each object will belong to a single cluster and the similarity between pairs of objects will be an indication as to whether they share the same cluster. In this case, if  $s$  follows the set-intersection function, we face the original correlation clustering problem, which is  $\mathcal{NP}$ -hard (Bansal et al. [11]). In Bonchi et al. [23], hardness proofs for the other cases are presented.

### 4.3 Biased random-key genetic algorithms and local search

For solving the OCC, we proposed biased random-key genetic algorithms (BRKGAs) with two different representations and two different decoders. For comprehensive description of BRKGAs, please refer to Section 1.2.

### 4.3.1 Representation

In most BRKGA implementations, a chromosome is encoded as a real vector  $\mathbf{x}' \in [0, 1]^n$ , following the procedure given by Bean [14]. However, other encodings are possible. Here, we use two forms of chromosomes to represent a labeling. Let  $n$  be the number of objects and  $k$  the number of available clusters. The first representation, called *compact*, is an integer vector  $\mathbf{x}^c \in \mathbb{N}^n$ , and the second, called *extended*, is a binary vector  $\mathbf{x}^e \in \{0, 1\}^{nk}$ .

For the compact chromosome  $\mathbf{x}^c$ , each  $x_i^c$  is a positive integer that represents the clusters to which object  $v_i$  belongs. In this case, we consider each bit as a set indicator in which the least-significant bit corresponds to the first cluster, and so on.<sup>1</sup> Figure 4.1 shows an example of a compact representation where there are eight objects and it is allowed only five different labels on each object. Supposing that eight bits are being used to represent the labels, the object four has the key 163 that can be decomposed in the bit sequence (1, 0, 1, 0, 1, 1, 0, 1). In this case, one can ignore the most significant bits and take the set  $\{1, 3, 4\}$  as the labels assigned to the object.

In the extended chromosome  $\mathbf{x}^e$ , the subvector  $\tilde{\mathbf{x}}^e = \tilde{\mathbf{x}}_{(i-1)k+1}^e, \dots, \tilde{\mathbf{x}}_{ik}^e$  is an indicator vector where, for each object  $v_i$ ,  $\tilde{\mathbf{x}}_j^e$  is 1 if  $v_i$  is in cluster  $j$ , or 0 otherwise. This representation can be used with any  $n > 1$  and  $k > 1$ .

Although both compact and extended representations are quite similar, they affect the evolutionary mechanism of BRKGA differently. In the compact representation, each allele is the full labeling of an object and therefore, in the mating process, the offspring

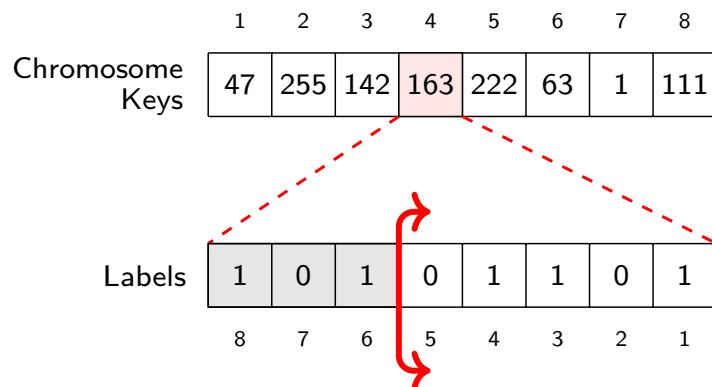


Figure 4.1: Compact representation of a solution by a chromosome. The grey area are the ignored bits.

<sup>1</sup> Although this representation limits the number of possible clusters ( $k < 64$  on a 64-bit machine), it enables very fast set operations since they are done bitwise, and for most applications, this limit is sufficient. If a specialized bitset structure like C++'s `bitset<>` template is used, this limitation can be overcome although at the expense of increased complexity.

inherits these full labelings. Learning occurs through the combination of entire sets of labels of each object. In the extended representation, each allele represents a specific label of an object. In the mating process, the offspring inherits each label individually, enabling learning to occur at the level of each label for each object.

### 4.3.2 Decoding a solution

To decode a solution from a chromosome using the representations of Section 4.3.1, we use a two-phase decoder in which the first phase extracts a solution and the second phase is committed to local search procedures. We assume that the input vector to the decoder is in compact representation. Note that in case of the extended representation, we need to “compactify” the vector obtaining from each subvector  $\tilde{\mathbf{x}}^e = \tilde{\mathbf{x}}_{(i-1)k+1}^e, \dots, \tilde{\mathbf{x}}_{ik}^e$ , for each object  $v_i$ , the integer that can be described by this sequence of bits. Algorithm 4.1 describes the first phase.

First of all, we must guarantee that the number of clusters of each object is limited to  $p$ . If the number  $|\ell(v)|$  of clusters containing object  $v$  is greater than  $p$ , we repair the chromosome by removing  $|\ell(v)| - p$  clusters from  $v$  uniformly at random. Lines 1–2 of Algorithm 4.1 summarize this procedure, in which by “ $s \in \mathbf{x}$ ” we mean that “ $s = x_h$  for some  $h$ ,” and by “ $|s|$ ,” the number of bits equal to one in the binary representation of  $s$ .

After the repair, we have a feasible solution whose value is computed in lines 4–14. We compare each pair of objects  $(u, v)$ , calculating the Jaccard or set-intersection similarity and adding the error to the solution value, according to Equation (4.1). Note that in lines 5 and 6, we abuse notation:  $l(u) \leftarrow \mathbf{x}_u$  gets the labeling of object  $u$ . We also split the amount of error caused by  $u$  and  $v$  into two parts: the positive error  $e^+$ , indicating that labels assigned to  $u$  and  $v$  are too similar, and the negative error  $e^-$ , indicating otherwise. These errors are used in the local search phase to improve the solution. Note that vectors  $e^+, e^- \in \mathbb{R}^+$ . After local search, the solution may be improved and, in such case, we consolidate the change of the solution in the chromosome (line 16).

The running time complexity of the first phase is  $O(kn^2)$  when  $k$  is unrestricted. Note that the repair phase runs in  $O(kn)$  time. The solution value computation has a quadratic factor due to the comparison of each pair of objects. The time complexity of function  $\mathcal{H}$  depends directly on  $k$  and the type of function. Assuming  $k$  is unrestricted, when  $\mathcal{H} = \mathcal{J}$ , we need to compute the union and intersection of the two sets of labels, which takes  $O(k)$  time each. In the case of  $\mathcal{H} = \mathcal{I}$ , we only need to calculate the intersection, which takes  $O(k)$  time. However as observed in the previous section, for most practical applications,  $k < 64$ , which enables the representation using a 64-bit integer. The intersection can be implemented using a single machine AND operation, the union using a single machine

$\text{OR}$  operation, and the bit counting running in  $O(1)$  using a lookup table<sup>2</sup>. Using this implementation, the time complexity of phase 1 is  $O(n^2)$ , if  $k < 64$ .

In the second phase, a local search procedure is applied to the solution found in phase 1. We developed a local search that explores neighborhood solutions based on error reduction. This procedure is detailed on Section 4.3.3. We also use the local search methods proposed by Bonchi et al. [23]. They developed two local searches, one for OCC using the Jaccard index as the  $\mathcal{H}$  function and another using the set-intersection function. Section 4.3.4 details these algorithms.

---

**Algorithm 4.1:** Decoder – Phase 1

---

**Input:** a vector  $\mathbf{x} \in \mathbb{N}^n$ , integers  $k, p \in \mathbb{N}$ , and set  $V$ .  
**Output:** modified  $\mathbf{x}$  and the fitness  $\mathcal{C}$  of  $\mathbf{x}$ .

```

// Repair chromosome
1 foreach  $s \in \mathbf{x}$  such that  $|s| > p$  do
2   Remove  $|s| - p$  elements from  $s$  uniformly at random;
3    $\mathcal{C} \leftarrow 0$ ;
4 foreach pair  $(u, v) \in V \times V$  do
5    $l(u) \leftarrow \mathbf{x}_u$ ;
6    $l(v) \leftarrow \mathbf{x}_v$ ;
7    $error \leftarrow \mathcal{H}(l(u), l(v)) - s(u, v)$ ;
8    $\mathcal{C} \leftarrow \mathcal{C} + \text{absval}(error)$ ;
9   if  $error > 0$  then
10     $e_u^+ \leftarrow e_u^+ + error$ ;
11     $e_v^+ \leftarrow e_v^+ + error$ ;
12  else
13     $e_u^- \leftarrow e_u^- - error$ ;
14     $e_v^- \leftarrow e_v^- - error$ ;
15 LocalSearch();
16 Rewrite the improved solution on the chromosome  $\mathbf{x}$ ;
17 return modified  $\mathbf{x}$  and solution value  $\mathcal{C}$ ;
```

---

<sup>2</sup>Another approach is to use the special hardware instruction `POPCNT` found in modern processors. For details, see Haque et al. [94].

### 4.3.3 Error Reduction Local Search

Algorithm 4.2 depicts the local search procedure based on error reduction, which we call **OLS**. In line 4, we sort the objects  $v$  in nonincreasing order of  $e_v^+ + e_v^-$ , such that we start with the object whose labeling leads to the largest error in the entire clustering process. For the first given  $\tau \leq n$  objects in this order, **OLS** attempts to reduce or augment the total similarity driven by the values of  $e^+$  and  $e^-$ . This is done by removing or adding labels that impact total similarity. For a given object  $v$ , if  $e_v^+ \geq e_v^-$ , we try to find a most common label between  $v$  and all other objects, and remove it from  $v$  (line 16). Otherwise, if  $e_v^+ < e_v^-$ , we try to add to  $v$  a most common label that was not assigned to it (line 18). If none of these *add* or *remove* operations can be done, we remove a label from  $v$  uniformly at random and replace it with a label not assigned to  $v$ , also chosen uniformly at random (line 10–14). In this case, in each exchange we only select a new label not previously chosen in the previous iterations. In the worst case, we try all  $k$  labels. If an improvement is reached, the solution value and the errors of each object are updated (lines 19–25) and a new iteration begins. After  $\tau$  iterations without improvement, the local search stops.

The time complexity of **OLS** is tricky to compute since it depends on the starting solution. The innermost loop (starting in line 10) iterates at most  $k$  times since each label is added or removed only once. Note that adding (respectively, removing) a label that was removed (respectively, added) in earlier iterations (of the loop starting on line 10) does not lead to an improvement in solution quality. The exchange on line 14 can be done in  $O(1)$  time. The operations in lines 16 and 18 can be done in  $O(\log k)$  time using a red-black tree data structure (Bayer [13]) for the histogram of labels that we use to keep track of the most used labels. However, note that for small values of  $k$ , it is worthwhile to use a naive linear approach running in time  $O(k)$ . The update operations in lines 19–25 take  $O(n)$  time since we need to recalculate the function  $\mathcal{H}$  for all pairs  $(v, w)$  with fixed  $v$  and  $w \neq u \in V$ . Therefore, the loop starting on line 6 has time complexity  $O(\tau(k \log k + kn))$ . If  $k < 64$  and  $\tau \leq n$ , then the time complexity is  $O(n^2)$ . As observed earlier, it is hard to estimate the complexity of the loop starting on line 3, but its main components are  $O(n \log n)$  from line 4 and the complexity of the loop starting on line 6. Therefore, each iteration has time complexity  $O(n^2)$ .

**Algorithm 4.2:** Error Reduction Local Search – OLS.

---

**Input:** labeling function  $\ell$ ; vectors  $e^+, e^- \in \mathbb{R}^+$ ; integers  $p, \tau \in \mathbb{N}$ ; set  $V$ .

- 1 Let  $\mathcal{C}$  be the cost of  $\ell$  computed with Obj. Func. (4.1);
- 2  $impr \leftarrow \text{True}$ ;
- 3 **while**  $impr = \text{True}$  **do**
- 4     Let  $V'$  be the set of objects  $v \in V$ , sorted in non-increasing order of  $e_v^+ + e_v^-$ ;
- 5      $impr \leftarrow \text{False}$ ;  $i \leftarrow 0$ ;
- 6     **while**  $i \leq \tau$  **do**
- 7         Take the next  $v \in V'$  in the given order;
- 8          $\hat{\ell}(v) \leftarrow \ell(v)$ ;
- 9          $local\_impr \leftarrow \text{True}$ ;
- 10         **while**  $local\_impr = \text{True}$  **do**
- 11              $\hat{\mathcal{C}} \leftarrow \mathcal{C}$ ;
- 12              $local\_impr \leftarrow \text{False}$ ;
- 13             **if**  $e_v^+ \geq e_v^-$  **and**  $|\ell(v)| = 1$  **then**
- 14                 Exchange the unique label  $c \in \ell(v)$  for a  $c' \notin \ell(v)$  chosen uniformly at random;
- 15             **if**  $e_v^+ \geq e_v^-$  **and**  $|\ell(v)| > 1$  **then**
- 16                 Remove from  $\ell(v)$  the  $c \in \ell(v)$  which corresponds to the largest cluster containing  $v$ ;
- 17             **if**  $e_v^+ < e_v^-$  **and**  $|\ell(v)| < p$  **then**
- 18                 Add to  $\ell(v)$  the  $c$  which corresponds to the largest cluster not containing  $v$ ;
- 19         **foreach**  $u \in V \setminus \{v\}$  **do**
- 20              $\delta \leftarrow \mathcal{H}(\ell(u), \hat{\ell}(v)) - \mathcal{H}(\ell(u), \ell(v))$ ;
- 21             **if**  $\delta > 0$  **then**
- 22                  $e^+(v) \leftarrow e^+(v) + \delta$ ;  $e^+(u) \leftarrow e^+(u) + \delta$ ;
- 23             **else**
- 24                  $e^-(v) \leftarrow e^-(v) + \delta$ ;  $e^-(u) \leftarrow e^-(u) + \delta$ ;
- 25              $\mathcal{C} \leftarrow \mathcal{C} + \delta$ ;
- 26         **if**  $\mathcal{C} \leq \hat{\mathcal{C}}$  **then**
- 27              $impr \leftarrow \text{True}$ ;  $local\_impr \leftarrow \text{True}$ ;
- 28         **else**
- 29              $\mathcal{C} \leftarrow \hat{\mathcal{C}}$ ;
- 30         **if**  $local\_impr = \text{True}$  **then** Go to line 3;
- 31         **else**  $\ell(v) \leftarrow \hat{\ell}(v)$ ;  $i++$ ;

32 **return** labeling  $\ell$  and solution value  $\mathcal{C}$ ;

---

### 4.3.4 Bonchi et al. Local Search

Bonchi et al. [23] proposed two local search algorithms to deal with OCC using the Jaccard index and set-intersection functions. Their framework is based on the relabeling of objects, one at time, by solving a system of linear equations in the case of Jaccard, and by applying a simple greedy algorithm in the case of set intersection. The main idea is to find a good labeling of a single object given a fixed labeling of the other objects.

Let  $v$  be an object and  $\ell$  be a labeling for all objects. The error incurred by  $v$  is defined as

$$C_v(\ell(v)|\ell) = \sum_{u \in V \setminus \{v\}} |\mathcal{H}(\ell(v), \ell(u)) - s(v, u)|, \quad (4.4)$$

and, consequently, the objective function (4.1) can be rewritten as

$$\frac{1}{2} \sum_{v \in V} C_v(\ell(v)|\ell). \quad (4.5)$$

Using these observations, Bonchi et al. proposed Algorithm 4.3, which is a simple local search algorithm.

In the case of the Jaccard index, the following approach is used. Let  $v$  be an object to be relabeled and  $\ell$  be a fixed labeling for all  $u \in V \setminus \{v\}$ . Let  $x^v$  be an indicator vector such that  $x_j^v = 1$  if label  $j$  is assigned to object  $v$ , and  $x_j^v = 0$ , otherwise. Assume that the number of labels assigned to  $v$  is  $t$ , that is,

$$\sum_{j \in L} x_j^v = t. \quad (4.6)$$

Ideally we would like to have  $\mathcal{J}(\ell(v), \ell(u)) = s(v, u)$  for all  $u \in V \setminus \{v\}$ . This can be written as

$$\mathcal{J}(\ell(v), \ell(u)) = \frac{\sum_{j \in \ell(u)} x_j^v}{|\ell(u)| + t - \sum_{j \in \ell(u)} x_j^v} = s(v, u),$$

which is equivalent to

$$(1 + s(v, u)) \sum_{j \in \ell(u)} x_j^v - s(v, u)t = s(v, u)|\ell(u)| \quad (4.7)$$

for all  $u \in V \setminus \{v\}$ . Although Equations (4.6) and (4.7) are linear with respect to the unknowns  $x_j^v$  and  $t$ , these variables are integral and therefore the system seemingly cannot be solved in polynomial time. Bonchi et al. [23] applied a nonnegative least-squares optimization method which led to possibly fractional  $x^v$  and  $t$  values. They then sort  $x^v$  in non-increasing order, breaking ties arbitrarily. Let  $\pi^v$  be the permutation of labels induced by this order and consider the  $p$  sets  $\{\pi_1^v, \dots, \pi_i^v\}$  for  $i = 1, \dots, p$ . The new set of labels for  $v$  is the set  $\{\pi_1^v, \dots, \pi_i^v\}$  that minimizes Equation (4.4).

---

**Algorithm 4.3:** Bonchi et al. Local Search – BSL

---

**Input:** labeling function  $\ell$ ; set  $V$ .

```

1 Let  $\mathcal{C}$  be the cost of  $\ell$  computed with Obj. Func. (4.1);
2 improvement  $\leftarrow$  True;
3 while improvement = True do
4   improvement  $\leftarrow$  False;
5   foreach  $v \in V$  do
6     Find the labeling  $L$  that minimizes  $C_v(L|\ell)$ ;
7     Let  $\hat{\mathcal{C}}$  be the cost of  $(L|\ell)$  computed with Obj. Func. (4.1);
8     if  $\hat{\mathcal{C}} < \mathcal{C}$  then
9        $\ell(v) \leftarrow L$ ;  $\hat{\mathcal{C}} \leftarrow \mathcal{C}$ ;
10      improvement  $\leftarrow$  True;
11 return labeling  $\ell$  and solution_value;

```

---

For the set-intersection indicator, Bonchi et al. presented a greedy approach which starts from an empty labeling for a given object and fixes the labeling for the other objects as is done in the approach for the Jaccard index. In each iteration, the label that causes the least error while improving the solution value is chosen. If such a label cannot be found, the algorithm stops the search for this object and goes on to the next. In fact, this greedy search solves  $C_v(\ell(v)|\ell)$  in line 6 of Algorithm 4.3.

As in Algorithm 4.2, the time complexity of Algorithm 4.3 is tricky to determine because of its dependency on the starting solution. For the Jaccard function, the dominant factor is the nonnegative least squares that can be computed in  $O(m^3)$  time, where  $m$  is the major dimension of the matrix (Householder [100]). Note that the matrix resulting from Equation (4.7) has dimension  $n \times (k + 1)$ . As observed earlier, if  $k < n$ , then the complexity of an iteration of the main loop (starting on line 3) is  $O(n^4)$ . For the set-intersection function, each label search takes  $O(k^2)$  time, which is done at most  $p$  times. As this operation is performed for each object, each iteration of the main loop takes  $O(nk^3)$  time.

## 4.4 Experimental results

### 4.4.1 Instances

We used several datasets, grouped in two major categories, to evaluate our algorithms. The first category has instances with known multi-label assignments. For this case, we

computed the similarities between the objects using the Jaccard index. These datasets were used to check the quality of the labelings produced by the algorithms when compared to the actual labelings. We used two datasets in this category. The first, named EMOTIONS, corresponds to psychological trials of people listening to music (Trohidis et al. [197]). There are 593 objects (trials) and six available labels. The second dataset, named YEAST, is formed by micro-array expression data and phylogenetic profiles with 2417 genes in a learning set for which 14 functional classes (labels) are assigned (Elisseeff and Weston [54]).

The second category contains instances with unknown multi-labelings. The first dataset in this category corresponds to animal trajectories from the Starkey Project (Rowland et al. [183]<sup>3</sup>). We used an instance with 88 trajectories of elk, mule deer, and cattle, and classify them using five labels. To calculate the similarity between each trajectory, we used the approach presented in Chen et al. [35], which defines the *Edit Distance in Real Sequences* (EDR). The EDR is defined as following: let  $P = [(x_1, y_1, t_1), \dots, (x_n, y_n, t_n)]$  be a trajectory such that each triple  $(x, y, t)$  is a position in space and time. Denote by  $r(P) = [(x_2, y_2, t_2), \dots, (x_n, y_n, t_n)]$  the remainder of the trajectory, i.e., the original trajectory without the first point. Let  $P$  and  $Q$  be two different trajectories. For  $p \in P$ ,  $q \in Q$ , we say that  $m(p, q) = 1$  if  $|p_x - q_x| < \varepsilon_x$  and  $|p_y - q_y| < \varepsilon_y$  and  $|p_t - q_t| < \varepsilon_t$ , i.e., the distance in space and time is not larger than a constant factor. We take  $m(p, q) = 0$  otherwise. We say the EDR is

$$EDR(P, Q) = \begin{cases} |P| & \text{if } |Q| = 0, \\ |Q| & \text{if } |P| = 0, \\ \min(EDR(r(P), r(Q)) + m(p_1, q_1), \\ \quad EDR(r(P), Q) + 1, \\ \quad EDR(P, r(Q)) + 1), & \text{otherwise.} \end{cases}$$

The similarity between two trajectories  $u$  and  $v$  is given by

$$s(u, v) = 1 - EDR(u, v). \quad (4.8)$$

The second group of instances in this category is from the field of biology. They consist of homologous groups of proteins from the SCOP taxonomy (Murzin et al. [151]<sup>4</sup>). This taxonomy is a hand-made tree classification of functional proteins. We tested four databases with 669, 587, 567, and 654 proteins (objects) for which we assigned at most with five labels. The similarities were calculated as follows: For a node  $u$  in the protein tree, let  $d(u)$  be the depth of  $u$  in the tree. If  $u$  is the root,  $d(u) = 0$ . For any pair of

---

<sup>3</sup>Available at <http://www.fs.fed.us/pnw/starkey>.

<sup>4</sup>Available at <http://scop.mrc-lmb.cam.ac.uk/scop>.

nodes  $u$  and  $v$ , let  $lca(u, v)$  denote the lowest common ancestor of  $u$  and  $v$ . Let  $V$  be the set of leaves of the classification tree. The similarity between different objects  $u \in V$  and  $v \in V$  is defined as

$$s(u, v) = \frac{d(lca(u, v))}{\max(d(u), d(v)) - 1}.$$

The third group in this category consists of 1,000 newsgroup messages spread over 20 newsgroups (Rennie et al. [177]). The similarities are calculated using a base dictionary of the 500 most common words excluding stop words and common names but keeping political and religious references. For each message, we construct a characteristic vector to reflect the *Term Frequency – Inverse Document Frequency* (TF-IDF) of each word in the dictionary (Jones [105]). TF-IDF is defined as

$$\text{TF-IDF}(t, d, D) = \left(0.5 + \frac{0.5 \times f(t, d)}{\max\{f(w, d) : w \in d\}}\right) \times \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

where  $t$  is the term,  $d$  is the message,  $D$  is the set of all messages, and  $f(t, d)$  is the frequency of term  $t$  in message  $d$ . Using these vectors, we applied a radial basis function to obtain the similarities

$$s(u, v) = e^{\|\mathbf{u}-\mathbf{v}\|^2} \quad (4.9)$$

where  $\mathbf{u}$  and  $\mathbf{v}$  are the characteristic vectors of messages  $u$  and  $v$ , respectively.

#### 4.4.2 Evaluated algorithms

Using the two representations of Section 4.3.1 and decoders from Sections 4.3.3 and 4.3.4, we consider the following variations of BRKGAs:

- **OLS-Comp**: BRKGA using the compact representation and local search from Section 4.3.3;
- **OLS-Ext**: Same as above but using the extended representation;
- **BLS-Comp**: BRKGA using the compact representation and Bochi et al. local search from Section 4.3.4;
- **BLS-Ext**: Same as above but using the extended representation;

We also tested the algorithms from Bonchi et al. [23]. Originally, each run of those algorithms starts with a simple random vector and ends when Algorithm 4.3 cannot find improvements. To fortify these algorithms, we built multi-start approaches around them, allowing each run to take several iterations, each starting with a different random vector. We always keep the best solution and the algorithms stop when a stopping criterion is

reached. These modifications have a great impact on the original algorithms in terms of solution quality. As expected, the multi-start approaches outperformed the original algorithms. Consequently, we use them here but still refer to them as Bonchi.

It worths to mention that we also try to build a integer linear programming to the set-intersection case but this approach is very slow even for the smallest instances. As we aim at large datasets, we discarded this approach.

### 4.4.3 Computational environment and parameters

The experiments were conducted on identical machines with two 6-core Intel Xeon 2.4 GHz CPUs (two threads per core) and 32 GBytes of RAM running GNU/Linux. Running times reported are UNIX real wall-clock times in seconds, excluding the effort to read the instance. The algorithms are implemented in C++ and we use the GNU `g++` compiler version 4.8. Random numbers were generated by an implementation of the Mersenne Twister [143]. For the nonnegative least-squares method, we use the Householder rank-revealing QR decomposition [100] provided by the Eigen library [89].

For the BRKGAs, we set the parameters following the advise of Gonçalves and Resende [81]. The population size was set to  $pop = 500$ , the elite size to  $pop_e = \lceil 0.30pop \rceil$ , and the number of mutants to  $pop_m = \lfloor 0.15pop \rfloor$ . The probability of inheriting each allele from the elite parent was  $\rho_e = 0.70$ . We used the island model (Whitley et al. [206]) with three independent and concurrent populations where every 100 generations each population exports its two best solutions to the other populations. After 300 generations without improvement, all populations are reset to vectors of random keys. We set  $\tau = 2\sqrt{n}$  after performing some short tuning trials and analyze the quality of solutions as a function of optimization time. We use 12 simultaneous cores for decoding.

Thirty independent runs were performed for all five algorithms. Each algorithm ran for a given maximum amount of time (instance dependent) or at most 1,000 generations (or iterations) without improvement of the best solution.

### 4.4.4 Defining maximum running times for BRKGA

Since we have several types of problems, a significant variation in BRKGA running times is expected on each problem. We carried out some preliminary experiments and observed that the BRKGAs obtained a large number of small improvements in solution quality. In fact, in the first hour of optimization, the BRKGAs improved the solution about every two or three generations. In light of this, we performed one long run for an instance of each scenario. For these long runs we did not set a time limit but rather stopped after 1,000 generations without improvement of the best solution. As we only did one long run per scenario, we cannot draw any statistically significant conclusion

about comparisons of the BRKGAs with the different  $\mathcal{H}$  functions. We limit ourselves to observing the general behavior of these BRKGAs.

Figure 4.2 depicts the convergence of the cost value as a function of time and iterations for the EMOTIONS dataset. The  $Y$ -axis shows the cost value. The  $X$ -axis represents the time in seconds in Figure 4.2a and the number of iterations in Figure 4.2b. The solid red lines show the convergence for the BRKGA with the set-intersection function for values of  $p$  from one to six. The dashed blue line shows the same but for the Jaccard index function. We note that the BRKGA takes a long time with small improvements in the best solution and, after 30,000 seconds (about 8.3 hours), the improvements are even rarer. For the BRKGA with the Jaccard index function, values very close to zero, a lower bound on the optimum, are reached. Using the set-intersection indicator, the results are worse but the convergence behavior is similar. Looking at the number of iterations, one can note that the BRKGA with set-intersection function converges faster than it does with respect to time. We can conclude, therefore, that the local search spends more time using this function.

For other instances, the curves have the same behavior and they can be found in Appendix C.1. For the YEAST dataset, running times are much larger than those on EMOTIONS. In this case, the BRKGA with the set-intersection function has slower convergence than it does with the Jaccard index but is able to obtain improvements even after running for 1,000,000 seconds (about 11 days). For the Starkey project dataset, the running times are very small for both  $\mathcal{H}$  functions, mainly due to the size of the problem

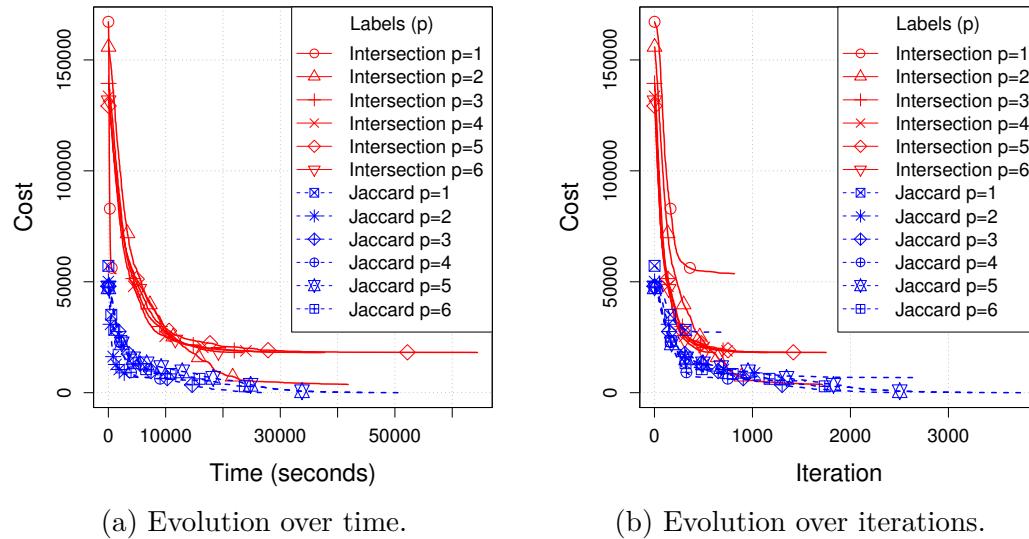


Figure 4.2: Evolution of the cost for the EMOTIONS dataset. Only 1% of the symbols correspondent to solutions are showed.

(88 objects). One notes that the BRKGA with set intersection is much worse than with the Jaccard index (except for the case of  $p = 2$ , in which they obtained similar results). For the SCOP dataset 1, we observe different behavior: although the convergence is similar, running times are smaller than for the previous scenarios and, more importantly, the BRKGA with set intersection shows poor results when compared to EMOTIONS and YEAST. Later, we discuss these results in more detail. For newsgroup messages, we omitted the curves for the BRKGA with set intersection since it obtained results 20 times worse than with the Jaccard index but with convergence about 400 times faster. This may indicate that set intersection is not appropriate for clustering in this type of instance.

From these experiments, we can estimate the running times for each scenario and set time bounds for the experiments to follow. For EMOTIONS and YEAST, we limit the experiments to at most eight hours (28,800 seconds) trying to balance running time and solution quality. This limit is less favorable to the BRKGA with set intersection which still finds solution improvements after 15 days (1,250,000 seconds). For Starkey and SCOP, we set the maximum running time to one hour (3,600 seconds). For newsgroup messages, the maximum running time is set to seven days (604,800 seconds).

#### 4.4.5 Evaluating the quality of the algorithms on ground-truth instances

To evaluate the quality of the algorithms, we first applied them on instances in which we know the actual labeling *a priori*. We compare the costs of the final solutions produced by the algorithms and also make use of two metrics, precision and recall. Define  $P(x) = \{\{u, v\}, u \neq v : x(u) \cap x(v) \neq \emptyset\}$ , the set of unordered pairs of objects with at least one common label in  $x$ . Let  $g$  be the labeling of the ground truth. The *precision* of a labeling  $\ell$  is defined as

$$\text{Precision}_g(\ell) = \frac{|P(\ell) \cap P(g)|}{|P(\ell)|}. \quad (4.10)$$

The precision is proportion of the number of relevant true positives obtained by the algorithm to all labeling assigned by the algorithm. In other words, it is the proportion of obtained labelings that are similar to the given labelings. The *recall* of a labeling can be seen as the relative quantity of correct labels (true positives) obtained by the algorithm. The recall of a labeling  $\ell$  is defined as

$$\text{Recall}_g(\ell) = \frac{|P(\ell) \cap P(g)|}{|P(g)|}. \quad (4.11)$$

Note that for  $\ell = g$ ,  $\text{Precision}_g(\ell) = \text{Recall}_g(\ell) = 1$ .

Figure 4.3 shows the performance of the algorithms for the EMOTIONS dataset. The  $X$ -axis on all plots corresponds to the maximum number of allowed labels per object.

For Figures 4.3a and 4.3c, the  $Y$ -axis corresponds to the scaled solution costs (lower being better). In the plots, the red line with circles shows results for Bonchi, the black lines with triangles for the BRKGAs with the OLS decoder, while the blue lines with squares show results for the BRKGAs with the BLS decoder. Solid symbols correspond to the compact chromosome representation and hollow symbols correspond to the extended representation. The description of Figures 4.3b and 4.3d is similar, but there, solid lines show precision and dashed lines, recall (with higher being better). Figures 4.3a and 4.3b

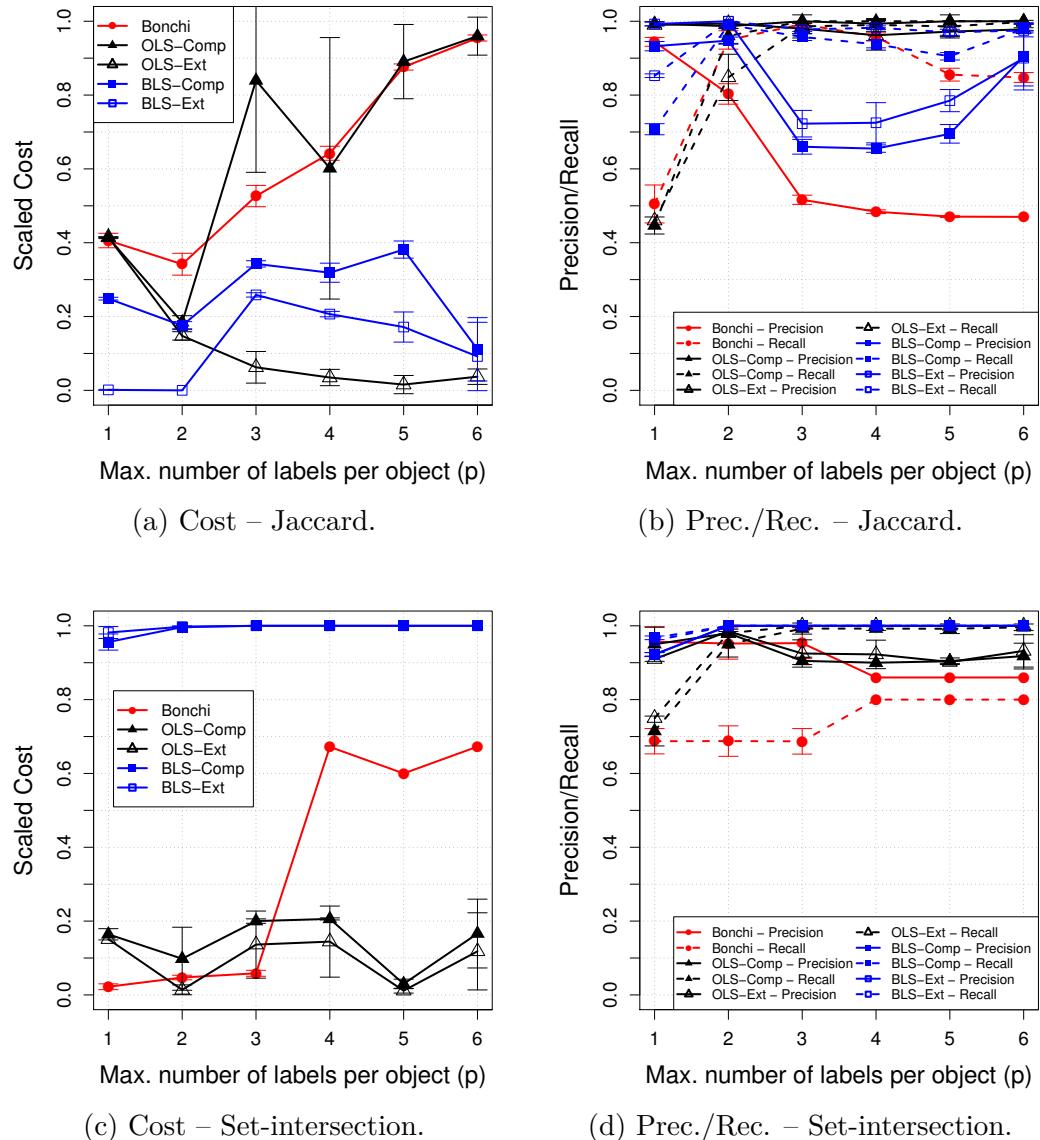


Figure 4.3: Comparison of costs, precision, and recall among the algorithms for the EMO-TIONS dataset.

correspond to the Jaccard index function and Figures 4.3c and 4.3d, to the set-intersection function.

The algorithms using the Jaccard index with  $p = 1$  have similar performance. One notes that this case is very close to traditional clustering and results in a partition of the objects. But as more labels per object are allowed, the performance of **Bonchi** degrades while all but one of the BRKGAs perform well. The exception is **OLS-Comp**, which followed the performance of **Bonchi** and displayed a large variation in its results. The same decoder using the extended representation (**OLS-Ext**) obtained the best results on average. Analyzing Figure 4.3b, one can see that the precision and the recall of the BRKGAs are always above those of **Bonchi**. Even **OLS-Comp**, which obtained costs similar to those of **Bonchi**, showed better precision and recall. The algorithms that use the Bonchi et al. approach for set intersection showed worse performance than those using the **OLS** decoder. It is worthwhile to mention that the solutions obtained using the Jaccard index function have different cost values from those obtained using the set-intersection function, as the objective functions are different.

Figure 4.4 depicts results for the YEAST dataset. Its description is identical to that of Figure 4.3. Here, we observe a very different behavior for both the Jaccard index and set-intersection functions. For the Jaccard index, note that the algorithms using the **OLS** decoder perform poorly (Figure 4.4a) when the number of labels allowed for each object is small, while algorithms using **BLS** performed quite well. As  $p$  increases, **OLS** outperforms **BLS**. In particular, when  $p$  is between 8 and 11, the algorithms switch their behavior.

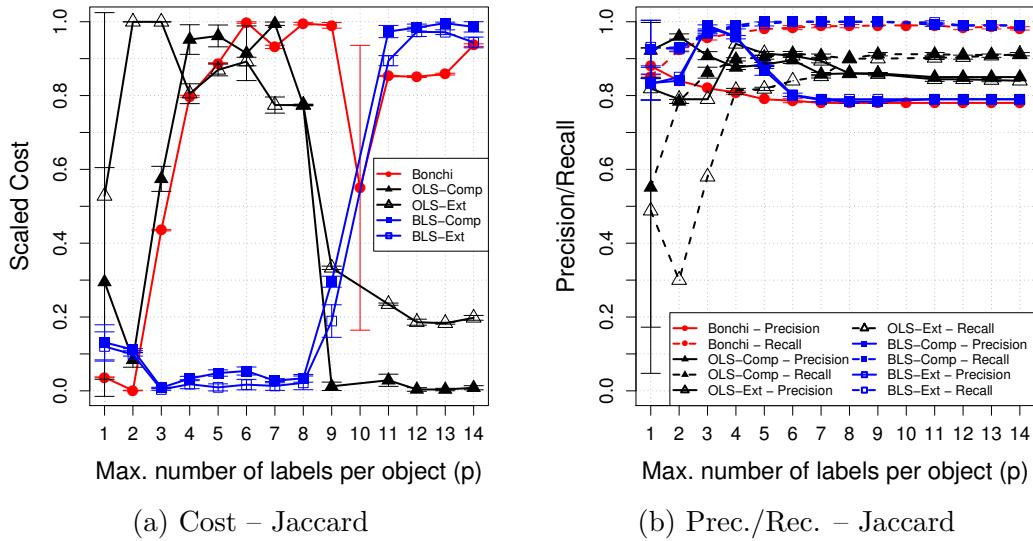


Figure 4.4: Comparison of costs, precision, and recall among the algorithms for the YEAST dataset (continue in the next page).

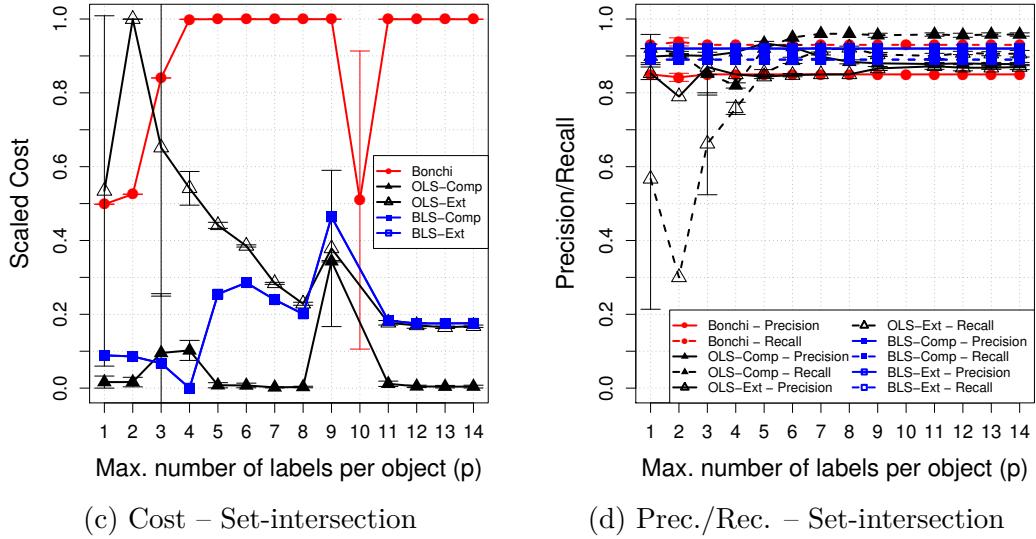


Figure 4.4: (Continued) Comparison of costs, precision, and recall among the algorithms for the YEAST dataset.

However, it is interesting to notice that both precision and recall are relatively stable. For the set-intersection function, the plots resemble the behavior of those for the Jaccard index function on the EMOTIONS dataset. Finally, note that the BRKGAs with BLS found better solutions on the YEAST dataset than they did on EMOTIONS. All BRKGAs obtained better precision on YEAST than they did on EMOTIONS.

#### 4.4.6 Evaluating the algorithms for instances with unknown multi-labeling

To evaluate the algorithms on instances with no multi-labeling given *a priori*, we first consider the Starkey project dataset. The algorithms were run for labelings of size  $p = 1, 2, 3$ , from a total of  $k = 5$  available labels. Figure 4.5 shows the costs of the labelings obtained by the algorithms. For each scenario (consisting of an  $\mathcal{H}$  function and a value of  $p$ ), we scaled the costs into the interval  $[0, 1]$  using the minimum and maximum costs of all algorithms. The box plots show the smallest cost (lowest whiskers), the first quartile (bottom box), the median cost (filled circles), the third quartile (upper box), the largest cost (highest whiskers), and the outliers (gray hollow circles). We observe that the sizes of the boxes for all configurations are very small, indicating that similar costs were found for all runs on a given algorithm.

For the Jaccard index, BLS-Comp and BLS-Ext were able to produce the best results, although slightly different. For set intersection, OLS-Comp found better solutions, although in some cases OLS-Ext also found good solutions as its outliers suggest. To confirm these

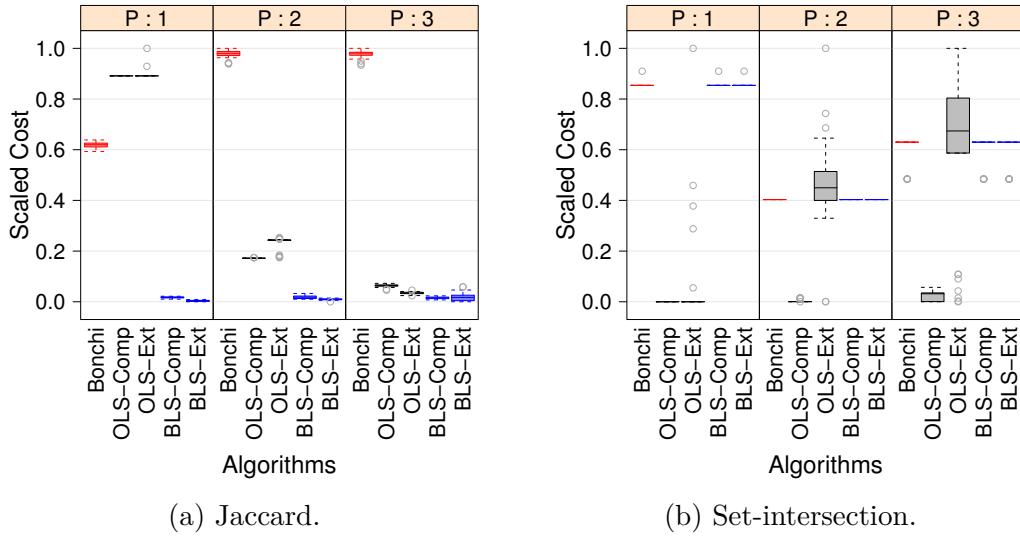


Figure 4.5: Boxplot of median and quartiles for each algorithm in Starkey dataset.

results, we applied the Wilcoxon-Mann-Whitney U test (Conover [38]). This test assumes as the null hypothesis that the location statistics are equal in both distributions. Assuming a confidence interval of 95%, almost all algorithms presented significant difference in their results when compared to each other for the Jaccard index. The exception is the pair **BLS-Comp** and **BLS-Ext** for  $p = 3$ , whose  $p$ -value is 0.67 and we cannot thus assure a significant difference. For the set-intersection function, **OLS-Comp** was significantly better than the other algorithms. **Bonchi**, **BLS-Comp**, and **BLS-Ext** found solutions having the same value on all runs. Detailed test results are presented in Table C.1 in Appendix C.2.

Figure 4.6 shows results on the four SCOP datasets. The structure is similar to that of Figure 4.5, except that the scaling was done for each dataset set separately and then combined in the plots for each configuration. For the Jaccard index, **BLS-Comp** and **BLS-Ext** found good solutions for  $p = 1$ , whereas **OLS-Comp**, **OLS-Ext**, and **BLS-Ext** had the best results for  $p = 2$ . In fact, the U test did not present significant difference among these three algorithms and value of  $p$  (for a confidence interval of 95%). For  $p = 3$ , **OLS-Ext** presented significantly better results than the other algorithms. For set intersection, the algorithms performed similarly and for most cases, no significant difference was found. Notice that the results are closer to 1.0 (worst solutions) which indicates that the algorithms converged to local minima frequently. But since the bottom whiskers are at 0.0, the algorithms did find a good solution. Again, refer to the Appendix C.2 for the complete tests.

Figure 4.7 shows results for the instance from the newsgroup messages dataset. For the Jaccard index function, the behaviors of the algorithms are similar to those on the

other instances. For  $p = 1$ , the algorithms using the Bonchi approach are able to find better results, but for  $p \geq 2$ , OLS-Comp found significantly better solutions. For set intersection, the algorithms using the Bonchi approach found the best results with no significant difference among them.

Table 4.1 lists the algorithms that obtained the best results for each dataset and configuration. For the Jaccard index, both OLS-Ext and BLS-Ext presented the best results for most cases, which indicates that algorithms using the extended representation

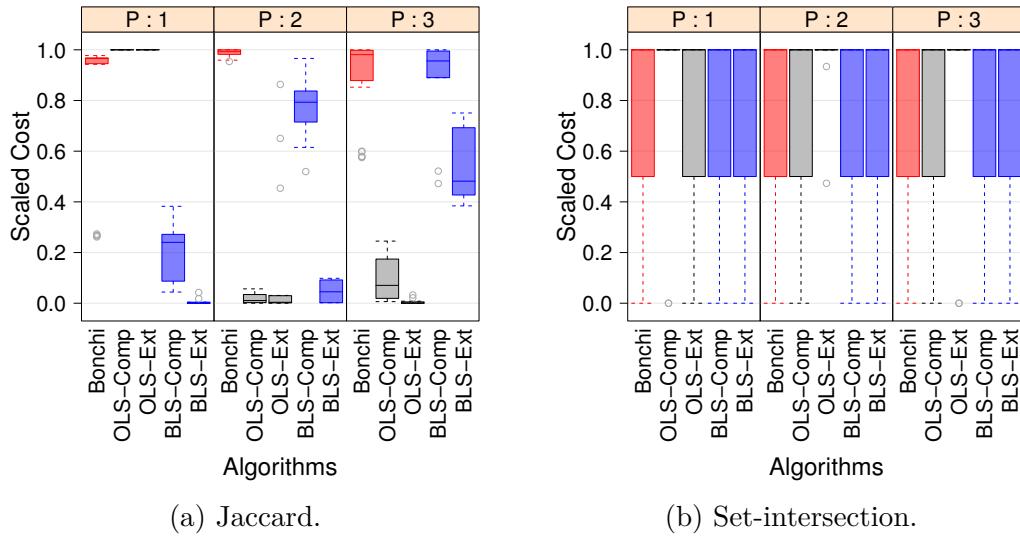


Figure 4.6: Boxplot of median and quartiles for each algorithm in SCOP dataset.

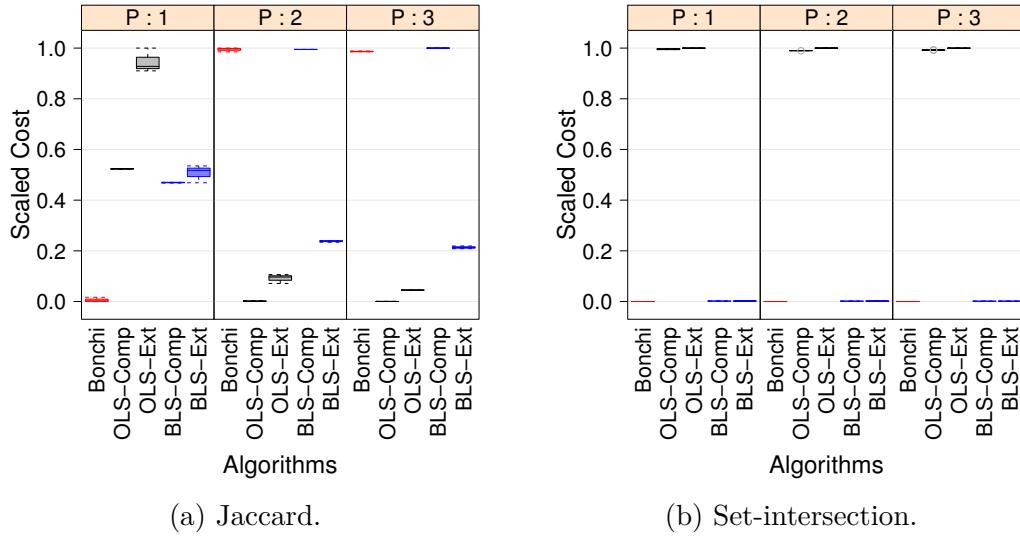


Figure 4.7: Boxplot of median and quartiles for each algorithm in newsgroup messages.

are able to find better solutions than those using the compact representation. The **OLS** approach obtained more best solutions than did the **BSL** approach. For set intersection, the performances of the algorithms were similar, reaching the best solution at least once in most cases. In the Starkey dataset, the algorithms using **OLS** performed better than those which did not. For the newsgroup message dataset, **Bonchi** found the best solutions.

Table 4.1: Algorithms that computed the best results for each instance,  $\mathcal{H}$  function, and  $p$ .

	Inst.	p	Jaccard	Intersec		Inst.	p	Jaccard	Intersec
Starkey	1	BLS-Ext		OLS-Comp/ OLS-Ext	SCOP3	1	BLS-Ext	All	
	2	BLS-Ext	OLS-Comp			2	OLS-Comp	Except OLS-Ext	
	3	BLS-Ext		OLS-Comp/ OLS-Ext		3	OLS-Ext	All	
SCOP1	1	BLS-Ext	All		SCOP4	1	BLS-Ext	All	
	2	OLS-Ext	All			2	OLS-Ext	All	
	3	OLS-Ext	All			3	OLS-Ext	All	
SCOP2	1	BLS-Ext	All		News.	1	Bonchi	Bonchi	
	2	OLS-Ext	All			2	OLS-Comp	Bonchi	
	3	OLS-Ext	All			3	OLS-Comp	Bonchi	

## 4.5 Concluding remarks

In general, the BRKGAs are effective at finding good solutions and are able to beat the Bonchi et al. approach on most cases when using the Jaccard index in the objective function. For set intersection, the algorithms based on Bonchi et al. approach presented better results than other algorithms in most scenarios. Also, the extended representation allows the BRKGAs to obtain better results when compared to the compact representation on most cases. On the negative side, running times to convergence for the BRKGAs can be high. We believe that this is not a major issue, since most applications of OCC are prospective in nature and therefore do not require real-time response.

# Chapter 5

## The Winner Determination Problem in Combinatorial Auctions

**T**HIS chapter addresses the Winner Determination Problem (WDP) in combinatorial auctions. This problem consists in picking a subset of bids in a general combinatorial auction to maximize the overall profit using the first-price model. This winner determination problem assumes that a single bidding round is held to determine both the winners and prices to be paid. Although this problem is not directly related to network design, combinatorial auctions have been widely used by governmental agencies to sell rights of electromagnetic spectrum. In these spectrum auctions, telecommunication companies place bids over spectrum ranges in a given region. The winner will have the rights to operate over that region. An extensively number of spectrum auctions have been conducted in the last 15 years mainly due to the increase of wireless data demand and the introduction of new technologies. There is abundant literature on spectrum auctions. Cramton et al. [41] present a good review and recent results can be found in Hoefer et al. [96] and Yang et al. [211].

In this thesis, we do not address spectrum auctions themselves but more general auctions that can be used in several scenarios. We introduce six variants of biased random-key genetic algorithms for this problem. Three of them use a novel initialization technique that makes use of solutions of intermediate linear programming relaxations of an exact mixed integer-linear programming model as initial chromosomes of the population. An experimental evaluation compares the effectiveness of the proposed algorithms with a standard mixed linear integer programming formulation, a specialized exact algorithm, and the best-performing heuristics proposed to date for this problem. The proposed algorithms are competitive and offer strong results, mainly for large-scale auctions. This chapter is based on Andrade et al. [7].

## 5.1 Introduction

An auction is a mechanism or negotiation protocol for exchanging goods and services. In general, such goods are offered for bid, followed by a pre-determined round of bids, after which the highest bidder is pronounced the winner and pays for the negotiated item. Procurement auctions, on the other hand, are defined as follows: the auctioneer requests a set of goods, and each bidder can submit bids for this set. The lowest bidder is pronounced the winner and the auctioneer is paid for the goods. Today, auctions are widespread and, more importantly, distributed, thanks mainly to the Internet. Examples can be found in advertisement and position auctions in search engines such as Google and Yahoo!, as well as those coordinated by governments to negotiate radio spectrum, offshore oil and gas exploration, general goods, and services, among others.

In this chapter, we are interested in general combinatorial auctions where bidders place bids (usually sealed) on subsets of goods, also known as *bundles*. Each bidder has access to a finite set of goods and is asked to come up with a list of bids, where each bid is an offer for a subset of goods. The objective of a bidder is to win the bid at an acceptable price. The greatest advantage of this type of auction is that it generates high economic efficiency since it allows the bidders to express both complementarity and substitutability of their preferences within bids. More formally, let  $M$  be a set of goods,  $g_1, g_2 \in M$  be two goods, and let  $f : 2^M \rightarrow \mathbb{R}$  be a valuation function for sets of these goods. Goods  $g_1$  and  $g_2$  are said to be *complementary* if and only if  $f(\{g_1\}) + f(\{g_2\}) \leq f(\{g_1, g_2\})$ , where  $\{g_1, g_2\}$  denotes a bundle of goods  $g_1$  and  $g_2$ . They are said to be *substitutes* if and only if  $f(\{g_1\}) + f(\{g_2\}) \geq f(\{g_1, g_2\})$ . For other variations, see Parsons et al. [162]. Since we allow bids for any subset of goods, there could be as many as  $n(2^m - 1)$  bids, where  $n$  is the number of bidders and  $m$  is the number of goods. Hence, one of the key problems arise in auction mechanisms is to determine the winners of the auction, i.e. selecting pairwise disjoint bids to maximize the sum of the values of the selected bids. For other associated problems, see Cramton et al. [41].

We focus on the *Winner Determination Problem* or WDP. In general, the WDP is equivalent to the weighted set packing problem, a well-known  $\mathcal{NP}$ -hard problem (Garey and Johnson [65]). Notice that solving the WDP in auctions with no additional constraints and where only simple bids are allowed (i.e., bids for a single good) can be easily done in  $O(nm)$ -time. The seminal work on the WDP is credited to Rothkopf et al. [182], who identified several special cases that can be solved in polynomial time. Such cases involve special bid structures like bid trees, geometrical regions, and cardinal restricted bids. However, these structures limit the expressiveness of the bids, potentially leading to an inefficient economy (Bichler et al. [19]).

Sophisticated exact approaches to solve the WDP were proposed by Sandholm [185, 186] and Escudero et al. [56], who also presented a polyhedral study applying cuts in an exact algorithm that scaled well in auctions with up to 300 bids. With regard to approximation algorithms, the general case cannot be approximated by a factor of  $O(m^{1/2+\epsilon})$  of the optimal total value of the selected bids (unless  $\mathcal{P} = \mathcal{NP}$ ), a bound inherited from the set packing problem (see Halldórsson [92], who also describes an algorithm with  $O(\ell/(\log \ell)^2)$ -approximation that runs in  $O(\max(\ell^c, m^2\ell^2))$ -time, where  $\ell$  is the number of bids and  $c$  is a constant). An approximation algorithm with factor  $O(\sqrt{m})$  is described in Lehmann et al. [122] for the case in which each bidder has interest in only one particular bundle. For more approximation algorithms for special formulations, see Dobzinski et al. [49] and Feige and Vondrák [58]. Several such algorithms and special cases of the winner determination problem are revisited in Blumrosen and Nisan [22].

The first heuristic addressing the WDP specifically is the **Casanova** algorithm (Hoos and Boutilier [98]), a multi-start stochastic local search algorithm that runs on top of greedy randomized initial solutions. A hill-climbing procedure can be found in Holte [97]. The first metaheuristic-based heuristics addressing the problem were a genetic algorithm and a simulated annealing heuristic (Schwind et al. [188]). A hybrid simulated annealing with local search called **SAGII** was proposed by Guo et al. [90]. To date, the strongest results come from a memetic algorithm by Boughaci [25] and Boughaci et al. [26].

It is interesting to observe that the WDP can also be modeled as the Multidimensional Knapsack Problem (MDKP), enabling the utilization of the algorithms developed to tackle the latter. As with the weighted set packing, the MDKP is a well-studied  $\mathcal{NP}$ -hard problem frequently used to evaluate new algorithms due to its intrinsic difficulty and its well-established benchmark test sets. One of the best heuristics to deal with MDKP was developed by Raidl and Gottlieb [174] and consists in a genetic algorithm with weight-biased representation using surrogate duality to modify item weights. Recently, Mansini and Speranza [140] presented an exact algorithm based on the idea of restricted core problems where a recursive variable fixing step is done until a given threshold is reached. The remaining subproblems are explored by a branch-and-bound approach. Several other approaches can be found, among them genetic algorithms (Chu and Beasley [36]), tabu search (Vasquez and Vimont [199]), ant-based optimization (Alaya et al. [2]), GRASP (Chardaire et al. [33]) and other hybridization techniques, e.g. Puchinger et al. [173] and Boyer et al. [27].

We address the winner determination problem of general combinatorial auctions using the first-price model for single goods. We restrict ourselves to sealed auctions that use a single round to determine winners and prices to be paid, where the bids can be placed with no constraints other than their non-negativity. We also consider that the bids are anonymous and that bidder identity is not used to model or solve the underlying prob-

lem. Six variants of biased random-key genetic algorithms (BRKGAs) are implemented to address this problem, three of them adopting a novel scheme that employs linear programming (LP) relaxations to initialize the population when the underlying problem can be modeled as a 0–1 integer linear program. In such problems, an LP relaxation directly serves as a chromosome of the BRKGA heuristic, since both are defined over the interval [0, 1]. Experiments comparing the BRKGAs with a standard mixed integer-linear programming formulation of the WDP solved with a commercial solver, as well as the best performing heuristics proposed for the problem, the best performing exact algorithm, and the best performing heuristic for the MDKP, are carried out to identify the strengths and drawbacks of each approach.

This chapter is organized as follows. In Section 5.2 we formalize the winner determination problem in combinatorial auctions. We then address biased random-key genetic algorithms follow up with a description of our heuristics in Section 5.3. Experimental results are provided and discussed in Sections 5.5 and 5.6, respectively. Concluding remarks are made in Section 5.7. Additional results are found in Appendix D.

## 5.2 General combinatorial auctions and their formulations

Several models for combinatorial auctions have been proposed in the literature, but most of them introduce additional constraints to limit the context of the auction so as to expose special properties that make the problem computationally easier. We next present a general description. Let  $N = \{1, 2, \dots, n\}$  be a set of bidders and  $M = \{1, 2, \dots, m\}$  be a set of goods. A collection of bids is represented by a tuple  $\mathcal{B} = (\mathcal{B}_1, \dots, \mathcal{B}_n)$  such that  $\mathcal{B}_i$  is the bid set of bidder  $i$ . Each bid  $B \in \mathcal{B}_i$ , for  $i = 1, \dots, n$ , is a list of desired goods (bundle), i.e.,  $B \subseteq M$  such that bidder  $i$  provides the function  $b_i : 2^M \rightarrow \mathbb{R}^+$  that measures how much bidder  $i$  is willing to pay for a bundle. An allocation of goods is represented by a tuple  $\mathcal{S} = (\mathcal{S}_1, \dots, \mathcal{S}_n)$  where  $\mathcal{S}_i$  is the set of winner bids of bidder  $i = 1, \dots, n$ . Note that

$$\left( \bigcup_{S \in \mathcal{S}_i} S \right) \cap \left( \bigcup_{R \in \mathcal{S}_j} R \right) = \emptyset, \text{ for all } i, j \in N.$$

We consider the *private information model* in which the auctioneer only knows the set of bids  $\mathcal{B}$  and the functions  $b_i$ , for all  $i = 1, 2, \dots, n$ . We restrict ourselves to first-price sealed auctions where the bidders submit one bid per desired bundle. This contrasts with *iterative auctions*, where the bidders may submit multiple bids to the same bundle in different rounds. Only the auctioneer can handle the bids and the winning bidders

pay what they offered in their winning bids, i.e.,  $b_i(\mathcal{S}_i)$ . Sealed auctions are used mainly in government and industry procurements. For more on the theory of auctions and its variants, see Krishna [114].

The major work done in the literature has related the WDP with both the weight set packing problem and the multidimensional knapsack problem (see Bikhchandani and Ostroy [20] for other models). In the set packing problem, we want to select weighted pairwise disjoint sets from a collection of items while maximizing the sum of the weights of the selected sets. A special case of this problem is the weighted stable set problem where we have a graph whose nodes have weights and one must choose a subset of nodes with no common incident edge and maximize the sum of weights. The WDP can be reduced to the weighted stable set problem in the following way: consider the intersection graph  $G = (V, E)$ , where each  $s \in V$  represents a bid. An edge  $(s, s') \in E$  exists if and only if  $B_s \cap B_{s'} \neq \emptyset$ , such that  $B_s \in \mathcal{B}_i$ ,  $B_{s'} \in \mathcal{B}_j$ ,  $i, j \in N$  and  $i \neq j$ , i.e., the edge exists if and only if two bids from different bidders request a common good. A stable set on this intersection graph corresponds to a set of pairwise-disjoint bids from different bidders. Mathematically, the stable set problem can be written as the standard integer programming model

$$\begin{aligned} \max \quad & \sum_{s \in V} b_s x_s \\ \text{s.t.} \quad & x_s + x_{s'} \leq 1 \quad \forall (s, s') \in E \\ & x_s \in \{0, 1\} \quad \forall s \in V, \end{aligned} \tag{5.1}$$

where we use the abbreviation  $b_s = b(B_s)$ , i.e.,  $b_s$  is the value offered for bundle  $s$ . Let the binary variable  $x_s = 1$  if and only if bid  $s$  is a winner. The above formulation enables overlapping among bids of a same bidder. Its main advantage is that the bidder in question need not present a bid for each subset of desired goods although the bidder may possibly overpay for some of them. In fact, this formulation is appropriate for super-additive valuations. The number of variables and constraints of this formulation is, respectively,  $\ell$  and  $O(\ell^2)$ , where  $\ell$  is the total number of bids.

Another very common way to deal with WDP is to model it as a multidimensional knapsack problem. We consider that each bid is an item to be packed in the dimensions induced by the goods. Let  $\hat{\mathcal{B}} = \bigcup_{i=1}^n \mathcal{B}_i$  be the set of all bids. In case two or more bids contain the same goods, we add a “dummy” good to each bid such that the new good uniquely identifies the bundle (Nisan [157]). Let  $w_{jk} = 1$  if good  $j \in M$  is considered in bid  $k \in \hat{\mathcal{B}}$ ,  $w_{jk} = 0$ , otherwise. The MDKP can be model as

$$\begin{aligned} \max \quad & \sum_{k \in \hat{\mathcal{B}}} b_k x_k \\ \text{s.t.} \quad & \sum_{k \in \hat{\mathcal{B}}} w_{jk} x_k \leq c_j \quad \forall j \in M \end{aligned} \tag{5.2}$$

$$x_k \in \{0, 1\} \quad \forall k \in \hat{\mathcal{B}}.$$

Again, we abuse the notation of  $b_k$  as the value offered for bundle  $k$  and we consider as winning bids, all  $k$  such that  $x_k = 1$ . For combinatorial auctions with single goods, we have that  $c_j = 1$  for all  $j \in M$ . A first observation is that this formulation can deal with multi-unit auctions where we can have multiple copies of good  $j$  (by allowing  $c_j \in \mathbb{N}$  for all  $j \in M$ ) and a bid can request a certain number of copies of the good (by allowing  $w_{jk} \in \mathbb{N}$ , for  $j \in M$  and  $k \in \hat{\mathcal{B}}$ ). The number of variables and constraints of this formulation are, respectively,  $\ell$  and  $O(m)$ , where  $\ell$  is the number of bids and  $m$  is the number of goods.

The choice between the stable set and MDKP models can be very tricky in the case of single-unit combinatorial auctions. There are two important aspects to analyze: the tightness of the formulations and their sizes. With respect to tightness, it is well-known that the MDKP model (again, for the single-unit case) generates tighter formulations than the stable set model, since the former contains more clique inequalities than the latter and, therefore, results in better linear programming relaxations (Padberg [161]). In fact, it is known that the stable set model, even with clique inequalities, leads to poor relaxations (Carr and Lancia [31]).

With respect to size, although the MDKP formulation has the size  $O(\ell m)$ , it can be much larger than the stable set formulation. The problem does not lie in the formulation itself, but in the input that can potentially be exponential for the MDKP in the number of goods. To illustrate this, suppose that a bidder has the following bids:  $B_1 = (\{1, 2\}, \$10)$ ,  $B_2 = (\{2, 3, 4\}, \$10)$ , and  $B_3 = (\{4, 5\}, \$10)$ . If we consider only these bids, the stable set formulation will have three variables and no constraint, since the overlapping bids belong to the same bidder. In the MDKP, the bidder must generate, besides the given bids, the bids  $B'_{12} = (\{1, 2, 3, 4\}, \$20)$ ,  $B'_{13} = (\{1, 2, 4, 5\}, \$20)$ ,  $B'_{23} = (\{2, 3, 4, 5\}, \$20)$ , and  $B'_{123} = (\{1, 2, 3, 4, 5\}, \$30)$  since the bidder cannot win overlapping bids in this model. Although we have only one constraint, the number of variables (bids) is exponential with respect to those in the stable set model. Note that if bidder identities are unknown, we must consider each bid individually and the MDKP becomes the best choice. In this work, we do not consider bidder identities and therefore adopt the MDKP model.

### 5.3 Biased random-key genetic algorithms for the winner determination problem

For solving the WDP, we proposed biased random-key genetic algorithms with three different decoders and two initialization schemes. We now focus on BRKGA decoders for the winner determination problem. Recall that an instance of the WDP consists of finite sets of bidders  $N = \{1, \dots, n\}$ , goods  $M = \{1, \dots, m\}$ , and a collection of bids

$\mathcal{B} = (\mathcal{B}_1, \dots, \mathcal{B}_n)$ , where  $\mathcal{B}_i$  is the bid set of bidder  $i \in N$ . As aforementioned, we have not addressed the bidder identities; instead, we consider the bids  $\hat{\mathcal{B}}$ , as defined in Section 5.2, but in some fixed order such that  $\hat{\mathcal{B}} = (B_1, B_2, \dots, B_t)$ , where  $t = |\hat{\mathcal{B}}|$ . Each bid  $B_j$  has value  $b_j$ . The decoders select a subset of bids that is maximal with respect to the sum of their values while respecting the pairwise-disjoint constraints among selected bids.

We develop three approaches for decoding a solution. These approaches are related in how they select and analyze the bids based on chromosome values and structural information of the problem. Define the size of each chromosome to be  $t$ . Our decoders associate each bid with an allele, i.e., the value of the  $j$ -th random key is associated with the  $j$ -th bid. The first step is to sort the bids in some particular order, generating a permutation of bids.

**Chromosomal approach:** The keys are sorted in non-increasing order of their values.

Ties are broken by element indices;

**Greedy approach:** We first choose the keys whose values are greater than or equal to a threshold  $\tau$ ; then, these keys are sorted in non-increasing order of the cost/benefit of their respective bids, i.e.,  $b_j/|B_j|$ . Notice that, in the greedy approach, the relative order of the bids is fixed for all possible chromosomes and, in fact, a permutation of the bids is not generated. Instead, we generate an ordered list containing a subset of the original bids. Ties are broken by element indices;

**Surrogate Duality approach:** Similar to the greedy approach but the cost/benefit is calculated differently. Let  $\alpha$  be the dual solution vector of the relaxation of Formulation (5.2) when  $x \in [0, 1]^t$ . Note that each  $\alpha_i$  is tied to good  $i$  and represents the “shadow price” of  $i$ . The cost/benefit of  $B_j$  is  $b_j/\sum_{i \in B_j} \alpha_i$ . This surrogate duality approach was first proposed by Pirkul [168]. As in the above greedy approach, the dual vector  $\alpha$  may be computed only once, and ordered lists can then be generated from it. Again, ties are broken by element indices;

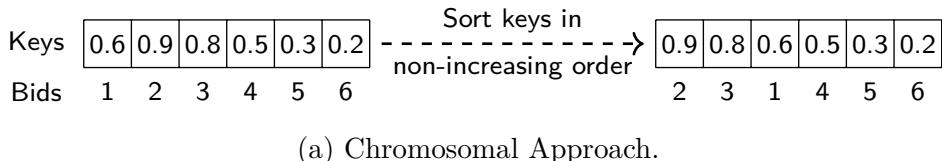
Note that in the greedy and surrogate approaches, the parameter  $\tau$  induces an implicit binary chromosome encoding and does not take advantage of the magnitude of the keys as does the chromosomal approach. The rationale behind the greedy approach is that the algorithm will take the most efficient bundles at first. This means that it will prefer the bids that most value the goods individually. The surrogate duality approach tries to capture the aggregate consumption levels of goods, meaning that bid efficiency is a measure of how much the bid impacts the entire system when it is chosen as the winner. In other words, if the marginal cost of the goods for a given bid is high, then this bid considers goods with high demand and it may not be worthwhile to choose it as winner if it were to offer a low value for these goods.

As an example, consider the chromosome in Figure 5.1. In chromosomal approach, we simply sort the keys generating a permutation of bids, as shown by the indices of the vector in Figure 5.1a. In the greedy and surrogate dual approaches, we first filter the bids by their keys (using  $\tau = 0.5$  in this example) and then sort the remaining bids in non-increasing order of their cost/benefit (as shown in the grey vector in Figure 5.1b).

One can note that in the chromosomal approach, the chromosome is used to generate a *permutation* of bids to be used in the subsequent processing. In the greedy and surrogate dual cases, the chromosome is used to generate a *subset* of bids whose size is controlled by parameter  $\tau$ . Note that if  $\tau = 0$ , then all bids are considered at once and, as the relative order of bids is fixed a priori due the cost/benefit relation, the decoder always returns the same solution. This way,  $\tau > 0$  can be viewed as a *separation threshold*. Lines 3–7 of Algorithm 5.1 summarize these procedures.

The next phase (lines 8–14) uses the sorted list of bids detailed above to construct a solution for the WDP. Initially, no bid is selected and all goods are unmarked. The decoder iterates over the bids in the supplied list, selecting a bid whenever all of its goods are not yet marked, thus maintaining the property that the winning bids are mutually exclusive with respect to their goods. If the current bid  $B_j$  is selected, its corresponding goods are then marked. Otherwise, if bid  $B_j$  has a conflicting good with another bid already selected, it is ignored and the value of the corresponding key, say  $\kappa_j$ , is reset to  $1 - \kappa_j$  if  $\kappa_j > 0.5$ , discouraging this bid from being considered in the following generations. Note that if  $\kappa_j \leq 0.5$ , this bid is already discouraged and its key value need not change.

After this primary construction phase, the algorithm has checked all bids in the chromosomal approach. Therefore, in this case, it returns the solution value. In the greedy



(a) Chromosomal Approach.

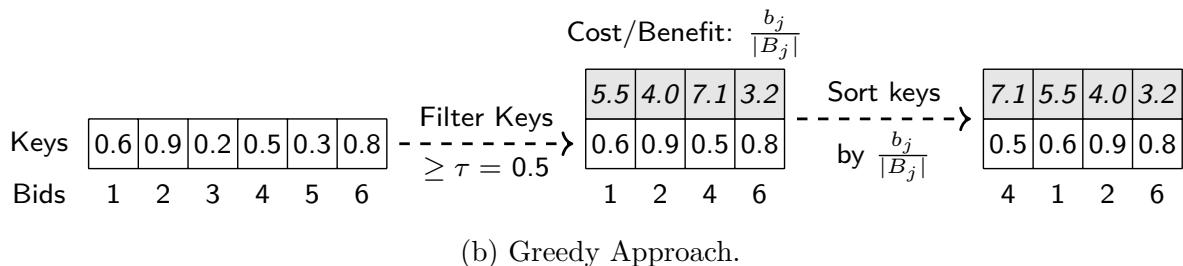


Figure 5.1: Example of sorting keys using the chromosomal and greedy approach.

---

**Algorithm 5.1:** Decoder for the Winner Determination Problem.

---

```

1 Let  $S$  be an empty list to hold the solution;
2 Let  $\kappa_j$  be the key associated with bid  $B_j$ ;
3 if the chromosomal approach is used then
4   Let  $L$  be a list of bid indexes ordered in non-increasing order of keys  $\kappa$ ;
5 else
6   Let  $L$  be a list of bid indexes such that  $\kappa_j \geq \tau$  for all bid  $B_j$ ;
7   Sort  $L$  in non-increasing order of cost/benefit according to greedy or surrogate dual approach;
8 foreach  $j \in L$  in the given order do
9   if  $B_j$  has no marked goods then
10     $S \leftarrow S \cup \{j\}$ ;
11    Mark all goods of  $B_j$ ;
12   else if  $\kappa_j > 0.5$  then
13     $\kappa_j \leftarrow 1 - \kappa_j$ ; // discourage bid  $B_j$ 
14     $L \leftarrow L \setminus \{j\}$ ;
15 if the chromosomal approach is used then
16   Go to Line 25;
// Process the remaining bids
17 Let  $L'$  be the list of indexes of remaining bids with unmarked goods;
18 Sort  $L'$  in non-increasing order of cost/benefit according to greedy or surrogate dual approach;
19 foreach  $j \in L'$  do
20   if  $B_j$  has no marked goods then
21     $S \leftarrow S \cup \{j\}$ ;
22    Mark all goods of  $B_j$ ;
23   if  $\kappa_j < 0.5$  then
24     $\kappa_j \leftarrow 1 - \kappa_j$ ; // encourage bid  $B_j$ 
25 return the fitness  $\sum_{j \in S} b_j$ .

```

---

and surrogate dual cases, some bids are not visited because of filtering by  $\tau$ . Therefore, the algorithm builds a secondary list containing those bids that include only unmarked goods (disregarding those that were not selected in the previous phase due to conflicts). The bids are then sorted according to one of the above criteria, and the algorithm iterates over this list adding the bids that do not create conflict with the bids already selected.

Each added bid  $B_j$  has its corresponding key  $\kappa_j$  reset to  $1 - \kappa_j$  if  $\kappa_j < 0.5$ , encouraging this bid to be considered in further generations. This secondary phase is described in lines 17–24.

The running time for this procedure to obtain the sorted list of bids is bounded by  $O(t \log t)$ , where  $t$  is the number of bids.<sup>1</sup> In the worst case, the sum of the number of iterations in the two **foreach** loops is at most  $t$ , given that all bids may be analyzed. Checking and marking of goods can be implemented in  $O(1)$  using a simple binary vector indexed by the goods. This implies that, for each iteration, we have  $O(m)$  checks and markings in the worst case, where  $m$  is the number of goods. Therefore, in the chromosomal case, we can bound the running time of the decoder by  $O(t \log t + tm)$ .

The running times for the greedy and surrogate duality cases are different from the previous cases since we have an extra sorting procedure and a second traversal over the bids. As argued in the start of this section, the relative order of bids is fixed and need only be calculated once a priori. In this case, the sort procedures of lines 7 and 18 can be done in linear time using an indicator vector, where each position corresponds to the position of a bid in the pre-calculated order. Note that each sort procedure is done over a partition of the bids and, therefore, both together have running times that can be bounded by  $O(t)$ . The first filtering traversal in line 6 takes  $t$  steps. The second traversal in line 17 is a function of  $\tau$  and takes less than  $t$  steps. Both loops together take  $t$  iterations over  $m$  goods. Thus, we can bound the total running time of these decoders by  $O(t) + 2t + tm = O(tm)$ .

## 5.4 Initializing the population of BRKGA

The most common approach to initialize the population of a BRKGA is to generate its chromosomes with uniformly drawn random keys over the interval  $[0, 1]$ . In an attempt to speed up the search, we introduce a novel approach where we use solutions to the linear programming (LP) relaxations of Equation (5.2) as chromosomes, given that the decision variables of these relaxations are such that  $0 \leq x_k \leq 1$  for all  $k \in \hat{\mathcal{B}}$ , where  $\hat{\mathcal{B}}$  represents the set of all bids and  $t = |\hat{\mathcal{B}}|$ . Therefore, a solution to the relaxed LP is a vector  $x \in [0, 1]^t$  that is compatible with the requirement of the keys of a BRKGA, and therefore we simply use the values of the optimal relaxed variables  $x_k$  as the corresponding alleles of an initial chromosome. An advantage of using such an individual in the initial population is that it is perhaps closer to a good solution than are most random individuals. In addition to the pure relaxation, we use relaxations generated by the insertion of cutting planes in

---

<sup>1</sup>Note that this term depends on the sort algorithm used, and, in fact, can be reduced to  $\Theta\left(t \frac{\log t}{\log \log t}\right)$  using fusion trees (Fredman and Willard [63]).

the original formulation. A *cutting plane* is an inequality that eliminates an infeasible solution for the original integer program. The insertion of cutting planes leads to tighter formulations with respect to the integer solutions (see e.g. Wolsey [209] for more details). We expect that chromosomes generated from these tighter relaxations will be decoded into solutions that are even closer to good integer solutions.

This process consists in two nested phases as shown in Algorithm 5.2. In the first phase (lines 5–7), cutting planes are generated and added to the formulation and its linear relaxation solved. This results in vector  $\tilde{x}$  such that  $0 \leq \tilde{x}_k \leq 1$ , for all  $k = 1, \dots, t$  except the fixed variables which have their values defined in next phase. Cut-generation procedures have been widely studied in the mathematical programming literature and can be implemented in different ways. Here, we do not make use of any particular cut-generation procedure but, rather, delegate their generation to the mixed integer programming (MIP) solver. To date, most modern MIP solvers, such as IBM ILOG Cplex [101], Gurobi Optimizer [91], and Fico Xpress [60], are able to generate general strong cuts, such as clique

---

**Algorithm 5.2:** Initialization by LP relaxations.

---

```

1 Let  $x_1, \dots, x_t$  be a vector such that  $x_k$  is the variable associated to bid  $B_k \in \hat{\mathcal{B}}$ ;
2 Let  $P$  be the empty initial population;
3  $k \leftarrow 1$ ;  $bound \leftarrow 0$ ;
4 while  $k < t$  and a stopping criterion is not reached do
5   while maximum cutting iterations or the time limit are not reached do
6     Insert cutting planes in the formulation if possible;
7     Solve the LP relaxation;
8   Let  $\tilde{x} \in [0, 1]^t$  be the relaxed optimal solution;
9    $P \leftarrow P \cup \{\tilde{x}\}$ ;
10  // Do variable fixing
11  Fix  $x_k$  to  $bound$ ;
12  if  $bound = 0$  then
13     $bound \leftarrow 1$ ;
14    if  $k \geq 2$  then
15      Unfix  $x_{k-1}$ ;
16  else
17  if  $P$  is not complete then
18    Generate random chromosomes to complete  $P$ ;

```

---

cuts (Nemhauser and Wolsey [155]) and Gomory fractional cuts (Gomory [74]) known for the tight relaxations they produce. The task of finding cutting planes and reoptimization can be time consuming and therefore we limit this procedure to at most a predetermined number of steps or stop after a maximum time limit is reached. The relaxed solution  $\tilde{x}$  is added to the initial population.

To generate several different chromosomes, we fix variables iteratively, generating other relaxations (lines 10–16). In alternating iterations, we fix some variable  $x_s$  to 0, meaning that the corresponding bid will not belong to any solution, or to 1, implying that the corresponding bid will belong to all solutions. This way, two consecutively generated chromosomes enforce the decision to select or not select the bid in question. One can note that we fix the variables in the order that they appear in set of bids. Another possible strategy is to choose a variable to fix uniformly at random saving the last fixed variable to restore its bounds. Both types of variable fixing procedures do not guarantee any solution quality, but diversify the search. Note that in the first iteration of Algorithm 5.2, no variable is fixed and a full relaxation of the model is solved. It is also possible, although unlikely, that two or more distinct variable fixings result in the same relaxation. In this case, we discard the duplicates.

Although the initialization with LP relaxations can speed up the convergence of the BRKGA, the time to generate these initial chromosomes is not negligible. It is worthwhile mentioning that solving an LP relaxation is a polynomial-time process, but finding cutting planes can be slow in certain situations, and several practical issues can contribute to this slowdown (Wolsey [209]). In this regard, we set the stopping criterion for this type of initialization to a specific running time or number of chromosomes, whichever comes first (line 4). The remaining chromosomes are generated at random as is usual in the standard BRKGA.

## 5.5 Experimental Setup

We conducted several experiments with three objectives. The first objective was to investigate the effectiveness of biased random-key genetic algorithms to find optimal solutions for instances where exact algorithms succeeded in finding one. The second was to evaluate the solution quality for those instances where an optimal solution could not be found. Finally, the third objective was to investigate the effectiveness of the initialization of BRKGA with LP relaxations. Throughout the experiments, we compare our results with state of the art algorithms for the WDP and the MKDP.

### 5.5.1 Instances

For the following experiments, we use two sets of instances. We first generated several instances using the Combinatorial Auction Test Suite, or CATS (Leyton-Brown et al. [125]), a standard generator of instances for combinatorial auction, largely adopted in the literature. The advantage of this suite lies in its ability to generate instances for several scenarios, such as time-scheduling auctions, matching auctions, region-border auctions, and even legacy distributions used in earlier works. We generated two blocks of instances: one of smaller instances containing from 40 to 400 bids whose number of goods vary between 10 and 100; and another comprised of larger instances with 1000 to 4000 bids and 256 to 1500 goods. Preliminary experiments showed that the number of goods does not considerably affect the running time of the algorithms. This fact was also observed by Buer and Pankratz [29]. Henceforth, we set the number of goods to be smaller than the number of bids in the test problems, seeking auctions with relevant conflicts among the bids, i.e. with several bids competing for the same sets of goods.

From CATS, we used legacy distributions L2, L3, L4, L6, L7, and the “arbitrary,” “matching,” “paths,” “regions,” and “scheduling” distributions (a total of 10 classes). We did not use the L1 and L5 distributions due to problems generating non-dominated bids.<sup>2</sup> These instances broadly cover general combinatorial auctions. Following, we replicate the description given by Leyton-Brown and Shoham [126]:

- L2:** Random distribution that chooses a number of goods  $g$  uniformly at random from  $[1, m]$  and assigns a price drawn uniformly from  $[g, 1000g]$ ;
- L3:** Uniform distribution that sets the number of goods to some constant  $c$  and draws the price offer from  $[0, 1000]$ . Default:  $c = 3$ ;
- L4:** Decay distribution that starts with a bundle size of 1, and repeatedly increments the bundle size and draws a uniform random number from  $[0, 1]$ , stopping when this number exceeds a given parameter  $\alpha$ . The prices is drawn uniformly from  $[g, 1000g]$  where  $g$  is the number of selected goods. Default:  $\alpha = 0.55$ ;
- L6:** Exponential distribution that requests  $g$  goods with probability  $e^{-g/q}$ , and assigns a price offer drawn uniformly at random from  $[g, 1000g]$ . Default:  $q = 5$ ;
- L7:** Binomial distribution that assigns each good an independent probability  $p$  of being included in a bundle, and assigns a price offer drawn uniformly at random from  $[1, 1000g]$ , where  $g$  is the number of selected goods;

---

<sup>2</sup> We said that a bid  $(B_i, b_i)$  is non-dominated if either  $B_i \not\subseteq B_j$  for all bid  $j$ , or if  $B_i \subseteq B_j$  then  $b_i > b_j$ , for any bid  $j$ .

**Paths:** Models an auction of transportation links between cities, or more generally of edges in a nearly-planar graph. Bids request sets of goods correspond to paths between randomly selected pairs of nodes. Substitutable bids are those that connect the same pairs of nodes. Prices depend on path lengths;

**Regions:** Models an auction of real estate, or more generally of any good over which two-dimensional adjacency is the basis for complementarity. Bids request goods that are adjacent in a planar graph;

**Arbitrary:** Similar to regions, but relaxes the planarity assumption and models arbitrary complementarities between discrete goods such as electronics parts or collectibles;

**Matching:** Models auctions for airline take-off and landing rights. Each bid requests one take-off and landing slot bundle, and each bidder submits a set of bids for acceptable bundles;

**Scheduling:** Models a distributed job-shop scheduling domain, with bidders requesting a set of resource time-slots that will satisfy their specific deadlines.

For further details, see Chapters 18 and 19 of Cramton et al. [41]. For each distribution, we generated three instances of each type according to Table 5.1 using the default parameters supplied by CATS. We also used the CATS hard mode (`-default_hard` flag), that generates three instances with approximately 1024 bids and 256 goods for each distribution with the objective of being hard to solve. The suite does not generate hard instances for “path” distributions. In summary, we used CATS to generate 120 small and 117 large instances.

A drawback of CATS is that instances appear to be easy in the sense that they can generally be solved by exact algorithms in reasonable time (see Boughaci et al. [26], Guo et al. [90]). In fact, such studies adopted a set of instances provided by Lau and Goh [120], which are indeed harder than instances generated by CATS. These instances were generated using several factors observed in real brokering systems as pricing of a bundle, preference of each bidder, and fairness of good distributions. We selected three

Table 5.1: Instance classes and their sizes.

	CATS							LG			
Bids	40	80	200	400	1000	1024 <sup>†</sup>	2000	4000	1000	1000	1500
Goods	10	10	50	50	256	256	512	1024	500	1000	1500
# of insts.	30	30	30	30	30	27	30	30	100	100	100

<sup>†</sup> Generated using `default_hard` flag.

classes of such instances and called them LG. Each class contains 100 instances, all having more than 1000 bids.

In short, we experimentally analyzed the proposed algorithms on 537 instances where 417 of them are large with respect to number of bids, i.e., they have more than 1000 bids. Table 5.1 summarizes the instances adopted in the subsequent analysis. The last line of this table shows the number of instances in each class.<sup>3</sup>

An important aspect of instances for the winner determination problem (WDP) is their tightness. This metric is used by Chu and Beasley [36] to craft instances of the Multidimensional Knapsack Problem (MDKP), largely used in the literature as the main benchmark for this problem. The tightness of a constraint  $j$  is defined as

$$t_j = \frac{c_j}{\sum_{k \in \hat{\mathcal{B}}} w_{jk}}, \quad (5.3)$$

where  $c_j$  is the availability of resource  $j$  and  $w_{jk}$  is the amount of resource  $j$  requested by  $k$ , as defined in Formulation (2) of the main text. Note that for the WDP, the tightness is

$$t_j = \frac{1}{|\{B : j \in B, B \in \hat{\mathcal{B}}\}|}, \quad \forall j \in M, \quad (5.4)$$

by definition, i.e., the tightness is defined as the inverse of the number of bids that request a certain good. Note that a low  $t_j$  indicates that good  $j$  is required by several bids, probably increasing the problem difficulty.

In the Chu and Beasley MDKP instances, every constraint of a given problem has the same tightness, which is either 0.25, 0.5, or 0.75. For the WDP instances, tightness varies for each constraint and depends heavily on the type and size of the problems. For the most classes, as the size increases, tightness decreases, notably for the L2, L7, and LG classes. By definition, for some classes tightness is almost constant as, e.g. L3 and matching. Table 5.2 shows the average tightness of each constraint for each class and problem size. Note that the hard “path” instances are not shown since the CATS suite does not generate hard instances for “path” distributions.

### 5.5.2 Algorithms

In our evaluation, we considered specialized algorithms for both the winner determination problem and the multi-dimensional knapsack problem. We test two exact algorithms and two heuristics, both considered to be state of the art for both problems.

To tune the parameters of the heuristics, we use the *iterated racing* procedure (Birattari et al. [21]). This method consists in sampling configurations from a particular distribution, evaluating them using either the Friedman test or the  $t$ -test, and refining the sampling

---

<sup>3</sup>These instances can be found in <http://www.loco.ic.unicamp.br/instances/wdp.html>

Table 5.2: Average of instances tightness.

Class	Size							
	40	80	200	400	1000	1024	2000	4000
L2	0.347	0.341	0.094	0.098	0.019	0.026	0.012	0.008
L3	0.333	0.333	0.333	0.333	0.333	0.208	0.333	0.333
L4	0.506	0.436	0.507	0.420	0.555	0.605	0.514	0.511
L6	0.433	0.381	0.355	0.314	0.351	0.578	0.351	0.344
L7	0.543	0.471	0.111	0.109	0.015	0.068	0.009	0.004
Arbitrary	0.192	0.207	0.132	0.140	0.134	0.138	0.130	0.131
Matching	0.347	0.333	0.333	0.333	0.333	0.333	0.333	0.333
Paths	0.562	0.567	0.350	0.375	0.192	—	0.171	0.143
Regions	0.196	0.220	0.133	0.135	0.133	0.135	0.134	0.131
Scheduling	0.317	0.249	0.190	0.252	0.202	0.114	0.195	0.216
Size	1000/500		1000/1000		1500/1500			
LG	0.317		0.249		0.190			

distribution with repeated applications of F-Race. We use the `irace` package (López-Ibáñez et al. [135]), implemented in R, for parameter tuning. For each heuristic, we use a budget of 2,000 experiments in the tuning procedure, where each experiment was limited to one hour. For this propose, we chose one instance of each size from each CATS class, and ten instances from each LG class, totalling 109 instances.

### Boughaci et al. Memetic Algorithm — $\text{BO}_{\text{MA}}$

Boughaci et al. [26] presented a specialized memetic algorithm for WDP. It is a genetic algorithm that uses random-key encoding tied to a local search procedure for exploitation. Their representation and decoding phase is similar to our chromosomal approach. But in the reproduction phase, the individuals are chosen to crossover only if their “differ” sufficiently based on a similarity metric. In this case, the similarity is the size of the intersection of the winning bid sets induced by these individuals.

For crossover, the algorithm chooses an individual  $X$  from the set  $C_1$  of best individuals and another individual  $Y$  from a set  $C_2$  that contains individuals with small similarity with respect to individuals in  $C_1$ . The crossover is done traversing the concatenation  $XY$  and choosing no-conflict bids in this order. The local search that characterizes the memetic flavor is the following: With probability  $wp$  choose the best bid (one that maximizes the auctioneer’s revenue) or, with probability  $1 - wp$ , a random bid. This bid is added to the solution and all other conflicting bids are removed. This process is repeated for a given number of iterations and returns the best individual found.

This algorithm outperformed other algorithms for the WDP that were previously proposed in the literature (**Casanova** of Hoos and Boutilier [98] and **SAGII** of Guo et al. [90]) and, indeed, presents competitive results, as we show in next sections.

We use the original C implementation provided to us by the author of Boughaci [25]. A slight modification was done to their implementation to support timing limits. In parameter tuning, we use the following ranges: population size  $pop\_size \in [300, 2000]$ ;  $|C_1| \in [5, 20]$ ;  $|C_2| \in [7, 30]$ ;  $wp \in [0.1, 0.5]$ ; and maximum local search iterations  $max\_lsi \in [100, 500]$ . The best setup indicated by **irace** was:  $pop\_size = 1400$ ;  $|C_1| = 12$ ;  $|C_2| = 24$ ;  $wp = 0.3$ ; and  $max\_lsi = 150$ .

### Raidl and Gottlieb Weight-Biased Genetic Algorithm — **RG<sub>RK</sub>**

Raidl and Gottlieb [174] proposed a genetic algorithm for the MDKP where a solution is represented by a weight-biased real vector using surrogate dual information in the decoding phase. The authors used several probability distributions to generate the biased vectors. Their experiments show that following a log-normal distribution often works best. The weighted vector  $w$  is generated such that  $w_j = (1 + \gamma)^{\mathcal{N}(0,1)}$  where  $\mathcal{N}$  denotes a normally distributed random number with mean 0 and unit standard deviation and  $\gamma > 0$  is a parameter that controls the intensity of biasing. Thus, the item  $j$  is biased by a new price  $p'_j = p_j w_j$ .

The decoding phase uses the approach of Pirkul [168] and is similar to our surrogate dual ordering. In this case, the pseudo utility of a item  $j$  is

$$u_j = \frac{p'_j}{\sum_{i=1}^m \alpha_i r_{ij}},$$

where  $m$  is the number of dimensions,  $\alpha_i$  is the dual value associated with dimension  $i$  and  $r_{ij}$  is the demand of item  $j$  in dimension  $i$ .

The offspring generation is done by selecting two parents via binary tournaments, performing uniform crossover in their characteristic vectors, flipping each bit with probability  $1/n$  (mutation probability), performing repair if a capacity constraint is violated, and always applying local improvement. If such a new candidate solution is different from all solutions in the current population, it replaces the worst of them only if the new candidate has a better fitness than the worst solution (Puchinger et al. [173]). This algorithm, to date, is one of the best heuristics for the MDKP.

We use the Java code provided to us by the authors of Pfeiffer and Rothlauf [167]. In parameter tuning with **irace**, we use the following ranges: population size  $pop\_size \in [300, 2000]$ ; tournament size  $tourn\_size \in [10, 30]$ ; and  $\gamma \in [0.01, 0.20]$ . The best results were obtained with:  $pop\_size = 500$ ;  $tourn\_size = 20$ ; and  $\gamma = 0.15$ .

### Mansini and Speranza Exact Algorithm — CORAL

Mansini and Speranza [140] presented an exact algorithm for MDKP using the idea of *core items*. This algorithm divides the problem into subproblems with a limited number of variables. For each subproblem, a recursive variable fixing procedure is applied trying to fix as many variables as possible. The remaining unfixed variables represent the core items for which it is difficult to decide whether they belong to an optimal solution. For these items, a *restricted core* problem is built and solved with a branch-and-bound procedure. To speed up the branch-and-bound, several pruning conditions are introduced. This algorithm has a non-trivial implementation and we omit several details here which can be found in the original publication.

CORAL is considered to be a state of the art exact algorithm for the MDKP. It works particularly well on instances with a large number of items. Its key feature is the ability to continually improve lower bounds using the optimal solutions from their restricted subproblems. However, in instances with a large number of constraints, CORAL has difficulty in finding good solutions, as observed in Mansini and Speranza [140].

We use the original Java implementation provided to us by the authors. Slight modifications were done to their implementation to support timing limits. All parameters were set as in the original paper.

### Standard Mixed Integer Programming Solver — CPLEX

We also used the IBM ILOG CPLEX Optimizer [101] as a standard mixed integer programming solver to deal with Formulation (5.2) directly. CPLEX uses a branch-and-cut algorithm which is a deterministic enumerative procedure that explores a solution space using a bounding process in the solution values of the tree built during the search. Further detail can be found in Wolsey [209]. This type of algorithm has exponential running time in the worst case. The implementation of the IBM ILOG CPLEX Optimizer uses linear programming relaxations to bound the solution values in addition to using primal heuristics to produce integer solutions. According to its documentation, the IBM ILOG CPLEX Optimizer is fully deterministic with default parameters that are used in our experiments.

Note that we use the cut generation procedure in our LP initialization approaches. In that situation, however, we only use solutions from the root node and the first level of the branching tree. In fact, we do not use CPLEX branching mechanism there but only solve the LP and apply cut generation procedures. All variable fixing is controlled by our procedure as shown in Algorithm 5.2.

We use the IBM ILOG CPLEX Optimizer version 12.5.0.0. All default control parameters were used, except time limit, which was set to 3,600 wall-clock seconds, and number

of threads, set to four. Using the default settings, CPLEX performs a preprocessing step to try to eliminate variables and constraints and calculate initial bounds. Unfortunately, we cannot know what methods are used to perform this preprocessing since CPLEX is a closed-source commercial package.

It is important to note that these settings are used only in the case where CPLEX is run stand-alone. To create the initial chromosomes for our algorithms based on LP relaxations, we set up CPLEX differently as described next.

## Our approaches

Our algorithms are described as follow:

**CA<sub>RA</sub>:** The proposed algorithm, defined in Section 5.3, using the chromosomal approach and random initialization;

**CA<sub>LP</sub>:** The proposed algorithm using the chromosomal approach but initialized with the optimal variables from the LP relaxations;

**GA<sub>RA</sub>:** The proposed algorithm using the greedy approach and random initialization;

**GA<sub>LP</sub>:** The proposed algorithm using the greedy approach and initialized with the optimal variables from the LP relaxations;

**SD<sub>RA</sub>:** The proposed algorithm using the surrogate duality approach and random initialization;

**SD<sub>LP</sub>:** The proposed algorithm using the surrogate duality approach and initialized with the optimal variables from the LP relaxations.

The proposed algorithms were written in C++ on top of the BRKGA API of Toso and Resende [196], which implements all of the problem-independent components described in Section 5.3. Random numbers were generated by an implementation of the Mersenne-Twister (Matsumoto and Nishimura [143]) and we used the standard sort algorithm of the C++ Standard Template Library. In particular, the used version implements the introsort algorithm whose worst case running time is  $O(n \log n)$  (Musser [152]). Our algorithms used four cores for simultaneous decoding.

To tune the BRKGA parameters with `irace`, we used the following ranges: elite percentage  $\in [0.10, 0.30]$ ; percentage of mutants introduced at each generation  $\in [0.05, 0.20]$ ; probability of inheriting each allele from elite parent  $\rho_e \in [0.5, 0.8]$ ; number of independent populations  $\pi \in [1, 3]$ ; exchange interval  $\delta \in [50, 200]$ ; and number of elite individuals in an exchange  $\eta \in [1, 2]$ . The population size was set to  $p = \min(10t, 2000)$ , where  $t$  is

the number of bids. The main reason for this upper bound is to bound the running time of each generation and allow the BRKGA to evolve for several generations. We noted that populations with over 2,000 individuals had slow convergence because of the time needed to evaluate each generation. This is true mainly on large instances. The tuning results obtained with `irace` were very close to the values suggested by Gonçalves and Resende [81]. The elite size was set to  $p_e = \lceil 0.20p \rceil$ , the number of mutants to  $p_\mu = \lfloor 0.15p \rfloor$ , and inheritance probability to  $\rho_e = 0.70$ . We evolved  $\pi = 3$  populations simultaneously and once every  $\delta = 100$  generations, each population exchanged its  $\eta = 2$  best solutions with the other populations.

For the greedy and surrogate duality approaches, we set the filter threshold  $\tau = 0.5$ . In these cases, we expect that half of the bids are assigned to the first phase allocation and the other half to the second phase. As the algorithm evolves, good bids will have random key values greater than or equal to  $\tau$  and the impact of the second phase will diminish since bad bids will have their goods marked in the first phase. Note that  $\tau$  has more impact on initial and mutants chromosomes than on others since on the latter the random keys evolved to a better solution.

For the approaches with LP-based initialization, we relaxed the integrality constraints of Formulation (5.2) to  $0 \leq x_k \leq 1$ . Since this procedure is time consuming, they were restricted as follows. The number of initial chromosomes was set to  $lp_{init} = \lfloor 0.1p \rfloor$ . Note that in Algorithm 5.2, this number may be small, since it is limited by the number of bids and duplications. If we obtain no duplicates,  $lp_{init} \leq 2t + 1$ , where  $t$  is the number of bids (the additive factor 1 is due the initial unrestricted relaxation). We allow two iterations of cut generation or five seconds to generate each chromosome. Both cut generation and the solution of the LP relaxation were done with the IBM ILOG CPLEX Optimizer version 12.5.0.0. We do not use other CPLEX features, such as variable fixing and branch-and-bound.

### 5.5.3 Computational environment and algorithm settings

The experiments were conducted on identical machines with quad-core Intel Xeon E5530 2.4 GHz CPUs and 32 GBytes of RAM running GNU/Linux. Running times reported are UNIX real wall-clock times in seconds, excluding the effort to read the instance. Each run was limited to 3,600 seconds for all algorithms. There are two reasons behind our choice of time limit. On the application side, procurement auctions are often managed in a short period of time to limit the response time of the bidders in order to achieve economic efficiency (for further details, see Chapters 2 and 23 in Cramton et al. [41]). On the algorithmic side, the exact approaches produced a pattern of slow convergence characterized by minimal decrease in the optimality gap throughout the execution on all

instances where an optimal solution was not found. This pattern can be clearly identified in the first hour of computation according to preliminary experiments. Notice in the results that follow that some running times are slightly over 3,600 seconds. This is because we wait for each algorithm to complete its current iteration before actually stopping it.

For the heuristics, we use an additional stopping criterion: 1,000 generations without improvement of the best solution found so far. In preliminary experiments, no instance presented an offset greater than 1,000 between two subsequent improvements. In fact, the average offset was about  $126.20 \pm 138.00$ , and the largest offset was 791 iterations. This way, we reduced the total computation time though we may have also reduced the long term effect of mutation in  $\text{BO}_{\text{MA}}$  and  $\text{RG}_{\text{RK}}$ , and perhaps more seriously, the effect of the introduction of mutants in the BRKGAs.

To compile the C/C++ code, we use the GNU `g++` compiler version 4.8.1 and `libstdc++` version 6.0.18. To compile the Java code, we use the Oracle Java 64-Bit JDK runtime environment version 1.7.0\_45. To allow for full memory utilization, we run the Java bytecodes with JVM parameter `-Xmx32g`.

## 5.6 Experimental Results and Discussion

This section presents our experimental results. For **CPLEX** and **CORAL**, we performed one run per instance since both are exact and deterministic algorithms. For the remaining algorithms, we performed 30 independent runs for each instance. One can note that this experimental setup is huge, and in fact it took more than 100 CPU days running over 32 identical machines. Each experiment was conducted individually on one machine to ensure the algorithm had exclusive use of all of the machine's resources. This way, we minimized external effects. All reported times are wall-clock times where we reported only the optimization time excluding the effort to load the instance and log the run.

### 5.6.1 Comparing revenue

To compare the algorithms with respect to revenue, it is necessary to scale the results since each instance can have very different revenue values and even different orders of magnitude. For each instance  $\mathcal{I}$ , let  $\chi_{\mathcal{I}}$  be the set of values of the solutions found for  $\mathcal{I}$ , and  $D_{\mathcal{I}} = \max(\chi_{\mathcal{I}}) - \min(\chi_{\mathcal{I}})$ . The scaling is done by the simple transformation

$$\chi'_{\mathcal{I}} = \begin{cases} (x - \min(\chi_{\mathcal{I}}))/D_{\mathcal{I}} & \forall x \in \chi_{\mathcal{I}} \text{ and } D_{\mathcal{I}} > 0, \\ 1 & \text{otherwise.} \end{cases}$$

where  $\chi'_{\mathcal{I}}$  is the set of scaled values. Note that all values are scaled to the range  $[0, 1]$ .

Using this scaling process, Figure 5.2 shows the distribution of revenues for each algorithm. The box plots show the location of the first quartile, the revenue median and the third quartile. The whiskers extend to the most extreme revenue no more than 1.5 times the length of the box. The dots are the outliers.

Figure 5.2c shows the revenue distribution over all instances. One can note that CORAL presented poor results when compared to the other algorithms. Indeed, this behavior was expected since Mansini and Speranza [140] reported large solution times for instances with 500 bids (items, in their case). This can be clearly observed if we compare the distribution for small instances ( $\leq 400$  bids) in Figure 5.2a with the distribution for large instances ( $\geq 1000$  bids) in Figure 5.2b. CORAL is able to achieve a large range of values on small instances with a median of 0.99, close to the other results. For the large instances, CORAL rarely found a good solution, as shown by its outliers: it obtained only 20 optimal solutions on the large instances. One could argue that these instances are hard to solve by exact algorithms, however we observe that CPLEX does very well on them. CPLEX's distributions are consistently good, with 113 optimal solutions found on large instances. The heuristics presented overall good results. Among them, RG<sub>RK</sub> experienced the worst results, followed by BO<sub>MA</sub>. Our approaches did better than the other algorithms, although among our approaches, it is tricky to determine their relative performance simply examining the box plots. Note that the utilization of the pseudo utility derived from surrogate dual vectors did not, as expected, have a favorable impact on the results since SD<sub>RA</sub> and SD<sub>LP</sub> presented greater variances and lower medians than our other approaches that did not use surrogate duality.

The box plots help shape our intuition that our approaches obtained better results than those of previous methods. To confirm our conclusions, we tested the normality of these distributions using the Shapiro-Wilk test and applied the Mann-Whitney-Wilcoxon U test, considered more effective than the *t*-test for distributions sufficiently far from normal and for sufficiently large sample sizes (Conover [38], Fay and Proschan [57]). For all tests, we assume a confidence interval of 99%. For small, large, and full distributions, the Shapiro-Wilk tests revealed that no revenue distribution fits a normal distribution since the *p*-values for all tests are less than  $2.2 \times 10^{-16}$ . Therefore, we applied the U test which assumes as null hypothesis that the location statistics are equal in both distributions. As several statistical tests were performed, we used a *p*-value correction procedure based on false discovery rate (FDR) to minimize the number of false positives (Type I error) as indicated by Benjamini and Hochberg [18].

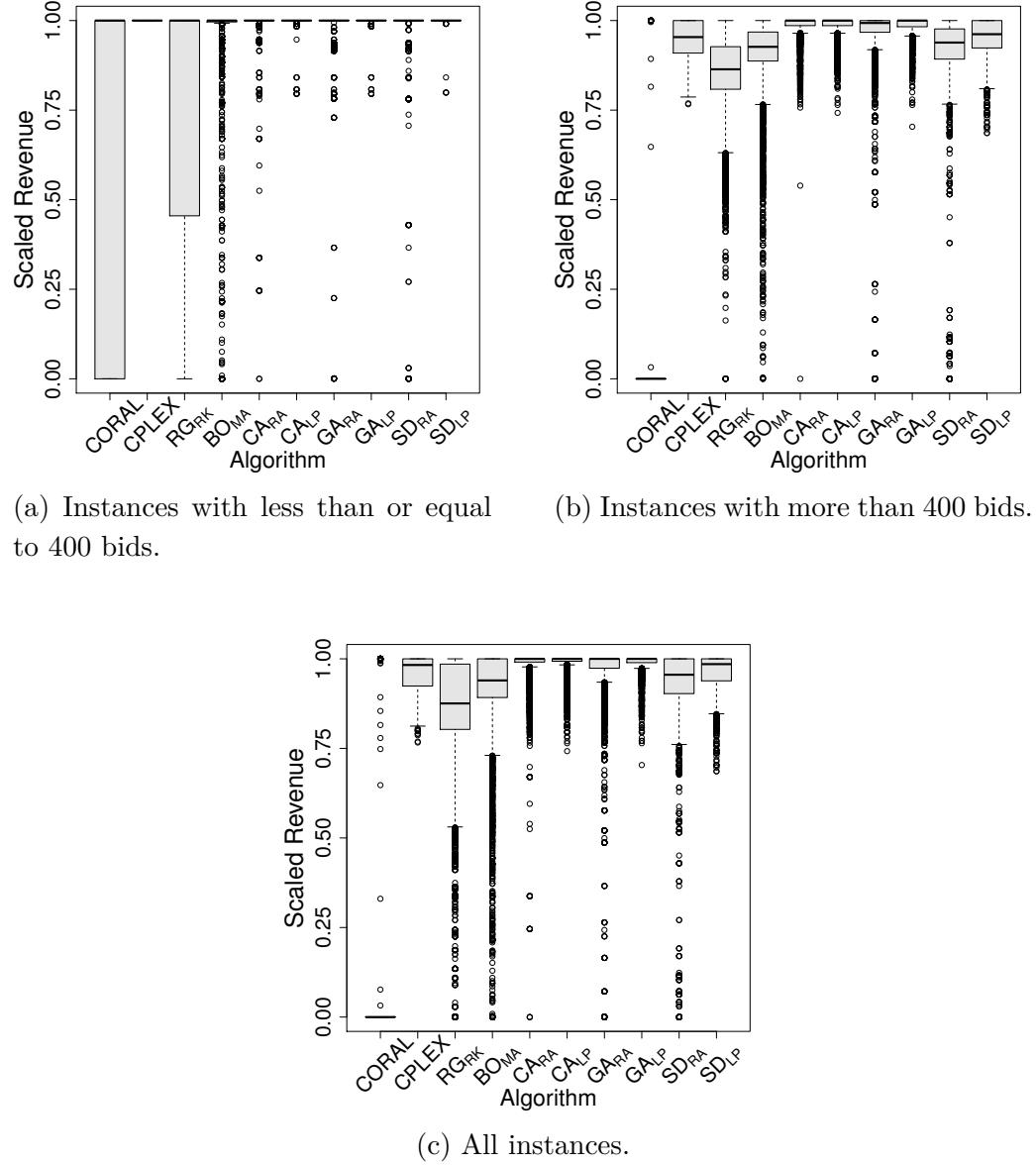


Figure 5.2: Dispersion of revenue for each algorithm.

We tested the results of each pair of algorithms for small instances, large instances, and the full instance dataset.<sup>4</sup> For a confidence level of 99%, almost all comparisons were statistically significant, indicating differences among the results of the different algorithms (almost all  $p$ -values are less than 0.01). CORAL presented significantly poor performance and we believe there are two major reasons for this: first, CORAL cannot handle instances with a large number of bids and goods as previously shown by Mansini and Speranza [140]

<sup>4</sup>The full results are available in the supplementary material in Appendix D.

and confirmed in Figure 5.2b. Second, CORAL takes advantage of the cardinality of each dimension constraint. In MDKP, each constraint  $j$  is limited to be at most equal to  $c_j$  (Formulation (5.2)), where, in general  $c_j \geq 1$ . In single-good WDPs, all constraints have cardinality  $c_j = 1$ , which does not allow CORAL to take advantage of them, limiting its major feature. Only on small instances does CORAL present good results, although worse than the other approaches.

CPLEX produced results that were, in general, better than those of  $\text{RG}_{\text{RK}}$  and  $\text{BO}_{\text{MA}}$ . With respect to the algorithms proposed here, CPLEX was worse except when compared to  $\text{SD}_{\text{RA}}$ . comparison between CPLEX and  $\text{SD}_{\text{LP}}$  was inconclusive ( $p$ -value  $> 0.07$ ). For all small instances, CPLEX obtained an optimal solution. Although the U test returned a value of 0.00 for the differences among the algorithms and CPLEX, we believe that this difference is very small in favor of CPLEX since the other algorithms display more variance than does CPLEX, as shown in Figure 5.2a. Note that for the approaches using LP-based initialization applied to small instances, the tests against CPLEX were inconclusive ( $p$ -values  $> 0.08$ ). We discuss this effect in Section 5.6.5. For large instances, its behavior was similar to that of the general case, where all instances are considered.

The performance of  $\text{RG}_{\text{RK}}$  was worse than that of all heuristics and even CPLEX. This is surprising since this algorithm has been considered to be one of the best heuristics for the MDKP (Pfeiffer and Rothlauf [167]). We argue that, like CORAL,  $\text{RG}_{\text{RK}}$  uses in its favor the cardinality of each constraint since cardinality is correlated with the dual variables which are used to build the pseudo utilities. In the MDKP, this information is very rich since the multiplicity of the demand of each dimension is weighted by its corresponding dual cost. The WDP has unit cardinalities and therefore the pseudo utilities are less effective. Note that even in our BRKGAs, both  $\text{SD}_{\text{RA}}$  and  $\text{SD}_{\text{LP}}$ , which make use of surrogate duality, performed worse than other other variants which did not use this.  $\text{BO}_{\text{MA}}$  performed quite well on the entire set of instances, producing results that are slightly worse than those of our approaches but better than previous algorithms.

Most variants of BRKGA outperformed all other algorithms in the general case. Among these variants, one can only point to small differences. The exception to this observation is  $\text{SD}_{\text{RA}}$ , whose location statistics are slightly lower those of CPLEX. It is surprising that the chromosomal approach (of  $\text{CA}_{\text{RA}}$  and  $\text{CA}_{\text{LP}}$ ), our most basic approach, showed better results than our other variants. We believe that this is so because the greedy and surrogate dual strategies can lead the algorithms to premature convergence to poor local maxima from which they are unable to escape. Note that for small instances, the tests comparing the LP-based initialized algorithms with CPLEX displayed  $p$ -values  $> 0.08$  and, therefore, we cannot reject the hypothesis that these algorithms have similar performance. If we consider only the large instances, the behavior was similar to the general case. In general, the results for  $\text{CA}_{\text{RA}}$  and  $\text{GA}_{\text{LP}}$  are inconclusive since in their test the  $p$ -value  $> 0.29$ .

Table 5.3 reports the performance of the algorithms considering the instances partitioned into two sets. The first column is the name of the algorithm. The first group of columns (2–6) shows the performance considering 202 instances for which an optimal solution was found by CPLEX. There, column “# OPT” represents the number of instances for which the algorithm found an optimal solution; column “% Opt” shows a percentage of the number of optimal solutions found; and column “% Run” shows a percentage of the number of runs on which the algorithm found an optimal solution. The two columns under label “Prod. diff.” show, respectively, the average of the proportional difference between the optimal solution value and the achieved value (%), and its corresponding standard deviation ( $\sigma$ ). As we used the optimum solutions obtained by CPLEX, its entries are presented at the maximum levels (minimum levels, in columns 5–6). CORAL found few optimal solutions (25.74% of the 202 instances for which optimal solutions are known) while the other solutions it produced varied widely with respect to quality. Note that both “% Opt” and “% Run” have the same value since a single run was performed per instance (see Section 5.6, first paragraph). The heuristics performed well, finding more than 60% of the optimal solutions. With the exception of  $SD_{RA}$ , they were never off by more than 4% of the optimal. As expected, the approaches using LP-based initialization often found optimal solutions. However, in some cases they did not reach an optimal solution, suggesting that the relaxation and variable-fixing process not always produced chromosomes that when evolved are decoded into an optimal solution (see Section 5.6.5).

The second group of columns (7–11) of Table 5.3 reports the performance of the algorithms on the 335 instances where no optimal solution is known. It follows the same

Table 5.3: Algorithm performance on instances with known and unknown optimum solutions.

Alg.	Known Optima (202 instances)					Unknown Optima (335 instances)				
	Optima			Prop. diff.		Best			Prop. diff.	
	# Opt	% Opt	% Run	%	$\sigma$	# Best	% Best	% Run	%	$\sigma$
CPLEX	202	100.00	100.00	0.00	0.00	31	9.25	9.25	4.26	2.70
CORAL	52	25.74	25.74	24.49	35.41	3	0.90	0.90	58.36	16.44
RG <sub>RK</sub>	123	60.89	19.93	3.02	3.22	3	0.90	0.56	8.64	3.57
BO <sub>MA</sub>	128	63.37	18.90	2.96	4.00	93	27.76	4.05	4.92	2.76
CA <sub>RA</sub>	171	84.65	24.48	0.95	0.95	200	59.70	26.36	1.00	1.29
CA <sub>LP</sub>	187	92.57	30.33	0.79	0.80	201	60.00	24.51	1.02	1.28
GA <sub>RA</sub>	184	91.09	23.66	2.33	3.68	142	42.39	18.89	1.54	1.55
GA <sub>LP</sub>	186	92.08	28.87	0.89	0.84	200	59.70	23.67	1.14	1.36
SD <sub>RA</sub>	144	71.29	24.24	7.99	14.74	42	12.54	4.02	4.24	2.45
SD <sub>LP</sub>	186	92.08	37.17	1.29	1.09	64	19.10	5.90	3.52	2.23

structure of columns 2–6 but instead of comparing the algorithms with the optimum solution values, we compared them using the best known solutions. CPLEX was able to find a best known solution on 9.25% of the 335 instances, while on the remaining instances it was about 4.26% off of the best values. CORAL and RG<sub>RK</sub> only found three best known solutions while the average gaps of the solutions it found with respect to the best known solution values were about 58.36% and 8.64%, respectively. Again, we emphasize that the pseudo utility approach did not work well since RG<sub>RK</sub>, SD<sub>RA</sub>, and SD<sub>LP</sub> presented the worst results among all heuristics. The LP-based initialization approaches again found the best results (with the exception of SD<sub>LP</sub>), but not as good as the quality of the solutions it found on instances with known optima.

### 5.6.2 Iterations and runtime analyses

Table 5.4 shows the average number of iterations taken by the heuristics to find a best solution. The last iterations without improvement performed in the value of the best solution found are disregarded. The first two columns of this table list, respectively, the instance classes and their corresponding sizes. Each following pair of columns shows the average number of iterations to find a best solution and standard deviation for each algorithm, respectively. For instances with 40 and 80 bids, all algorithms converged very early to an optimal solution. An analysis of CA<sub>RA</sub>, the “most random” of all algorithms, show that all instances having 40 and 80 bids are easy. This is so because all but six runs shows only a single iteration to reach an optimal solution on 40 bids instances (one took three iterations and another five took two). Note that for the LP-based approaches on instances with 40 and 80 bids, the BRKGA framework did not play any role in the optimization since all runs took a single iteration (see Section 5.6.5). All algorithms presented a similar number of iterations in most cases, although those using LP-based initialization required slightly fewer iterations than did the others.

Figure 5.3 shows performance profiles (Dolan and Moré [50]) for all algorithms. In performance profiles, the abscissa shows the time needed to reach a target solution value (in log scale), while the ordinate shows the cumulative probability to reach a target solution value for the given time in the abscissa. Each algorithm is characterized by a different performance profile curve made up of (time, cumulative probability) pairs, one for each execution of the algorithm on a particular instance. Runs that took over 3,600 seconds are not shown in the figure. Therefore, the percentage of runs that concluded within the time limit can be seen as the intersection of the profile with the left side (ordinate) of the figure.

Figure 5.3a shows performance profiles considering only target values of instances for which an optimum solution was found. Figure 5.3b has as target the values of the best

Table 5.4: Average of iterations in finding the best solution. The last iterations without improvement in the best solution found are disregarded.

Class	Size	RG <sub>RK</sub>		BO <sub>MA</sub>		CA <sub>RA</sub>		CA <sub>LP</sub>		
		Iter.	$\sigma$	Iter.	$\sigma$	Iter.	$\sigma$	Iter.	$\sigma$	
CATS	40	2	0.07	2	14.25	1	0.14	1	0.00	
	80	2	0.47	21	95.38	1	0.45	1	0.00	
	200	17	64.24	287	516.44	29	114.36	1	0.00	
	400	64	167.19	348	675.30	109	227.47	30	137.06	
	1000	243	360.07	523	629.76	322	456.50	118	271.27	
	1024	257	345.40	480	561.63	284	448.99	121	282.51	
	2000	232	204.81	435	572.54	651	767.01	220	422.79	
	4000	121	86.07	155	147.78	579	521.75	319	459.41	
	LG	1000	95	195.78	509	515.63	197	311.99	197	317.58
		1500	46	91.45	469	531.82	178	258.24	166	240.78
GA <sub>RA</sub>					GA <sub>LP</sub>		SD <sub>RA</sub>		SD <sub>LP</sub>	
		Iter.	$\sigma$		Iter.	$\sigma$	Iter.	$\sigma$	Iter.	
CATS	40	1	0.55	1	0.00	1	2.65	1	0.00	
	80	4	38.27	1	0.00	2	8.25	1	0.00	
	200	44	142.41	1	0.00	76	182.31	1	0.00	
	400	71	189.38	20	115.19	93	215.54	22	104.42	
	1000	322	511.15	143	361.45	352	447.98	106	256.41	
	1024	308	450.09	127	277.23	287	449.97	100	270.16	
	2000	585	689.49	209	399.79	700	765.56	153	330.10	
	4000	748	745.53	393	567.50	777	643.72	413	637.15	
	LG	1000	173	286.25	194	303.76	262	371.30	186	303.13
		1500	140	231.15	150	224.75	212	280.29	171	248.40

solution found on instances with unknown optimal solution. Finally, Figure 5.3c takes, as target, the values of best solutions found for all instances. The solid black line with white squares shows the performance profile for CPLEX. As previously reported, CPLEX is quite fast to find these optimal solutions: in 82% of runs it required less than one second and in 99% less than 1,000 seconds. Only one run took 1,230 seconds. Since CPLEX spent about one hour on instances with unknown optima and did not find any best solution, it does not appear on Figure 5.3b. In general, CPLEX has the empirical probability of approximately 37% to find a best solution in less than 1,000 seconds. CORAL is represented by the solid black line with filled squares. It found 20% of optimal solutions in less than ten seconds but only 26% in less than 3,600 seconds. For the same reason as CPLEX, CORAL does not appear in Figure 5.3b and has around a 10% probability of finding a best solution in less

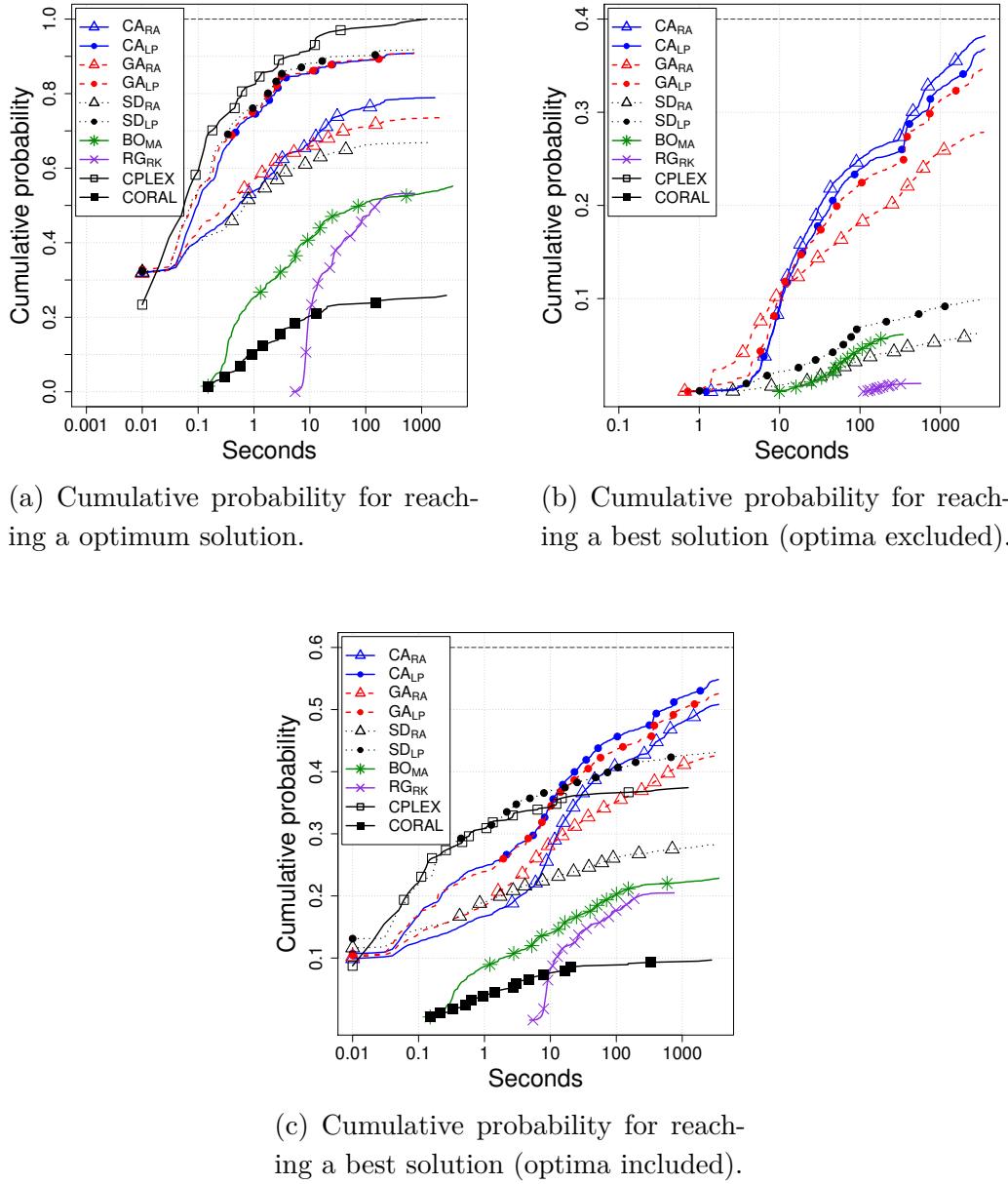


Figure 5.3: Running time distributions to optimal assignment of winner bids. The identification marks correspond to 0.2% of the points plotted for each algorithm.

than 1,000 seconds. BO<sub>MA</sub> (solid green line with asterisks) and RG<sub>RK</sub> (solid purple line with crosses) are slower than the other algorithms (except CORAL) in most cases. Note that BO<sub>MA</sub> has a small advantage over SD<sub>RA</sub> (dense dashed black line with triangles) when we consider Figure 5.3b. RG<sub>RK</sub> and BO<sub>MA</sub> found about 55% of the optimal solutions in less than 3,600 seconds. But, in general, BO<sub>MA</sub> presents a slightly better probability than that of

$\text{RG}_{\text{RK}}$ , as shown in Figure 5.3c. In general, BRKGA variants using LP-based initialization (lines with solid dots) are slower in the first ten seconds, due to the initialization process, but outperformed their corresponding counterparts after this. This fact is due to the time needed to create the first LP-based individuals. In fact, the average time of this procedure is  $50.71 \pm 78.13$  seconds and the maximum time was 1377 seconds. The 377 additional seconds are due to instance setup as a CPLEX model. Considering optimal solutions, CPLEX presents the best time/probability tradeoff. Among the heuristics,  $\text{SD}_{\text{LP}}$  (dense dashed line with solid dots) presents the highest probability (approximately 92%). Considering instances with unknown optima,  $\text{CA}_{\text{RA}}$  (solid blue line with triangles) presents the highest probability (approximately 38%). Overall, the best empirical probability was approximately 55% for  $\text{CA}_{\text{LP}}$  (solid blue line with solid dots).

### 5.6.3 Comparing the heuristics on hard instances

Since the exact methods can only solve to optimality the small or easy instances, the heuristics play a major role in solving the large instances. The following analysis uses the set of LG 1500/1500 instances which proved themselves to be the hardest instances considered here. All algorithms, except  $\text{BO}_{\text{MA}}$ , reached the time limit on most runs and presented a relatively small number of iterations.

Figure 5.4 shows the distributions of revenues considering only the heuristics on the LG 1500/1500 instances. The values were scaled in the same fashion as was done in Section 5.6.1. We also performed the U test for each pair of algorithms at confidence level of 99%.<sup>5</sup> For the pairs of algorithms  $(\text{CA}_{\text{RA}}, \text{CA}_{\text{LP}})$ ,  $(\text{CA}_{\text{LP}}, \text{GA}_{\text{LP}})$ , and  $(\text{SD}_{\text{RA}}, \text{SD}_{\text{LP}})$ , we cannot reject the hypothesis that the results of each pair are similar, since the  $p$ -values obtained from the U tests are greater than 0.01. For the other pairs, the differences presented in Figure 5.4 are statistically significant. Note that the revenues of  $\text{BO}_{\text{MA}}$  are inferior to those of the BRKGAs, which confirms our previous suspicion that  $\text{BO}_{\text{MA}}$  converges prematurely, as shown by its number of iterations in Table 5.4. As in the previous analyses, the algorithms using surrogate duality presented results below those of the other approaches, while  $\text{CA}_{\text{RA}}$ ,  $\text{CA}_{\text{LP}}$ , and  $\text{GA}_{\text{LP}}$  found the best values.

### 5.6.4 Comparing heuristics on small number of generations

Standard genetic algorithms, such as  $\text{RG}_{\text{RK}}$  and  $\text{BO}_{\text{MA}}$ , often converge quickly, i.e., in a small number of generations, to locally optimal, globally sub-optimal, solutions. Most of the diversification in their population is due to the initial population, since during evolution, new individuals are created only by crossover or mutation. Besides creating

---

<sup>5</sup>The full results are available in the supplementary material in Appendix D.

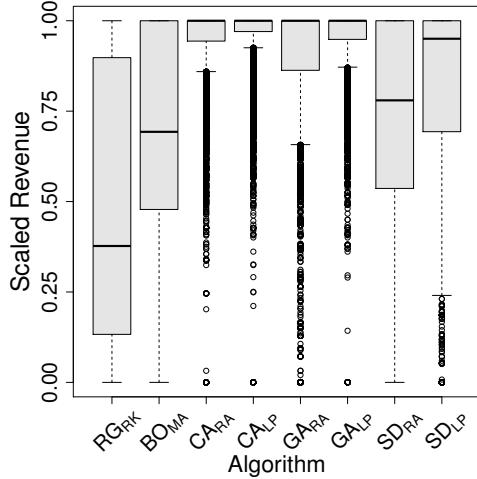


Figure 5.4: Dispersion of revenue for each algorithm on LG 1500/1500 instances.

new individuals by crossover, BRKGAs insert new genetic material into the population at each generation in the form of mutants. This diversification can lead to long runs, i.e., having a large number of generations. Tables 5.3 and 5.4 show us that the BRKGAs and  $\text{BO}_{\text{MA}}$  are able to find better solutions using, systematically, more iterations than  $\text{RG}_{\text{RK}}$ , suggesting that the adopted stopping criterion of 1,000 generations without improvement of the best solution may favor BRKGAs and  $\text{BO}_{\text{MA}}$  over  $\text{RG}_{\text{RK}}$ .

To address this possible bias, we limit ourselves in this section to consider only the best solutions found by the algorithms in their first 100 generations. We chose 100 generations because this value is close to 95.49, the average number of generations taken by  $\text{RG}_{\text{RK}}$  to first find the best solution in a given run. This way, we expect to reduce the impact of inserting new genetic material into the population of a BRKGA. We used the experimental results of previous sections but extracted the values after 100 generations. Note that for large instances, the algorithms were not able to reach the 100<sup>th</sup> generation due to the time limit. This is particularly true on large instances with 4,000 bids (and also on some instances with 2,000 bids). Therefore, these large instances are omitted from the following discussion.

Figure 5.5 shows the boxplots for these results. Their description is similar to Figure 5.2. We also performed U tests for each pair of algorithms using a confidence level of 99%.<sup>5</sup> In general,  $\text{BO}_{\text{MA}}$  outperformed  $\text{RG}_{\text{RK}}$  but not the BRKGAs, as observed in previous sections. The exception here is that  $\text{BO}_{\text{MA}}$  is significantly better than all other algorithms on small instances (Figure 5.5b). In general,  $\text{CA}_{\text{LP}}$  and  $\text{GA}_{\text{LP}}$  presented the best results, although the test between them was inconclusive ( $p$ -value > 0.84). For large instances,  $\text{CA}_{\text{LP}}$  and  $\text{GA}_{\text{LP}}$  reached the best results. As before, we cannot affirm which of the two is

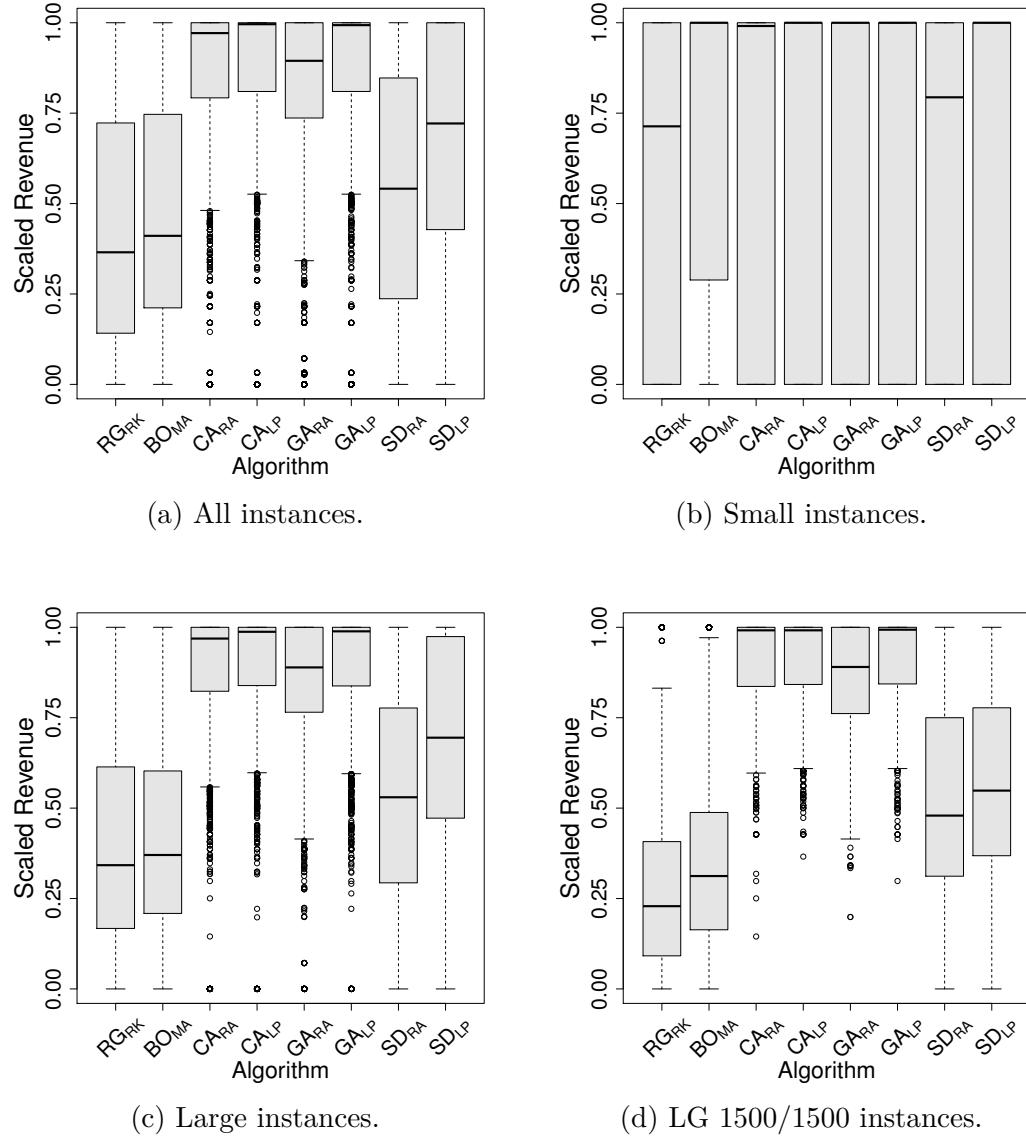


Figure 5.5: Dispersion of revenue for each heuristic using 100 generations at most.

best ( $p$ -value  $> 0.77$ ). For hard LG 1500/1500 instances, the tests for  $\text{CA}_{\text{RA}}$ ,  $\text{CA}_{\text{LP}}$ , and  $\text{GA}_{\text{LP}}$  were inconclusive due  $p$ -values  $> 0.66$ .

We conclude this section by observing that even with a small number of generations, the BRKGAs outperformed the other algorithms (except for  $\text{BO}_{\text{MA}}$  on small instances). Another interesting observation was the performance of the algorithms with LP-initialization, which are able to produce better results than those with random initialization. We discuss this further in the next section.

### 5.6.5 Effect of LP-based initialization

One can notice in the tables of Section 5.6.1 and, with some difficulty in Figure 5.2, that the approaches using LP-based initialization performed better than the approaches that use only random vectors as the initial population. This could suggest that some chromosomes generated by the LP relaxation are often decoded into an optimum solution. In fact, this is not the case, as shown in Table 5.5. This table shows the average ratio  $R_{LP}$  of the revenues of the best chromosome generated by LP relaxations and the best chromosome in the final population, i.e.,  $R_{LP} = \text{best\_lp}/\text{best\_final}$ , for each algorithm and all the instances (hard and easy). The best LP-based chromosome values were obtained with Algorithm 5.2 and do not include any random individuals. Among all LP-based chromosomes, we select the one with the highest revenue. Note that we also consider random initialized algorithms. In this case, we consider the best randomly chromosome from the first generation.

Table 5.5 contains three blocks with respect to the size of the instances and each block has a column labeled “Ratio” that shows the average ratio  $R_{LP}$  and a column labeled  $\sigma$  with the corresponding standard deviation. One can note that on small instances, the LP-based chromosomes generate revenues very close to those in the final population, indicating a possible dominance of LP initial solutions. But note also that the ratio of random initialization and LP-based initialization revenues is very small. This ratio is about 4% larger when we consider large instances exclusively. It is interesting to note that  $\text{GA}_{RA}$  displays a better ratio than that of  $\text{GA}_{LP}$ , implying that the results between the first and last generations of  $\text{GA}_{RA}$  are closer together than the corresponding ones for  $\text{GA}_{LP}$ . However,  $\text{GA}_{LP}$  presented better results than  $\text{GA}_{RA}$  as shown in Section 5.6.1.

Table 5.6 presents the  $R_{LP}$  ratios for the perspective instance class. Each column lists the average ratio of each instance class for each algorithm, with the exception of the last two, which are the overall average  $r$  and standard deviation  $\sigma$ . The results are

Table 5.5: Ratio between the revenue of LP-based chromosomes and the best chromosome.

Alg.	Size $\leq 400$		Size $\geq 1000$		All	
	Ratio	$\sigma$	Ratio	$\sigma$	Ratio	$\sigma$
$\text{CA}_{RA}$	0.9848	0.02	0.9035	0.06	0.9183	0.06
$\text{CA}_{LP}$	0.9999	0.00	0.9471	0.03	0.9574	0.03
$\text{GA}_{RA}$	0.9963	0.00	0.9669	0.02	0.9724	0.02
$\text{GA}_{LP}$	0.9999	0.00	0.9529	0.03	0.9617	0.03
$\text{SD}_{RA}$	0.9931	0.01	0.9295	0.05	0.9427	0.05
$\text{SD}_{LP}$	0.9998	0.00	0.9640	0.03	0.9721	0.03

very close to those of Table 5.5. There are, however, some important points to consider. Note that on the instance class L2, Matching, and Scheduling, the LP-based initialization algorithms achieved a ratio of  $R_{LP} = 1.00$  with a standard deviation of 0.00. This implies that for all instances of these classes, the best solution was found in the first generation. This, in turn, implies that the LP relaxations are able to produce the best solution. This is expected since such instance classes are known to be easy to solve. In fact, CPLEX was able to find the optimal solutions for these instances in the root node of the branch-and-bound tree, i.e. its heuristics were able to find these solutions without enumeration. In these cases, the evolutionary mechanism of BRKGA plays no role in the optimization. However, for other instance classes, BRKGA improves the solution value.

Note that for the CATS instances, the algorithms with LP-based initialization have very tight  $R_{LP}$  ratios, i.e. around 0.99, especially when compared with the random initialization approach (for which they are around 0.95). For LG instances, the  $R_{LP}$  ratios are larger than those for CATS: For both initialization approaches, the ratios are around 0.94. These results were expected since the CATS instances are easier than the LG instances. Again, it is interesting to note that  $GA_{RA}$  presented tighter ratios than  $GA_{LP}$  but only on the LG instances. This is not in contradiction with the results of Table 5.5 since the number of LG instances total more than half of all instances.

Table 5.6: Ratio between the revenue of LP-based chromosomes and the best chromosome by instance class.

Alg	$CA_{RA}$	$CA_{LP}$	$GA_{RA}$	$GA_{LP}$	$SD_{RA}$	$SD_{LP}$	General	
							$r$	$\sigma$
L2	0.99	<b>1.00</b>	0.99	<b>1.00</b>	0.92	<b>1.00</b>	0.99	0.05
L3	0.83	0.99	0.95	0.99	0.94	0.99	0.95	0.08
L4	0.92	0.99	0.98	0.99	0.97	0.99	0.97	0.04
L6	0.88	0.99	0.96	0.99	0.95	0.99	0.96	0.06
L7	0.98	0.99	0.99	0.99	0.97	0.99	0.99	0.02
Arbitrary	0.89	0.96	0.95	0.96	0.93	0.97	0.94	0.05
Matching	0.89	<b>1.00</b>	0.97	<b>1.00</b>	0.96	<b>1.00</b>	0.97	0.06
Paths	0.90	0.99	0.96	0.99	0.95	0.99	0.96	0.05
Regions	0.89	0.97	0.95	0.97	0.93	0.97	0.94	0.06
Scheduling	0.99	<b>1.00</b>	0.99	<b>1.00</b>	0.99	<b>1.00</b>	0.99	0.01
LG	0.91	0.93	0.97	0.94	0.93	0.94	0.94	0.03

## 5.7 Final Remarks

In this chapter, we introduced six variants of biased random-key genetic algorithms applied to the winner determination problem in combinatorial auctions. We also proposed a novel initialization scheme for BRKGAs based on intermediate solutions to the LP relaxation of the integer programming model for the that problem. Such scheme can be easily applied in BRKGAs for other 0–1 integer linear programs, since solutions for the LP relaxation of such problems are natural vectors of random-keys, and thus BRKGA chromosomes. The proposed algorithms have outperformed the standard LP model using a commercial mixed integer programming solver and other recent heuristic approaches on large CATS instances (Leyton-Brown et al. [125]), as well as on the LG instances (Lau and Goh [120]). On small CATS instances, where optimal solutions were found with the commercial mixed integer programming solver but not the BRKGAs, the maximum gap between the two was less than 3%, showing that the BRKGAs can still obtain high-quality solutions. Another advantage of BRKGAs is their ability to find good solutions in a very short time, enabling their application in iterative auctions with thousands of goods and bids.

# Chapter 6

## Concluding remarks

THE capacity of telecommunication networks has been pushed continuously in the last years mainly because of the huge increase in data utilization. One reason is the popularization of streaming services as such high resolution video streaming, known for its huge use of bandwidth. Another reason is the high increase in the number of connected devices, particularly mobile devices such as smart phones and tablets. In this sense, it is necessary to increase the capacity of existing networks and deploy new ones. Provided that this is a multi-billionaire market, optimization is essential since any small improvement can represent a change of millions of dollars and determine the success or failure of the operation. In this thesis, we proposed new optimization algorithms to solve several problems that appear in the planning and deployment of telecommunication networks and related problems. We considered problems of planning wireless access networks and core networks. We also studied two support problems: a clustering problem that can be used to study customer demands; and an auction problem to commercialize electromagnetic spectra.

In Chapter 2, we proposed a new problem called Wireless Backhaul Network Design Problem (WBNDP). The objective of this problem is to provide an access network using wireless technologies such as Wi-Fi and LTE, such that the cost/benefit ratio is the best possible. Although the WBNDP resembles some problems in the family of facility location and Steiner tree problems, it has some particularities that differ it from previous problems in the literature. In the first of these, the objective is a function of the maximum flow that can be routed through the network. Second, the underlying flow problem cannot be reduced, as far as we know, to a classical maximum flow problem which means that we cannot use previously proposed algorithms to solve it. We considered real-world constraints such as radio interference between pieces of equipment, capacities, and geographical location. We proposed a biased random-key genetic algorithm (BRKGA) and a mixed integer linear programming model for the WBNDP and tested them on 30 real-

world instances. Our approaches were able to delivery excellent solutions that resulted in profit margins that, in most cases, were greater than 100%. To date, our algorithms have been used in production environment in one of the largest telecommunication companies of the United States.

In Chapter 3, we proposed a BRKGA and a multi-start algorithm to solve the  $k$ -Interconnected Multi-Depot Multi Traveling Salesman Problem ( $k$ -IMDMTSP). This problem can be used to model core networks that offer certain degrees of resilience. Our algorithms use sophisticated local search procedures in the decoding that are able to deliver good results. We proposed five scenarios and performed experiments with high statistical rigor. We were able to assert the performance of the BRKGA learning process, and provide very good results.

In Chapter 4, we studied the Overlapping Correlation Clustering Problem (OCC). In this problem one wants to find a multi labeling for objects that have a measure of similarity between them. The OCC can be used to perform data analysis such as demand prediction and consumption paths in networks, among several other applications. We proposed four BRKGAs which combine two representations and two decoders. For one type of decoder, we proposed a new local search procedure. For the other type, we used the state of the art algorithm for OCC as a subroutine. We also improved this state of the art algorithm creating a multi-start algorithm from it. We used the algorithms to solve problems from different fields such as Biology, mobility, and text analysis. Our approaches were able to overcome the state of the art algorithm in the most cases and delivery substantially better results. The downside was that the BRKGAs were slower than other approaches because of the long convergence time. As these clustering problems are generally used in exploratory research, the slow convergence, in our opinion, may not be an issue.

In Chapter 5, we studied the Winner Determination Problem (WDP) in combinatorial auctions. In these auctions, the bidder can place bids on bundles of objects that can represent complementarity or substitutability among these objects. The objective is to choose bids such the sum of their values is maximized. Note that one must grant that winner bids are pairwise disjoint. We proposed six BRKGA variants and compare them with a linear programming model solved by a commercial solver, two state of the art algorithms for the WDP (one heuristic and one exact algorithm) and a popular genetic algorithm. We performed experiments on 537 instances of different sizes and levels of difficulty. We were able to statistically show the BRKGAs variants produced the best results using very simple decoders. Moreover, BRKGA produced high quality solutions specially compared with optimal solutions.

During the development of this thesis, we could assert the capability of biased random-key genetic algorithms to delivery excellent solutions for a variety of problems. Even using different types of encoding and several decoding paradigms, BRKGAs presented solid and

reliable results. For example, in Chapter 5, the decoders are simple and have no local search. Nevertheless, the BRKGAs were able to overcome a memetic algorithm with local search procedures in its decoding phase. Some important facts about the BRKGA must be noted. First, the insertion of mutants instead of standard mutation in the genes, allows much more diversification in the population and, consequently, better chances of escape from local minima/maxima. Second, the restart approach is fundamental to obtain good solutions. Third, local search in the decoding process is not always necessary as one can observe in Chapters 2 and 5. Another important point is the choice of the representation and the effects that this choice has in the BRKGA learning process as we can observe in Chapter 4.

Summarizing, our major contributions are:

- The proposition of a new problem called the Wireless Backhaul Network Design Problem (WBNDP) and the methods to solve it. This is a real-world problem that occurs frequently in network design. One may note that it is possible to derive a whole new family of problems from the WBNDP that can be applied in other contexts;
- The proposition of new very effective algorithms for solving the overlapping correlation clustering problem (OCC), the  $k$ -interconnected multi-depot multi traveling salesman problem ( $k$ -IMDMTSP), and the winner determination problem (WDP). Such algorithms were shown statistically, to provide solutions of excellent quality;
- The assertion that the biased random-key genetic algorithm (BRKGA) is an effective tool for solving different kinds of combinatorial optimization problems. BRKGA shows itself as a reliable tool from simple (but hard) problems (e.g. WDP) to complex ones (e.g. WBNDP). The results produced by the BRKGA are usually very stable and can present several alternative solutions. Such solution can be used to better understand the problem and improve modeling and solution process.



# Bibliography

- [1] N. Ailon and E. Liberty. Correlation clustering revisited: The “true” cost of error minimization problems. In S. Albers, A. Marchetti-Spaccamela, Y. Matias, S. Nikoletseas and W. Thomas (editors), *Automata, Languages and Programming*, volume 5555 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, pages 24–36, 2009.
- [2] I. Alaya, C. Solnon and K. Ghédira. Ant algorithm for the multi-dimensional knapsack problem. In *International Conference on Bioinspired Optimization Methods and their Applications (BIOMA 2004)*. Pages 63–72, 2004.
- [3] A. Allahverdi, C. T. Ng, T. C. E. Cheng and M. Y. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985–1032, 2008.
- [4] C. E. Andrade, F. K. Miyazawa and M. G. C. Resende. Evolutionary algorithm for the  $k$ -interconnected multi-depot multi-traveling salesmen problem. In *Proceedings of the 15<sup>th</sup> Annual Conference on Genetic and Evolutionary Computation*, GECCO’13. ACM, New York, NY, USA, pages 463–470, 2013.
- [5] C. E. Andrade, M. G. C. Resende, H. J. Karloff and F. K. Miyazawa. Evolutionary algorithms for overlapping correlation clustering. In *Proceedings of the 16<sup>th</sup> Conference on Genetic and Evolutionary Computation*, GECCO’14. ACM, New York, NY, USA, pages 405–412, 2014<sup>a</sup>.
- [6] C. E. Andrade, M. G. C. Resende, W. Zhang, R. K. Sinha, K. C. Reichmann, R. D. Doverspike and F. K. Miyazawa. A biased random-key genetic algorithm for wireless backhaul network design, 2014<sup>b</sup>. Submitted to Applied Soft Computing.
- [7] C. E. Andrade, R. F. Toso, M. G. C. Resende and F. K. Miyazawa. Biased random-key genetic algorithms for the winner determination problem in combinatorial auctions. *Evolutionary Computation*, 2015. DOI 10.1162/EVCO\_a\_00138. To appear.

- [8] D. L. Applegate, R. E. Bixby, V. Chvátal and W. J. Cook. Concorde TSP solver. <http://www.math.uwaterloo.ca/tsp/concorde>, 2003. Accessed on Oct 12<sup>nd</sup>, 2014.
- [9] D. L. Applegate, R. E. Bixby, V. Chvátal and W. J. Cook. *The traveling salesman problem: A computational study*. Princeton University Press, 2007.
- [10] A. Banerjee, C. Krumpelman, J. Ghosh, S. Basu and R. J. Mooney. Model-based overlapping clustering. In *Proceedings of the 11<sup>th</sup> ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD'05. ACM, New York, NY, USA, pages 532–537, 2005.
- [11] N. Bansal, A. Blum and S. Chawla. Correlation clustering. *Machine Learning*, 56:89–113, 2004.
- [12] S. Barreto, C. Ferreira, J. Paixão and B. S. Santos. Using clustering analysis in a capacitated location-routing problem. *European Journal of Operational Research*, 179(3):968–977, 2007.
- [13] R. Bayer. Symmetric binary B-Trees: Data structure and maintenance algorithms. *Acta Informatica*, 1(4):290–306, 1972.
- [14] J. C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal On Computing*, 2(6):154–160, 1994.
- [15] R. Becker, R. Cáceres, K. Hanson, S. Isaacman, J. M. Loh, M. Martonosi, J. Rowland, S. Urbanek, A. Varshavsky and C. Volinsky. Human Mobility Characterization from Cellular Network Data. *Communications of the ACM*, 56(1):74–82, 2013.
- [16] J.-M. Belenguer, E. Benavent, C. Prins, C. Prodhon and R. W. Calvo. A branch-and-cut method for the capacitated location-routing problem. *Computers & Operations Research*, 38(6):931–941, 2011.
- [17] R. E. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.
- [18] Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society. Series B (Methodological)*, 57(1):289–300, 1995.
- [19] M. Bichler, A. Pikovsky and T. Setzer. An analysis of design problems in combinatorial procurement auctions. *Business & Information Systems Engineering*, 1:111–117, 2009.

- [20] S. Bikhchandani and J. M. Ostroy. From the assignment model to combinatorial auctions. In *Combinatorial auctions*. MIT Press, pages 189–214, 2006.
- [21] M. Birattari, Z. Yuan, P. Balaprakash and T. Stützle. F-Race and iterated F-Race: an overview. In *Experimental methods for the analysis of optimization algorithms*. Springer Berlin Heidelberg, pages 311–336, 2010.
- [22] L. Blumrosen and N. Nisan. Combinatorial auctions. In *Algorithmic game theory*. Cambridge University Press, pages 267–299, 2007.
- [23] F. Bonchi, A. Gionis and A. Ukkonen. Overlapping correlation clustering. *Knowledge and Information Systems*:1–32, 2012.
- [24] O. Borůvkra. O jistém problému minimálním. *Práce mor. přírodověd. spol.*, 3:37–58, 1926.
- [25] D. Boughaci. Metaheuristic approaches for the winner determination problem in combinatorial auction. In X.-S. Yang (editor), *Artificial Intelligence, Evolutionary Computing and Metaheuristics*, volume 427 of *Studies in Computational Intelligence*. Springer Berlin Heidelberg, pages 775–791, 2013.
- [26] D. Boughaci, B. Benhamou and H. Drias. A memetic algorithm for the optimal winner determination problem. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 13:905–917, 2009.
- [27] V. Boyer, D. E. Baz and M. Elkihel. Solution of multidimensional knapsack problems via cooperation of dynamic programming and branch and bound. *European Journal of Industrial Engineering*, 4:434–449, 2010.
- [28] J. Brodkin. Verizon led massive astroturf campaign to end NJ broadband obligation. Ars Technica, 2014. <http://arstechnica.com/tech-policy/2014/04/verizon-led-massive-astroturf-campaign-to-end-nj-broadband-obligation>. Accessed on Sep 6<sup>th</sup>, 2014.
- [29] T. Buer and G. Pankratz. Solving a bi-objective winner determination problem in a transportation procurement auction. *Logistics Research*, 2:65–78, 2010.
- [30] S. A. Canuto, M. G. C. Resende and C. C. Ribeiro. Local search with perturbations for the prize-collecting Steiner tree problem in graphs. *Networks*, 38(1):50–58, 2001.
- [31] R. D. Carr and G. Lancia. Ramsey theory and integrality gap for the independent set problem. *Operations Research Letters*, 42(2):137–139, 2014.

- [32] V. Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, 1985.
- [33] P. Chardaire, G. P. McKeown and J. A. Maki. Application of GRASP to the multiconstraint knapsack problem. In E. J. W. Boers (editor), *Applications of Evolutionary Computing*, volume 2037 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, pages 30–39, 2001.
- [34] M. Charikar, V. Guruswami and A. Wirth. Clustering with qualitative information. *Journal of Computer and System Sciences*, 71(3):360–383, 2005.
- [35] L. Chen, M. T. Özsü and V. Oria. Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD’05. ACM, New York, NY, USA, pages 491–502, 2005.
- [36] P. C. Chu and J. E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4:63–86, 1998.
- [37] Cisco. VNI Mobile forecast highlights 2013–2018. Cisco website, 2014. [http://www.cisco.com/assets/sol/sp/vni/forecast\\_highlights\\_mobile](http://www.cisco.com/assets/sol/sp/vni/forecast_highlights_mobile). Accessed on Oct 23<sup>rd</sup>, 2014.
- [38] W. J. Conover. *Practical nonparametric statistics*. John Wiley & Sons, 2<sup>nd</sup> edition, 1980.
- [39] A. M. Costa, J.-F. Cordeau and G. Laporte. Fast heuristics for the Steiner tree problem with revenues, budget and hop constraints. *European Journal of Operational Research*, 190(1):68–78, 2008.
- [40] A. M. Costa, J.-F. Cordeau and G. Laporte. Models and branch-and-cut algorithms for the Steiner tree problem with revenues, budget and hop constraints. *Networks*, 53(2):141–159, 2009.
- [41] P. Cramton, Y. Shoham and R. Steinberg. *Combinatorial auctions*. MIT Press, 2006.
- [42] A. S. da Cunha, A. Lucena, N. Maculan and M. G. C. Resende. A relax-and-cut algorithm for the prize-collecting Steiner problem in graphs. *Discrete Applied Mathematics*, 157(6):1198–1217, 2009.

- [43] G. Dahl, L. Gouveia and C. Requejo. On formulations and methods for the hop-constrained minimum spanning tree problem. In M. G. Resende and P. M. Pardalos (editors), *Handbook of Optimization in Telecommunications*. Springer US, pages 493–515, 2006.
- [44] G. B. Dantzig. *Linear programming and extensions*. Princeton University Press, Princeton, NJ, 1963.
- [45] R. S. de Camargo, G. de Miranda and A. Løkketangen. A new formulation and an exact approach for the many-to-many hub location-routing problem. *Applied Mathematical Modelling*, 37(12-13):7465–7480, 2013.
- [46] E. D. Demaine, D. Emanuel, A. Fiat and N. Immorlica. Correlation clustering in general weighted graphs. *Theoretical Computer Science*, 361(2–3):172–187, 2006.
- [47] B. Dezső, A. Jüttner and P. Kovács. LEMON - An open source C++ graph template library. *Electronic Notes in Theoretical Computer Science*, 264(5):23–45, 2011.
- [48] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.
- [49] S. Dobzinski, N. Nisan and M. Schapira. Approximation algorithms for combinatorial auctions with complement-free bidders. In *Proceedings of the 37<sup>th</sup> annual ACM Symposium on Theory of Computing*, STOC’05. ACM, New York, NY, USA, pages 610–618, 2005.
- [50] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.
- [51] M. Dorigo, V. Maniezzo and A. Colorni. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 26(1):29–41, 1996.
- [52] M. Dorigo and T. Stützle. *Ant colony optimization*. MIT Press, Cambridge, MA, USA, 2004.
- [53] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972.
- [54] A. Elisseeff and J. Weston. A kernel method for multi-labelled classification. In T. G. Dietterich, S. Becker and Z. Ghahramani (editors), *Advances in Neural Information Processing Systems 14*. Pages 681–687, 2001.

- [55] M. Ericsson, M. G. C. Resende and P. M. Pardalos. A genetic algorithm for the weight setting problem in OSPF routing. *Journal of Combinatorial Optimization*, 6(3):299–333, 2002.
- [56] L. F. Escudero, M. Landete and A. Marín. A branch-and-cut algorithm for the Winner Determination Problem. *Decision Support Systems*, 46(3):649–659, 2009.
- [57] M. P. Fay and M. A. Proschan. Wilcoxon-Mann-Whitney or t-test? On assumptions for hypothesis tests and multiple interpretations of decision rules. *Statistics Surveys*, 4:1–39, 2010.
- [58] U. Feige and J. Vondrák. The submodular welfare problem with demand queries. *Theory of Computing*, 6(1):247–290, 2010.
- [59] T. A. Feo and M. G. C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71, 1989.
- [60] FICO. FICO Xpress Optimization Suite. <http://www.fico.com/en/products/fico-xpress-optimization-suite>, 2014. Accessed on Oct 27<sup>th</sup>, 2014.
- [61] T. Fischer and P. Merz. Embedding a chained Lin-Kernighan algorithm into a distributed algorithm. In K. F. Doerner, M. Gendreau, P. Greistorfer, W. Gutjahr, R. F. Hartl, M. Reimann, R. Sharda and S. Voß (editors), *Metaheuristics*, volume 39 of *Operations Research/Computer Science Interfaces Series*. Springer US, pages 277–295, 2007.
- [62] L. R. Ford Jr. and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.
- [63] M. L. Fredman and D. E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47:424–436, 1993.
- [64] Z.-H. Fu and J.-K. Hao. Breakout local search for the Steiner tree problem with revenue, budget and hop constraints. *European Journal of Operational Research*, 232(1):209–220, 2014.
- [65] M. R. Garey and D. S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. Freeman, San Francisco, 1979.
- [66] M. Gendreau and J.-Y. Potvin. *Handbook of metaheuristics*. Springer Publishing Company, Incorporated, 2<sup>nd</sup> edition, 2010.

- [67] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986.
- [68] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.
- [69] M. Goemans and D. Williamson. The primal dual method for approximation algorithms and its application to network design problems. In D. Hochbaum (editor), *Approximation Algorithms for NP-Hard Problems*. P.W.S. Publishing Co., Boston, MA, USA, pages 144–191, 1996.
- [70] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.
- [71] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *Journal of ACM*, 35(4):921–940, 1988.
- [72] D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Professional, Boston, MA, USA, 1<sup>st</sup> edition, 1989.
- [73] M. K. Goldberg, M. Hayvanovych and M. Magdon-Ismail. Measuring similarity between sets of overlapping clusters. In *IEEE Second International Conference on Social Computing*, (SocialCom’10). IEEE Computer Society, Washington, DC, USA, pages 303–308, 2010.
- [74] R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278, 1958.
- [75] J. F. Gonçalves, J. J. M. Mendes and M. G. C. Resende. A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, 167(1):77–95, 2005.
- [76] J. F. Gonçalves, J. J. M. Mendes and M. G. C. Resende. A genetic algorithm for the resource constrained multi-project scheduling problem. *European Journal of Operational Research*, 189(3):1171–1190, 2008.
- [77] J. F. Gonçalves, J. J. M. Mendes and M. G. C. Resende. A random key based genetic algorithm for the resource constrained project scheduling problems. *Computers and Operations Research*, 36:92–109, 2009.
- [78] J. F. Gonçalves and M. G. C. Resende. An extended Akers graphical method with a biased random-key genetic algorithm for job-shop scheduling. *International Transactions in Operational Research*, 21(2):215–246, 2014.

- [79] J. F. Gonçalves, M. G. C. Resende and J. J. M. Mendes. A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem. *Journal of Heuristics*, 17:467–486, 2011.
- [80] J. F. Gonçalves and J. R. de Almeida. A hybrid genetic algorithm for assembly line balancing. *Journal of Heuristics*, 8(6):629–642, 2002.
- [81] J. F. Gonçalves and M. G. C. Resende. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17:487–525, 2011<sup>a</sup>.
- [82] J. F. Gonçalves and M. G. C. Resende. A parallel multi-population genetic algorithm for a constrained two-dimensional orthogonal packing problem. *Journal of Combinatorial Optimization*, 22:180–201, 2011<sup>b</sup>.
- [83] J. F. Gonçalves and M. G. C. Resende. A parallel multi-population biased random-key genetic algorithm for a container loading problem. *Computers & Operations Research*, 39(2):179–190, 2012.
- [84] J. F. Gonçalves and M. G. C. Resende. A biased random key genetic algorithm for 2D and 3D bin packing problems. *International Journal of Production Economics*, 145(2):500–510, 2013.
- [85] L. Gouveia. Using the Miller-Tucker-Zemlin constraints to formulate a minimal spanning tree problem with hop constraints. *Computers & Operations Research*, 22(9):959–970, 1995.
- [86] L. Gouveia, A. Paias and D. Sharma. Restricted dynamic programming based neighborhoods for the hop-constrained minimum spanning tree problem. *Journal of Heuristics*, 17(1):23–37, 2011<sup>a</sup>.
- [87] L. Gouveia, L. Simonetti and E. Uchoa. Modeling hop-constrained and diameter-constrained minimum spanning tree problems as Steiner tree problems over layered graphs. *Mathematical Programming*, 128(1-2):123–148, 2011<sup>b</sup>.
- [88] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966.
- [89] G. Guennebaud, B. Jacob et al. Eigen v3. <http://eigen.tuxfamily.org>, 2010. Accessed on Dec 29<sup>th</sup>, 2014.
- [90] Y. Guo, A. Lim, B. Rodrigues and Y. Zhu. Heuristics for a bidding problem. *Computers & Operations Research*, 33(8):2179–2188, 2006.

- [91] Gurobi Optimization. Gurobi Optimizer. <http://www.gurobi.com>, 2014. Accessed on Oct 27<sup>th</sup>, 2014.
- [92] M. M. Halldórsson. Approximations of weighted independent set and hereditary subset problems. *Journal of Graph Algorithms and Applications*, 4(1):1–16, 2000.
- [93] P. Hansen, N. Mladenović and J. A. M. Pérez. Variable neighbourhood search: Methods and applications. *Annals of Operations Research*, 175(1):367–407, 2010.
- [94] I. S. Haque, V. S. Pande and W. P. Walters. Anatomy of high-performance 2D similarity calculations. *Journal of Chemical Information and Modeling*, 51(9):2345–2351, 2011.
- [95] K. Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.
- [96] M. Hoefer, T. Kesselheim and B. Vöcking. Approximation algorithms for secondary spectrum auctions. In *Proceedings of the 23<sup>rd</sup> Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA’11. ACM, New York, NY, USA, pages 177–186, 2011.
- [97] R. Holte. Combinatorial auctions, knapsack problems, and hill-climbing search. In E. Stroulia and S. Matwin (editors), *Advances in artificial intelligence*, volume 2056 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, pages 57–66, 2001.
- [98] H. H. Hoos and C. Boutilier. Solving combinatorial auctions using stochastic local search. In *Proceedings of the 17<sup>th</sup> National Conference on Artificial Intelligence and 20<sup>th</sup> Conference on Innovative Applications of Artificial Intelligence*. AAAI Press, pages 22–29, 2000.
- [99] R. Horst, P. M. Pardalos and N. V. Thoai. *Introduction to global optimization*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2<sup>nd</sup> edition, 2000.
- [100] A. S. Householder. Unitary triangularization of a nonsymmetric matrix. *Journal of ACM*, 5(4):339–342, 1958.
- [101] IBM. IBM ILOG CPLEX Optimizer. <http://www.ibm.com/software/commerce/optimization/cplex-optimizer>, 2014. Accessed on Oct 21<sup>st</sup>, 2014.
- [102] P. Jaccard. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bulletin de la Société Vaudoise des Sciences Naturelles*, 37:547–579, 1901.

- [103] T. Jansen. On the analysis of dynamic restart strategies for evolutionary algorithms. In J. J. M. Guervós, P. Adamidis, H. G. Beyer, H. P. Schwefel and J. L. Fernández-Villacañas (editors), *Parallel Problem Solving from Nature - PPSN VII*, volume 2439 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, pages 33–43, 2002.
- [104] D. S. Johnson, M. Minkoff and S. Philips. The prize collecting tree problem: Theory and practice. In *Proceedings of the 11<sup>th</sup> Annual ACM-SIAM Symposium on Discrete Algorithms*. Baltimore, Maryland, USA, Pages 790–769, 1999.
- [105] K. S. Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21, 1972.
- [106] L. V. Kantorovic. Mathematical methods of organizing and planning production. *Management Science*, 6:363–422, 1960.
- [107] D. R. Karger and M. Minkoff. Building Steiner trees with incomplete global knowledge. In *Proceedings of 41<sup>st</sup> Annual Symposium on Foundations of Computer Science*, FOCS’00. IEEE Computer Society, pages 613–623, 2000.
- [108] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.
- [109] J. Kennedy and R. Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks Proceedings*, volume 4. Pages 1942–1948, 1995.
- [110] L. Khachiyan. A polynomial algorithm in linear programming. *Soviet Math.*, 20(1):191–194, 1979.
- [111] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
- [112] G. W. Klau, I. Ljubić, A. Moser, P. Mutzel, P. Neuner, U. Pferschy, G. Raidl and R. Weiskircher. Combining a memetic algorithm with integer programming to solve the prize-collecting Steiner tree problem. In K. Deb (editor), *Genetic and Evolutionary Computation - GECCO 2004*, volume 3102 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, pages 1304–1315, 2004.
- [113] E. Kosman and K. J. Leonard. Similarity coefficients for molecular markers in studies of genetic relationships between individuals for haploid, diploid, and polyploid species. *Molecular Ecology*, 14(2):415–424, 2005.
- [114] V. Krishna. *Auction theory*. Academic Press, 2<sup>nd</sup> edition, 2010.

- [115] J. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7:48–50, 1956.
- [116] M. J. Kuby and R. G. Gray. The hub network design problem with stopovers and feeders: the case of Federal Express. *Transportation Research Part A: Policy and Practice*, 27(1):1–12, 1993.
- [117] M. Labbé, I. Rodríguez-Martín and J. J. Salazar-Gonzalez. A branch-and-cut algorithm for the plant-cycle location problem. *Journal of the Operational Research Society*, 55(5):513–520, 2004.
- [118] R. Lahyani, M. Khemakhem and F. Semet. Rich vehicle routing problems: From a taxonomy to a definition. *European Journal of Operational Research*, 2014. DOI 10.1016/j.ejor.2014.07.048. To appear.
- [119] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 18(3):497–520, 1960.
- [120] H. C. Lau and Y. G. Goh. An intelligent brokering system to support multi-agent web-based 4<sup>th</sup>-party logistics. In *Proceedings of the 14<sup>th</sup> IEEE International Conference on Tools with Artificial Intelligence*, ICTAI '02. IEEE Computer Society, Washington, DC, USA, pages 154–161, 2002.
- [121] S. B. Layeb, I. Hajri and M. Haouari. Solving the Steiner tree problem with revenues, budget and hop constraints to optimality. In *5<sup>th</sup> International Conference on Modeling, Simulation and Applied Optimization (ICMSAO)*. Hammamet, Algeria, Pages 1–4, 2013.
- [122] D. Lehmann, L. I. O'Callaghan and Y. Shoham. Truth revelation in approximately efficient combinatorial auctions. *Journal of ACM*, 49:577–602, 2002.
- [123] J. Leskovec, A. Rajaraman and J. Ullman. *Mining of massive datasets*. Cambridge University Press, New York, NY, USA, 2012.
- [124] D. Levine. Application of a hybrid genetic algorithm to airline crew scheduling. *Computers & Operations Research*, 23(6):547–558, 1996.
- [125] K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden and Y. Shoham. CATS: The Combinatorial Auction Test Suite, 2011. <http://www.cs.ubc.ca/~kevinlb/CATS>. Accessed on Dec 29<sup>th</sup>, 2014.
- [126] K. Leyton-Brown and Y. Shoham. A test suite for combinatorial auctions. In *Combinatorial auctions*. MIT Press, pages 451–478, 2006.

- [127] C.-C. Lin, J.-Y. Lin and Y.-C. Chen. The capacitated  $p$ -hub median problem with integral constraints: An application to a Chinese air cargo network. *Applied Mathematical Modelling*, 36(6):2777–2787, 2012.
- [128] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21(2):498–516, 1973.
- [129] A. Lingas, M. Persson and D. Sledneu. Iterative merging heuristics for correlation clustering. *International Journal of Metaheuristics*, 3:105–117, 2014.
- [130] I. Ljubić and S. Gollowitzer. Layered graph approaches to the hop constrained connected facility location problem. *INFORMS Journal on Computing*, 25(2):256–270, 2013.
- [131] I. Ljubić, R. Weiskircher, U. Pferschy, G. W. Klau, P. Mutzel and M. Fischetti. An algorithmic framework for the exact solution of the prize-collecting Steiner tree problem. *Mathematical Programming*, 105(2-3):427–449, 2006.
- [132] S. P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.
- [133] M. C. Lopes, C. E. Andrade, T. A. Queiroz, M. G. C. Resende and F. K. Miyazawa. Heuristics for a hub location-routing problem. Submitted to *Networks*, 2014<sup>a</sup>. To appear.
- [134] M. C. Lopes, T. A. Queiroz, C. E. Andrade and F. K. Miyazawa. Solving a variant of the (hub) location-routing problem. In Z. Zhang, Z. M. Shen, J. Zhang and R. Zhang (editors), *Proceedings of 2014 International Conference on Logistics, Informatics and Service Science*, LISS 2014. New York, NY, USA, Pages 1–1, 2014<sup>b</sup>.
- [135] M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle and M. Birattari. The `irace` package, iterated race for automatic algorithm configuration. techreport TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium. <http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf>, 2011.
- [136] M. Luby, A. Sinclair and D. Zuckerman. Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, 47(4):173–180, 1993.
- [137] A. Lucena and M. G. C. Resende. Strong lower bounds for the prize collecting Steiner problem in graphs. *Discrete Applied Mathematics*, 141(1-3):277–294, 2004.

- [138] M. L. Lucena, C. E. Andrade, M. G. C. Resende and F. K. Miyazawa. Some extensions of biased random-key genetic algorithms. In *Proceedings of the 46<sup>th</sup> Brazilian Symposium of Operational Research*, XLVI SBPO. Pages 1–12, 2014.
- [139] W. Malik, S. Rathinam and S. Darbha. An approximation algorithm for a symmetric generalized multiple depot, multiple travelling salesman problem. *Operations Research Letters*, 35(6):747–753, 2007.
- [140] R. Mansini and M. G. Speranza. CORAL: An exact algorithm for the multidimensional knapsack problem. *INFORMS Journal on Computing*, 24(3):399–415, 2012.
- [141] S. Martello and P. Toth. *Knapsack problems: Algorithms and computer implementations*. John Wiley & Sons, Inc., New York, NY, USA, 1990.
- [142] O. Martin, S. W. Otto and E. W. Felten. Large-step markov chains for the TSP incorporating local search heuristics. *Operations Research Letters*, 11(4):219–224, 1992.
- [143] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8:3:3–30, 1998.
- [144] C. C. McGeoch. *A guide to experimental algorithmics*. Cambridge University Press, 2012.
- [145] Y. Z. Mehrjerdi and A. Nadizadeh. Using greedy clustering method to solve capacitated location-routing problem with fuzzy demands. *European Journal of Operational Research*, 229(1):75–84, 2013.
- [146] M. T. Melo, S. Nickel and F. Saldanha da Gama. Facility location and supply chain management – A review. *European Journal of Operational Research*, 196(2):401–412, 2009.
- [147] M. Mitchell. *An introduction to genetic algorithms*. MIT Press, Cambridge, MA, USA, 1996.
- [148] M. Mitzenmacher and E. Upfal. *Probability and computing : Randomized algorithms and probabilistic analysis*. Cambridge University Press, New York, NY, USA, 2005.
- [149] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

- [150] R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge University Press, New York, NY, USA, 1995.
- [151] A. G. Murzin et al. SCOP: structural classification of proteins, 2009. <http://scop.mrc-lmb.cam.ac.uk/scop>. Accessed on Oct 16<sup>th</sup>, 2013.
- [152] D. R. Musser. Introspective sorting and selection algorithms. *Software: Practice and Experience*, 27(8):983–993, 1997.
- [153] G. Nagy and S. Salhi. The many-to-many location-routing problem. *Top*, 6(2):261–275, 1998.
- [154] G. Nagy and S. Salhi. Location-routing: Issues, models and methods. *European Journal of Operational Research*, 177(2):649–672, 2007.
- [155] G. L. Nemhauser and L. A. Wolsey. *Integer and combinatorial optimization*. Wiley-Interscience, New York, NY, USA, 1988.
- [156] V.-P. Nguyen, C. Prins and C. Prodhon. Solving the two-echelon location routing problem by a GRASP reinforced by a learning process and path relinking. *European Journal of Operational Research*, 216(1):113–126, 2012.
- [157] N. Nisan. Bidding and allocation in combinatorial auctions. In *Proceedings of the 2<sup>nd</sup> ACM Conference on Electronic Commerce*. ACM, New York, NY, USA, pages 1–12, 2000.
- [158] P. Oberlin, S. Rathinam and S. Darbha. A transformation for a heterogeneous, multiple depot, multiple traveling salesman problem. In *American Control Conference, ACC'09*. Pages 1292–1297, 2009.
- [159] A. E. Olsson. *Particle swarm optimization: Theory, techniques and applications*. Nova Science Publishers, Inc., Commack, NY, USA, 2010.
- [160] J. O'Toole. Mobile apps overtake PC Internet usage in U.S. CNN Money, February 28, 2014. <http://money.cnn.com/2014/02/28/technology/mobile/mobile-apps-internet>. Accessed on Oct 23<sup>rd</sup>, 2014.
- [161] M. W. Padberg. On the facial structure of set packing polyhedra. *Mathematical Programming*, 5:199–215, 1973.
- [162] S. Parsons, J. A. Rodriguez-Aguilar and M. Klein. Auctions and bidding: A guide for computer scientists. *ACM Computing Surveys*, 43:10:1–10:59, 2011.

- [163] P. Pathak and R. Dutta. A survey of network design problems and joint design approaches in wireless mesh networks. *IEEE Communications Surveys Tutorials*, 13(3):396–428, 2011.
- [164] L. L. C. Pedrosa. *Approximation algorithms for facility location problems and other supply chain problems*. PhD thesis, University of Campinas, Campinas, SP, 2014.
- [165] G. Perboli, R. Tadei and D. Vigo. The two-echelon capacitated vehicle routing problem: models and math-based heuristics. *Transportation Science*, 45(3):364–380, 2011.
- [166] A. Pérez-Suárez, J. F. Martínez-Trinidad, J. A. Carrasco-Ochoa and J. E. Medina-Pagola. OClustR: A new graph-based algorithm for overlapping clustering. *Neurocomputing*, 121:234–247, 2013.
- [167] J. Pfeiffer and F. Rothlauf. Analysis of greedy heuristics and weight-coded EAS for multidimensional knapsack problems and multi-unit combinatorial auctions. In *Proceedings of the 9<sup>th</sup> annual Conference on Genetic and Evolutionary Computation*, GECCO ’07. ACM, New York, NY, USA, pages 1529–1529, 2007.
- [168] H. Pirkul. A heuristic solution procedure for the multiconstraint zero-one knapsack problem. *Naval Research Logistics (NRL)*, 34(2):161–172, 1987.
- [169] R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957.
- [170] C. Prins, C. Prodhon, A. Ruiz, P. Soriano and R. W. Calvo. Solving the capacitated location-routing problem by a cooperative Lagrangean relaxation-granular tabu search heuristic. *Transportation Science*, 41(4):470–483, 2007.
- [171] C. Prodhon and C. Prins. A survey of recent research on location-routing problems. *European Journal of Operational Research*, 238(1):1–17, 2014.
- [172] H. J. Prömel and A. Steger. *The Steiner tree problem: A tour through graphs, algorithms, and complexity*. Vieweg Teubner Verlag, 2002.
- [173] J. Puchinger, G. R. Raidl and U. Pferschy. The multidimensional knapsack problem: Structure and algorithms. *INFORMS Journal on Computing*, 22(2):250–265, 2010.
- [174] G. R. Raidl and J. Gottlieb. Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem. *Evolutionary Computation*, 13(4):441–475, 2005.

- [175] F. Rayal. LTE peak capacity explained: How to calculate it? Personal Blog, June 27, 2011. <http://frankrayal.com/2011/06/27/lte-peak-capacity>. Accessed on Sep 26<sup>th</sup>, 2014.
- [176] G. Reinelt. TSPLIB. <http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95>, 2008. Accessed on Dec 29<sup>th</sup>, 2014.
- [177] J. Rennie et al. 20 Newsgroups. <http://qwone.com/~jason/20Newsgroups>, 2008. Accessed on Oct 13<sup>rd</sup>, 2013.
- [178] M. G. C. Resende. Biased random-key genetic algorithms with applications in telecommunications. *TOP*, 20(1):130–153, 2012.
- [179] M. G. C. Resende and C. C. Ribeiro. Greedy randomized adaptive search procedures: advances, hybridizations, and applications. In M. Gendreau and J.-Y. Potvin (editors), *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*. Springer US, pages 283–319, 2010.
- [180] C. C. Ribeiro, I. Rossetti and R. Vallejos. Exploiting run time distributions to compare sequential and parallel stochastic local search algorithms. *Journal of Global Optimization*, 54:405–429, 2012.
- [181] I. Rodríguez-Martín, J. J. Salazar-González and H. Yaman. A branch-and-cut algorithm for the hub location and routing problem. *Computers & Operations Research*, 50(0):161–174, 2014.
- [182] M. H. Rothkopf, A. Pekc and R. M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44(8):1131–1147, 1998.
- [183] M. M. Rowland, P. K. Coe, R. J. Stussy, A. A. Ager, N. J. Cimon, B. K. Johnson and M. J. Wisdom. The Starkey habitat database for ungulate research: construction, documentation, and use. techreport, U.S. Forest Service General Technical Report PNW-GTR-430, Portland, Oregon, USA, 1998.
- [184] F. Samanlioglu, M. B. Kurz, W. G. Ferrell and S. Tangudu. A hybrid random-key genetic algorithm for a symmetric travelling salesman problem. *International Journal of Operational Research*, 2(1):47–63, 2006.
- [185] T. Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135(1–2):1–54, 2002.
- [186] T. Sandholm. Optimal winner determination algorithms. In *Combinatorial auctions*. MIT Press, pages 331–361, 2006.

- [187] M. W. P. Savelsbergh and M. Sol. The general pickup and delivery problem. *Transportation Science*, 29(1):17–29, 1995.
- [188] M. Schwind, T. Stockheim and F. Rothlauf. Optimization heuristics for the combinatorial auction problem. In *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, volume 3. Pages 1588–1595, 2003.
- [189] I. Shamshurin. Data representation in machine learning-based sentiment analysis of customer reviews. In S. O. Kuznetsov, D. P. Mandal, M. K. Kundu and S. K. Pal (editors), *Pattern Recognition and Machine Intelligence*, volume 6744 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, pages 254–260, 2011.
- [190] R. N. Shepard and P. Arabie. Additive clustering: Representation of similarities as combinations of discrete overlapping properties. *Psychological Review*, 86(2):87–123, 1979.
- [191] O. V. Shylo, T. Middelkoop and P. M. Pardalos. Restart strategies in optimization: Parallel and serial cases. *Parallel Computing*, 37(1):60–68, 2011<sup>a</sup>.
- [192] O. V. Shylo, O. A. Prokopyev and J. Rajgopal. On algorithm portfolios and restart strategies. *Operations Research Letters*, 39(1):49–52, 2011<sup>b</sup>.
- [193] L. V. Snyder and M. S. Daskin. A random-key genetic algorithm for the generalized traveling salesman problem. *European Journal of Operational Research*, 174(1):38–53, 2006.
- [194] W. M. Spears and K. A. DeJong. On the virtues of parameterized uniform crossover. In *Proceedings of the 4<sup>th</sup> International Conference on Genetic Algorithms*. Pages 230–236, 1991.
- [195] C. Swamy. Correlation clustering: Maximizing agreements via semidefinite programming. In *Proceedings of the 15<sup>th</sup> Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'04. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pages 526–527, 2004.
- [196] R. F. Toso and M. G. C. Resende. A C++ application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software*, 30(1):81–93, 2015.
- [197] K. Trohidis, G. Tsoumakas, G. Kalliris and I. Vlahavas. Multilabel classification of music into emotions. In *Proceedings of International Conference on Music Information Retrieval*, ISMIR 2008. Pages 325–330, 2008.

- [198] P. H. Vance, C. Barnhart, E. L. Johnson and G. L. Nemhauser. Airline crew scheduling: A new formulation and decomposition algorithm. *Operations Research*, 45:188–200, 1997.
- [199] M. Vasquez and Y. Vimont. Improved results on the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, 165(1):70–81, 2005.
- [200] V. V. Vazirani. *Approximation algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.
- [201] I. Vorontsov, I. Kulakovskiy and V. Makeev. Jaccard index based similarity measure to compare transcription factor binding site models. *Algorithms for Molecular Biology*, 8(1):23, 2013.
- [202] S. Voß. Steiner tree problems in telecommunications. In M. G. Resende and P. M. Pardalos (editors), *Handbook of Optimization in Telecommunications*. Springer US, pages 459–492, 2006.
- [203] N. Wang and J. Li. Restoring: A greedy heuristic approach based on neighborhood for correlation clustering. In H. Motoda, Z. Wu, L. Cao, O. Zaiane, M. Yao and W. Wang (editors), *Advanced Data Mining and Applications*, volume 8346 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, pages 348–359, 2013.
- [204] Z. Wang, C. Lin and C.-K. Chan. Demonstration of a single-fiber self-healing CWDM metro access ring network with unidirectional OADM. *Photonics Technology Letters, IEEE*, 18(1):163–165, 2006.
- [205] G. Wäscher, H. Haußner and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operation Research*, 183:1109–1130, 2007.
- [206] D. Whitley, S. Rana and R. B. Heckendorn. The island model genetic algorithm: On separability, population size and convergence. *Journal of Computing and Information Technology*, 7:33–47, 1998.
- [207] D. P. Williamson and D. B. Shmoys. *The design of approximation algorithms*. Cambridge University Press, New York, NY, USA, 2011.
- [208] P. Winter. Steiner problem in networks: A survey. *Networks*, 17(2):129–167, 1987.
- [209] L. A. Wolsey. *Integer programming*. Wiley-Interscience, New York, NY, USA, 1998.
- [210] S. Yadlapalli, W. Malik, S. Darbha and M. Pachter. A Lagrangian-based algorithm for a multiple depot, multiple travelling salesmen problem. In *American Control Conference, 2007. ACC'07*. Pages 4027–4032, 2007.

- [211] D. Yang, X. Zhang and G. Xue. PROMISE: A framework for truthful and profit maximizing spectrum double auctions. In *INFOCOM, 2014 Proceedings IEEE*. Pages 109–117, 2014.
- [212] V. F. Yu, S.-W. Lin, W. Lee and C.-J. Ting. A simulated annealing heuristic for the capacitated location routing problem. *Computers & Industrial Engineering*, 58(2):288–299, 2010.
- [213] M. H. F. Zarandi, A. Hemmati, S. Davari and I. B. Turksen. Capacitated location-routing problem with time windows under uncertainty. *Knowledge-Based Systems*, 37(0):480–489, 2013.



# Appendix A

## Additional results for Chapter 2

### A.1 Instance details

Table A.1: Characteristics of the instances. The areas were calculated from the convex hull considering all locations and their geodesic characteristics.

Class	Name	Poles	VRADs	Macros	Blocks	Demand (Mbps)	Area (km <sup>2</sup> )
Small	re01	454	17	7	2507	5270.32	61.88
	re02	484	15	1	2739	5762.16	7.61
	re03	630	7	3	293	594.66	77.40
	re04	667	104	0	5064	10653.00	5.75
	re05	670	3	1	494	990.04	134.95
	re06	673	244	15	5130	10822.60	17.91
	re07	687	1	28	4139	8724.86	6.55
	re08	922	41	7	11465	24162.70	16.39
	re09	969	142	29	4694	9805.00	20.95
	re10	1018	48	5	2544	5321.20	9.77
Medium	re11	2045	157	28	25288	53110.20	26.62
	re12	2113	53	9	6678	13993.10	252.26
	re13	2254	139	9	13472	28260.70	24.39
	re14	2288	84	22	28202	59364.20	58.21
	re15	2305	49	8	12519	26320.90	62.44
	re16	2327	93	8	18550	39036.60	8.69
	re17	2345	43	10	13067	27434.90	23.06
	re18	2347	123	24	15007	31411.00	233.10
	re19	2363	86	6	17646	37125.40	4.82
	re20	2423	27	10	22630	47422.30	27.78
Large	re21	5153	106	11	20053	42069.30	128.04
	re22	5228	260	16	24730	51805.40	50.29
	re23	5266	230	20	19356	40472.90	63.64
	re24	5423	163	10	20411	42844.10	23.73
	re25	5544	206	16	24555	51546.70	46.40
	re26	6562	297	45	35376	74382.30	134.03
	re27	6773	264	23	24714	51866.90	55.65
	re28	7606	272	23	23759	49652.50	147.14
	re29	7660	233	17	26948	56437.60	268.09
	re30	8740	390	30	35754	74933.50	411.71

## A.2 Experimental results

Table A.2: Instance characteristics after preprocessing (restricted scenario). The first two columns represents the class and instance names. The next three columns represent, respectively, the original number of poles, the number of poles after the preprocessing, and the reduction percentage. The next three columns represent the same as the last, but with respect to the demand blocks. The three last columns show the graph characteristics, respectively, number of vertices, number of arcs, and graph density given by  $2|A|/(|V| \cdot (|V| - 1))$ .

Class	Name	Poles			Blocks			Graph		
		Orig.	Prep.	% Red.	Orig.	Prep.	% Red.	Vert.	Arcs	Density
Small	re01	454	417	8.15	2507	60	97.41	1356	9949	0.0108
	re02	484	392	19.01	2739	90	96.68	1287	11083	0.0134
	re03	630	296	53.02	293	11	92.15	929	10498	0.0244
	re04	667	656	1.65	5064	91	97.35	2208	18275	0.0075
	re05	670	217	67.61	494	51	88.26	717	6917	0.0269
	re06	673	512	23.92	5130	92	98.05	1927	11525	0.0062
	re07	687	659	4.08	4139	68	97.92	2150	21217	0.0092
	re08	922	828	10.20	11465	262	97.59	2824	30949	0.0078
	re09	969	960	0.93	4694	228	94.95	3348	38649	0.0069
	re10	1018	618	39.29	2544	66	97.05	1994	21117	0.0106
Medium	re11	2045	2033	0.59	25288	922	96.29	7279	103755	0.0039
	re12	2113	1465	30.67	6678	216	96.57	4706	40176	0.0036
	re13	2254	2150	4.61	13472	539	95.90	7171	82290	0.0032
	re14	2288	2254	1.49	28202	799	97.05	7746	99077	0.0033
	re15	2305	1665	27.77	12519	406	96.67	5487	54049	0.0036
	re16	2327	2273	2.32	18550	546	97.01	7493	98855	0.0035
	re17	2345	2131	9.13	13067	501	96.09	6979	73646	0.0030
	re18	2347	1998	14.87	15007	437	96.91	6654	53547	0.0024
	re19	2363	2241	5.16	17646	502	97.08	7344	84962	0.0032
	re20	2423	2313	4.54	22630	1031	95.38	8044	115970	0.0036
Large	re21	5153	4671	9.35	20053	784	95.95	14966	211363	0.0019
	re22	5228	5170	1.11	24730	1093	95.26	16991	249935	0.0017
	re23	5266	5106	3.04	19356	1018	94.62	16651	205176	0.0015
	re24	5423	5350	1.35	20411	771	96.13	17034	224754	0.0015
	re25	5544	5416	2.31	24555	900	96.21	17434	286106	0.0019
	re26	6562	5174	21.15	35376	1140	96.70	17123	126525	0.0004
	re27	6773	6685	1.30	24714	956	96.05	21365	314229	0.0014
	re28	7606	7238	4.84	23759	1195	94.62	23335	307691	0.0011
	re29	7660	7127	6.96	26948	1240	95.21	22958	300128	0.0011
	re30	8740	7903	9.58	35754	1426	95.81	25690	253804	0.0008
Average		13.00			95.96			0.0057		

Table A.3: Instance characteristics after preprocessing (unrestricted scenario). The description is the same of Table A.2.

Class	Name	Poles			Blocks			Graph		
		Orig.	Prep.	% Red.	Orig.	Prep.	% Red.	Vert.	Arcs	Density
Small	re01	454	436	3.96	2507	60	97.41	1413	10224	0.0102
	re02	484	443	8.47	2739	91	96.64	1441	11743	0.0113
	re03	630	598	5.08	293	28	85.67	1854	20619	0.0120
	re04	667	666	0.15	5064	91	97.35	2238	18351	0.0073
	re05	670	406	39.40	494	56	87.04	1290	10517	0.0126
	re06	673	653	2.97	5130	92	98.05	2350	14827	0.0054
	re07	687	682	0.73	4139	68	97.92	2219	21567	0.0088
	re08	922	907	1.63	11465	264	97.58	3063	32427	0.0069
	re09	969	962	0.72	4694	228	94.95	3354	38665	0.0069
	re10	1018	993	2.46	2544	101	95.68	3154	32914	0.0066
Medium	re11	2045	2041	0.20	25288	922	96.29	7303	103943	0.0039
	re12	2113	1724	18.41	6678	226	96.41	5494	44860	0.0030
	re13	2254	2209	2.00	13472	539	95.90	7348	83174	0.0031
	re14	2288	2282	0.26	28202	799	97.05	7830	99499	0.0032
	re15	2305	2158	6.38	12519	412	96.61	6973	63240	0.0026
	re16	2327	2319	0.34	18550	546	97.01	7631	100071	0.0034
	re17	2345	2328	0.72	13067	501	96.09	7570	78601	0.0027
	re18	2347	2177	7.24	15007	438	96.91	7192	56369	0.0022
	re19	2363	2297	2.79	17646	502	97.08	7512	85752	0.0030
	re20	2423	2405	0.74	22630	1031	95.38	8320	118320	0.0034
Large	re21	5153	5131	0.43	20053	804	95.85	16366	223525	0.0017
	re22	5228	5204	0.46	24730	1093	95.26	17093	250388	0.0017
	re23	5266	5213	1.01	19356	1018	94.62	16972	207085	0.0014
	re24	5423	5398	0.46	20411	771	96.13	17178	225471	0.0015
	re25	5544	5507	0.67	24555	901	96.21	17708	287775	0.0018
	re26	6562	6078	7.38	35376	1141	96.70	19836	143535	0.0007
	re27	6773	6736	0.55	24714	956	96.05	21518	315330	0.0014
	re28	7606	7554	0.68	23759	1195	94.62	24283	312417	0.0011
	re29	7660	7538	1.59	26948	1240	95.21	24191	305839	0.0010
	re30	8740	8445	3.38	35754	1430	95.79	27322	261222	0.0007
<b>Average</b>		<b>4.04</b>			<b>95.65</b>			<b>0.0043</b>		

Table A.4: Best results for instances in restricted scenarios ( $H = 2$  and  $\delta_{bh}^+ = 5$ ) considering at most five hours of running time. Column “Best” is in monetary units. Column “Value” is the proportion of the best value and the star (\*) indicates that algorithm reached the best value.. The time is in seconds.

Class	Name	Best	BRKGA		MIP	
			Value	Time	Value	Time
Small	re01	223318.91	96.20	426	*	11535
	re02	276283.54	98.40	206	*	18000
	re03	777.70	*	2	*	718
	re04	503390.32	99.03	811	*	18001
	re05	34260.77	98.20	109	*	11767
	re06	312983.30	94.65	818	*	9776
	re07	408958.12	94.72	1181	*	18001
	re08	1120700.40	93.38	949	*	18003
	re09	969767.30	94.82	2196	*	18002
	re10	208447.26	*	459	97.64	18001
Medium	re11	3337910.41	95.91	17932	*	18013
	re12	763482.03	95.47	2671	*	18004
	re13	2274004.01	98.47	8643	*	18008
	re14	3595743.00	97.03	17729	*	18016
	re15	1003845.55	93.40	2645	*	18006
	re16	2918574.32	96.82	13198	*	18011
	re17	1665169.72	92.51	14950	*	18008
	re18	1012156.38	94.59	5276	*	18008
	re19	3127893.63	98.85	17196	*	18011
	re20	1947590.10	97.80	17413	*	18013
Large	re21	2446582.47	*	17890	53.37	18029
	re22	4672597.84	*	17968	53.58	18037
	re23	3306754.86	*	17841	94.13	18031
	re24	3468301.68	*	17930	44.67	18036
	re25	3523112.09	*	17947	44.46	18048
	re26	3240689.63	85.54	17895	*	18046
	re27	4105130.94	*	17848	47.89	18054
	re28	3333164.09	*	17921	42.66	18058
	re29	2870427.50	*	17952	31.45	18069
	re30	3494309.59	*	17934	32.96	18077

Table A.5: Best results for instances in unrestricted scenarios ( $H = \infty$  and  $\delta_{bh}^+ = \infty$ ). The shown results are from BRKGA since MIP could not generate results for this scenarios. Column “Value” is in monetary units and column “Time” is in seconds.

Class	Name	Value	Time
Small	re01	169616.48	133
	re02	223468.57	1069
	re03	5308.44	298
	re04	435721.68	618
	re05	30915.48	301
	re06	213459.59	1153
	re07	319970.34	2110
	re08	754766.38	3465
	re09	621946.84	3545
	re10	165799.06	661
Medium	re11	920374.03	3588
	re12	418432.29	2041
	re13	888349.09	3591
	re14	948616.36	3585
	re15	471514.34	3564
	re16	981844.03	3588
	re17	601561.67	3592
	re18	498607.85	3591
	re19	1020244.96	3589
	re20	574179.03	3585
Large	re21	563981.69	3571
	re22	793855.12	3564
	re23	767027.56	3572
	re24	785345.52	3568
	re25	800893.55	3561
	re26	627323.55	3540
	re27	832122.19	3552
	re28	629435.90	3546
	re29	495815.35	3538
	re30	594627.10	3515



## Appendix B

# Additional results for Chapter 3

## B.1 Detailed results

The following tables describe the best solution found for each instance in each scenario evaluated. The first column shows the instance name. Note that this name is constituted by a description and the number of vertices in the graph. The second and third columns represent the size of the inner cycle ( $k$ ) and size of outer cycles ( $C$ ), respectively. The fourth column shows the best value found to that instance ( $v_b$ ). The fifth column shows the additional percentage ( $v\%$ ) of the mean on 30 independent runs for the BRKGA with respect to the best solution, which is calculated by the simple equation

$$v\% = \frac{(\bar{v} - v_b)}{v_b} \times 100 \quad (\text{B.1})$$

where  $\bar{v}$  is the mean of the values obtained by BRKGA and  $v_b$  is the best solution value (which is the same of forth column). The sixth column brings the standard deviation of previous mean ( $\sigma_v$ ). Columns seven and eight represents the mean ( $t$ ) and standard deviations ( $\sigma_t$ ) of the running times (presented in seconds). Column nine shows the time to reach the best solution ( $t$  to  $v_b$ ), if BRKGA could do it. Otherwise, a bar is shown. The columns 10–14 have the same description of columns 5–9 but applied to the MSH.

Table B.1: Best results for ST scenario.

Inst.	$k$	$C$	Best	BRKGA					MSH				
				$v\%$	$\sigma_v$	$t$	$\sigma_t$	$t$ to $v_b$	$v\%$	$\sigma_v$	$t$	$\sigma_t$	$t$ to $v_b$
ali535	107	5	1403446.00	0.21	0.09	904	2	900	0.21	0.09	1232	0	—
att48	10	5	16776.00	15.78	6.32	73	28	96	86.94	0.01	35	0	—
att532	107	5	175972.00	0.15	0.08	695	2	681	0.16	0.07	1010	0	—
bayg29	6	5	2223.00	3.75	2.26	17	26	17	29.42	0.00	13	0	—
bays29	6	5	2759.00	4.69	2.44	16	31	29	46.73	2.60	23	35	—
berlin52	11	5	12409.00	8.64	4.36	107	24	107	33.50	1.48	43	0	—
bier127	26	5	310919.00	3.94	3.28	144	1	147	2.99	3.27	218	0	217
brazil58	12	5	43250.00	22.44	9.52	167	34	287	96.48	2.46	113	36	—
brg180	36	5	345320.00	6.83	3.01	492	30	579	21.92	2.02	555	36	—
burma14	3	5	4428.00	0.20	0.46	2	19	2	0.44	0.49	4	34	2
ch130	26	5	20812.00	0.59	0.53	114	2	114	0.57	0.49	163	1	181
ch150	30	5	21111.00	0.47	0.40	131	2	130	0.49	0.42	189	2	188
d657	132	5	322367.00	0.21	0.10	812	0		0.18	0.09	1217	0	1218
dantzig42	9	5	1153.00	6.69	4.04	51	27	73	42.63	0.96	27	0	—
eil101	21	5	1558.00	7.25	1.31	92	22	203	7.14	0.47	125	2	—
eil51	11	5	631.00	15.69	6.67	70	28	58	69.35	0.16	36	16	—
eil76	16	5	944.00	29.98	12.67	148	70	396	44.69	0.73	70	2	—
fri26	6	5	1296.00	6.54	2.72	14	40	13	38.57	2.64	18	33	—
gil262	53	5	10774.00	0.46	0.17	245	1		0.45	0.20	348	0	348
gr120	24	5	23853.00	4.81	1.12	125	25	247	5.13	0.22	172	1	—
gr137	28	5	255489.00	0.11	0.12	138	1	146	0.11	0.12	217	1	216
gr17	4	5	2481.00	2.18	1.95	5	33	10	11.64	3.51	8	29	—
gr202	41	5	127079.00	2.75	1.45	233	1	232	2.37	1.45	372	0	—
gr21	5	5	3870.00	4.59	3.26	9	33	14	27.75	2.60	11	37	—
gr229	46	5	549056.00	0.68	0.31	256	3		0.61	0.30	413	0	412
gr24	5	5	1716.00	1.32	1.34	13	33	21	32.23	2.30	15	32	—
gr431	87	5	747273.00	0.74	0.46	608	2	591	0.59	0.45	971	0	—
gr48	10	5	7555.00	16.64	8.98	69	34	47	96.50	2.20	58	41	—
gr666	134	5	1724806.00	0.32	0.17	975	1	957	0.36	0.20	1345	0	—
gr96	20	5	133823.00	22.20	4.54	114	79	499	29.90	1.77	118	3	—
hk48	10	5	18765.00	9.70	4.87	71	24	104	79.97	1.90	62	33	—
kroA100	20	5	62746.00	0.05	0.10	88	2	88	0.05	0.10	130	1	129
kroA150	30	5	112327.00	0.01	0.01	137	3	137	0.01	0.01	195	2	197
kroA200	40	5	125867.00	0.69	0.16	182	2		0.65	0.24	266	1	265
kroB100	20	5	55864.00	30.58	5.05	113	70	489	32.29	0.08	134	2	—
kroB150	30	5	106388.00	2.12	0.39	137	6	187	2.20	0.06	195	2	—
kroB200	40	5	156552.00	0.13	0.07	183	2	200	0.12	0.08	271	2	301
kroC100	20	5	70945.00	0.42	0.26	88	3	86	0.40	0.22	129	2	128
kroD100	20	5	82294.00	0.19	0.01	92	3		0.18	0.03	131	2	130
kroE100	20	5	52419.00	41.30	5.43	113	83	626	42.74	0.05	133	1	—
lin105	21	5	41678.00	17.82	2.96	158	53	611	18.86	0.79	183	1	—
lin318	64	5	260132.00	0.24	0.20	343	1	341	0.18	0.15	492	0	—
nrwl379	276	5	430369.00	0.11	0.06	2596	0		0.13	0.06	3601	0	3601
pa561	113	5	15471.00	0.18	0.12	618	0	616	0.23	0.12	895	0	895
pr107	22	5	146993.00	56.15	13.44	447	105	2111	68.62	0.13	232	0	—
pr124	25	5	279457.00	17.51	2.81	155	71	757	18.26	0.13	183	2	—
pr136	28	5	374920.00	0.39	0.20	136	1	139	0.37	0.17	190	1	191
pr144	29	5	329380.00	0.15	0.10	150	2		0.19	0.11	196	2	197
pr152	31	5	408929.00	0.11	0.15	262	3	250	0.07	0.06	322	0	—
pr226	46	5	707932.00	0.23	0.13	289	2	294	0.19	0.14	372	1	371
pr264	53	5	411099.00	0.20	0.09	630	1		0.20	0.11	971	0	971
pr299	60	5	303013.00	0.34	0.23	294	1	292	0.40	0.22	442	0	—
pr439	88	5	654763.00	0.21	0.18	601	4	604	0.20	0.13	858	0	—
pr76	16	5	221016.00	20.11	3.70	69	63	281	21.18	0.45	74	2	—
rd100	20	5	22100.00	0.27	0.08	84	2	83	0.29	0.06	120	2	—
rd400	80	5	75434.00	0.17	0.09	405	0		0.16	0.12	577	0	576
si175	35	5	37171.00	8.06	3.80	861	65	3011	24.79	0.39	446	34	—
si535	107	5	119258.00	10.56	2.74	3381	14	3600	22.90	0.33	2141	33	—
st70	14	5	1165.00	29.68	14.80	161	57	389	56.51	0.12	68	0	—
swiss42	9	5	1912.00	15.54	7.44	51	30	62	77.24	2.43	50	32	—
ts225	45	5	635554.00	0.32	0.14	203	1		0.32	0.12	289	2	287
ulysses16	4	4	8873.00	0.00	0.00	4	21	4	2.35	0.88	9	34	—
ulysses22	5	5	8461.00	1.86	1.48	8	27	9	27.37	2.98	12	41	—

Table B.2: Best results for SL scenario.

Inst.	<i>k</i>	<i>C</i>	Best	BRKGA					MSH				
				<i>v</i> %	$\sigma_v$	<i>t</i>	$\sigma_t$	<i>t</i> to $v_b$	<i>v</i> %	$\sigma_v$	<i>t</i>	$\sigma_t$	<i>t</i> to $v_b$
ali535	107	10	478199.00	0.50	0.33	907	3	—	0.43	0.31	1223	0	1223
att48	10	10	13985.00	3.24	1.44	34	22	60	5.78	0.32	30	0	—
att532	107	10	56985.00	0.52	0.25	678	4	651	0.58	0.22	969	0	—
bayg29	6	10	1805.00	3.43	2.07	12	34	9	37.59	0.18	10	0	—
bays29	6	10	2264.00	2.83	2.26	15	28	25	37.32	0.97	10	18	—
berlin52	11	10	9642.00	16.71	7.47	72	66	139	26.03	0.14	40	0	—
bier127	26	10	217101.00	0.62	0.25	143	1	145	0.58	0.19	209	0	—
brazil58	12	10	32668.00	12.02	6.14	180	31	157	114.18	2.52	104	27	—
brg180	36	10	122070.00	16.61	7.67	408	26	449	118.92	4.05	625	35	—
burma14	3	10	3540.00	0.24	1.06	1	22	1	1.35	1.74	3	33	3
ch130	26	10	8858.00	0.80	0.68	98	2	98	0.62	0.80	137	1	136
ch150	30	10	9745.00	0.73	0.85	111	1	110	1.30	1.10	156	1	156
d657	132	10	120426.00	0.66	0.21	784	0	—	0.59	0.27	1093	0	1105
dantzig42	9	10	876.00	8.92	6.13	40	29	46	24.09	1.04	21	0	—
eil101	21	10	905.00	1.05	0.85	74	1	74	1.19	0.83	105	1	105
eil51	11	10	562.00	8.27	7.18	45	26	48	27.09	0.87	29	0	—
eil76	16	10	798.00	2.47	0.94	42	1	—	2.38	0.81	59	1	65
fri26	6	10	1095.00	2.52	1.19	10	27	14	35.77	2.87	15	38	—
gil262	53	10	3948.00	1.37	0.52	211	0	—	1.35	0.59	293	0	293
gr120	24	10	10205.00	1.06	0.66	105	3	102	0.94	0.62	155	0	155
gr137	28	10	118373.00	1.06	0.26	129	2	—	1.06	0.36	199	0	198
gr17	4	10	2155.00	0.01	0.07	3	21	3	9.66	2.62	6	32	—
gr202	41	10	66222.00	1.27	0.37	224	1	—	1.31	0.45	355	0	355
gr21	5	10	3186.00	0.64	1.29	6	31	5	20.97	3.36	10	35	—
gr229	46	10	214706.00	1.61	0.71	252	2	253	1.42	0.50	383	0	—
gr24	5	10	1470.00	1.07	0.61	7	38	17	25.52	2.73	14	31	—
gr431	87	10	269601.00	1.23	0.70	631	3	642	1.68	1.12	962	0	—
gr48	10	10	6245.00	12.03	6.32	55	21	60	90.04	2.54	40	33	—
gr666	134	10	621541.00	0.67	0.42	953	1	961	0.92	0.42	1277	0	—
gr96	20	10	79811.00	1.36	0.65	73	3	71	1.18	0.53	100	1	108
hk48	10	10	14029.00	10.35	4.74	62	25	53	95.55	1.66	52	31	—
kroA100	20	10	31405.00	0.47	0.38	80	1	83	0.24	0.36	112	1	120
kroA150	30	10	43554.00	0.04	0.03	121	1	121	0.05	0.07	168	1	—
kroA200	40	10	48249.00	0.51	0.34	165	1	164	0.39	0.23	228	1	227
kroB100	20	10	40344.00	0.20	0.29	84	2	87	0.22	0.24	119	0	118
kroB150	30	10	37335.00	0.16	0.15	123	1	126	0.15	0.14	170	1	179
kroB200	40	10	67539.00	0.20	0.25	168	2	—	0.19	0.12	236	0	236
kroC100	20	10	31616.00	0.76	0.20	79	4	—	0.82	0.23	111	1	111
kroD100	20	10	35187.00	0.09	0.11	81	2	78	0.15	0.11	114	2	112
kroE100	20	10	32259.00	0.36	0.23	85	2	82	0.41	0.22	115	1	—
lin105	21	10	30369.00	0.34	0.19	130	4	—	0.28	0.14	168	0	167
lin318	64	10	101604.00	0.68	0.48	313	2	313	0.53	0.40	423	0	—
nrwl379	276	10	86506.00	0.85	0.25	2469	0	—	0.71	0.31	3505	0	3505
pa561	113	10	6337.00	0.66	0.29	560	0	—	0.64	0.28	793	0	793
pr107	22	10	92314.00	1.16	0.43	194	9	287	1.22	0.42	214	0	—
pr124	25	10	198997.00	7.39	1.30	127	21	256	7.69	0.14	154	1	—
pr136	28	10	142115.00	0.70	0.48	117	2	126	0.88	0.50	163	1	162
pr144	29	10	92727.00	0.38	0.31	136	2	—	0.32	0.38	165	1	164
pr152	31	10	170696.00	0.45	0.24	243	3	235	0.42	0.30	298	1	297
pr226	46	10	433901.00	0.18	0.04	262	2	—	0.16	0.07	324	0	324
pr264	53	10	170593.00	0.14	0.14	636	1	—	0.08	0.08	940	0	942
pr299	60	10	134869.00	0.46	0.32	284	1	285	0.47	0.38	397	0	—
pr439	88	10	171725.00	0.74	0.65	599	4	593	0.83	0.60	806	0	—
pr76	16	10	165235.00	0.18	0.16	44	1	47	0.16	0.25	69	0	69
rd100	20	10	13735.00	0.07	0.02	73	8	105	0.09	0.05	99	2	—
rd400	80	10	27035.00	0.45	0.24	355	0	357	0.46	0.20	496	0	—
si175	35	10	33659.00	4.50	1.89	418	46	1100	22.00	0.62	338	31	—
si535	107	10	108002.00	1.98	0.92	3196	17	3362	19.80	0.45	2112	27	—
st70	14	10	944.00	1.51	0.40	38	22	72	1.59	0.22	50	2	—
swiss42	9	10	1545.00	3.32	2.77	35	25	36	80.26	1.92	37	35	—
ts225	45	10	226870.00	0.15	0.24	174	1	186	0.26	0.49	240	1	260
ulysses16	4	8	7081.00	0.16	0.25	3	22	3	8.15	3.43	5	30	—
ulysses22	5	10	7392.00	0.23	0.28	9	37	9	25.09	2.95	12	42	—

Table B.3: Best results for LT scenario.

Inst.	$k$	$C$	Best	BRKGA					MSH				
				$v\%$	$\sigma_v$	$t$	$\sigma_t$	$t$ to $v_b$	$v\%$	$\sigma_v$	$t$	$\sigma_t$	$t$ to $v_b$
ali535	268	2	1976741.00	0.08	0.06	1966	1	1998	0.08	0.06	1966	1	1998
att48	24	2	17290.00	9.63	5.89	49	21	55	9.63	5.89	49	21	55
att532	266	2	215450.00	8.93	0.07	1537	1	—	4.52	4.21	2205	0	2217
bayg29	15	2	2251.00	5.67	3.89	6	34	12	49.74	3.88	10	31	—
bays29	15	2	2810.00	5.59	3.36	5	26	7	47.90	4.01	11	30	—
berlin52	26	2	12439.00	13.05	10.73	107	33	137	50.47	2.10	42	0	—
bier127	64	2	388875.00	2.25	1.32	202	1	—	4.35	1.70	315	0	319
brazil58	29	2	39324.00	26.02	9.27	400	30	615	128.60	3.03	239	37	—
brg180	90	2	517910.00	11.26	3.41	737	30	617	15.17	2.58	1047	32	—
burma14	7	2	4552.00	0.18	0.45	1	25	1	1.03	0.84	3	34	3
ch130	65	2	14347.00	63.00	18.61	244	119	813	83.57	2.14	136	1	—
ch150	75	2	16256.00	82.36	13.58	156	127	802	93.70	1.63	140	1	—
d657	329	2	361342.00	3.22	0.09	2579	0	—	1.69	1.50	3602	0	3603
dantzig42	21	2	1181.00	9.48	5.63	24	34	45	64.10	2.27	16	0	—
eil101	51	2	1295.00	40.75	12.06	121	105	331	60.46	6.50	76	0	—
eil51	26	2	700.00	8.80	4.03	26	20	28	77.40	4.90	22	51	—
eil76	38	2	963.00	16.50	13.65	94	54	100	55.59	8.50	37	0	—
fri26	13	2	1429.00	3.04	2.42	8	42	7	33.98	3.62	11	34	—
gil262	131	2	13100.00	0.15	0.09	264	0	—	1.21	1.03	378	0	377
gr120	60	2	14374.00	22.59	25.01	570	44	798	84.97	2.43	139	0	—
gr137	69	2	155397.00	12.91	8.72	1112	27	1532	130.72	2.77	204	0	—
gr17	9	2	2733.00	0.64	0.42	3	28	3	8.59	3.19	6	40	—
gr202	101	2	166493.00	1.48	0.52	387	0	387	1.84	0.44	651	0	—
gr21	11	2	4112.00	5.40	2.57	4	33	4	22.97	3.58	7	37	—
gr229	115	2	443023.00	56.04	9.22	487	116	2426	62.46	1.62	524	0	—
gr24	12	2	1848.00	7.10	3.42	4	31	8	36.20	2.99	8	35	—
gr431	216	2	913594.00	32.36	11.81	1838	57	3600	41.24	0.33	1890	0	—
gr48	24	2	7713.00	10.06	4.82	21	22	25	104.88	2.45	29	37	—
gr666	333	2	2479425.00	0.13	0.05	2750	0	—	1.58	1.43	3601	0	3600
gr96	48	2	105491.00	11.96	4.95	475	28	557	109.88	0.45	95	1	—
hk48	24	2	20223.00	7.20	3.94	34	28	35	80.11	3.11	37	34	—
kroA100	50	2	45063.00	10.21	6.34	435	29	587	116.72	6.38	85	0	—
kroA150	75	2	65467.00	14.37	6.77	827	24	968	147.35	2.17	165	0	—
kroA200	100	2	82026.00	18.62	8.99	1532	21	1472	127.82	2.97	268	0	—
kroB100	50	2	47210.00	6.61	4.73	415	22	478	129.60	4.17	92	0	—
kroB150	75	2	64545.00	10.11	6.30	803	29	1385	118.72	2.25	159	0	—
kroB200	100	2	84458.00	32.75	29.21	1334	47	2119	115.40	3.45	274	0	—
kroC100	50	2	44181.00	9.60	7.09	418	22	365	160.53	5.50	83	8	—
kroD100	50	2	43451.00	12.80	6.91	469	22	613	116.53	0.03	94	0	—
kroE100	50	2	43988.00	29.89	22.42	344	49	407	87.87	0.06	85	2	—
lin105	53	2	32814.00	16.04	12.09	797	34	916	115.93	0.05	146	0	—
lin318	159	2	297594.00	0.24	0.14	542	2	522	0.25	0.09	797	0	—
nrw1379	690	2	554094.00	0.09	0.04	3609	0	3604	0.07	0.03	3609	0	—
pa561	281	2	18754.00	0.17	0.09	1457	0	1449	0.18	0.09	1960	0	1968
pr107	54	2	68663.00	22.03	8.42	3406	9	3600	310.45	0.04	621	0	—
pr124	62	2	154141.00	45.31	31.19	947	74	1230	100.03	0.15	255	0	—
pr136	68	2	204334.00	66.69	23.34	519	110	1069	96.62	0.12	199	0	—
pr144	72	2	312468.00	0.12	0.12	295	4	276	0.13	0.10	376	0	—
pr152	76	2	464736.00	19.86	3.09	1055	45	3600	20.57	0.27	1358	0	—
pr226	113	2	341413.00	82.50	24.08	1472	86	3600	109.27	0.08	933	0	—
pr264	132	2	243113.00	12.74	6.32	3600	0	3600	125.41	0.04	2211	0	—
pr299	150	2	186292.00	37.44	28.83	2549	48	3600	108.51	0.12	576	0	—
pr439	220	2	886444.00	0.12	0.08	1175	2	1216	0.16	0.09	1619	0	—
pr76	38	2	232517.00	30.59	10.33	121	93	424	40.19	0.23	71	0	—
rd100	50	2	16096.00	11.67	6.46	397	24	504	102.83	0.15	85	0	—
rd400	200	2	105561.00	0.18	0.07	699	0	—	0.20	0.09	972	0	973
si175	88	2	38717.00	6.53	5.07	1103	37	1886	40.07	0.55	406	34	—
si535	268	2	125285.00	4.28	2.44	3601	0	3601	36.11	0.56	3577	2	—
st70	35	2	1205.00	12.73	6.57	90	25	80	70.23	0.64	35	0	—
swiss42	21	2	2070.00	13.62	5.54	17	28	23	77.89	2.84	23	33	—
ts225	113	2	761726.00	0.27	0.16	194	2	193	0.31	0.20	292	0	—
ulysses16	8	2	9098.00	0.65	0.45	3	32	3	4.10	1.34	5	33	—
ulysses22	11	2	9420.00	6.93	3.80	9	30	11	22.43	2.83	17	36	—

Table B.4: Best results for LL scenario.

Inst.	<i>k</i>	<i>C</i>	Best	BRKGA					MSH				
				<i>v</i> %	$\sigma_v$	<i>t</i>	$\sigma_t$	<i>t</i> to $v_b$	<i>v</i> %	$\sigma_v$	<i>t</i>	$\sigma_t$	<i>t</i> to $v_b$
ali535	268	4	493588.00	0.37	0.25	2017	1	—	0.35	0.21	2954	0	2954
att48	24	4	14624.00	2.21	0.69	47	21	55	3.24	0.33	40	0	—
att532	266	4	81028.00	0.39	0.18	1633	1	1592	0.39	0.15	2333	0	—
bayg29	15	4	1959.00	7.31	2.81	14	27	17	20.72	0.20	8	0	—
bays29	15	4	2443.00	5.57	2.25	11	31	16	23.96	0.03	8	0	—
berlin52	26	4	10492.00	17.61	6.08	53	57	161	22.33	0.19	48	0	—
bier127	64	4	164116.00	1.27	0.68	217	1	218	1.24	0.55	343	0	—
brazil58	29	4	31719.00	20.72	8.47	539	30	451	131.79	2.69	256	30	—
brg180	90	4	333670.00	8.98	4.37	794	35	540	31.24	2.80	1105	31	—
burma14	7	4	3819.00	1.34	1.47	2	33	1	7.67	0.88	2	19	—
ch130	65	4	9363.00	1.25	0.50	117	2	—	1.06	0.58	168	0	167
ch150	75	4	136200.00	1.18	0.42	124	1	121	1.11	0.53	177	0	—
d657	329	4	102304.00	0.54	0.24	2999	0	2989	0.54	0.22	3602	0	—
dantzig42	21	4	932.00	7.41	2.48	33	18	36	15.67	0.00	21	0	—
eil101	51	4	991.00	1.25	0.61	71	1	71	1.05	0.55	100	0	100
eil51	26	4	579.00	5.82	4.89	51	25	70	34.12	0.32	22	0	—
eil76	38	4	768.00	5.64	1.66	41	22	90	5.49	1.47	51	0	—
fri26	13	4	1157.00	4.66	2.54	12	44	15	43.14	3.40	14	29	—
gil262	131	4	4647.00	0.64	0.39	321	0	315	0.50	0.27	445	0	—
gr120	60	4	10887.00	1.26	0.66	107	2	—	1.07	0.63	168	0	168
gr137	69	4	103672.00	1.17	0.53	150	1	151	1.23	0.52	237	0	—
gr17	9	4	2273.00	2.80	1.76	5	33	7	17.80	3.70	6	35	—
gr202	101	4	73467.00	1.31	0.34	444	1	—	1.12	0.49	706	0	705
gr21	11	4	3256.00	5.20	4.43	6	32	3	34.10	3.71	8	29	—
gr229	115	4	239960.00	1.01	0.54	387	3	389	0.86	0.46	577	0	—
gr24	12	4	1559.00	2.97	1.68	10	35	5	38.05	2.87	10	33	—
gr431	216	4	405363.00	0.54	0.37	1296	1	1286	0.46	0.39	1994	0	—
gr48	24	4	6220.00	11.15	4.09	61	28	63	107.18	3.37	35	31	—
gr666	333	4	658983.00	0.52	0.25	2845	0	—	0.48	0.30	3602	0	3602
gr96	48	4	77422.00	0.38	0.44	79	8	78	0.33	0.50	117	0	117
hk48	24	4	15338.00	12.77	6.39	77	19	65	92.29	2.69	49	30	—
kroA100	50	4	38543.00	0.41	0.30	72	2	—	0.39	0.16	110	0	110
kroA150	75	4	52564.00	0.82	0.46	139	2	134	0.76	0.53	202	0	—
kroA200	100	4	50933.00	0.28	0.26	213	1	215	0.28	0.26	320	0	319
kroB100	50	4	39313.00	0.82	0.42	80	2	80	0.83	0.31	116	0	—
kroB150	75	4	48190.00	0.76	0.33	130	1	131	0.78	0.34	197	0	—
kroB200	100	4	56982.00	0.36	0.32	221	2	216	0.43	0.26	324	0	—
kroC100	50	4	39277.00	0.40	0.25	71	2	72	0.29	0.24	106	0	106
kroD100	50	4	32942.00	0.04	0.08	77	2	80	0.08	0.17	119	0	122
kroE100	50	4	40033.00	0.03	0.07	74	2	72	0.02	0.09	108	0	112
lin105	53	4	29518.00	0.18	0.17	110	3	112	0.14	0.15	170	0	174
lin318	159	4	77281.00	1.22	0.44	605	1	—	1.20	0.53	875	0	875
nrw1379	690	4	114921.00	0.35	0.17	3608	0	3600	0.38	0.15	3608	0	—
pa561	281	4	6793.00	0.83	0.21	1558	0	—	0.72	0.28	2098	0	2099
pr107	54	4	77943.00	0.17	0.12	436	1	437	0.23	0.14	645	0	644
pr124	62	4	84521.00	1.25	0.64	188	3	—	1.12	0.59	284	0	284
pr136	68	4	222771.00	0.39	0.31	150	3	148	0.38	0.36	232	0	—
pr144	72	4	88436.00	0.35	0.29	310	3	313	0.28	0.23	409	0	409
pr152	76	4	123344.00	1.77	1.45	1003	1	—	1.37	1.24	1383	0	1383
pr226	113	4	193582.00	0.57	0.36	673	2	675	0.56	0.29	983	0	—
pr264	132	4	95447.00	0.48	0.24	1492	1	1479	0.41	0.18	2271	0	—
pr299	150	4	90494.00	0.97	0.52	436	1	—	0.71	0.42	647	0	646
pr439	220	4	262339.00	0.51	0.32	1231	2	1211	0.58	0.42	1725	0	—
pr76	38	4	148851.00	0.55	0.53	54	8	—	0.61	0.51	85	0	85
rd100	50	4	14816.00	0.85	0.63	77	2	76	1.03	0.67	110	0	—
rd400	200	4	35972.00	0.56	0.27	777	0	775	0.61	0.28	1072	0	—
si175	88	4	33123.00	10.16	3.50	769	39	1119	32.53	0.52	476	37	—
si535	268	4	101823.00	4.48	2.71	3601	0	3601	34.33	0.35	3573	1	—
st70	35	4	1039.00	19.50	3.13	39	78	204	20.62	0.60	49	0	—
swiss42	21	4	1656.00	12.50	6.51	39	24	36	83.95	2.82	34	29	—
ts225	113	4	241754.00	0.93	0.66	240	2	231	1.10	0.58	350	0	—
ulysses16	8	4	7692.00	1.41	1.06	4	31	5	9.53	2.46	7	35	—
ulysses22	11	4	7654.00	5.21	1.55	11	24	15	6.89	0.20	10	0	—

Table B.5: Best results for SQ scenario.

Inst.	$k$	$C$	Best	BRKGA					MSH				
				$v\%$	$\sigma_v$	$t$	$\sigma_t$	$t$ to $v_b$	$v\%$	$\sigma_v$	$t$	$\sigma_t$	$t$ to $v_b$
ali535	24	24	495241.00	1.49	0.74	1349	0	1352	1.49	0.74	1349	0	1352
att48	7	7	15685.00	7.79	3.77	42	20	48	7.79	3.77	42	20	48
att532	24	24	78677.00	0.65	0.22	1067	1	1059	0.62	0.21	1049	0	—
bayg29	6	6	2086.00	3.38	2.08	16	41	24	19.35	0.15	11	0	—
bays29	6	6	2596.00	3.69	2.54	15	38	16	45.35	0.28	11	9	—
berlin52	8	8	10496.00	5.47	3.06	72	29	70	42.78	0.16	35	0	—
bier127	12	12	191417.00	22.57	7.77	293	99	1268	29.05	0.79	192	0	—
brazil58	8	8	38758.00	7.65	3.39	101	29	183	78.17	1.50	83	37	—
brg180	14	14	5210.00	541.45	51.61	521	64	863	1507.58	4.72	381	31	—
burma14	4	4	4320.00	0.45	0.48	3	28	2	4.63	1.77	6	30	6
ch130	12	12	13119.00	0.27	0.12	101	1	101	0.25	0.11	115	1	—
ch150	13	13	14393.00	14.86	2.53	134	75	679	15.05	0.88	130	1	—
d657	26	26	115829.00	0.35	0.16	834	0	839	0.29	0.13	656	0	—
dantzig42	7	7	1019.00	5.88	3.32	35	21	39	23.23	0.53	23	0	—
eil101	11	11	926.00	1.46	1.27	74	1	74	1.56	1.23	92	0	92
eil51	8	8	548.00	10.24	5.63	52	25	66	58.99	0.84	30	0	—
eil76	9	9	725.00	5.72	3.44	129	24	163	26.18	0.24	54	0	—
fri26	6	6	1211.00	3.26	3.03	12	30	8	43.10	2.75	16	35	—
gil262	17	17	5847.00	0.22	0.16	230	1	232	0.25	0.13	218	0	—
gr120	11	11	11319.00	22.57	9.80	260	87	496	30.93	0.18	141	0	—
gr137	12	12	141900.00	0.50	0.65	149	0	155	0.65	0.68	188	0	188
gr17	5	5	2403.00	1.51	2.26	4	35	8	18.93	3.46	7	24	—
gr202	15	15	84018.00	1.56	0.77	243	5	312	1.32	0.55	309	0	—
gr21	5	5	3870.00	4.39	3.38	7	26	9	29.24	2.48	11	39	—
gr229	16	16	274928.00	0.14	0.16	309	1	309	0.07	0.07	360	0	360
gr24	5	5	1716.00	1.43	1.90	12	32	8	32.70	2.71	15	34	—
gr431	21	21	392557.00	0.65	0.56	1089	1	1082	0.71	0.53	1363	0	—
gr48	7	7	7553.00	8.87	3.58	49	23	40	66.75	1.35	40	28	—
gr666	26	26	716090.00	0.56	0.32	1341	0	—	0.61	0.43	1224	0	1224
gr96	10	10	125671.00	0.67	0.47	73	0	73	0.56	0.40	89	1	88
hk48	7	7	16699.00	7.63	3.92	45	25	50	70.69	2.42	46	37	—
kroA100	10	10	38157.00	36.70	11.21	180	79	489	46.96	0.46	101	0	—
kroA150	13	13	67920.00	0.09	0.05	138	1	139	0.09	0.06	158	0	—
kroA200	15	15	73659.00	0.42	0.42	207	0	205	0.39	0.42	206	1	204
kroB100	10	10	40591.00	26.50	13.86	280	75	751	45.22	0.14	106	1	—
kroB150	13	13	53275.00	35.90	9.34	411	102	1869	47.39	0.26	159	1	—
kroB200	15	15	84398.00	0.21	0.17	201	0	199	0.24	0.24	214	0	213
kroC100	10	10	39666.00	32.66	13.54	276	74	823	51.79	0.45	102	1	—
kroD100	10	10	36779.00	15.54	13.68	408	42	465	51.61	0.16	103	1	—
kroE100	10	10	41415.00	37.66	12.94	197	88	468	49.34	0.69	98	1	—
lin105	11	11	29364.00	34.31	4.88	126	48	406	36.04	0.44	135	0	—
lin318	18	18	117651.00	0.16	0.13	353	0	—	0.14	0.10	292	0	292
nrw1379	38	38	154675.00	0.38	0.24	2494	1	2535	0.49	0.25	1686	0	—
pa561	24	24	6719.00	0.27	0.15	621	0	622	0.28	0.13	644	0	—
pr107	11	11	88417.00	52.33	12.41	265	105	867	63.96	0.03	132	1	—
pr124	12	12	136011.00	0.13	0.20	120	0	122	0.20	0.28	125	1	134
pr136	12	12	188640.00	29.45	11.37	370	79	854	42.57	0.13	128	0	—
pr144	12	12	175679.00	0.01	0.02	132	1	132	0.02	0.03	130	0	136
pr152	13	13	237141.00	0.47	0.18	186	1	—	0.47	0.16	188	0	188
pr226	16	16	306104.00	43.43	6.08	492	125	3600	46.52	0.09	200	1	—
pr264	17	17	246782.00	0.20	0.11	633	1	—	0.16	0.09	698	0	698
pr299	18	18	170702.00	0.33	0.08	463	0	—	0.34	0.12	450	0	450
pr439	21	21	321399.00	0.15	0.12	756	0	757	0.19	0.14	679	0	—
pr76	9	9	163369.00	23.38	15.77	167	59	302	50.76	0.08	59	1	—
rd100	10	10	13302.00	22.91	10.87	174	63	311	34.42	1.03	86	0	—
rd400	20	20	37937.00	0.39	0.16	408	0	411	0.44	0.16	352	0	—
si175	14	14	28666.00	6.22	3.86	737	40	1236	23.91	0.26	272	29	—
si535	24	24	82273.00	3.22	1.07	3278	13	3601	9.73	0.05	1788	0	—
st70	9	9	1019.00	6.40	3.10	126	23	94	43.77	0.55	49	0	—
swiss42	7	7	1810.00	6.70	3.97	34	29	51	64.22	2.64	36	30	—
ts225	15	15	350432.00	0.37	0.32	176	0	175	0.32	0.30	181	1	193
ulysses16	4	4	8873.00	0.00	0.01	4	24	6	2.58	0.76	7	26	—
ulysses22	5	5	8533.00	1.57	1.55	9	35	7	26.69	2.46	11	35	—

## Appendix C

# Additional results for Chapter 4

## C.1 Additional plots of Section 4.4.4

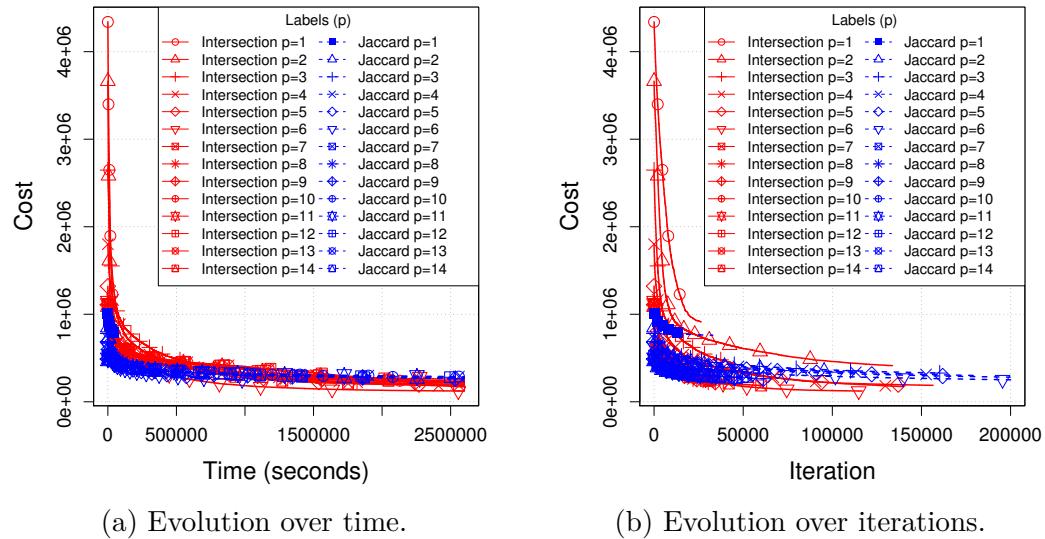


Figure C.1: Evolution of the cost for the YEAST dataset. Only 1% of the symbols corresponding to solutions are shown.

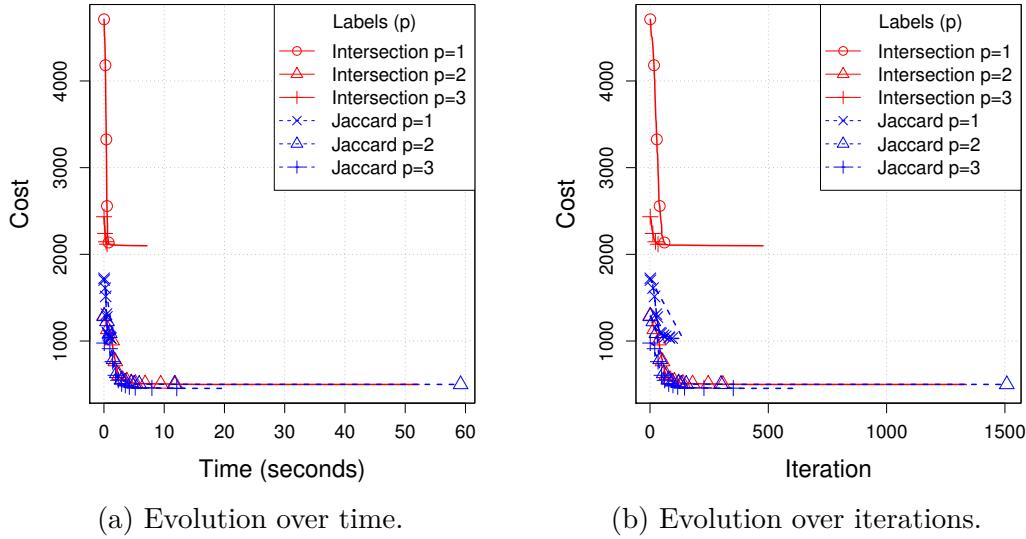


Figure C.2: Evolution of the cost for the Starkey project dataset. Only 1% of the symbols corresponding to solutions are shown.

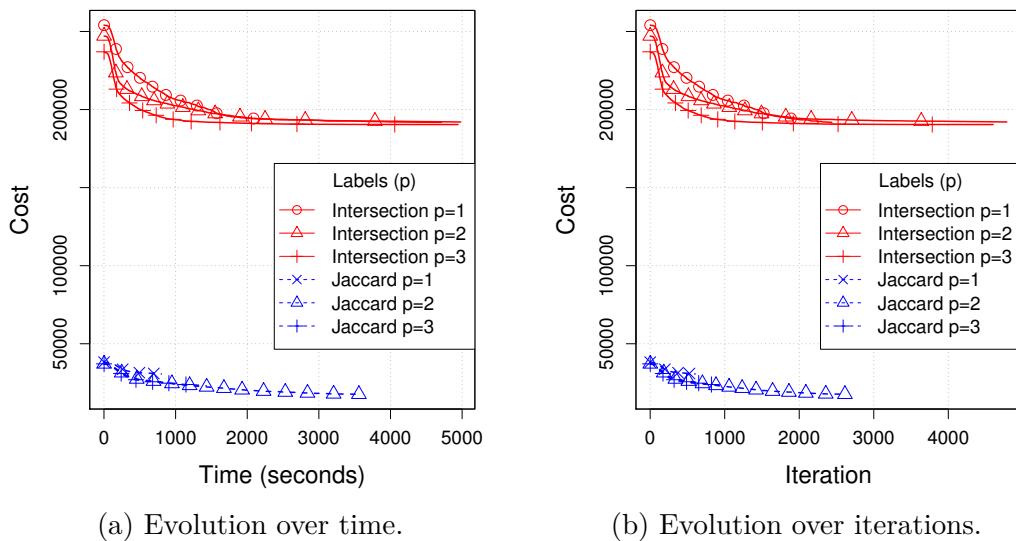


Figure C.3: Evolution of the cost for the protein alignment dataset 1. Only 1% of the symbols corresponding to solutions are shown.

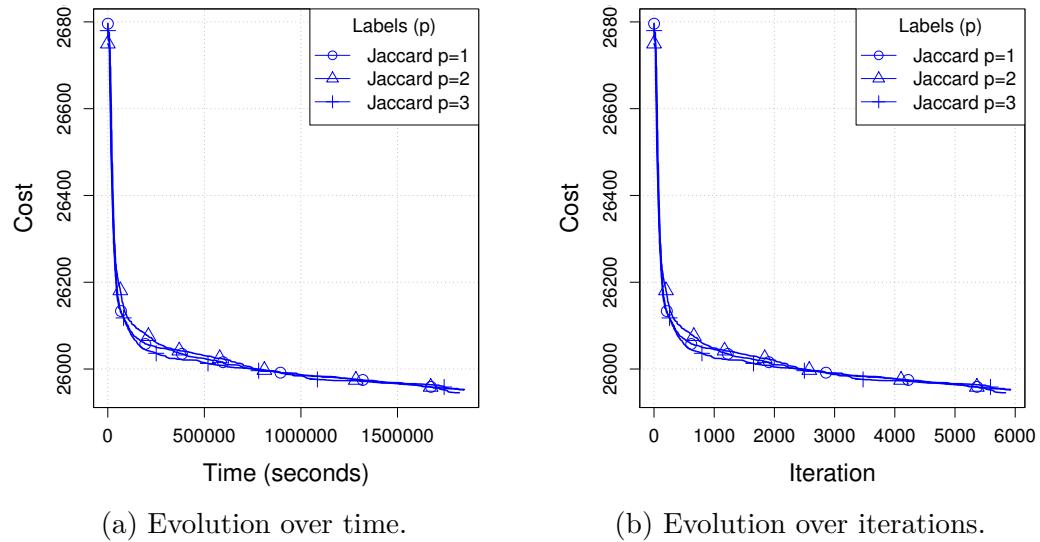


Figure C.4: Evolution of the cost for the newsgroup messages. Only 1% of the symbols corresponding to solutions are shown. We omitted the curves for set intersection since it obtained results 20 times worse than with the Jaccard index but with convergence about 400 times faster. This may indicate that set intersection is not appropriate for clustering in this type of instance.

## C.2 Statistical test tables of Section 4.4.6

Table C.1: Difference in median location for cost distributions for Starkey dataset using Wilcoxon-Mann-Whitney U test with 95% of confidence. The bottom-left block shows  $p$ -values that are greater than 0.05. A negative value means that the median of the “line” algorithm is smaller/better than the “column” algorithm. A dash (—) indicates that the results for that pair of algorithms are identical.

$\mathcal{H}$	$p$	Algorithm	Bonchi	OLS-Comp	OLS-Ext	BLS-Comp	BLS-Ext
Jaccard	1	Bonchi	<b>0.61</b>	-0.27	-0.27	0.60	0.61
		OLS-Comp		<b>0.89</b>	0.00	0.87	0.88
		OLS-Ext		0.16	<b>0.89</b>	0.87	0.88
		BLS-Comp				<b>0.01</b>	0.01
		BLS-Ext					<b>0.003</b>
Set-intersection	2	Bonchi	<b>0.98</b>	0.80	0.73	0.96	0.97
		OLS-Comp		<b>0.17</b>	-0.07	0.15	0.16
		OLS-Ext			<b>0.24</b>	0.22	0.23
		BLS-Comp				<b>0.01</b>	0.008
		BLS-Ext					<b>0.009</b>
Jaccard	3	Bonchi	<b>0.98</b>	0.91	0.94	0.96	0.96
		OLS-Comp		<b>0.06</b>	0.02	0.04	0.04
		OLS-Ext			<b>0.03</b>	0.01	0.01
		BLS-Comp				<b>0.01</b>	-0.001
		BLS-Ext				0.67	<b>0.01</b>
Set-intersection	1	Bonchi	<b>0.85</b>	0.85	0.85	—	—
		OLS-Comp		<b>0.00</b>	0.00	-0.85	-0.85
		OLS-Ext			<b>0.00</b>	-0.85	-0.85
		BLS-Comp				<b>0.85</b>	—
		BLS-Ext					<b>0.85</b>
Jaccard	2	Bonchi	<b>0.40</b>	0.40	-0.04	—	—
		OLS-Comp		<b>0.003</b>	-0.44	-0.40	-0.40
		OLS-Ext			<b>0.44</b>	0.04	0.04
		BLS-Comp				<b>0.40</b>	—
		BLS-Ext					<b>0.04</b>
Set-intersection	3	Bonchi	<b>0.62</b>	0.59	-0.07	—	—
		OLS-Comp		<b>0.03</b>	-0.65	-0.59	-0.59
		OLS-Ext	0.06		<b>0.67</b>	0.07	0.07
		BLS-Comp			0.06	<b>0.62</b>	—
		BLS-Ext					<b>0.62</b>

Table C.2: Difference in median location for cost distributions for SCOP datasets using Wilcoxon-Mann-Whitney U test with 95% of confidence. The bottom-left block shows  $p$ -values that are greater than 0.05. A negative value means that the median of the “line” algorithm is smaller/better than the “column” algorithm. A dash (—) indicates that the results for that pair of algorithms are identical.

$\mathcal{H}$	$p$	Algorithm	Bonchi	OLS-Comp	OLS-Ext	BLS-Comp	BLS-Ext
Jaccard	1	Bonchi	<b>0.96</b>	-0.03	-0.03	0.70	0.95
		OLS-Comp		<b>1.00</b>	—	0.75	0.99
		OLS-Ext			<b>1.00</b>	0.75	0.99
		BLS-Comp				<b>0.24</b>	0.23
		BLS-Ext					<b>0.00</b>
Jaccard	2	Bonchi	<b>0.99</b>	0.97	0.98	0.19	0.92
		OLS-Comp		<b>0.01</b>	0.001	-0.77	-0.02
		OLS-Ext	0.62		<b>0.00</b>	-0.74	-0.00
		BLS-Comp				<b>0.79</b>	0.73
		BLS-Ext	0.28		0.32		<b>0.04</b>
Jaccard	3	Bonchi	<b>0.98</b>	0.85	0.97	0.00002	0.40
		OLS-Comp		<b>0.07</b>	0.07	-0.85	-0.43
		OLS-Ext			<b>0.0003</b>	-0.95	-0.46
		BLS-Comp	0.92			<b>0.95</b>	0.42
		BLS-Ext					<b>0.48</b>
Set-intersection	1	Bonchi	<b>1.00</b>	-0.00001	-0.00002	-0.00002	-0.00002
		OLS-Comp	0.52	<b>1.00</b>	0.00003	0.00003	0.00003
		OLS-Ext	0.65		<b>1.00</b>	—	—
		BLS-Comp	0.65			<b>1.00</b>	—
		BLS-Ext	0.65				<b>1.00</b>
Set-intersection	2	Bonchi	<b>1.00</b>	-0.00002	-0.00002	-0.00002	-0.00002
		OLS-Comp		<b>1.00</b>	-0.00003	—	—
		OLS-Ext	0.34	0.48	<b>1.00</b>	0.0000004	0.0000004
		BLS-Comp			0.48	<b>1.00</b>	—
		BLS-Ext			0.48		<b>1.00</b>
Set	3	Bonchi	<b>1.00</b>	-0.00002	-0.00001	-0.00002	-0.00002
		OLS-Comp		<b>1.00</b>	-0.00003	—	—
		OLS-Ext	0.52	0.65	<b>1.00</b>	0.00003	0.00003
		BLS-Comp			0.65	<b>1.00</b>	—
		BLS-Ext			0.65		<b>1.00</b>

Table C.3: Difference in median location for cost distributions for newsgroup messages using Wilcoxon-Mann-Whitney U test with 95% of confidence. The bottom-left block shows  $p$ -values that are greater than 0.05. A negative value means that the median of the “line” algorithm is smaller/better than the “column” algorithm. A dash (—) indicates that the results for that pair of algorithms are identical.

$\mathcal{H}$	$p$	Algorithm	Bonchi	OLS-Comp	OLS-Ext	BLS-Comp	BLS-Ext
Jaccard	1	Bonchi	<b>0.001</b>	-0.52	-0.92	-0.46	-0.51
		OLS-Comp		<b>0.52</b>	-0.40	0.05	0.006
		OLS-Ext			<b>0.92</b>	0.45	0.44
		BLS-Comp				<b>0.46</b>	-0.04
		BLS-Ext		0.70			<b>0.51</b>
Jaccard	2	Bonchi	<b>0.99</b>	0.99	0.90	0.002	0.75
		OLS-Comp		<b>0.00</b>	-0.09	-0.99	-0.23
		OLS-Ext			<b>0.09</b>	-0.89	-0.14
		BLS-Comp	0.70			<b>0.99</b>	0.75
		BLS-Ext					<b>0.23</b>
Jaccard	3	Bonchi	<b>0.98</b>	0.98	0.94	-0.01	0.77
		OLS-Comp		<b>0.0002</b>	-0.04	-0.99	-0.21
		OLS-Ext			<b>0.04</b>	-0.95	-0.16
		BLS-Comp	0.10			<b>0.99</b>	0.78
		BLS-Ext					<b>0.21</b>
Set-intersection	1	Bonchi	<b>0.000007</b>	-0.99	-0.99	-0.001	-0.001
		OLS-Comp		<b>0.99</b>	-0.005	0.992	0.99
		OLS-Ext			<b>0.99</b>	0.998	0.99
		BLS-Comp	0.10			<b>0.001</b>	-0.0001
		BLS-Ext	0.10			0.70	<b>0.001</b>
Set-intersection	2	Bonchi	<b>0.00001</b>	-0.98	-0.99	-0.001	-0.001
		OLS-Comp		<b>0.98</b>	-0.01	0.98	0.98
		OLS-Ext			<b>0.99</b>	0.99	0.99
		BLS-Comp	0.10			<b>0.001</b>	-0.0003
		BLS-Ext	0.10			0.10	<b>0.001</b>
Set-intersection	3	Bonchi	<b>0.00003</b>	-0.99	-0.99	-0.001	-0.001
		OLS-Comp		<b>0.99</b>	-0.007	0.99	0.99
		OLS-Ext			<b>0.99</b>	0.99	0.99
		BLS-Comp	0.01			<b>0.001</b>	-0.000
		BLS-Ext	0.01			0.10	<b>0.001</b>

## Appendix D

# Additional results for Chapter 5

## D.1 Statistical tests

Tables D.1–D.8, show U test results for each pair of algorithms and different instance sizes, at a 99% confidence level. The structure of these tables is as follows: Each row and column is indexed by one algorithm. Each element in the diagonal (bold) is the median of the corresponding algorithm. The upper-right diagonal elements are the differences in location statistics for each pair of algorithms. A positive difference indicates that the “row algorithm” has its location statistics higher (better) than the “column algorithm”, and the negative difference is the opposite. The bottom-left diagonal elements are the  $p$ -values of each test. We omitted all  $p < 0.01$  values, that indicate that the difference is statistically significant for those pairs. We also omitted confidence intervals since for all tests the values lie in these intervals and they are very narrow. For instance, in Table D.1 we can see that the location statistics for **CPLEX** (2<sup>nd</sup> line) are higher (better) than for **RG<sub>RK</sub>** (4<sup>th</sup> column) since the value 0.0806 is positive. Since the  $p$ -value for this pair was omitted (3<sup>rd</sup> line, 3<sup>rd</sup> column), the table indicates that **CPLEX** performed significantly better than **RG<sub>RK</sub>** in these tests. We chose to display a large number of significant digits since for some pairs of algorithms the differences are very small they are still statistically significant. This is the case, for example, of algorithms **GA<sub>RA</sub>** and **SD<sub>LP</sub>** in Table D.1 where the difference is only 0.000009 but is still significant (in terms of the U test) in favor of **GA<sub>RA</sub>**.

Since several tests were performed, we applied a  $p$ -value correction procedure based on false discovery rate ([18]) aiming to minimize the number of false positives (Type I error).

Table D.1: Difference in median location for revenue distributions for all instances, using a confidence interval of 99%. The omitted  $p$ -values are less than 0.0009.

	CORAL	CPLEX	RG <sub>RK</sub>	BO <sub>MA</sub>	CA <sub>RA</sub>	CA <sub>LP</sub>	GA <sub>RA</sub>	GA <sub>LP</sub>	SD <sub>RA</sub>	SD <sub>LP</sub>
CORAL	<b>0.000000</b>	-0.950720	-0.840038	-0.916819	-0.999908	-0.999944	-0.988866	-0.999941	-0.930584	-0.961331
CPLEX		<b>0.982822</b>	0.080600	0.018096	-0.001477	-0.003086	-0.000017	-0.002008	0.004974	-0.000078
RG <sub>RK</sub>			<b>0.875503</b>	-0.051401	-0.114096	-0.115350	-0.104717	-0.114523	-0.062547	-0.088295
BO <sub>MA</sub>				<b>0.939643</b>	-0.051826	-0.053154	-0.042700	-0.051845	-0.000850	-0.025455
CA <sub>RA</sub>					<b>1.000000</b>	-0.000036	0.000002	0.000014	0.034406	0.000695
CA <sub>LP</sub>						<b>1.000000</b>	0.000005	0.000037	0.035707	0.003093
GA <sub>RA</sub>							<b>1.000000</b>	-0.000067	0.026002	0.000012
GA <sub>LP</sub>								<b>1.000000</b>	0.034429	0.001266
SD <sub>RA</sub>									<b>0.955629</b>	-0.010476
SD <sub>LP</sub>										<b>0.951900</b>
										<i>p &gt; 0.29</i>
										<i>p &gt; 0.07</i>

Table D.2: Difference in median location for revenue distributions for instances with 400 bids or less, using a confidence interval of 99%. The omitted  $p$ -values are less than 0.004.

	CORAL	CPLEX	RG <sub>RK</sub>	BO <sub>MA</sub>	CA <sub>RA</sub>	CA <sub>LP</sub>	GA <sub>RA</sub>	GA <sub>LP</sub>	SD <sub>RA</sub>	SD <sub>LP</sub>
CORAL	<b>0.999984</b>	-0.000062	-0.000052	-0.000033	-0.000012	-0.000042	-0.000031	-0.000037	-0.000017	-0.000039
CPLEX		<b>1.000000</b>	0.000083	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
RG <sub>RK</sub>			<b>1.000000</b>	-0.000049	-0.000057	-0.000029	-0.000070	-0.000077	-0.000007	-0.000026
BO <sub>MA</sub>				<b>1.000000</b>	-0.000039	-0.000011	-0.000041	-0.000016	-0.000046	-0.000025
CA <sub>RA</sub>					<b>1.000000</b>	-0.000047	0.000012	-0.000077	0.000063	-0.000068
CA <sub>LP</sub>			<i>p &gt; 0.09</i>			<b>1.000000</b>	0.000071	0.000022	0.000060	-0.000089
GA <sub>RA</sub>						<i>p &gt; 0.25</i>	<b>1.000000</b>	-0.000037	0.000041	-0.000007
GA <sub>LP</sub>			<i>p &gt; 0.08</i>				<i>p &gt; 0.86</i>	<b>1.000000</b>	0.000083	-0.000032
SD <sub>RA</sub>								<i>p &gt; 0.01</i>	<b>1.000000</b>	-0.000076
SD <sub>LP</sub>										<b>0.999300</b>
										<i>p &gt; 0.30</i>

Table D.3: Difference in median location for revenue distributions for instances with 1000 bids or more, using a confidence interval of 99%. The omitted  $p$ -values are less than 0.001.

	CORAL	CPLEX	RG <sub>RK</sub>	BO <sub>MA</sub>	CA <sub>RA</sub>	CA <sub>LP</sub>	GA <sub>RA</sub>	GA <sub>LP</sub>	SD <sub>RA</sub>	SD <sub>LP</sub>
CORAL	<b>0.000000</b>	-0.946779	-0.856397	-0.922194	-0.999946	-0.999927	-0.990230	-0.999915	-0.931543	-0.957204
CPLEX		<b>0.953821</b>	0.079737	0.022089	-0.035372	-0.035907	-0.023595	-0.034921	0.014744	-0.000004
RG <sub>RK</sub>			<b>0.863907</b>	-0.056439	-0.124063	-0.124426	-0.112816	-0.123114	-0.062318	-0.088282
BO <sub>MA</sub>				<b>0.926657</b>	-0.062474	-0.062700	-0.053631	-0.061591	-0.005172	-0.029241
CA <sub>RA</sub>					<b>1.000000</b>	-0.000065	0.000065	0.000018	0.051891	0.027844
CA <sub>LP</sub>						<b>1.000000</b>	0.000086	0.000071	0.052948	0.028406
GA <sub>RA</sub>							<b>0.993222</b>	-0.000044	0.043039	0.017885
GA <sub>LP</sub>								<b>1.000000</b>	0.051161	0.027183
SD <sub>RA</sub>									<b>0.938537</b>	-0.021078
SD <sub>LP</sub>										<b>0.951700</b>
$p > 0.27$										
$p > 0.03$										

Table D.4: Difference in median location for revenue distributions for LG 1500/1500 instances, using a confidence interval of 99%. The omitted  $p$ -values are less than 0.00001.

$R_{G_{RK}}$	$B_{O_{MA}}$	$C_{A_{RA}}$	$C_{A_{LP}}$	$G_{A_{RA}}$	$G_{A_{LP}}$	$S_{D_{RA}}$	$S_{D_{LP}}$	
<b><math>R_{G_{RK}}</math></b>	<b><math>0.827570</math></b>	-0.081040	-0.163875	-0.162074	-0.154572	-0.160352	-0.102957	-0.109679
$B_{O_{MA}}$	<b><math>0.910548</math></b>	-0.083920	-0.083313	-0.076227	-0.079144	-0.022304	-0.028658	
$C_{A_{RA}}$		<b><math>1.000000</math></b>	0.000027	0.000036	0.000045	0.057928	0.053322	
$C_{A_{LP}}$		$p > 0.05$	<b><math>1.000000</math></b>	0.000036	0.000057	0.057150	0.052600	
$G_{A_{RA}}$				<b><math>0.998888</math></b>	-0.000042	0.050617	0.042824	
$G_{A_{LP}}$			$p > 0.01$		<b><math>1.000000</math></b>	0.054981	0.047942	
$S_{D_{RA}}$						<b><math>0.935822</math></b>	-0.005051	
$S_{D_{LP}}$							$p > 0.01$	<b><math>0.942900</math></b>

Table D.5: Difference in median location of revenue distributions for all instances, considering the best solutions until 100 generations. A confidence interval of 99% was used. The omitted  $p$ -values are less than 0.000009.

	RG <sub>RK</sub>	BO <sub>MA</sub>	CA <sub>RA</sub>	CA <sub>LP</sub>	GA <sub>RA</sub>	GA <sub>LP</sub>	SD <sub>RA</sub>	SD <sub>LP</sub>
RG <sub>RK</sub>	<b>0.365256</b>	-0.027354	-0.484062	-0.495047	-0.439106	-0.495536	-0.082448	-0.251942
BO <sub>MA</sub>		<b>0.410741</b>	-0.433579	-0.443894	-0.389594	-0.444571	-0.028155	-0.208201
CA <sub>RA</sub>			<b>0.971370</b>	-0.000033	0.000045	-0.000016	0.313978	0.115553
CA <sub>LP</sub>				<b>0.995975</b>	0.000040	0.000046	0.321063	0.124509
GA <sub>RA</sub>					<b>0.894743</b>	-0.000053	0.274147	0.069280
GA <sub>LP</sub>						<b>0.993376</b>	0.321429	0.126418
SD <sub>RA</sub>							<b>0.541106</b>	-0.126061
SD <sub>LP</sub>								<b>0.721500</b>

Table D.6: Difference in median location of revenue distributions for instances with 400 bids or less, considering the best solutions until 100 generations. A confidence interval of 99% was used. The omitted  $p$ -values are less than 0.009.

	RG <sub>RK</sub>	BO <sub>MA</sub>	CA <sub>RA</sub>	CA <sub>LP</sub>	GA <sub>RA</sub>	GA <sub>LP</sub>	SD <sub>RA</sub>	SD <sub>LP</sub>
RG <sub>RK</sub>	<b>0.713545</b>	-0.000017	-0.000011	-0.000029	-0.000019	-0.000020	-0.000009	-0.000034
BO <sub>MA</sub>		<b>1.000000</b>	0.000005	0.000051	0.000030	0.000067	0.000038	0.000059
CA <sub>RA</sub>	$p > 0.01$		<b>0.990978</b>	-0.000027	0.000008	-0.000006	-0.000000	-0.000015
CA <sub>LP</sub>			$p > 0.01$	<b>0.999988</b>	0.000003	0.000033	0.000021	-0.000044
GA <sub>RA</sub>				$p > 0.37$	$p > 0.11$	<b>0.999943</b>	-0.000037	0.000018
GA <sub>LP</sub>					$p > 0.04$	$p > 0.70$	$p > 0.27$	<b>0.999986</b>
SD <sub>RA</sub>	$p > 0.41$					$p > 0.09$	$p > 0.01$	<b>0.793841</b>
SD <sub>LP</sub>						$p > 0.52$	$p > 0.03$	$p > 0.37$

Table D.7: Difference in median location of revenue distributions for instances with more than 400 bids, considering the best solutions until 100 generations. A confidence interval of 99% was used. The omitted  $p$ -values are less than 0.0001.

Table D.8: Difference in median location of revenue distributions for LG 1500/1500 instances, considering the best solutions until 100 generations. A confidence interval of 99% was used. The omitted  $p$ -values are less than 0.00004.

$R_{G_{RK}}$	$B_{O_{MA}}$	$C_{A_{RA}}$	$C_{A_{LP}}$	$G_{A_{RA}}$	$G_{A_{LP}}$	$S_{D_{RA}}$	$S_{D_{LP}}$	
<b><math>R_{G_{RK}}</math></b>	<b><math>0.228827</math></b>	-0.064896	-0.661234	-0.661403	-0.608903	-0.665038	-0.242486	-0.306383
$B_{O_{MA}}$	<b><math>0.312164</math></b>	-0.599983	-0.601567	-0.544650	-0.604113	-0.172053	-0.239376	
$C_{A_{RA}}$		<b><math>0.991615</math></b>	-0.000039	0.009253	-0.000065	0.418796	0.334902	
$C_{A_{LP}}$		$p > 0.97$	<b><math>0.991615</math></b>	0.012762	-0.000056	0.419685	0.334964	
$G_{A_{RA}}$				<b><math>0.890265</math></b>	-0.018571	0.351878	0.290690	
$G_{A_{LP}}$			$p > 0.68$	$p > 0.68$	<b><math>0.993368</math></b>	0.421907	0.336256	
$S_{D_{RA}}$						<b><math>0.479321</math></b>	-0.057048	
$S_{D_{LP}}$							<b><math>0.548400</math></b>	

## D.2 Additional running time results

Table D.9 shows the average time in seconds taken by each algorithm to find the best solution (recall that we limited runs to at most 3,600 seconds). The additional time in the last iterations without improvement in the best solution found is disregard. We also exclude the time used loading instances and logging. To be fair with the Java implementations, each run began with a warm-up phase so that Java virtual machine could load and optimize all necessary bytecode. The first two columns of this table list, respectively, the instance classes and their corresponding sizes. Each following pair of columns shows the average time and standard deviation for each algorithm, respectively.

Table D.9: Running time comparison among the algorithms. For each algorithm, the table shows the average time to find the best solutions. The over time used in the last iterations without improvement is disregarded. Time in seconds.

Class	Size	CORAL		CPLEX		RG <sub>RK</sub>		BO <sub>MA</sub>		CA <sub>RA</sub>		CA <sub>LP</sub>		GA <sub>RA</sub>		GA <sub>LP</sub>		SD <sub>RA</sub>		SD <sub>LP</sub>	
		Time	$\sigma$	Time	$\sigma$	Time	$\sigma$	Time	$\sigma$	Time	$\sigma$	Time	$\sigma$	Time	$\sigma$	Time	$\sigma$	Time	$\sigma$	Time	$\sigma$
CATS	40	1	1	1	1	10	6	1	1	1	0	1	0	2	1	1	0	2	3	1	0
	80	2	2	1	1	12	7	1	1	2	1	1	0	5	38	1	0	2	4	1	0
	200	792	1438	1	1	18	15	17	32	14	74	1	0	31	123	1	0	46	132	1	0
	400	923	1418	1	1	40	56	51	121	58	161	28	124	47	139	18	96	65	172	20	89
	1000	2883	1436	382	1075	473	651	736	936	53	136	30	96	60	134	36	104	75	167	32	112
	1024	2803	1494	960	1568	460	591	778	1130	54	142	33	114	72	165	30	106	79	181	28	96
	2000	2903	1411	1377	1708	1463	1151	1909	1324	72	181	35	106	66	135	39	114	78	166	29	99
	4000	3012	1344	1802	1799	2527	1540	2611	1361	52	115	28	75	68	133	42	119	54	123	30	87
LG	1000	3606	12	3601	1	289	529	75	68	94	196	93	192	94	196	95	197	78	181	71	168
	1500	3624	23	3601	1	425	702	66	58	118	211	113	200	94	187	100	187	92	188	83	179

### D.3 Best results for each instance

This section presents the results obtained for the LG instances. The format of the tables is the following: the first column and second columns are the instance name and the best revenue obtained for this instance, respectively. The following columns show the percentage of the revenue from the best solution obtained by the algorithm that names the column. A high percentage indicates that the obtained solution is closer to the best. A star (\*) indicates that the algorithm found the best solution.

Table D.10: Best results for CATS instances with less than 400 bids. The names of the instances are composed by the class, number of bids, number of goods, and serial number of the instance.

Inst.	Best	CORAL	CPLEX	RG <sub>RK</sub>	BO <sub>MA</sub>	CA <sub>RA</sub>	CA <sub>LP</sub>	GA <sub>RA</sub>	GA <sub>LP</sub>	SD <sub>RA</sub>	SD <sub>LP</sub>
L2_40_10_1	8774.7200	*	*	*	*	*	*	*	*	*	*
L2_40_10_2	9229.3400	*	*	*	*	*	*	*	*	*	*
L2_40_10_3	8967.4300	*	*	*	*	*	*	*	*	*	*
L2_80_10_1	9828.2500	77.64	*	*	*	*	*	*	*	*	*
L2_80_10_2	9786.7100	*	*	*	*	*	*	*	*	*	*
L2_80_10_3	9441.1700	*	*	*	*	*	*	*	*	*	*
L2_200_50_2	45785.7000	*	*	*	*	*	*	*	*	92.15	*
L2_200_50_3	49031.9000	87.41	*	*	*	*	*	*	*	*	*
L2_400_50_1	46588.7410	*	*	*	*	*	*	*	*	98.34	*
L2_400_50_2	47706.0000	*	*	*	*	*	*	*	*	*	*
L2_400_50_3	47819.7160	*	*	*	*	*	*	*	*	*	*
L3_40_10_1	2474.2480	*	*	*	*	*	*	*	*	*	*
L3_40_10_2	2682.7890	*	*	*	*	*	*	*	*	*	*
L3_40_10_3	2929.2870	*	*	*	*	*	*	*	*	*	*
L3_80_10_1	2862.1650	*	*	*	*	*	*	*	*	*	*
L3_80_10_2	2779.9100	*	*	*	*	*	*	*	*	*	*
L3_80_10_3	2938.3120	*	*	*	*	*	*	*	*	*	*
L3_200_50_1	12178.8010	*	*	*	*	*	*	*	*	*	*
L3_200_50_3	12612.9650	*	*	*	*	*	*	*	*	*	*
L3_400_50_1	14338.1150	*	*	*	*	*	*	*	*	*	*
L3_400_50_2	14747.9490	*	*	99.07	*	*	*	*	*	*	*
L3_400_50_3	14495.9880	*	*	99.54	*	*	*	*	*	*	*
L4_40_10_1	9543.8540	*	*	*	*	*	*	*	*	*	*
L4_40_10_2	8870.7760	*	*	*	*	*	*	*	*	*	*
L4_40_10_3	9249.9330	*	*	*	*	*	*	*	*	*	*
L4_80_10_1	9770.0770	*	*	99.69	*	*	*	*	*	*	*
L4_80_10_2	9817.6040	*	*	99.50	*	*	*	*	*	*	*
L4_80_10_3	9759.7910	*	*	*	*	*	*	*	*	*	*
L4_200_50_1	45191.2690	86.61	*	99.65	*	*	*	*	*	*	*
L4_200_50_2	44275.5990	92.24	*	99.56	*	*	*	*	*	*	*
L4_200_50_3	46496.4650	93.73	*	*	*	*	*	*	*	*	*
L4_400_50_1	47748.4440	89.23	*	99.42	*	*	*	*	*	*	*
L4_400_50_2	47988.4200	*	*	99.56	99.62	*	*	*	*	*	*
L4_400_50_3	48410.5140	*	*	99.40	99.55	*	*	*	*	*	*
L6_40_10_1	8791.5910	*	*	*	*	*	*	*	*	*	*
L6_40_10_2	9297.1700	*	*	*	*	*	*	*	*	*	*
L6_40_10_3	9217.2400	*	*	*	*	*	*	*	*	*	*

Continue on next page...

Table D.10: (continued).

Inst.	Best	CORAL	CPLEX	RG <sub>RK</sub>	B0 <sub>MA</sub>	CA <sub>RA</sub>	CA <sub>LP</sub>	GA <sub>RA</sub>	GA <sub>LP</sub>	SD <sub>RA</sub>	SD <sub>LP</sub>
L6_80_10_1	9290.9270	*	*	*	*	*	*	*	*	*	*
L6_80_10_2	9836.4500	*	*	*	*	*	*	*	*	*	*
L6_80_10_3	9593.9010	*	*	97.69	*	*	*	*	*	*	*
L6_200_50_1	41639.9910	96.86	*	95.31	*	*	*	*	*	*	*
L6_200_50_2	38873.5410	98.44	*	99.62	*	*	*	*	*	99.53	*
L6_200_50_3	40561.3300	*	*	*	*	*	*	*	*	*	*
L6_400_50_1	44990.9010	99.12	*	*	*	*	*	*	*	*	*
L6_400_50_2	46366.8710	99.45	*	97.48	*	*	*	*	*	*	*
L6_400_50_3	45216.8660	96.18	*	96.83	95.86	99.92	*	*	*	*	*
L7_40_10_1	8309.1230	*	*	*	*	*	*	*	*	*	*
L7_40_10_2	9090.6580	*	*	*	*	*	*	*	*	*	*
L7_40_10_3	8553.2690	*	*	*	*	*	*	*	*	*	*
L7_80_10_1	9818.5880	*	*	99.32	*	*	*	*	*	*	*
L7_80_10_2	9435.4580	*	*	*	*	*	*	*	*	*	*
L7_80_10_3	9775.6220	*	*	99.81	*	*	*	*	*	*	*
L7_200_50_1	28286.0100	*	*	*	*	*	*	*	*	*	*
L7_200_50_2	30478.8250	69.46	*	*	*	*	*	*	*	*	*
L7_200_50_3	29014.3000	*	*	*	*	*	*	*	*	*	*
L7_400_50_1	32505.1200	*	*	*	*	*	*	*	*	98.15	*
L7_400_50_2	33512.5500	*	*	96.52	*	*	*	*	*	*	*
L7_400_50_3	29829.2200	*	*	98.74	*	*	*	*	*	*	*
arbitrary_40_10_1	1019.7600	*	*	*	*	*	*	*	*	*	*
arbitrary_40_10_2	749.5520	99.98	*	*	*	*	*	*	*	*	*
arbitrary_40_10_3	679.8568	*	*	*	*	*	*	*	*	*	*
arbitrary_80_10_1	1038.7469	*	*	*	*	*	*	*	*	*	*
arbitrary_80_10_2	775.6040	*	*	*	*	*	*	*	*	*	*
arbitrary_80_10_3	581.9593	*	*	*	*	*	*	*	*	*	*
arbitrary_200_50_1	3007.5090	*	*	96.00	*	*	*	*	*	99.13	*
arbitrary_200_50_2	3260.1100	*	*	97.54	*	*	*	*	*	*	*
arbitrary_200_50_3	3271.8422	*	*	96.60	*	*	*	*	*	*	*
arbitrary_400_50_1	4038.0004	*	*	93.09	90.48	*	*	*	*	*	*
arbitrary_400_50_2	3791.8860	*	*	93.80	85.02	*	*	*	*	*	*
arbitrary_400_50_3	4289.3898	*	*	88.70	*	*	*	*	*	*	*
matching_40_10_1	10.7792	*	*	*	*	*	*	*	*	*	*
matching_40_10_2	7.1517	*	*	*	*	*	*	*	*	*	*
matching_40_10_3	15.6388	*	*	*	*	*	*	*	*	*	*
matching_80_10_1	13.7825	*	*	*	*	*	*	*	*	*	*
matching_80_10_2	1.5328	*	*	*	*	*	*	*	*	*	*
matching_80_10_3	8.1853	*	*	*	*	*	*	*	*	*	*
matching_200_50_1	32.2974	*	*	*	*	*	*	*	*	*	*
matching_200_50_2	32.0509	*	*	*	*	*	*	*	*	*	*
matching_200_50_3	23.8792	*	*	*	*	*	*	*	*	*	*
matching_400_50_1	41.8873	*	*	*	*	*	*	*	*	*	*
matching_400_50_2	55.8732	*	*	*	*	*	*	*	*	*	*
matching_400_50_3	27.1398	*	*	*	*	*	*	*	*	*	*
paths_40_10_1	4.5826	*	*	*	*	*	*	*	*	*	*
paths_40_10_2	6.1875	*	*	*	*	*	*	*	*	*	*
paths_40_10_3	5.3516	*	*	*	*	*	*	*	*	*	*
paths_80_10_1	7.0575	*	*	*	*	*	*	*	*	*	*
paths_80_10_2	5.0659	*	*	*	*	*	*	*	*	*	*
paths_80_10_3	6.0272	*	*	*	*	*	*	*	*	*	*
paths_200_50_1	20.2063	97.32	*	*	*	*	*	*	*	*	*
paths_200_50_2	20.0969	*	*	*	*	*	*	*	*	*	*

Continue on next page...

Table D.10: (continued).

Inst.	Best	CORAL	CPLEX	RG <sub>RK</sub>	BO <sub>MA</sub>	CA <sub>RA</sub>	CA <sub>LP</sub>	GA <sub>RA</sub>	GA <sub>LP</sub>	SD <sub>RA</sub>	SD <sub>LP</sub>
paths_200_50_3	22.1016	*	*	*	*	*	*	*	*	*	*
paths_400_50_1	26.8886	*	*	*	99.54	*	*	*	*	*	*
paths_400_50_2	22.9762	*	*	*	*	*	*	*	*	*	*
paths_400_50_3	23.7947	*	*	*	99.72	*	*	*	*	*	*
regions_40_10_1	942.2510	*	*	*	*	*	*	*	*	*	*
regions_40_10_2	750.5340	*	*	*	*	*	*	*	*	*	*
regions_40_10_3	659.7730	*	*	*	*	*	*	*	*	*	*
regions_80_10_1	808.6960	*	*	96.58	*	*	*	*	*	*	*
regions_80_10_2	957.2460	*	*	*	*	*	*	*	*	*	*
regions_80_10_3	1159.6219	*	*	98.13	98.13	*	*	*	*	*	*
regions_200_50_1	3616.3098	*	*	97.69	*	*	*	*	*	*	*
regions_200_50_2	3292.0154	*	*	87.48	*	*	*	*	*	*	*
regions_200_50_3	3401.2610	*	*	92.77	*	*	*	*	*	*	*
regions_400_50_1	4177.5069	*	*	89.33	*	*	*	*	*	*	*
regions_400_50_2	3606.1991	*	*	91.53	*	*	*	*	*	*	*
regions_400_50_3	3482.6069	*	*	93.81	*	*	*	*	*	*	*
scheduling_40_10_1	14.7840	*	*	*	*	*	*	*	*	*	*
scheduling_40_10_2	15.1235	*	*	*	*	*	*	*	*	*	*
scheduling_40_10_3	21.7354	*	*	*	*	*	*	*	*	*	*
scheduling_80_10_1	22.6557	*	*	*	*	*	*	*	*	*	*
scheduling_80_10_2	15.7918	*	*	*	*	*	*	*	*	*	*
scheduling_80_10_3	22.8672	*	*	*	*	*	*	*	*	*	*
scheduling_200_50_1	22.6139	*	*	*	*	*	*	*	*	*	*
scheduling_200_50_2	53.8402	*	*	*	*	*	*	*	*	*	*
scheduling_200_50_3	48.8045	*	*	95.26	*	*	*	*	*	*	*
scheduling_400_50_1	58.2749	*	*	97.38	*	*	*	*	*	*	*
scheduling_400_50_2	70.3185	*	*	*	*	*	*	*	*	*	*
scheduling_400_50_3	43.8167	1.50	*	*	*	*	*	*	*	*	*

Table D.11: Best results for CATS instances more than 400 bids. The names of the instances are composed by the class, number of bids, number of goods, and serial number of the instance. Instances with **hard** in the name have 1024 bids and 256 goods.

Inst.	Best	CORAL	CPLEX	RG <sub>RK</sub>	BO <sub>MA</sub>	CA <sub>RA</sub>	CA <sub>LP</sub>	GA <sub>RA</sub>	GA <sub>LP</sub>	SD <sub>RA</sub>	SD <sub>LP</sub>
L2_1000_256_1	244098.0000	*	*	*	*	*	*	*	*	88.40	*
L2_1000_256_2	241158.0000	*	*	*	*	*	*	*	*	90.88	*
L2_1000_256_3	254988.0000	*	*	*	*	*	*	*	*	81.73	*
L2_2000_512_1	495448.0000	*	*	*	*	*	*	*	*	72.84	*
L2_2000_512_2	501810.0000	4.27	*	*	*	*	*	*	*	80.74	*
L2_2000_512_3	505625.0000	*	*	*	*	*	*	*	*	87.47	*
L2_4000_1024_1	1000590.0000	*	*	*	*	*	*	*	*	37.53	*
L2_4000_1024_2	1010991.6920	10.66	*	*	100.00	*	*	*	*	59.42	*
L2_4000_1024_3	996744.0000	*	*	*	*	*	*	*	*	34.50	*
L2_hard_1	262.5110	*	*	*	*	*	*	*	*	63.91	*
L2_hard_2	456.5370	*	*	*	*	*	*	*	*	*	*
L2_hard_3	317.4120	*	*	*	*	*	*	*	*	66.52	*
L3_1000_256_1	64626.2530	75.64	*	99.26	99.60	99.60	99.60	99.60	99.56	96.60	99.50
L3_1000_256_2	66106.1710	79.20	*	99.08	99.31	99.75	99.78	99.78	99.66	98.58	99.78
L3_1000_256_3	64987.7430	75.45	*	99.68	*	99.84	*	99.81	*	99.79	*
L3_2000_512_1	128004.7893	81.84	*	98.80	97.74	99.47	99.92	99.92	99.72	98.43	99.95

Continue on next page...

Table D.11: (continued).

Inst.	Best	CORAL	CPLEX	RG <sub>RK</sub>	BO <sub>MA</sub>	CA <sub>RA</sub>	CA <sub>LP</sub>	GA <sub>RA</sub>	GA <sub>LP</sub>	SD <sub>RA</sub>	SD <sub>LP</sub>
L3_2000_512_2	132229.2010	93.59	*	99.18	98.78	99.49	99.71	99.35	99.78	97.62	99.69
L3_2000_512_3	133133.1410	82.48	*	98.95	98.83	99.07	99.64	99.43	99.62	97.09	99.62
L3_4000_1024_1	263970.8210	78.25	*	90.99	97.45	99.88	99.74	98.87	99.50	96.46	99.25
L3_4000_1024_2	263936.8590	75.72	*	91.47	97.29	99.48	99.62	99.60	99.55	96.62	99.53
L3_4000_1024_3	263404.1096	80.03	*	91.46	96.98	99.09	99.30	99.17	98.94	96.70	98.93
L3_hard_1	75.4074	75.32	*	99.35	97.11	99.28	99.39	99.18	99.35	98.89	99.18
L3_hard_2	34.1897	80.59	99.33	96.96	95.45	99.27	*	99.09	97.74	97.17	97.74
L3_hard_3	20.8636	82.47	96.79	97.63	96.19	98.44	98.63	98.44	*	95.31	*
L4_1000_256_1	228752.1550	4.41	*	99.52	99.17	99.85	*	*	*	99.67	*
L4_1000_256_2	229601.7270	92.34	*	99.27	98.48	99.71	*	*	*	99.43	*
L4_1000_256_3	229349.1950	2.85	*	99.58	99.70	99.90	*	*	*	99.43	*
L4_2000_512_1	461218.2640	3.56	*	99.48	99.04	99.71	*	*	*	99.07	*
L4_2000_512_2	459425.3230	2.75	*	99.83	98.97	99.73	*	*	*	99.32	*
L4_2000_512_3	458536.7260	2.64	*	99.50	99.07	99.65	*	*	*	99.50	*
L4_4000_1024_1	914322.9910	2.30	*	98.33	96.11	99.32	*	*	*	99.15	*
L4_4000_1024_2	920786.4330	2.22	*	98.39	95.88	99.09	*	*	*	98.93	*
L4_4000_1024_3	920294.1540	3.27	*	98.37	96.36	99.16	99.99	99.99	99.99	99.00	99.99
L4_hard_1	290.2399	16.81	*	99.53	98.80	*	*	*	*	*	*
L4_hard_2	383.8526	13.76	*	99.36	99.14	*	*	*	*	*	*
L4_hard_3	282.6879	7.06	*	99.55	98.90	*	*	*	*	*	*
L6_1000_256_1	199757.0790	45.81	*	97.22	99.41	98.23	98.75	98.32	98.72	98.22	98.22
L6_1000_256_2	200559.8373	69.95	*	97.41	96.57	98.80	99.39	97.61	99.39	97.38	99.11
L6_1000_256_3	201208.1706	3.72	*	99.56	99.15	98.72	99.96	98.36	98.92	97.16	98.36
L6_2000_512_1	405788.3937	72.95	*	98.00	95.24	97.76	99.00	97.85	99.00	94.39	99.00
L6_2000_512_2	411091.1370	5.22	*	97.83	94.81	97.77	98.16	97.22	98.16	96.02	97.77
L6_2000_512_3	402472.3077	71.43	*	98.29	95.41	97.04	97.93	97.73	97.93	94.55	97.93
L6_4000_1024_1	785686.0929	0.88	*	98.15	93.36	97.30	97.62	97.54	97.73	97.60	97.75
L6_4000_1024_2	801026.0135	75.21	*	98.90	92.95	97.91	97.66	97.33	97.84	95.53	95.95
L6_4000_1024_3	791849.8150	73.60	*	98.19	93.02	97.06	97.03	96.92	97.38	95.83	96.82
L6_hard_1	377.5873	13.69	*	99.57	99.51	*	*	*	*	*	*
L6_hard_2	330.2240	18.25	*	99.58	99.65	*	*	*	*	*	*
L6_hard_3	446.4472	6.50	*	99.62	99.68	*	*	*	*	*	*
L7_1000_256_1	68830.4000	95.17	*	*	*	*	*	*	*	86.30	*
L7_1000_256_2	79025.8000	96.74	*	*	100.00	*	*	*	*	96.74	*
L7_1000_256_3	81981.6000	100.00	*	*	100.00	*	*	*	*	*	*
L7_2000_512_1	121043.0000	*	*	*	*	*	*	*	*	97.56	*
L7_2000_512_2	119058.0000	*	*	*	*	*	*	*	*	93.95	*
L7_2000_512_3	122346.0000	99.99	*	*	*	*	*	*	*	92.35	*
L7_4000_1024_1	244374.0000	*	*	*	*	*	*	*	*	90.59	*
L7_4000_1024_2	229826.0000	*	*	*	*	*	*	*	*	99.68	*
L7_4000_1024_3	228342.0000	*	*	*	*	*	*	*	*	88.19	*
L7_hard_1	233.0348	73.22	*	*	*	*	*	*	98.25	*	*
L7_hard_2	127.4510	100.00	*	*	*	*	*	*	*	*	*
L7_hard_3	261.2782	83.15	97.72	97.70	*	99.18	*	99.18	*	95.56	95.56
arbitrary_1000_256_1	17186.3016	70.40	96.39	93.12	95.91	*	*	96.87	*	94.46	95.53
arbitrary_1000_256_2	15782.8217	6.06	98.02	96.27	95.63	98.41	*	98.56	99.40	96.39	98.03
arbitrary_1000_256_3	17280.1375	9.31	98.04	92.55	97.45	98.69	*	99.28	99.28	97.28	97.28
arbitrary_2000_512_1	32267.8600	0.17	96.98	96.15	93.59	99.74	98.89	*	99.56	95.35	96.13
arbitrary_2000_512_2	32159.7621	1.56	95.83	96.42	94.28	99.97	*	99.39	99.19	98.02	97.21
arbitrary_2000_512_3	32181.8011	2.52	95.86	96.79	97.81	99.54	99.27	*	99.04	97.77	98.95
arbitrary_4000_1024_1	62694.3745	1.43	95.70	93.86	91.57	99.64	98.48	98.56	*	96.90	96.76
arbitrary_4000_1024_2	61809.4598	0.35	97.64	93.42	90.90	97.84	98.90	*	97.92	96.32	95.92
arbitrary_4000_1024_3	62366.9031	79.65	96.20	93.46	90.80	98.82	99.01	97.84	*	96.39	95.77

Continue on next page...

Table D.11: (continued).

Table D.12: Best results for LG 1000/500 instances.

Inst.	Best	CORAL	CPLEX	RG <sub>RK</sub>	BO <sub>MA</sub>	CA <sub>RA</sub>	CA <sub>LP</sub>	GA <sub>RA</sub>	GA <sub>LP</sub>	SD <sub>RA</sub>	SD <sub>LP</sub>
in101	72724.6180	37.66	92.27	92.27	96.03	*	*	*	*	95.02	98.63
in102	72518.2220	45.17	98.03	96.72	98.22	*	*	*	*	97.44	99.82
in103	72129.5000	40.67	96.64	95.13	96.76	*	*	97.41	*	*	98.43
in104	72709.6470	65.30	98.04	94.46	97.42	*	*	*	*	92.45	*
in105	75646.1406	39.96	89.05	90.91	*	*	*	*	*	94.99	*
in106	71258.6130	51.23	89.77	93.23	94.31	*	*	*	*	94.49	*
in107	69713.4030	38.64	98.38	98.38	99.24	*	*	99.55	*	*	*
in108	75813.2109	11.33	98.39	99.12	*	99.95	*	99.12	99.95	99.31	99.30
in109	69475.8950	38.47	91.99	95.09	95.34	*	*	*	*	*	*
in110	68295.2890	16.29	*	92.75	99.79	*	*	*	*	*	*
in111	75133.2900	42.87	96.74	95.16	97.12	*	*	*	*	95.53	97.12
in112	71342.4830	60.02	99.25	94.80	99.81	*	*	*	*	*	*
in113	73365.8906	53.84	92.73	96.02	*	*	*	*	*	98.43	*
in114	69224.7656	13.31	94.35	94.58	*	*	98.58	99.56	99.56	96.04	96.04
in115	70221.5610	48.33	94.85	93.15	96.06	*	*	99.55	*	95.95	96.95
in116	70032.4609	48.56	98.32	93.80	*	*	*	*	*	98.32	98.32
in117	69982.8330	59.34	94.96	99.92	99.92	*	*	*	*	95.33	98.99
in118	72160.9870	57.56	95.02	93.06	97.21	*	*	*	*	95.36	95.36
in119	67038.4297	58.44	96.86	*	*	*	*	98.36	*	98.27	98.27
in120	75514.9300	58.27	98.85	93.41	99.95	*	*	99.13	99.13	97.67	98.87
in121	67639.1250	34.44	96.47	94.24	*	*	*	*	*	96.34	96.34
in122	69546.2730	40.98	96.73	98.24	98.24	*	*	99.97	*	95.69	*
in123	70618.3130	49.50	92.25	94.96	99.97	*	*	*	99.97	98.78	99.93
in124	71686.0469	39.54	97.28	99.72	*	*	*	*	*	96.63	99.72
in125	69233.1220	51.98	95.79	95.79	97.41	*	*	*	*	98.14	*
in126	70671.7700	9.53	98.45	93.05	98.61	*	*	98.61	*	95.98	96.62
in127	69273.3203	42.94	98.39	92.27	*	*	*	*	*	98.74	*
in128	72179.4310	17.30	94.35	90.70	98.32	*	*	*	*	95.27	96.20
in129	65751.6490	37.24	97.51	97.51	97.59	*	*	*	*	97.51	97.51
in130	71075.3000	48.78	97.39	97.90	97.90	*	*	99.14	*	96.04	97.90
in131	71177.9062	2.62	95.49	99.62	*	*	*	*	*	96.39	*
in132	75510.0469	43.34	96.88	99.90	*	*	*	*	*	*	*
in133	71253.5610	54.48	97.85	94.16	99.35	*	*	99.35	*	94.95	97.67
in134	75781.7490	46.70	96.61	91.22	98.73	*	*	*	*	97.39	*
in135	72138.1172	2.42	95.49	90.72	*	*	*	*	*	*	*
in136	68903.0938	43.37	94.79	96.29	*	*	*	99.86	99.86	96.61	99.04
in137	70072.0469	48.96	*	90.56	*	*	*	99.99	*	*	*
in138	71989.6330	28.25	97.43	99.24	99.24	99.24	*	99.24	*	99.24	97.71
in139	72840.3940	35.02	94.24	92.53	98.79	*	*	*	*	96.13	98.94
in140	73665.2310	43.72	*	92.15	92.42	*	*	*	*	*	*
in141	69605.0770	40.43	98.91	96.15	99.67	*	*	*	*	95.42	98.91
in142	74777.9850	49.90	97.26	94.59	96.20	*	*	*	*	97.52	97.52
in143	69699.0547	34.12	95.14	98.81	*	*	*	*	*	98.19	98.89
in144	73197.0730	49.56	94.95	93.48	99.03	*	*	*	*	*	*
in145	73695.0150	39.38	96.88	92.77	96.25	*	*	97.81	*	*	97.80
in146	73746.9375	38.30	95.29	93.65	*	*	*	*	*	97.29	97.29
in147	65878.3020	58.53	95.28	97.17	97.17	*	*	*	*	94.88	94.88
in148	72116.9690	51.94	96.01	95.66	98.84	99.81	*	98.84	*	99.81	99.81
in149	70800.1800	46.53	97.27	95.68	98.61	*	*	99.30	*	99.02	*
in150	72839.4240	46.46	94.35	93.20	98.91	*	*	*	*	*	*
in151	68834.5010	45.83	99.99	97.90	99.13	*	*	*	*	98.85	99.75
in152	76224.7812	41.04	93.71	93.62	*	*	*	*	*	97.94	97.94
in153	70110.7650	43.46	99.49	96.81	99.49	*	*	99.60	*	96.54	98.00

Continue on next page...

Table D.12: (continued).

Inst.	Best	CORAL	CPLEX	RG <sub>RK</sub>	B0 <sub>MA</sub>	CA <sub>RA</sub>	CA <sub>LP</sub>	GA <sub>RA</sub>	GA <sub>LP</sub>	SD <sub>RA</sub>	SD <sub>LP</sub>
in154	69215.5240	7.16	94.14	96.66	98.48	★	99.27	99.27	99.27	99.09	99.27
in155	74936.7730	36.64	96.51	96.26	97.22	★	★	99.75	★	99.75	★
in156	69704.1300	50.74	93.01	99.24	99.24	★	★	★	★	96.64	96.64
in157	73934.8438	33.75	91.20	93.38	★	★	★	★	★	92.71	★
in158	69489.5430	47.97	★	92.69	97.71	★	★	★	★	★	95.13
in159	71091.8047	55.58	96.38	95.46	★	★	★	★	★	98.53	98.53
in160	70606.9180	46.45	96.31	99.48	99.48	★	★	★	★	98.61	98.61
in161	66266.3710	15.81	92.56	93.91	98.53	★	99.34	★	★	97.88	★
in162	74720.7940	54.27	93.32	95.58	97.44	99.44	99.44	★	★	99.44	99.44
in163	64976.9910	46.23	98.63	98.45	99.06	★	99.86	99.86	99.86	99.06	99.06
in164	67950.6230	43.67	93.99	91.64	99.41	★	★	★	★	98.38	98.38
in165	70361.9531	39.37	95.13	95.97	★	★	★	98.61	★	97.19	97.19
in166	71460.8930	34.20	92.35	95.05	97.99	99.80	★	99.45	99.80	97.56	97.83
in167	74523.7656	26.61	★	96.58	★	★	★	★	★	96.86	96.86
in168	72097.3210	43.31	97.37	96.68	96.86	★	★	99.54	★	★	★
in169	71827.3400	45.18	96.43	97.32	98.45	★	★	98.62	★	95.21	★
in170	74564.7490	42.64	92.70	92.51	95.80	★	★	96.46	★	90.03	95.15
in171	71279.4840	37.03	96.53	94.32	97.33	★	★	98.48	★	98.48	98.45
in172	70361.8070	3.68	99.57	93.91	96.82	★	★	99.57	★	97.41	98.66
in173	73677.2030	57.65	96.29	93.20	99.78	★	★	★	★	94.10	97.49
in174	73523.6094	44.59	96.31	92.89	★	★	★	★	★	96.48	95.46
in175	72924.8740	49.45	97.54	91.49	97.26	★	99.67	99.67	99.67	99.67	99.67
in176	67761.4830	38.10	97.51	95.42	99.35	★	★	★	★	98.33	99.43
in177	70187.1540	49.13	94.27	95.49	98.94	★	★	★	★	98.88	98.94
in178	70833.3720	53.70	90.85	92.84	95.71	★	★	★	★	94.42	95.20
in179	72205.2980	42.51	96.80	95.57	96.60	★	★	98.97	★	96.60	★
in180	70513.3520	54.50	93.16	94.07	96.53	★	★	★	★	96.26	96.31
in181	72238.0859	37.33	95.06	97.16	★	★	★	★	★	96.76	99.07
in182	71645.0312	37.70	97.82	★	★	★	★	★	★	98.80	98.80
in183	71520.4688	37.89	98.38	93.86	★	★	★	★	★	99.57	99.57
in184	74377.5380	1.74	92.64	87.92	94.41	★	★	★	★	★	★
in185	73714.9531	47.24	★	94.44	★	★	★	99.52	★	99.52	★
in186	70736.2480	47.66	97.98	94.98	98.72	★	★	★	★	97.98	97.98
in187	70166.3660	31.20	95.26	94.28	97.34	★	★	★	★	98.55	98.55
in188	70485.1950	40.11	93.17	95.47	96.52	★	★	98.86	★	99.49	★
in189	69786.0220	38.77	95.35	96.84	98.82	★	★	★	★	98.54	★
in190	73765.2090	38.54	97.07	97.07	98.60	★	★	★	★	★	99.72
in191	72587.0780	8.24	99.65	97.87	98.63	★	★	99.65	★	★	★
in192	71156.8280	34.93	93.02	94.22	99.45	★	★	★	★	99.61	99.61
in193	72526.4688	33.95	97.21	94.22	★	★	★	★	★	97.46	97.46
in194	75803.5156	47.87	94.14	★	★	★	★	★	★	99.91	99.91
in195	69066.8672	29.99	91.41	96.21	★	★	★	★	★	96.25	96.25
in196	69776.2220	51.81	98.39	98.47	99.91	★	★	99.91	★	98.70	97.77
in197	68457.8040	55.49	★	★	98.33	★	★	★	★	97.41	97.41
in198	73474.3830	41.26	98.84	92.37	97.19	★	★	★	★	97.19	97.19
in199	70955.9130	37.03	93.84	95.59	99.98	★	★	99.98	★	98.21	98.34
in200	76803.1830	46.02	95.19	95.17	98.09	★	★	98.88	★	★	★

Table D.13: Best results for LG 1000/1000 instances.

Inst.	Best	CORAL	CPLEX	RG <sub>RK</sub>	B0 <sub>MA</sub>	CA <sub>RA</sub>	CA <sub>LP</sub>	GA <sub>RA</sub>	GA <sub>LP</sub>	SD <sub>RA</sub>	SD <sub>LP</sub>
in201	81557.7578	45.28	94.10	*	*	*	*	*	*	99.71	96.02
in202	90708.1406	38.91	98.60	93.30	*	*	*	*	*	99.96	*
in203	86239.2266	7.67	95.45	93.96	*	*	*	99.12	*	97.69	97.69
in204	87075.4453	38.39	94.14	94.39	*	*	*	*	*	95.49	98.62
in205	86515.9510	34.40	93.59	92.44	97.11	*	*	*	*	94.14	96.80
in206	91518.9640	19.12	94.93	93.41	94.93	*	*	*	*	*	*
in207	93129.2900	27.22	*	97.75	99.99	*	*	99.99	*	94.91	94.91
in208	94904.6953	25.73	88.80	90.18	*	*	*	96.71	*	96.71	96.71
in209	87268.9650	47.58	98.88	93.37	99.41	*	*	99.41	*	98.88	96.83
in210	89962.4062	39.71	96.64	95.73	*	*	*	*	*	98.38	*
in211	84913.6840	55.07	93.20	92.87	99.54	*	*	*	*	97.60	98.48
in212	90778.2188	40.06	96.81	91.38	*	*	*	*	*	98.97	98.97
in213	85369.1850	34.71	95.81	97.87	97.87	*	*	*	*	98.97	98.97
in214	85181.6090	39.16	97.34	96.19	99.58	*	*	*	*	99.58	*
in215	91531.7031	46.31	95.42	93.46	*	*	*	*	*	99.56	97.91
in216	91580.9800	48.61	*	93.53	94.72	*	*	*	*	*	*
in217	86962.9270	52.45	97.72	93.77	98.33	*	*	*	*	97.72	99.92
in218	94965.2109	45.14	90.34	91.55	*	*	*	*	*	*	*
in219	93586.4380	46.99	90.94	96.02	96.02	*	*	*	*	96.08	96.08
in220	89792.9219	44.44	97.87	96.82	*	*	*	98.48	*	98.63	98.63
in221	87410.7800	41.62	*	93.56	97.23	*	*	*	*	96.00	96.23
in222	89905.5391	45.82	94.77	90.80	*	*	*	*	*	*	*
in223	83045.4297	40.13	96.04	88.95	*	*	*	*	*	94.30	94.71
in224	87105.2770	49.10	96.86	98.39	99.92	*	*	*	*	97.13	97.40
in225	89430.1094	38.68	95.90	91.21	*	*	*	*	*	*	*
in226	88176.1220	34.96	91.75	90.75	95.92	*	*	95.09	*	95.92	95.92
in227	92613.3710	44.80	96.95	95.57	98.94	*	*	*	*	*	*
in228	92684.0781	56.28	96.70	96.70	*	*	*	*	*	98.86	95.33
in229	90468.1420	49.34	96.50	91.38	96.75	*	*	*	*	96.76	96.76
in230	91559.1562	48.44	96.66	94.13	*	*	*	*	*	97.74	97.74
in231	101458.6094	40.11	93.07	88.09	*	*	*	*	*	*	*
in232	87270.8630	17.55	95.18	91.66	99.45	*	*	*	*	92.66	*
in233	86151.8980	39.85	96.84	94.09	98.81	*	*	*	*	96.91	97.09
in234	88874.3359	49.60	98.17	92.70	*	*	*	*	*	96.84	96.84
in235	93246.5700	38.90	*	89.98	*	*	*	*	*	93.01	97.24
in236	87876.7891	38.94	98.10	91.59	*	*	*	*	*	95.25	96.84
in237	87616.0450	54.09	96.48	94.61	98.30	*	*	*	*	97.16	99.78
in238	87004.0781	49.72	98.22	90.70	*	*	*	99.57	*	98.70	*
in239	81435.3020	41.92	99.86	92.88	99.86	*	*	*	*	98.41	98.41
in240	86608.4120	45.38	98.11	98.61	*	*	*	*	*	95.04	98.90
in241	89961.1641	39.00	98.80	92.63	*	*	*	*	*	98.80	98.80
in242	92480.5420	35.73	90.80	91.44	92.68	*	*	*	*	95.12	96.56
in243	91839.5970	37.24	99.91	91.99	99.91	*	*	*	*	96.47	96.47
in244	91029.7940	42.40	95.61	92.89	98.11	*	*	98.11	*	97.04	97.04
in245	90590.5630	34.27	95.38	96.10	96.10	*	*	*	*	94.46	99.06
in246	87158.2344	24.83	99.39	*	*	*	*	99.17	*	99.39	99.17
in247	89044.3828	45.42	96.32	96.01	*	*	*	*	*	99.54	99.54
in248	93058.1406	57.39	91.53	92.92	*	*	*	*	*	95.73	95.73
in249	95169.5190	62.17	93.98	93.98	98.01	*	*	*	*	96.22	96.22
in250	93775.8359	48.37	*	*	*	*	*	*	*	98.82	98.82
in251	88734.0770	43.94	92.56	92.35	96.09	*	*	96.42	*	96.11	96.11
in252	89504.9220	53.90	93.49	98.03	98.03	*	*	*	*	99.86	99.86
in253	88253.3125	24.40	95.78	96.70	*	*	*	*	*	96.87	*

Continue on next page...

Table D.13: (continued).

Inst.	Best	CORAL	CPLEX	RG <sub>RK</sub>	BO <sub>MA</sub>	CA <sub>RA</sub>	CA <sub>LP</sub>	GA <sub>RA</sub>	GA <sub>LP</sub>	SD <sub>RA</sub>	SD <sub>LP</sub>
in254	85897.5010	31.00	96.01	96.37	96.37	*	*	*	*	98.85	*
in255	89368.1990	37.11	94.69	94.32	98.13	*	*	97.74	*	98.67	98.67
in256	89253.2656	38.86	93.03	92.12	*	*	*	96.74	*	*	95.03
in257	88605.5950	12.67	96.54	94.88	99.49	*	*	*	*	97.23	99.17
in258	85183.9110	44.59	99.33	97.74	98.65	*	*	*	*	*	*
in259	95397.3516	37.58	*	87.77	*	*	*	*	*	93.80	93.80
in260	90407.2050	42.48	99.25	92.46	96.03	*	*	*	*	99.38	*
in261	89790.1900	46.72	*	92.80	*	*	*	*	*	97.30	97.30
in262	88470.1100	50.02	*	92.98	99.03	*	*	*	*	96.68	*
in263	93087.8530	37.55	94.59	94.24	97.35	*	*	*	*	98.98	98.98
in264	86498.9141	48.56	97.86	91.86	*	*	*	*	*	99.00	99.00
in265	83621.1700	41.51	95.48	97.91	98.94	*	*	98.47	*	*	99.16
in266	90038.9920	31.15	96.12	94.84	98.48	*	*	*	*	98.50	98.50
in267	91438.2109	24.48	99.40	92.66	*	*	*	*	*	*	*
in268	89482.2790	41.41	97.93	93.04	99.78	*	*	*	*	98.80	98.80
in269	83546.6830	48.56	99.19	96.77	99.88	*	*	*	*	99.46	99.46
in270	87509.4062	34.87	97.20	92.81	*	*	*	*	*	95.73	95.73
in271	85951.6810	42.83	95.87	93.42	97.72	*	*	*	*	98.51	98.51
in272	88642.8220	49.07	92.82	95.86	97.28	*	*	*	*	*	*
in273	87909.9070	39.27	99.20	95.06	99.96	*	*	99.21	*	*	*
in274	83417.7890	45.77	98.89	93.02	99.18	*	*	98.89	*	98.33	98.33
in275	89915.1500	37.12	98.05	94.19	99.17	*	*	99.57	*	98.79	98.79
in276	86626.4375	50.65	99.36	99.40	*	*	*	99.78	*	97.18	99.36
in277	88537.7270	37.49	96.56	98.09	98.79	*	*	98.79	*	96.58	96.58
in278	91326.9531	52.88	95.55	96.72	*	*	*	99.28	*	99.28	99.28
in279	87058.9800	46.83	*	89.91	96.27	*	*	98.88	*	97.82	98.45
in280	86529.5938	38.92	97.14	93.11	*	*	*	*	*	99.46	99.46
in281	88470.4141	53.98	96.51	95.42	*	*	*	*	*	99.75	99.75
in282	88985.3290	46.25	92.23	91.56	96.27	*	*	*	*	95.57	96.27
in283	88915.6590	49.94	95.47	96.33	96.33	*	*	*	*	*	*
in284	88241.9750	39.20	96.30	96.86	96.86	*	*	*	*	*	97.14
in285	85953.2490	45.59	96.57	93.35	97.89	*	*	*	*	99.16	99.16
in286	88323.4844	57.41	92.98	*	*	*	*	*	*	92.53	92.31
in287	91652.7400	32.42	99.46	88.37	99.46	*	*	*	*	93.58	*
in288	85639.0090	41.68	95.93	96.81	97.90	*	*	*	*	96.81	96.17
in289	86032.8140	49.01	96.03	91.94	96.03	*	*	*	*	97.12	97.48
in290	92103.2070	42.87	95.79	90.63	96.06	*	*	*	*	94.55	95.24
in291	94188.2910	59.11	92.98	95.55	96.30	*	*	*	*	94.64	94.64
in292	94063.9650	57.29	97.14	95.96	96.56	*	*	*	*	97.90	*
in293	85810.6210	51.78	98.18	95.25	98.82	*	*	*	*	97.04	*
in294	91167.3160	49.30	94.62	91.92	96.80	*	*	*	*	*	*
in295	89267.5156	34.74	94.01	93.08	*	*	*	*	*	*	95.56
in296	90000.2970	22.43	98.55	93.35	98.55	*	*	*	*	*	*
in297	89725.9360	27.45	*	94.05	94.84	*	*	*	*	97.43	97.43
in298	89166.7422	47.96	98.65	93.52	*	*	*	*	*	*	98.77
in299	92218.6094	41.60	98.03	95.49	*	*	*	99.53	*	93.10	93.10
in300	88373.3281	48.68	97.91	*	*	*	*	*	*	99.30	96.75

Table D.14: Best results for LG 1500/1500 instances.

Inst.	Best	CORAL	CPLEX	RG <sub>RK</sub>	BO <sub>MA</sub>	CA <sub>RA</sub>	CA <sub>LP</sub>	GA <sub>RA</sub>	GA <sub>LP</sub>	SD <sub>RA</sub>	SD <sub>LP</sub>
in601	108800.4450	58.25	95.76	91.03	96.77	*	*	98.18	*	97.43	97.43
in602	105611.4760	24.41	93.92	92.95	95.78	*	*	*	*	94.12	94.12
in603	105121.0220	39.04	92.40	88.06	92.54	*	*	97.57	*	*	*
in604	107733.8050	50.52	96.29	96.13	96.29	98.96	98.96	98.96	*	98.04	98.04
in605	109840.9840	52.38	93.23	92.38	94.98	*	*	*	*	*	*
in606	107113.0670	37.26	92.47	97.42	98.23	*	*	*	*	93.83	93.83
in607	113180.2840	43.74	90.23	93.54	93.54	*	*	*	*	91.09	*
in608	105266.1070	50.50	96.26	88.48	97.88	*	*	*	*	99.07	99.07
in609	109472.3320	3.23	96.71	90.87	95.77	*	*	*	*	94.18	95.55
in610	113716.9650	32.91	93.89	88.04	95.86	*	*	98.04	*	95.87	98.04
in611	106666.3438	10.31	94.57	88.69	*	*	*	*	*	98.76	98.76
in612	109796.7400	54.31	*	96.59	*	*	*	*	*	96.91	94.91
in613	107980.1570	71.82	93.49	86.40	93.09	*	*	*	*	96.58	*
in614	108364.5859	49.66	*	89.98	*	*	*	*	*	*	*
in615	110508.8281	37.36	97.15	86.14	*	*	*	98.92	*	92.40	92.40
in616	109740.4922	44.02	88.30	91.67	*	*	*	*	*	95.39	96.64
in617	113302.4340	45.99	92.27	91.02	93.78	*	*	*	*	91.75	95.29
in618	111385.0810	47.03	94.81	88.56	99.93	*	*	99.58	*	95.53	99.57
in619	107571.5930	43.20	97.72	90.46	94.97	*	*	*	*	97.72	*
in620	110937.9750	59.54	92.13	93.65	95.07	*	*	97.96	*	97.96	96.75
in621	106133.8500	41.62	*	93.40	93.40	*	*	*	*	98.14	98.14
in622	107551.7370	55.58	91.12	94.14	98.49	*	*	*	*	96.71	97.53
in623	109487.0290	42.38	94.77	94.57	94.99	*	*	*	*	96.90	96.90
in624	104386.9790	48.61	92.76	92.27	95.69	*	*	*	*	*	*
in625	109065.3594	43.83	96.90	88.55	*	*	*	*	*	94.30	97.57
in626	114704.0340	50.12	89.36	88.28	96.60	*	*	*	*	97.77	92.88
in627	108846.2344	37.55	91.65	95.31	*	*	*	99.17	*	99.17	99.17
in628	108169.6953	42.91	94.53	*	*	*	*	97.07	*	97.51	96.74
in629	107929.2600	40.10	95.76	94.64	97.16	*	*	*	*	98.12	98.29
in630	105830.0620	54.00	94.55	93.68	99.65	*	*	*	*	99.75	99.75
in631	116505.2440	31.18	94.57	94.02	96.91	*	*	*	*	*	*
in632	104631.7140	52.88	92.98	90.76	95.18	*	*	99.59	*	98.28	*
in633	105564.4000	65.72	*	90.90	98.72	*	*	*	*	94.20	94.02
in634	108901.7300	47.86	94.31	91.85	93.34	*	*	*	*	93.80	93.80
in635	112902.6340	39.75	92.44	86.62	92.44	*	*	*	*	94.72	94.72
in636	106574.7480	48.82	92.64	91.03	98.23	*	*	99.07	*	97.76	93.89
in637	107989.7280	33.07	92.70	91.77	99.01	*	*	*	*	99.01	99.01
in638	112899.6320	30.01	97.48	88.95	92.88	*	*	97.48	*	97.48	97.48
in639	108894.4550	43.77	92.99	94.68	95.35	*	*	*	*	*	*
in640	108275.1328	53.59	96.04	91.48	*	*	*	99.08	*	96.60	96.60
in641	109744.0625	56.06	98.77	92.27	*	*	*	99.73	*	98.77	98.77
in642	114182.9688	40.41	91.12	90.13	*	*	*	*	*	*	*
in643	104015.0240	13.04	94.62	91.75	97.97	*	*	*	*	97.97	*
in644	108025.7490	60.10	98.22	93.57	98.34	*	*	99.03	*	99.03	98.25
in645	105841.6720	37.99	92.97	90.05	96.51	*	*	*	*	96.08	97.25
in646	107800.1030	33.71	93.84	94.90	95.98	*	*	*	*	*	*
in647	107701.7109	51.25	90.90	93.95	*	*	*	95.92	*	95.37	95.37
in648	105790.5900	37.28	*	96.66	*	*	*	99.93	*	99.55	99.55
in649	107587.3710	40.30	88.79	94.52	95.70	*	*	98.63	*	*	99.79
in650	103330.9010	45.80	92.36	93.79	96.86	*	*	*	*	97.02	97.50
in651	103827.2970	55.97	95.17	94.21	98.85	*	*	*	*	*	*
in652	107760.2480	28.48	94.24	97.48	97.48	*	*	*	*	97.20	96.19
in653	113946.4766	34.60	91.38	89.41	*	*	*	*	*	94.22	94.22

Continue on next page...

Table D.14: (continued).

Inst.	Best	CORAL	CPLEX	RG <sub>RK</sub>	BO <sub>MA</sub>	CA <sub>RA</sub>	CA <sub>LP</sub>	GA <sub>RA</sub>	GA <sub>LP</sub>	SD <sub>RA</sub>	SD <sub>LP</sub>
in654	111738.2310	35.91	94.25	89.29	98.26	★	★	★	★	★	★
in655	111785.0640	44.33	91.74	88.65	97.39	★	★	★	★	96.13	96.13
in656	112259.2750	43.93	90.25	96.67	96.67	★	★	★	★	94.21	95.51
in657	112708.6560	37.47	★	93.86	96.88	★	★	★	★	95.65	★
in658	110751.5340	38.17	91.70	91.29	93.87	★	★	96.80	★	★	★
in659	106545.4270	39.45	94.76	96.16	96.16	★	★	99.03	★	★	★
in660	112293.6080	39.96	98.65	91.81	96.61	★	★	99.72	★	99.72	98.65
in661	113106.6290	30.29	97.20	87.06	92.81	★	★	★	★	95.86	97.63
in662	108298.0790	58.18	97.41	91.26	91.26	★	★	★	★	94.29	94.29
in663	104826.7800	52.39	95.93	95.40	95.40	★	★	★	★	99.03	92.33
in664	112866.8650	42.89	95.93	91.67	94.66	★	★	99.38	★	99.68	99.68
in665	113002.6720	39.05	98.78	94.75	97.33	★	★	98.78	★	96.68	96.68
in666	106441.1562	46.49	★	91.88	★	★	★	★	★	98.54	98.54
in667	104683.7500	65.93	97.55	91.77	★	★	★	★	★	97.75	97.75
in668	107483.1580	45.33	94.12	93.41	98.89	★	★	★	★	98.12	98.12
in669	108163.4690	42.49	97.80	93.65	96.78	★	★	★	★	95.71	94.23
in670	110200.8160	50.35	94.90	92.73	96.53	★	★	★	★	99.98	99.85
in671	109306.8438	48.13	99.75	★	★	★	★	★	★	99.75	99.75
in672	107534.8870	43.05	93.33	95.58	95.58	★	★	★	★	96.40	95.58
in673	112320.2500	44.61	92.20	92.34	★	★	★	★	★	98.43	98.43
in674	109558.2344	37.01	91.87	87.59	★	★	★	★	★	95.40	95.40
in675	108131.9880	47.04	★	92.14	97.81	★	★	★	★	★	★
in676	107052.1910	37.62	94.64	88.05	96.10	★	★	★	★	★	★
in677	107831.5370	45.33	96.19	97.47	99.68	★	★	99.68	★	97.03	97.22
in678	102422.8290	29.45	★	95.83	96.87	★	★	★	★	★	★
in679	107982.4560	48.90	90.90	92.06	99.21	★	★	98.61	★	96.35	96.11
in680	107500.5000	44.67	96.69	91.50	★	★	★	★	★	98.92	98.92
in681	105237.2870	53.94	93.19	92.10	99.85	★	★	★	★	★	★
in682	107948.1260	38.39	97.33	89.67	98.25	★	★	★	★	98.25	99.93
in683	107777.6130	7.06	95.19	93.47	96.08	★	★	★	★	98.25	98.25
in684	114153.7410	62.07	91.14	85.74	94.52	★	★	★	★	92.55	94.26
in685	106686.6160	39.69	94.83	92.42	92.74	★	★	97.71	★	97.81	97.81
in686	106364.3580	19.52	99.45	92.48	98.55	★	99.53	99.45	99.53	97.70	97.70
in687	108301.4710	44.81	97.06	94.98	97.05	★	★	★	★	99.10	99.10
in688	112012.5703	50.12	93.49	94.55	★	★	★	99.83	★	95.27	97.60
in689	105968.1680	48.45	92.72	94.96	97.70	★	★	★	★	98.39	98.39
in690	108489.7109	34.23	92.05	92.90	★	★	★	★	★	97.02	97.02
in691	105564.6090	37.21	93.06	96.78	96.78	★	★	★	★	98.50	98.50
in692	109226.0700	44.39	93.71	91.40	97.15	★	★	★	★	98.99	97.40
in693	106719.6950	31.31	97.08	93.34	97.58	★	★	99.56	★	99.56	97.08
in694	114477.0540	47.90	89.45	94.44	94.44	★	★	96.94	★	93.66	93.66
in695	110240.9860	14.09	91.30	93.91	93.91	★	★	98.04	★	★	★
in696	104559.9530	39.47	★	95.00	98.94	★	★	★	★	99.43	99.43
in697	105958.6570	23.47	98.78	92.49	98.46	★	★	★	★	98.78	98.78
in698	105463.0312	26.21	95.12	92.14	★	★	★	★	★	97.86	97.77
in699	107132.3340	41.24	96.37	96.85	98.42	★	★	99.26	★	99.14	99.14
in700	106730.6770	45.46	95.37	95.11	95.11	★	★	★	★	97.09	94.31