UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

FERNANDO STEFANELLO

Heuristic approaches for network problems

Thesis presented in partial fulfillment
of the requirements for the degree of
Doctor of Computer Science

Advisor: Prof. Dra. Luciana S. Buriol
Coadvisor: Prof. Dr. Mauricio G. C. Resende

Porto Alegre
2015

CIP — CATALOGING-IN-PUBLICATION

Stefanello, Fernando

Heuristic approaches for network problems / Fernando Stefanello. – Porto Alegre: PPGC da UFRGS, 2015.

115 f.: il.

Research supported by CAPES and project PRH PB-217 of Petrobras, Brazil.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2015. Advisor: Luciana S. Buriol; Coadvisor: Mauricio G. C. Resende.

1. Heuristic. 2. Network problems. 3. BRKGA. 4. Cloud computing. I. Buriol, Luciana S.. II. Resende, Mauricio G. C.. III. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos Alexandre Netto
Vice-Reitor: Prof. Rui Vicente Oppermann
Pró-Reitor de Pós-Graduação: Prof. Vladimir Pinheiro do Nascimento
Diretor do Instituto de Informática: Prof. Luis da Cunha Lamb
Coordenador do PPGC: Prof. Luigi Carro
Bibliotecária-chefe do Instituto de Informática: Beatriz Regina Bastos Haro

A minha família e
minha amada Doralice.

AGRADECIMENTOS

Agradeço a minha orientadora, Luciana Buriol, pela oportunidade da realização do doutorado, orientação, ajuda e incentivo. Agradeço também ao meu coorientador Mauricio G. C. Resende, especialmente pelo inestimável apoio durante a realização do doutorado sanduíche.

Agradeço a minha família. Vocês foram e são a base forte nos momentos difíceis e fonte de inúmeras alegrias. Mais especificamente agradeço aos meus pais Edemar e Clede. Se fisicamente não pude estar mais presente, meu coração sempre esteve e estará com vocês. Vocês são a base de tudo, meu porto seguro, meus exemplos. Aos meus irmãos Edevandro e Sandra, o apoio de vocês foi mais que decisivo para que hoje eu pudesse estar concluindo mais esta etapa. Vocês são especiais! Aos meus cunhados Maicon e Gisele. Vocês são mais que cunhados, são meus irmãos. Aos meus afilhados Gustavo, Gabriela e minha sobrinha Letícia (e Larissa que está por vir). Vocês representam muito para mim.

Aos meus colegas de laboratório, em especial ao Árton com quem dividi um apartamento barulhento por anos. Obrigado a todos pelas inúmeras discussões e ideias.

À querida Lúcia Resende, que juntamente com o Mauricio me receberam tão bem no meu período de sanduíche nos EUA. Agradeço ainda, à Julliany Brandão e ao Carlos Eduardo de Andrade pela valiosa companhia e amizade.

A todos os professores, colaboradores e revisores que contribuíram no desenvolvimento deste trabalho.

A CAPES e ao projeto PRH PB-217 da Petrobras pelo suporte financeiro. Ao Instituto de Informática da UFRGS pela oportunidade e pela infraestrutura fornecida. A AT&T Lab Research pela oportunidade da realização do doutorado sanduíche.

A todos os meus amigos e amigas que estiveram presentes me aconselhando e incentivando com carinho e dedicação. A todas as pessoas que, direta ou indiretamente, contribuíram de uma forma ou de outra para a conclusão desta tese, o meu mais profundo e sincero muito obrigado.

Por fim, gostaria de dedicar um agradecimento especial a minha amada noiva Doralice Gall pelo inestimável apoio, paciência, carinho e compreensão durante essa longa jornada. Sempre faltarão palavras para te agradecer. Te amo!

ABSTRACT

In our highly connected world, new technologies provide continuous changes in the speed and efficiency of telecommunication and transportation networks. Many of these technologies come from research on network optimization problems with applications in different areas. In this thesis, we investigate three combinatorial optimization problems that arise from optimization on networks. First, traffic engineering problems in transportation networks are addressed. The main objective is to investigate the effects of changing the cost of some links in the network regarding some well-defined user behavior. The goal is to control the flow in the network and seek a better flow distribution over the network and then minimize the traffic congestion or maximize the flow on a subset of links over network conditions. The first problem considered is to install a fixed number of tollbooths and define the values of tariffs to minimize the average user travel time. The second problem considered is to define the values of tariffs to maximize the revenue collected in the tolled arcs. In both problems, users choose the routes based on the least cost paths from source to destination. From telecommunication networks, a placement problem subjected to network conditions is considered. The main objective is to place a set of resources minimizing the communication cost. An application from cloud computing is considered, where the resources are virtual machines that should be placed in a set of data centers. Network conditions, such as bandwidth and latency, are considered in order to ensure the service quality. For all these problems, mathematical models are presented and evaluated using a general-purpose commercial solver as an exact method. Furthermore, new heuristics approaches are proposed, including some based on biased random-key genetic algorithm (BRKGA). Experimental results demonstrate the good performance of the proposed heuristic approaches, showing that BRKGA is an efficient tool for solving different kinds of combinatorial optimization problems, especially over network structures.

Keywords: Heuristic. Network problems. BRKGA. Cloud computing.

Abordagens heurísticas para problemas em redes

RESUMO

Em nosso mundo altamente conectado, novas tecnologias provêm contínuas mudanças na velocidade e eficiência das redes de telecomunicações e de transporte. Muitas dessas tecnologias são originárias de pesquisas em problemas de otimização em redes aplicadas a diferentes áreas. Nesta tese, investigamos três problemas de otimização combinatória que podem ser abordados como estruturas de redes. Primeiramente, são abordados problemas de engenharia de tráfego em redes de transporte. O objetivo principal é investigar os efeitos de alterar o custo de um subconjunto de arcos da rede, considerando que os clientes desta rede agem com um comportamento bem definido. O objetivo é controlar o fluxo na rede de modo a obter uma melhor distribuição do fluxo, minimizando o congestionamento ou maximizando o fluxo em um subconjunto de arestas. No primeiro problema considera-se instalar um número fixo de postos de pedágios e definir os valores das tarifas para minimizar o tempo médio de viagem dos usuários. No segundo problema abordado, o objetivo é definir os valores das tarifas para maximizar a receita arrecadada nos arcos com pedágios. Em ambos os problemas, os usuários escolhem as rotas com base nos caminhos de menor custo da origem para o destino. Em redes de telecomunicações, um problema de alocação sujeito às condições da rede é considerado. O objetivo é alocar um conjunto de recursos, minimizando o custo de comunicação. Uma aplicação de computação em nuvem é considerada, onde os recursos são máquinas virtuais que devem ser alocadas em um conjunto de centros de dados. Condições da rede como largura de banda e latência são consideradas de modo a garantir a qualidade dos serviços. Para todos estes problemas, os modelos matemáticos são apresentados e avaliados usando um solver comercial de propósito geral como um método exato. Além disso, abordagens heurísticas são propostas, incluindo uma classe de algoritmo genético de chaves aleatórias viciadas (BRKGA). Resultados experimentais demonstram o bom desempenho das abordagens heurísticas propostas, mostrando que o BRKGA é uma ferramenta eficiente para resolver diferentes tipos de problemas de otimização combinatória, especialmente sobre estruturas de rede.

Palavras-chave: Heurística. Problemas em redes. BRKGA. Computação em nuvem.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

In our highly connected world, new technologies provide continuous changes in the speed and efficiency of telecommunication and transportation networks. Many of these technologies come from research on network optimization problems with applications in different areas. Connections as roads in transportation networks, links between computers in telecommunication networks, and the relation between people in social networks are some examples of connections that can be modeled as a network topology. These networks have common aspects as, for example, be easily represented by graphs, allowing to represent many different optimization network problems over this mathematical structure.

In a transportation network, predicting the flow distribution in the network allows a service distribution company to guide some agents to take alternative routes and thus minimize the transportation costs. Also, estimating the user behavior makes it possible to determine the flow in the network, allowing network administrators agents make decisions to reduce congestion at peak hours. In general, knowing how the flow behaves in a network helps to predict the behavior and provides information for decision-making, enabling the economy of resources, a better usage of the infrastructure, a better quality of services among other numerous benefits. Having knowledge of how the flow moves in a network, we can control it. Control the flow means having at hand the possibility to make decisions to supply a demand, reduce costs, or achieve a desired goal.

Telecommunication networks comprise the telephone system and data transmission systems over the Internet, for example. In these networks, the flow can mean data packets or phone calls travelling through computers or mobile phones. Ensuring connections between the source and destination nodes, allowing the exchange of information without violating the network capacity while the operational cost is minimized is an example of a optimization problem coming from these networks. In general, the objective is to obtain a least cost network configuration ensuring that constraints related to the network flow such as delay, reliability, and link capacity are respected.

This thesis investigates some optimization problems that can be described over network structures. Traffic engineering problems in transportation networks are considered. The main objective is to investigate the effects of changing the cost of some links in the network regarding some well-defined user behavior. The goal is to control the flow in the network seeking a better flow distribution over the network and then minimizing the traffic congestion or maximizing the flow on a subset of links over network conditions.

From telecommunication network, a placement problem subjected to network conditions is considered. The goal is to define a placement for a set of resources minimizing the communication cost, that implies a direct impact on the network structures.

The problems studied in this thesis are described in a contextualized manner over applications in networks. At the same time, we also provide the description through a formal mathematical structure, defining the problem more accurately. The main objective is to develop algorithms to obtain solutions for the problems, so that the algorithm can solve every instance of the problem, including the practical scenarios. Algorithms are used to find optimal solutions or analyze which scenarios can be solved exactly. As an exact approach, we use mathematical programming considering a nonlinear model with a convex cost function, a bilevel model, a quadratic model, and linear models. The linear models are reached from the previously cited nonlinear models using reformulations techniques while piecewise-linear functions have been proposed to approximate the convex cost function. When it is not possible to solve a problem optimality, heuristic algorithms are proposed to obtain near-optimal solutions. A class of evolutionary algorithm called biased random-key genetic algorithm is widely used in this work to successfully solve theses problems. Intensification strategies as local search and path-relinking methods are also used to improve the quality of the obtained solution.

In the next subsections, we review some basic concepts considered in the following chapters. Section 1.1 reviews some concepts of combinatorial optimization and the main techniques used to solve problems of this class. Section 1.2 presents a general framework of the biased random-key genetic algorithm that is the most used technique to solve the problems addressed in this thesis. Finally, Section 1.3 presents the organization and the main contributions of this thesis.

1.1 Definitions and main techniques

Combinatorial optimization is a lively field of applied mathematics with the objective to solve optimization problems over discrete structures, by combining techniques from combinatorics, linear programming, and the theory of computation. One of the main challenges of combinatorial optimization is to develop efficient algorithms, with their corresponding running times bounded by a polynomial of the same the size as their representation (POP, 2012). In general, the objective is to solve problems related to resource allocation, typically limited to achieve certain goals. Such problems arise in many practical areas, such as the development of digital circuits, scheduling, facility location, assignment, transportation planning, among others. The importance of these problems in the industry cannot be underestimated. For example, Anbil et al. (1991) describes a crew scheduling problem in an airline whose operating costs are in the order of billions of dollars, and even small improvements in staffing efficiency can lead to substantial cost reduction.

Combinatorial optimization problems are often easy to describe but difficult to solve. The difficulty comes from finding a solution that receives the highest rating possible and at the same time satisfy all the imposed constraints. This solution is called optimal solution. Whereas there is a discrete and usually finite set of solutions, a combinatorial optimization problem can be solved generating, evaluating and comparing solutions. Thus, inspecting all solutions of the problem domain ensures to find the best solution or proves problem infeasibility. Therefore, any problem can be solved in principle, but that does not mean it can be solved in an acceptable time. This is mainly addressed in Complexity Theory (GAREY; JOHNSON, 1979), which deals with the efficiency of computing algorithms and classify them according to the difficulty to solve.

Since many combinatorial optimization problems are characterized as NP-hard, combinatorial optimization has challenged researchers from different areas, including mathematics, computer science, engineering, economics and management. There are several methods for solving combinatorial optimization problems, and the solution approaches are basically exact and heuristic.

Exact algorithms have an important feature that is the guarantee of finding the optimal solution when carried out completely. However, these methods are often effective only in small and medium-size instances, since the runtime often increases exponentially with the size of the instance, limiting the practical use of these algorithms. Methods such branch-and-bound (LAWLER; WOOD, 1966), branch-and-cut (GOMORY, 1958), and branch-and-price (BARNHART et al., 1998), are among the most used exact algorithms. These algorithms check the gap between the lower and upper bound values for feasible solutions, and use techniques to prune the search space.

Many of these techniques are designed to be flexible and independent of the problem domain, to be applicable to a wide variety of practical problems without an intensive use of specific strategies. Research on independent domain techniques for combinatorial optimization has resulted in general purpose tools to mixed integer programming such as

CPLEX[1], GUROBI[2], COIN[3], LINDO[4], GLPK[5], to name a few. Once these techniques operate with mathematical models to represent a problem, they provide the possibility to quickly adapt the change of requirements that often occurs in real environments.

A heuristic is a method to obtain good solutions 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 are methods to obtain an initial solution to a given problem, in which the solution is built element by element following a well-defined sequence of steps. Local search heuristic usually starts from a feasible solution and, at each iteration, explores the search space using one or more neighborhood structures seeking for changes that lead to improvement in the solution. While heuristics are developed to a particular problem, metaheuristics are developed to be a more general approach to guide in a more efficiently way the search process. Techniques as the genetic algorithm (GOLDBERG, 1989), simulated annealing (KIRKPATRICK et al., 1983; ČERNÝ, 1985), ant colony optimization (DORIGO et al., 1996), tabu search (GLOVER, 1986; GLOVER; LAGUNA, 1993), greedy randomized adaptive search procedure (FEO; RESENDE, 1989; RESENDE; RIBEIRO, 2010), variable neighborhood search (MLADENOVIĆ; HANSEN, 1997; HANSEN et al., 2010), and biased random-key genetic algorithm (GONÇALVES; RESENDE, 2011) are some examples of metaheuristics successfully applied to combinatorial optimization problems. In general, these techniques are specialized for solving a specific class of optimization problems, unlike the exact methods that are more general in the sense that they can be applied to various types of structures. The utilization of a metaheuristic is, in many cases, the only way to find good solutions for large instances. In this case, the guarantee of finding optimal solutions is sacrificed for the sake of obtaining good solutions in a reasonable time. In Blum and Roli (2003), Talbi (2009), Gendreau and Potvin (2010) the authors present an overview of heuristic methods.

Among classes of heuristic methods, approximation algorithms can produce, in polynomial time, a solution with quality guarantee. These algorithms provide solutions with an approximation factor to the optimal solution. This thesis does not address approximation algorithms, but an interested reader can find more details and examples in Vazirani (2003) and Williamson and Shmoys (2011).

[1]<www.cplex.com>
[2]<www.gurobi.com>
[3]<www.coin-or.org>
[4]<www.lindo.com>
[5]<www.gnu.org/software/glpk>

1.2 Biased random-key genetic algorithm

In this section, we describe the general framework of the biased random-key genetic algorithm (BRKGA), since this algorithm is widely used to solve the problems addressed in this thesis. The algorithm can be described in problem-dependent component and problem-independent component. In this section, we describe the problem-independent part, while the problem-dependent part are presented in the corresponding chapter of the problem.

A biased random-key genetic algorithm is a metaheuristic for finding optimal or near-optimal solutions to optimization problems. BRKGAs encode solutions as vectors of random keys, i.e. randomly generated real numbers in the interval $(0, 1]$.

Algorithm 1 presents a general scheme of a BRKGA. The algorithm starts with a set (or *population*) of p random vectors of size n (line 2). Parameter n depends on the encoding while parameter p is user-defined. Starting from the initial population (lines 2 and 3), the algorithm generates a series of populations. Each iteration of the algorithm is called a *generation* (lines 4 to 10). The generation starts sorting the solution by they fitness value (line 5) and classifying the solutions as elite and non-elite (line 6). Following, elite solutions are copied to the next population (line 7), and mutants are added to the next population (line 8). The remaining individuals are generated by combining pairs of solutions from one generation to produce offspring solutions for the following generation (lines 9 and 10). The algorithm returns the best solution found during the process (line 11).

Algorithm 1: Pseudo-code of BRKGA

```
1  Procedure BRKGA()
2      Generate $p$ vectors of random keys;
3      Decode each vector of random keys;
4      while stopping criterion is not satisfied do
5          Sort solutions by their fitness;
6          Classify solutions as elite and non-elite;
7          Copy elite solutions to next population;
8          Generate mutants in the next population;
9          Combine an elite and a non-elite and add offspring to next population;
10         Decode each vector of random keys;
11     return best solution $S$;
```

BRKGAs rely on *decoders* to translate a vector of random keys into a solution of the optimization problem being solved. A decoder is a deterministic algorithm that takes as input a vector of random keys and returns a solution of the optimization problem as well as its cost (or *fitness*). The complexity of the decoder can range from very simple and direct as a mapping between the random-key and the solution, to a very complex construction that combine algorithms, or even black box computation.

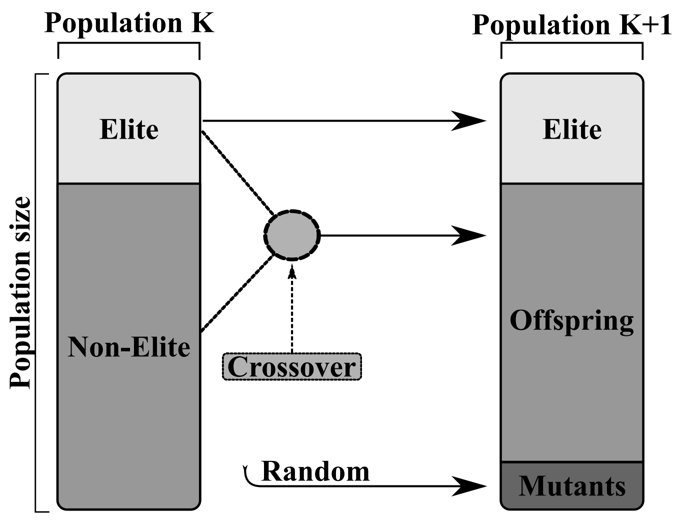
At each generation k, the decoder is applied to all newly created random keys and the population is partitioned into a smaller set of p_e elite solutions, i.e., the fittest p_e solutions in the population and another larger set of $p - p_e > p_e$ non-elite solutions. Population $k+1$ is generated as follows. All p_e elite solutions of population k are copied without change to population $k+1$. This elitist strategy maintains the best solution on hand. In biology, as well as in genetic algorithms, evolution only occurs if mutation is present. As opposed to most genetic algorithms, BRKGAs do not use a mutation operator, where each component of the solutions is modified with small probability. Instead p_m *mutants* are added to population $k+1$. A mutant is simply a vector of random keys, generated in the same way a solution of the initial population is generated.

With $p_e + p_m$ solutions accounted for population $k+1$, $p - p_e - p_m$ additional solutions must be generated to complete the p solutions that make up population $k+1$. This is done through *mating* or *crossover* (line 9). A parent-A is selected randomly from the elite solutions, and the parent-B is selected randomly between the set of non-elite solutions. A child C is produced by combining the parents using parameterized uniform crossover. Let $\rho_A > 1/2$ be the probability that the offspring solution inherits the key of parent-A and $\rho_B = 1 - \rho_A$ be the probability that it inherits the key of parent-B, i.e. $c_i = a_i$ with probability ρ_A or $c_i = b_i$ with probability $\rho_B = 1 - \rho_A$, where a_i and b_i are, respectively, the i-th key of parent-A and parent-B, for $i = 1, \ldots, n$.

Random-key genetic algorithms (RKGA) were first introduced in Bean (1994). BRKGA differs from the original approach in the way parents are chosen from the population and how the child inherits the key (GONÇALVES; RESENDE, 2011). Though the difference between RKGAs and BRKGAs is small, the resulting heuristics behave quite differently. Experimental results in Gonçalves et al. (2012) show that BRKGAs are almost always faster and more effective than RKGAs.

Figure 1.1 illustrates the evolutionary scheme of one iteration of BRKGA. On the left of the figure is the current population. After all vectors are sorted by their fitness values, the population is partitioned into two subsets, the elite and the non-elite. The elite vectors from the population k are copied without modification to the population $k+1$. After, the mutants are generated and added to the next population. The remainder of the population of the next generation is generated by crossover.

Figure 1.1 – General scheme of BRKGA evolution



Source: from the author (2015).

Gonçalves and Resende (2011) describe BRKGA as a general-purpose metaheuristic framework, where the framework has a clear division in a problem-dependent and problem-independent part. The problem-independent part operates without knowledge of the problem being solved. This involves the generation if the initial population, sort the population by fitness, classify in elite and non-elite, copy elite vector to the next population, generate the mutants, and the crossover operator that combines an elite and a non-elite to produce an offspring and add to next population. The only portion of the algorithm that has connection with the problem is the decoder process, which takes a random-key vector and produce a solution to the combinatorial optimization problem. To describe a BRKGA heuristic, one need only show how solutions are encoded and decoded, what choice of parameters were made, and how the algorithm stops.

The parameters that must be specified are the chromosome size n which define how the solution is encoded, population size p, the percentage of the elite vectors p_e, the percentage of the mutants vectors p_m, and the inheritance probability ρ_A. Additional parameters to restart the population after n generations without improvement and the number of the independent population also must be chosen if these features are used. The stop criterion for the algorithm can be defined as a time limit, a fixed number of generation, a number of generations without improvement, a number of restarts of the population, or a target solution value. Advice for the parameter setup can be found in Gonçalves and Resende (2011).

Biased random-key genetic algorithms have been successfully applied to many combinatorial optimization problems. For example, Resende (2012), Andrade et al. (2014), Andrade et al. (2015b), Andrade et al. (2015a), and Andrade (2015) uses BRKGA on some telecommunication problems. Buriol et al. (2010) use this technique for a transportation network problem. Gonçalves and Resende (2013) apply to 2D and 3D bin packing problem. Martinez et al. (2011) uses BRKGA for capacitated arc routing problem, while Ruiz et al. (2015) uses for capacitated mininum spanning three problem. Gonçalves and Resende (2011), Resende (2013), and Gonçalves et al. (2012) detail some experiments with the BRKGA framework and components, and provide an extended survey of applications and decodes for many applications of BRKGA. Toso and Resende (2014) proposed algorithms written in C++ on top of the BRKGA.

1.3 Results and thesis organization

This thesis presents a study of three optimization problems that can be represented over network structures. Chapters 2 and 3, present two problems with applications on transportation network planning. Chapter 4 presents a problem on telecommunication networks that is also an application on cloud computing. Each chapter is self-contained presenting the respective study of each problem, except for the references of the biased random-key algorithm that are presented in Section 1.2. Each chapter starts with a brief introduction to contextualize the application and also related works in the literature. Next, each problem is defined in terms of the mathematical formulation. Algorithms, experimental results and concluding remarks of each chapter are also presented.

Chapter 2 presents the *Tollbooth problem*. Supposing that users of a transportation network always select the least cost path from the source to the destination, in this bilevel problem we seek to install a fixed number of tollbooths and define the value of each toll with the objective of minimize a measure function of traffic congestion. This problem was first introduced in Buriol et al. (2010). From this work, we formalize mathematically the problem and propose two piecewise linear functions to approximate the convex cost function used to evaluate the congestion cost. Furthermore, we present a large study with the previously proposed biased random-key genetic algorithm considering a new arc weight value to calculate shortest paths. We also present a review of the algorithm components, such as the local search, and a more detailed review of the behavior of the algorithm, including a new set of instances and an analysis of characteristics of the final solutions.

We reported these contributions in two papers. The first (STEFANELLO et al., 2012), two piecewise linear functions are proposed and tested in general models of traffic flow. Experiments with BRKGA are also presented, including a new set of instances. The second (STEFANELLO et al., 2015c) presents the mathematical formulation for the problem, and a new arc weight value is considered. An evaluation of each BRKGA component is also presented.

Chapter 3 investigates the *Stackelberg network pricing problem*. Given a transportation network, supposing that the users always choose the least cost path route, in this problem the objective is to determine the values of tariffs of a given subset arcs in order to maximize the revenue collected in the tolled arcs. In this chapter, after the formalization of the problem, a biased random-key algorithm is proposed to solve this problem. An initial solution based on the values of tariffs obtained by solving the relaxed model is used to improve the algorithm. Experimental tests with a commercial solver show that this exact approach is limited to small instances, while the proposed BRKGA was an efficient approach in large instances. These results were reported in a conference paper (STEFANELLO et al., 2013).

Chapter 4 considers the problem of placement of virtual machines across geo-separated data centers (*VMPlacement problem*). In this problem, the objective is to minimize the traffic cost in a network. More specifically, give a set of geo-separated data centers, the goal is to select the best data center for allocating a set of virtual machines in order to minimize the communication cost between each pair of data centers. Additional network requirements that ensure the service quality are added to the problem. To the best of our knowledge, we are the first to introduce this problem considering a specific cost function and set of constraints, which is a generalization of the classic generalized quadratic assignment problem (GQAP). We formalize the problem mathematically using a quadratic model and two mixed integer linear models extended from the GQAP. Also, we propose local search with an intensive exploration of the neighborhood, and a path-relinking as an intensification method incorporated in two metaheuristic approaches. The first is a greedy randomized adaptive search procedure (GRASP), and the second is a biased random-key genetic algorithm. An extensive set of experiments is performed to evaluate different configurations of the algorithms. We test our algorithms in a set of synthetic instances where we show that the exact approach based on mathematical programming has limited performance. We also compare our approaches with a state-of-art algorithm for a set of instances from the literature for GQAP. We observe a good performance showing that the proposed approaches are competitive to produce good quality solutions. We reported the first experiments with BRKGA in a conference paper on genetic algorithms area (STEFANELLO et al., 2015a). A new mathematical formulation and comparison with the previously proposed mathematical model were reported in a national conference (STEFANELLO et al., 2015b). Finally, a new paper based on Chapter 4 is being prepared to be submitted to European Journal of Operational Research (EJOR).

Finally, Chapter 5 reviews and summarizes the main contributions. We also report some studies and strategies that did not obtain significant results or are promising direction for future research since they are not explored in depth in this work.

2 THE TOLLBOOTH PROBLEM

Urban transportation systems, which is composed of the network infrastructure, and public transport, is developed to provide mobility for users, beyond the carriage of products and consumer goods. Changes in the transport system involve considerable costs, besides to affecting user behavior that needs spend a time to adapt and follow the new rules. Therefore the importance of making changes that actually improve the traffic condition, with a correct estimation of the impact regarding congestion, travel time, emissions, safety and other factors.

These estimates are made using models of transportation systems, providing a simplified description based on the behavior of the users. These models developed for traffic assignment can be categorized as microscopic, mesoscopic and macroscopic. In microscopic models, the trajectory and behavior of each vehicle are described in detail, including position, speed, and acceleration. The simulation of the interaction between each vehicle in the system is a result of the relationship between the flow and travel time. This relationship can not be expressed through well-defined functions since this relationship depends on the interaction and behavior of users over time. These models are often used to design new roads configurations, and analyze the interactions between traffic components. In macroscopic models, the traffic conditions are described in terms of measures as flow and speed. Also, measures that express the relation between cost and flow are generally described in a simplified manner. Mesoscopic models represent a balance between microscopic and macroscopic models. Usually model a single vehicle behavior or in groups.

Macroscopic model is divided into static and dynamic. If the demand and the flows are not time-dependent, the model is considered *static*. Static models are used to describing the average journey times, traffic flows, and the flow demand for a single period. On the other hand, if the time-dependence is considered, the model is considered *dynamic*. Dynamic models can describe a flow congestion over time and allow the introduction of the time dependence for travellers, for the flow, and for the traffic demand. Naturally, dynamic models are more complex, both computationally and conceptually, that static models. Even that the flow is never time-independent, for a particular analysis, such as analysis of the flow in peak time, static models can provide good approximation.

The demand matrix in equilibrium problems and traffic allocation are often divided into two cases: fixed demand and elastic demands. In the fixed demand case, the origin-destination demand matrix is assumed given. For elastic demands, the demands are modelled as a function of least travel cost between each origin-destination pair.

Another important topic to be addressed in transportation networks is the cost that a flow unit has to transverse a link, i.e., the cost to user define your route. In general, the cost is a composition of n different factor. Mathematically, the cost in a arc 'a' can be represented by $w'_a(\ell_a) = \sum_{i=1}^{n} \gamma_i f_i(\ell_a)$, where γ_i is a conversion factor for the unit cost of the function f_i, that in general depends of ℓ_a. However, in the cases where is flow-independent, w'_a can be a constant value.

In this chapter, we address the Tollbooth problem that is an approach to control and guide the flow in a transportation network. Our approach is based on static macroscopic models, considering fixed demand matrix, and constant values for link costs. Supposing that, in general, users choose your routes based on a well-defined creation, as the shortest path route for example, changing the cost of some roads or arcs in the network it is possible to induce users to take alternative routes, i.e., guide the flow to take different paths and consequently obtain a better distribution of the flow in the network. Knowing how the flow is distributed over the network, it is possible to evaluate this distribution by many objective cost function, as congestion cost, the level of pollutants emission, fuel consumption, delays, etc. In this work, the focus is to obtain a flow distribution such that the average user travel time delay is minimized. To control the flow we evaluate different scenarios by the number of arcs that the cost is changed. In the analogy to the transportation network, we evaluate install tolls on a different number of roads and evaluate the efficiency to induce users to take alternative routes and the quality of the objective function. We extend the experiments of Buriol et al. (2010), by including different ways to evaluate the shortest path route, formalization of the problem through mathematical formulation, and extensive experiments with the biased random-key genetic algorithms. This chapter is based on Stefanello et al. (2015c).

2.1 Introduction

Transportation systems play an important role in modern life. Due to population growth and the massive production of vehicles, traffic congestion problems in metropolitan areas have become a common daily occurrence. To a commuter or traveler, congestion means loss of time, potentially missed business opportunities, and increased stress and frustration. To an employer, congestion means lost worker productivity, reduced trade opportunities, delivery delays, and increased costs (WEN, 2008). For example, a significant aspect is the value of wasted fuel and loss of productivity. In 2010, traffic congestion cost about US$115 billion in the 439 urban areas of the United States alone (SCHRANK et al., 2011).

Minimizing driving time directly impacts quality of life. One way to reduce travel time is by lowering congestion through the redistribution of traffic throughout the network. Improvements in transportation systems require a careful analysis of several factors. Different alternatives are evaluated using models that attempt to capture the nature of transportation systems and thus allow the estimation of the effect of future changes in system performance. Performance measures include efficiency in time and cost, security, and social and environmental impact, among others.

Several strategies have been proposed to reduce traffic congestion. Among them, the deployment of tolls on certain roads can induce drivers to choose alternative routes, thus reducing congestion as the result of better traffic flow distribution. Naturally, tolls can increase the cost of a trip, but this can be compensated with less travel time, reduced fuel cost, and lower amounts of stress. In the 1950s, Beckmann et al. (1956) proposed the use of tolls with this objective. This idea has made its way into modern transportation networks. In 1975, Singapore implemented a program called *Electronic Road Pricing* or ERP. Several cities in Europe and the United States, such as in London and San Diego, have begun to charge toll on their transportation networks (BAI et al., 2010). In fact, tolls are being deployed for traffic engineering in many small as well as large cities around the world.

Determining the location of tollbooths[1] and their corresponding tariffs is a combinatorial optimization problem. This problem has aroused interest in the scientific community not only because of its intrinsic difficulty, but also because of the social importance and

[1]We use the term *tollbooth* to refer to both traditional tollbooths as well as to sensors that read radio-frequency identification (RFID) tags from vehicles.

impact of its solution.

The optimization of transportation network performance has been widely discussed in the literature. The minimum tollbooth problem (MINTB), first introduced by Hearn and Ramana (1998), aims at minimizing the number of toll locations to achieve system optimality. Yang and Zhang (2003) formulate second-best link-based pricing as a bi-level program and solve it with a genetic algorithm. In Bai et al. (2010) it is shown that the problem is *NP-hard* and a local search heuristic is proposed. Another similar problem is to minimize total revenue (MINREV). MINREV is similar to MINSYS, but in this class of problems tolls can be negative as well as positive, while MINSYS does not accept negative tolls (HEARN; RAMANA, 1998; DIAL, 1999; DIAL, 2000; HEARN; YILDRIM, 2002; BAI et al., 2004). For a complete review of the design and evaluation of road network pricing schemes we refer the reader to the survey by Tsekeris and Voß (2008).

Two important transportation network concepts were introduced by Wardrop (1952): user equilibrium (UE) and system optimal (SO). The former is related to the equilibrium obtained when each user chooses a route that minimizes his/her costs in a congested network. In an UE state, any user can reduce his/her own travel cost by changing routes. Differently, SO is related to a state of equilibrium with minimum average journey time. This occurs when the users cooperate to choose their routes. However, the user usually chooses his/her own route in a non-cooperative manner. In a simplistic modeling behavior, users can choose their routes by different criteria. One possible simplification assumes that users choose their routes considering only fixed costs such as time to travel, or a value that depends on the congestion, or even only the toll values. These situations do not correspond to user equilibrium, but model different behaviors of the users.

In this work, we approach the tollbooth problem by routing on shortest paths as first studied in Buriol et al. (2010). The objective is to determine the location of a fixed number $\mathcal{K}$ of tollbooths and set their corresponding tariffs so that users travel on shortest paths between origin and destination, reducing network congestion. We calculate shortest paths according to two weight functions. In the first, the weights correspond to the tariffs of the tolled arcs. The second function considers as the weight of each arc its toll tariff added to its free flow time, where free flow time of an arc is defined to be the congestion-free time to traverse the arc. We also present a mathematical model for the minimum average link travel time and the tollbooth problem. We further propose two piecewise-linear functions that approximate an adapted convex travel cost function of the Bureau of Public Roads (1964) for measuring link congestion. Finally, we extend the work in Buriol et al. (2010) presenting a larger set of experiments, considering a new arc value to calculate shortest paths, a review of the algorithm components, such as the local search, and a more detailed review of the behavior of the algorithm, including a new set of instances and an analysis of characteristics of the final solutions.

This chapter is organized as follows. In Section 2.2 we present mathematical models for the minimum average link travel time, the tollbooth problem, and two approximate piecewise-linear functions for travel cost. The biased random-key genetic algorithm with local search proposed in Buriol et al. (2010) is presented in Section 2.3. Computational results are reported in Section 2.4. Finally, conclusions are drawn in Section 2.5.

2.2 Problem formulation

A road network can be represented as a directed graph $G = (V, A)$ where V represents the set of nodes (street or road intersections or points of interest), and A the set of arcs (street or road segments). Each arc $a \in A$ has an associated capacity c_a, and a time t_a, called the *free flow time*, necessary to transverse the unloaded arc a. To calculate the congestion on each link, a potential function Φ_a is computed as a function of the load or flow ℓ_a on arc a, along with α_a and β_a, two real-valued arc-tuning parameters. The parameters α_a and β_a intensify or mitigate the delay penalty value for the traffic congestion for the potential function BPR proposed by Bureau of Public Roads (1964) and described in the Subsection 2.2.1. Parameter β (often set to 0.15) is the ratio of travel time per unit distance, and parameter α (often set to 4) determines how fast the estimated average link speed decreases from free-flow to congested conditions. In addition, let

$$K = \{(o(1), d(1)), (o(2), d(2)), \dots, (o(|K|), d(|K|)\} \subseteq V \times V$$

denote the set of commodities or origin-destination (OD) pairs, where $o(k)$ and $d(k)$ represent, respectively, the origination and destination nodes for $k = 1, \dots, |K|$. Each commodity k has an associated demand of traffic flow $d_k = d_{o(k),d(k)}$, i.e., for each OD pair $(o(k), d(k))$, there is an associated flow d_k that emanates from node $o(k)$ and terminates in node $d(k)$. In this work we address the problem in which all the demand is routed on the network, such that traffic congestion is minimized. To encourage traffic to take on particular routes, we resort to levying tolls on selected street or road segments.

Before we describe our mathematical models, some notation is introduced. We denote by $IN(v)$ the set of incoming arcs to node $v \in V$, by $OUT(v)$ the set of outgoing arcs from node $v \in V$, by $a = (a_t, a_h) \in A$ a directed arc of the network, where $a_t \in V$ and $a_h \in V$ are, respectively, the tail and head nodes of arc a, by $\mathcal{S} = \sum_{k=1}^{|K|} d_k$ the total sum of demands, and by $\mathcal{Q} \subseteq V$ the set of destination nodes. Moreover, we denote by Φ_a the traffic congestion of arc $a \in A$, and by $\mathcal{K}$ the number of tollbooths to deploy (tolls are levied on users of the network at tollbooths). The values of φ_a^u and φ_a^l are approximations of traffic congestion cost on arc $a \in A$ given by piecewise-linear functions. We note that throughout the chapter we refer to flow and load interchangeably, as we do for commodity and demand.

In the following we summarize the notation and variables used in this section.

Notation:

V	set of nodes G;
A	set of arcs;
v	node $\in V$;
$IN(v)$	set of incoming arcs to node v;
$OUT(v)$	set of outgoing arcs from node v;
$a = (a_t, a_h)$	a directed arc of the network;
a_t, a_h	tail and head node of arc a, respectively;
c_a	capacity of arc a;
t_a	free flow time of arc a;
α_a, β_a	arc-tuning parameters;
K	set of commodities or OD pairs;
k	commodity $\in K$;
$o(k)$, $d(k)$	origination and destination node of commodity k, respectively;
$d_k = d_{o(k),d(k)}$	demand of commodity k;
$\mathcal{Q} \subseteq V$	set of destination nodes;
$\mathcal{S} = \sum_{k=1}^{\lvert K \rvert} d_k$	total sum of demands;
Φ_a	traffic congestion of arc a;
φ_a^l, φ_a^u	underestimation and overestimation of Φ_a, respectively;
$\mathcal{K}$	number of tollbooths to deploy;
C_a	fix weight cost of arc a;
M_1, M_2, M_3	sufficiently larger values;
P_l, P_u	minimum and maximum tariff values, respectively.

Variables:

ℓ_a	flow on arc a;
x_a^q	flow on arc a to destination $q \in \mathcal{Q}$;
w_a	toll tariff levied on arc a;
δ_v^q	the shortest-path distance from node v to destination node $q \in \mathcal{Q}$;
y_a^q	indicate if the arc a belongs to the shortest path to destination $q \in \mathcal{Q}$;
p_a	indicate if a tollbooth is deployed on arc $a \in A$.

In the next subsection we present a mathematical model of a relaxation of the tollbooth problem that does not take into account shortest paths. In Subsection 2.2.2 a complete model for the tollbooth problem is presented and in Subsection 2.2.3 we propose two piecewise-linear functions that approximate the convex cost function.

2.2.1 Model for minimization of average user travel time (MM1)

The evaluation of the traffic congestion cost can be defined in different ways according to specific goals. In this work we use the potential function

$$\Phi = \sum_{a \in A} \Phi_a, \text{ where } \Phi_a = \tfrac{\ell_a}{\mathcal{S}} t_a \Big[1 + \beta_a (\tfrac{\ell_a}{c_a})^{\alpha_a}\Big], \text{ for all } a \in A,$$

which is the convex travel cost function of the Bureau of Public Roads (1964) for measuring link congestion scaled by the term $\ell_a/\mathcal{S}$. This way, the potential function evaluates the average user travel time over all trips. Function Φ_a is convex and nonlinear and is a strictly increasing function of ℓ_a.

A mathematical programming model of average user travel time is

$$\min \Phi = \sum_{a \in A} \ell_a t_a \Big[1 + \beta_a (\ell_a / c_a)^{\alpha_a}\Big] / \mathcal{S} \tag{2.1}$$

subject to:

$$\ell_a = \sum_{q \in \mathcal{Q}} x_a^q, \ \forall a \in A, \tag{2.2}$$

$$\sum_{a \in OUT(v)} x_a^q - \sum_{a \in IN(v)} x_a^q = d_{v,q}, \ \forall v \in V \backslash \{q\}, \ \forall q \in \mathcal{Q}, \tag{2.3}$$

$$x_a^q \geq 0, \forall a \in A, \ \forall q \in \mathcal{Q}, \tag{2.4}$$

$$\ell_a \geq 0, \ \forall a \in A. \tag{2.5}$$

Its goal is to determine flows on each arc such that the average user travel time is minimized. In this model, decision variables $x_a^q \in \mathbb{R}^+$ represent the total flow to destination $q \in \mathcal{Q}$ on arc $a \in A$, and variables $\ell_a \in \mathbb{R}^+$ represent the total flow on arc $a \in A$. Objective function (2.1) minimizes average user travel time. Constraints (2.2) define total flow on each arc $a \in A$ taking into consideration the contribution of all commodities. Constraints (2.3) guarantee flow conservation, and (2.4)–(2.5) define the domains of the variables.

This model computes flow distribution without taking into account that users take a least cost route, providing a lower bound for the tollbooth problem to be described in the next subsection.

2.2.2 Model for the tollbooth problem (MM2)

A mathematical programming model for the tollbooth problem is

$$\min \ \Phi = \sum_{a \in A} \ell_a t_a \Big[1 + \beta_a (\ell_a / c_a)^{\alpha_a}\Big] / \mathcal{S} \tag{2.6}$$

subject to:

$$\ell_a = \sum_{q \in \mathcal{Q}} x_a^q, \ \forall a \in A, \tag{2.7}$$

$$\sum_{a \in OUT(v)} x_a^q - \sum_{a \in IN(v)} x_a^q = d_{v,q}, \ \forall v \in V \backslash \{q\}, \ \forall q \in \mathcal{Q}, \tag{2.8}$$

$$C_a + w_a + \delta_{a_h}^q - \delta_{a_t}^q \geq 0, \ \forall a \in A, \ \forall q \in \mathcal{Q} \ , \tag{2.9}$$

$$\delta_q^q = 0, \ \forall q \in \mathcal{Q}, \tag{2.10}$$

$$C_a + w_a + \delta_{a_h}^q - \delta_{a_t}^q \geq (1 - y_a^q)/M_1, \ \forall a \in A, \ \forall q \in \mathcal{Q}, \tag{2.11}$$

$$C_a + w_a + \delta_{a_h}^q - \delta_{a_t}^q \leq (1 - y_a^q) M_2, \ \forall a \in A, \ \forall q \in \mathcal{Q}, \tag{2.12}$$

$$M_3 y_a^q \geq x_a^q, \ \forall a \in A, \ \forall q \in \mathcal{Q}, \tag{2.13}$$

$$M_3 y_a^q + M_3 y_b^q \leq 2M_3 - x_a^q + x_b^q, \ \forall a, b \in \mathsf{A}^2_{OUT(v)}, \forall v \in V, \ \forall q \in \mathcal{Q}, \tag{2.14}$$

$$P_l p_a \leq w_a \leq P_u p_a, \ \forall a \in A, \tag{2.15}$$

$$\sum_{a \in A} p_a = \mathcal{K}, \ \forall a \in A, \tag{2.16}$$

$$x_a^q \geq 0, \ \forall a \in A, \ \forall q \in \mathcal{Q}, \tag{2.17}$$

$$\ell_a \geq 0, \ \forall a \in A, \tag{2.18}$$

$$w_a \in \mathbb{N}, \ \forall a \in A, \tag{2.19}$$

$$\delta_v^q \geq 0, \ \forall q \in \mathcal{Q}, \ \forall v \in V, \tag{2.20}$$

$$y_a^q \in \{0, 1\} \ \forall a \in A, \forall q \in \mathcal{Q}, \tag{2.21}$$

$$p_a \in \{0, 1\}, \ \forall a \in A. \tag{2.22}$$

This model seeks to levy tolls on $\mathcal{K}$ arcs of the transportation network such that the average user travel time is minimized if traffic is routed on least-cost paths. Here, the cost of a path is defined to be the sum of the tolls levied on the arcs of the path, or the sum of tolls and free flow times. We later describe in more detail these arc weight functions and how they are considered in the model using the fix weight cost C_a.

The decision variables for this model determine whether an arc will host a tollbooth and the amount of toll levied at each deployed tollbooth. Denote by $w_a \in \{0, P_l, P_l + 1, \ldots, P_u\}$ the toll tariff levied on arc $a \in A$, where $P_l, P_u \in \mathbb{N}^+$ are the minimum and maximum tariff values, respectively. For convenience we define $P_l = 1$. If no toll is levied on arc a, then $w_a = 0$. The binary decision variable $p_a = 1$ if a tollbooth is deployed on arc $a \in A$. The auxiliary binary variable $y_a^q = 1$ if arc $a \in A$ is part of a shortest path to destination node $q \in \mathcal{Q}$. Finally, auxiliary variable δ_v^q is the shortest-path distance from node $v \in V$ to destination node $q \in \mathcal{Q}$, and the constants M_1, M_2, and M_3 are sufficiently larger numbers.

Objective function (2.6) minimizes average user travel time. Constraints (2.7) define the total flow on each arc $a \in A$ while constraints (2.8) impose flow conservation. The other constraints force the flow of each commodity to follow the shortest path between the corresponding OD pair. An arc a belongs to the shortest path to destination q if the distance $\delta_{a_h}^q - \delta_{a_t}^q$ is equal to the arc cost, which in this case is $C_a + w_a$. Thus, constraints (2.9) define the shortest path distance for each node $v \in V$ and each destination $q \in \mathcal{Q}$. For consistency, constraints (2.10) require, for all $q \in \mathcal{Q}$, that the shortest distance from q to itself be zero. Constraints (2.11) and (2.12) together with (2.9) and (2.10) determine whether arc $a \in A$ belongs to the shortest path and thus determine the values of y_a^q, for $q \in \mathcal{Q}$. Constraints (2.11) require that an arc that does not belong to the shortest path have reduced cost $C_a + w_a + \delta_{a_h}^q - \delta_{a_t}^q > 0$. Constraints (2.12) assure that if the reduced cost of arc $a \in A$ and destination $q \in \mathcal{Q}$ is equal to zero, then arc a belongs to the shortest path to destination q, i.e. $y_a^q = 1$. In the computational experiments of Subsection 2.4.2, we used $M_1 = 100$ and $M_2 = 1000$. Constraints (2.13) assure that flow is sent only on arcs belonging to a shortest path. Constraints (2.14) are the even-split constraints. They guarantee that flow is split evenly among all shortest paths. In these constraints, $\mathsf{A}^2_{OUT(v)}$ is the set of all ordered groups of two distinct elements of $OUT(v)$. We later discuss these constraints in more detail. Constraints (2.15) limit the minimum and maximum tariff for a deployed tollbooth. Constraints (2.16) require that exactly $\mathcal{K}$ tolls be deployed. The remaining constraints define the domains of the variables.

Constraints (2.14) come in pairs for each node $v \in V$. For every pair of outgoing links $a \in OUT(v)$ and $b \in OUT(v)$: $\{a, b\} \in \mathsf{A}^2_{OUT(v)}$ and $\{b, a\} \in \mathsf{A}^2_{OUT(v)}$, there are two corresponding constraints. They model load balancing by assuring that if the flow from node $v \in V$ to destination $q \in \mathcal{Q}$ is routed on both arcs $a \in A$ and $b \in A$, i.e. if $y_a^q = y_b^q = 1$, then the flow on these arcs must be evenly split, i.e. $x_a^q = x_b^q$. To see this, suppose $y_a^q = y_b^q = 1$. The constraint for pair $\{a, b\} \in \mathsf{A}^2_{OUT(v)}$ implies that $x_a^q \leq x_b^q$. By symmetry the constraint for pair $\{b, a\} \in \mathsf{A}^2_{OUT(v)}$ implies that $x_a^q \geq x_b^q$. Consequently, $x_a^q = x_b^q$. Note that taking $M_3 = \max_{q \in \mathcal{Q}} \left(\sum_{v \in V} d_{v,q} \right)$ we assure that the right-hand-side of constraint (2.14) is bounded from below by M_3, making these constraints redundant for pairs of links with at most one of either y_a^k or y_b^k equal to one. Note that the even split constraints are part of the definition of shortest path in this work, since we seek to simulate the probability of user decision in an intersection.

A model for OSPF routing, which also considers shortest paths and even flow splitting, was previously proposed in Broström and Holmberg (2006). In their model a shortest path graph is built for each OD pair, while we opted for building a shortest path graph from all nodes to each node $q \in \mathcal{Q}$. This modification reduces the number of variables and constraints of the model.

We evaluate shortest paths according to two weight functions. In the first approach, called SPT (Shortest Path Toll), we define the weight of an arc $a \in A$ to be the tariff w_a levied on that arc. In this case, we set $C_a = \epsilon$, a sufficiently small value. This way, when there are one or more zero-cost paths, the flow is always sent along paths having smallest hop count. In the second approach, called SPTF (Shortest Path Toll+Free flow time), we define the weight of an arc $a \in A$ to be the tariff w_a levied on the arc plus the free flow time t_a of the arc, i.e. parameter $C_a = t_a + \epsilon$. The value $\epsilon > 0$ is added to the cost with the same goal as in the case of SPT since it is possible that $t_a = 0$ for one or more arcs $a \in A$.

2.2.3 Piecewise-linear functions for the models

The performance of mixed integer linear programming solvers has improved considerably over the last few years. The two mathematical programming models presented so far have a nonlinear objective function Φ. To apply these solvers, one must first linearize Φ, resulting in an approximation of the nonlinear objective function. One possible option is to approximate the nonlinear function by a piecewise linear function. Fortz and Thorup (2004) proposed a piecewise-linear function for a general routing problem to approximate network congestion cost. Ekström et al. (2012) describe an iterative approximation by piecewise linear function for the travel time and total travel time, resulting in a mixed integer linear program.

In this subsection, we propose two piecewise-linear approximations of the function $\Phi = \sum_{a \in A} \Phi_a$. The first linearization φ^u, is an overestimation, and under certain conditions is an upper bound of Φ. The second linearization φ^l is an underestimation and provides a lower bound of Φ. It is possible to apply these linearizations to any model with this type of nonlinear function. We apply them to models MM1 and MM2.

Let Ω be the set of constraints (2.2)–(2.5) or (2.7)–(2.22) of the previously described mathematical models. For the case where Ω represents the constraints of the MM1 model the approximation is called LMM1. On the other hand, when Ω represents the constraints of the MM2 model, we call the approximation LMM2.

In approximation φ^u, the cost function of each arc $a \in A$ is composed of a series of line segments sequentially connecting coordinates

$$(X_0, \Phi_a(X_0)), (X_1, \Phi_a(X_1)), \ldots, (X_n, \Phi_a(X_n)),$$

where values $X_0, X_1, \ldots, X_n$ are given such that $X_0 = 0$, and for $i = 1, \ldots, n$, $X_i \in \mathbb{R}$ and $X_i > X_{i-1}$.

If we denote the cost on arc $a \in A$ by the variables ψ_a, then the resulting mathematical programming model of the overestimation $\varphi^u = \sum_{a \in A} \varphi^u_a$ is

$$\min \varphi^u = \sum_{a \in A} \psi_a \tag{2.23}$$

subject to:

$$\text{Constraints } \Omega \text{ are satisfied,} \tag{2.24}$$

$$(m^i_a / c_a)\ell_a + b^i_a \leq \psi_a, \ \forall a \in A, \ \forall i = 1, \ldots, n, \tag{2.25}$$

$$\psi_a \geq 0, \ \forall a \in A, \tag{2.26}$$

where

$$m^i_a = (\Phi_a(X_i) - \Phi_a(X_{i-1}))/(X_i - X_{i-1}),$$
$$b^i_a = \Phi_a(X_i) - X_i m^i_a,$$

where

$$\Phi_a(X_i) = X_i c_a t_a (1 + \beta_a (X_i)^{\alpha_a})/\mathcal{S}$$

for $X_0 = 0 < X_1 < \cdots < X_n$. Objective function (2.23) minimizes the approximation of average user travel time. Constraints (2.25) evaluate the partial cost on each arc by determining the approximate value φ_a^u for Φ_a according to load ℓ_a. Constraints (2.26) define the domain of the variables.

The linearization requires the definition of the terms $X_0, X_1, \ldots, X_n$ whose values are computed as a function of ℓ_a/c_a. These values are input parameters, and can be defined by a predefined set for all arcs, or even a set defined for each arc based on the estimation for the relation ℓ_a/c_a. The number of these terms can be arbitrarily defined according to the accuracy required for the linearization of the cost function, or according to characteristics of the set of instances. This linearization requires a balance between the accuracy of the computed solution and the time to compute the linearization. With a large number n, the linearization tends to provide a better approximation of the original value, while a small value of n can save time while solving the model since each element entails $|A|$ additional constraints.

A second linearization, which we denote by φ_a^l, is an underestimation and gives us a lower bound on Φ_a. The mathematical model of this linearization is similar to that of the overestimation. However, to estimate φ_a^l, we first compute the slope $m_a(x)$ of Φ_a at $x = (X_{i-1} + X_i)/2$, for $i = 1, \ldots, n$, as

$$m_a(x) = \frac{\partial \Phi_a}{\partial x} = \frac{t_a}{\mathcal{S}} + \frac{(\alpha_a + 1) t_a \beta_a x^{\alpha_a}}{c_a^{\alpha_a} \mathcal{S}}.$$

Given x and $m_a(x)$, the independent term can be easily computed.

Linearizations φ^l and φ^u produce, respectively, an underestimation and an overestimation of Φ, as Proposition 1 states.

Proposition 1 *Let $\varphi^u = \sum_{a\in A} \varphi_a^u$, $\varphi^l = \sum_{a\in A} \varphi_a^l$, and as before $\Phi = \sum_{a\in A} \Phi_a$. Let $X_0, X_1, \ldots, X_n$ be the values for which the approximation is computed. If $\ell_a/c_a \leq X_n, \forall a \in A$, then $\varphi^l \leq \Phi \leq \varphi^u$.*

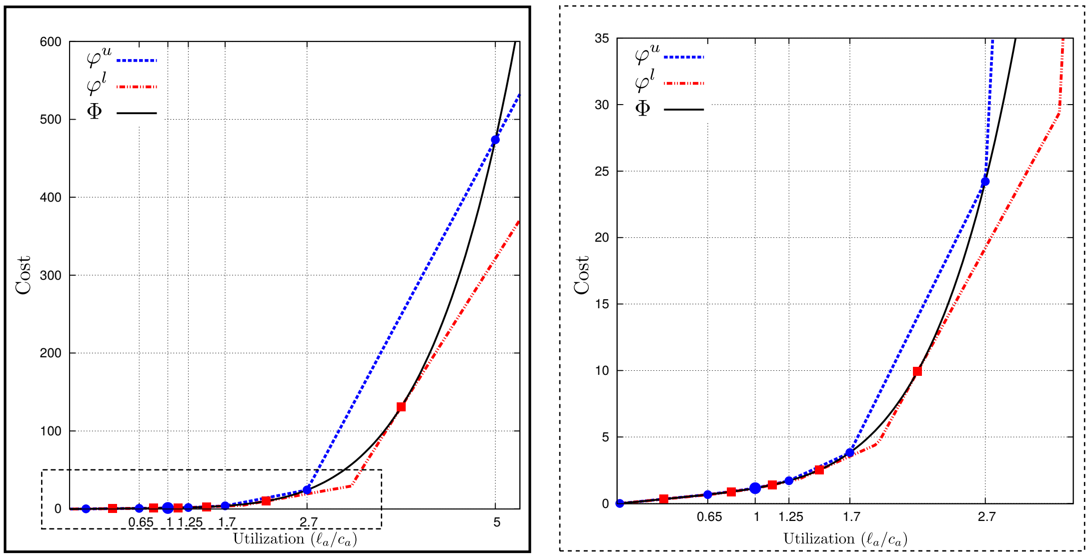
Proof. As Φ is convex, by construction $\varphi_a^l \leq \Phi_a$, since each line segment is tangent of Φ, then $\Phi = \sum_{a\in A} \Phi_a \geq \sum_{aA\in} \varphi_a^l = \varphi^l$. Thus $\Phi \geq \varphi^l$. Furthermore, if $\ell_a/c_a \leq X_n$, then by construction $\varphi_a^u \geq \Phi_a$, since each line segment is secant of Φ, which implies that $\varphi^u = \sum_{a\in A} \varphi_a^u \geq \sum_{a\in A} \Phi_a = \Phi$. Thus $\varphi^u \geq \Phi$. Therefore $\varphi^l \leq \Phi \leq \varphi^u$. □

Note that in the Proposition 1 the underestimation φ^l is always a lower bound of Φ, while the overestimation φ^u requires that $\ell_a/c_a \leq X_n, \forall a \in A$ be true for the proposition to hold. This can be easily obtained defining $X_n = \mathcal{S}/c_a, \forall a \in A$.

A representation of the functions φ_a^u, φ_a^l, and Φ_a is depicted in Figure 2.1. It shows the cost function Φ (solid line) as well as the piecewise-linear cost functions

φ^u and φ^l for an arc $a \in A$ with $t_a = 5$, $c_a = 200$, $\alpha_a = 4$, $\beta_a = 0.15$, and $\mathcal{S} = 1000$ using with $\{X_0, X_1, \ldots, X_6\} = \{0, 0.65, 1, 1.25, 1.7, 2.7, 5\}$. Observe that there is a higher concentration of points X in the range $\frac{l_a}{c_a} = [0.65; 1.25]$. This dense concentration of points in this region is used because the flow on the majority of the arcs is concentrated around their capacity. Thus, to obtain a good approximation requires that several X values be set to values around $\frac{l_a}{c_a} = 1$. Note that a ratio of $\frac{l_a}{c_a} > 1$ indicates that the arc is overloaded.

Figure 2.1 – Comparison of the cost function with the linear piecewise-linear cost function



Source: from the author (2015).

The potential function Φ is based on the BPR function (Bureau of Public Roads, 1964) that is the most used function to estimate the traffic congestion in transportation networks. This function has no asymptotic behavior in relation to the road capacity. In this case, even allowing capacity overloading, the function allows to analyze different situations and costs including roads with a high occupation or even infeasible cases. In contrast to the non-asymptotic behavior of the BPR, Davidson (1966) proposed a cost function with asymptotic behaviour for the capacity. However, this can result in infeasible solutions for some networks.

2.3 A biased random-key genetic algorithm

In this section we describe the biased random-key genetic algorithm (BRKGA) for the tollbooth problem. The general framework is described in section 1.2. Following we describe the encoding and decoding procedures. The values for parameters as well as the stopping criterion are given in Section 2.4.

Solutions are encoded as a $2 \times |A|$ vector $\mathcal{X}$, where $|A|$ is the cardinality of the set A of arcs in the network. The first $|A|$ keys correspond to the random keys which define the toll tariffs while the last $|A|$ keys correspond to a binary vector b, with $\mathcal{K}$ positions set to one, used to indicated tolled arcs.

The decoder has two phases. In the first phase tolls are selected and arc tariffs are set directly from the random keys. In the second phase, a local improvement procedure attempts to change the tariffs with the goal of reducing the value of the objective function. Each tolled arc a has a tariff in the interval $[1, w_{\max}]$, where $w_{\max}$ is an input parameter. The tariff for arc a is simply decoded as $b_a \cdot \lceil \mathcal{X}_a \cdot w_{\max} \rceil$. In an initial solution, the $\mathcal{K}$ tolled arcs are selected randomly by uniform distribution. In a crossover, if both parents have a toll in arc a, the same arc is tolled in the child. The remaining tolls are selected randomly among the arcs whose parents have different values.

Demands are routed forward to their destinations on shortest weight paths. For SPT, tolled links have weights equal to their tariffs and untolled links are assumed to have weight zero. For SPFT, we add to the tariff the free flow time to define the weight of all tolled arcs, while each untolled arc has weight equal to its free flow time. Depending on the number of tolls and the network, there can be several shortest paths of cost zero (especially for SPT). In this case, we use the path with the least number of hops. Traffic at intermediate nodes is split equally among all outgoing links on shortest paths to the destination. After the flow is defined, the fitness of the solution is computed by evaluating the objective function Φ.

The second phase of the decoder is a local improvement. Local search is applied to the solution produced in the first phase of the decoder. In short, it works as follows. Let q_{ls} be an integer parameter and $A^* \subseteq A$ be the $q = \min\{|A|, q_{ls}\}$ arcs having the largest congestion costs Φ_a, i.e. $|A^*| = q$ and $\Phi_{a^*} \geq \Phi_a$, for all pairs $\{a^*, a\}$ such that $a^* \in A^*$ and $a \in A \setminus A^*$. For each arc $a^* \in A^*$, in case it is tolled, its weight is increased by one unit at a time, to induce a reduction of its load. The unit-increase is repeated until either the weight reaches w_{max} or Φ no longer improves. If no improvement in the objective function is achieved, the weight is reset to its initial value. In case the arc is not currently tolled, a new toll is installed on the arc with initial weight one, and a toll is removed from some other link tested in circular order. If no reduction in the objective function is achieved, the solution is reversed to its original state. Every time a reduction in Φ is achieved, a new set A^* is computed and the local search restarts. The procedure stops at a local minimum when there is no improved solution changing the weights of the candidate arcs in set A^*.

In the local improvement, every time a weight is changed (added by one unit, inserted or removed) the current shortest paths are updated (BURIOL et al., 2008) instead of recomputed from scratch, thus saving a considerable amount of running time.

2.4 Computational results

In this section we present computational experiments with the models and algorithms introduced in the previous sections. Initially, we describe the dataset used in the experiments. Then, we detail three sets of experiments. The first set evaluates the mathematical models MM1 and LMM1. The second set of experiments evaluates the full model MM2 with piecewise linear function, which considers the shortest-path constraints with even split of loads. The last set of experiments evaluates the biased random-key genetic algorithm presented in Section 2.3.

The experiments were done on a computer with an Intel Core i7 930 processor running at 2.80 GHz, with 12 GB of DDR3 RAM of main memory, and Ubuntu 10.04 Linux operating system. The biased random-key genetic algorithms (BRKGA) were implemented in C and compiled with the `gcc` compiler, version 4.4.3, with optimization flag `-O3`. The commercial solver CPLEX 12.3[2] was used to solve the proposed linearizations of the mathematical linear models, while MOSEK[3] was used to solve the mathematical model MM1 (with convex objective function).

To test model LMM2, we created the instances from set *S1* from instance `SiouxFalls` of *S2* by removing from `SiouxFalls` some of its nodes and their adjacent links as well as all OD pairs where these nodes are either origin or destination nodes. Let $n < |V|$ be the new number of nodes. We choose to remove nodes

$$v \in \ V : v = \left\lfloor k\tfrac{|V|}{|V|-n} + 1 \right\rfloor \text{ with } k = 0, \dots, |V| - n - 1.$$

Let $a, b \in A$, and $v \in V$ be nodes such that $a \in OUT(v)$ and $b \in IN(v)$. Furthermore, let a_t and a_h be, respectively, the tail and head nodes of a links a. We create a link a' from a_h to b_t if there does not already exist a link between a_h and b_t and furthermore $|OUT(b_t)| < 4$ or $|IN(a_h)| < 4$. Link a' has the same characteristics (t_a, c_a, etc.) of link a. After all extensions, we remove from the network all arcs $a \in OUT(v) \cup IN(v)$ and v.

[2] <www.cplex.com>
[3] <www.mosek.com>

Table 2.1 details six synthetic instances (*S1*) and ten real-world instances (*S2*) considered in our experiments and made available at <www.bgu.ac.il/~bargera/tntp/>.

Table 2.1 – Attributes for the instances are given in each column. For each instance, its row lists the set identification (*S1* or *S2*), instance name, number of vertices, links, OD pairs, number of vertices in which traffic originates (Source nodes), and number of nodes in which traffic terminates (Sink nodes)

Set	Instance	Vertices	Links	OD pairs	Source nodes	Sink nodes
S1	`SiouxFalls_08`	8	16	48	8	8
	`SiouxFalls_09`	9	26	68	9	9
	`SiouxFalls_10`	10	36	84	10	10
	`SiouxFalls_12`	12	38	126	12	12
	`SiouxFalls_14`	14	36	172	14	14
	`SiouxFalls_16`	16	50	218	16	16
S2	`SiouxFalls`	24	76	528	24	24
	`Friedrichshain Center`	223	514	506	23	23
	`Prenzlauerberg Center`	350	717	1406	38	38
	`Tiergarten Center`	361	749	644	26	26
	`Mitte Center`	398	857	1260	36	36
	`Anaheim`	416	914	1406	38	38
	`MPF Center`	974	2153	9505	98	98
	`Barcelona`	1020	2522	7922	97	108
	`Winnipeg`	1052	2836	4345	135	138
	`ChicagoSketch`	933	2950	9351	386	386

Source: from the author (2015).

2.4.1 Results for models MM1 and LMM1

The first set of experiments evaluates the models when solved with commercial solvers. Table 2.2 presents, for each instance, the objective functions Φ, and the lower and upper bounds φ^l and φ^u, respectively.

In the first two columns after the name of the instance, we present the objective function values Φ and the computational times for model MM1 obtained with the nonlinear solver MOSEK 6.0 using the modeling system GAMS[4]. A few nonlinear solvers are part of the GAMS system and we evaluated the performance of all of them. Some of them are general nonlinear solvers, and have no specific routines for convex functions. Most were not able to solve the larger instances. MOSEK presented the best performance in terms of running times and for this reason we report only the results obtained with MOSEK. The next columns present results for CPLEX 12.3 with the proposed piecewise-linear functions φ^l and φ^u, respectively the lower and upper estimations of function Φ. In each case, we

[4] <www.gams.com>

show the objective function values in columns φ^l and φ^u, as well as $\Phi\{\varphi^l\}$ and $\Phi\{\varphi^u\}$, the values of Φ considering the arc loads obtained by the different approximations. The computational times are reported in seconds.

Table 2.2 – Computational results for MM1 and LMM1

	MM1		**LMM1-lower estimate**			**LMM1-upper estimate**		
Instance	$\mathbf{\Phi}$	**Time(s)**	$\varphi^{\mathbf{l}}$	$\mathbf{\Phi}\{\varphi^{\mathbf{l}}\}$	**Time(s)**	$\varphi^{\mathbf{u}}$	$\mathbf{\Phi}\{\varphi^{\mathbf{u}}\}$	**Time(s)**
`SiouxFalls_08`	8.77	0.0	8.69	8.77	0.0	8.97	8.77	0.0
`SiouxFalls_09`	6.32	0.0	6.26	6.32	0.0	6.46	6.32	0.0
`SiouxFalls_10`	6.71	0.0	6.62	6.72	0.0	6.81	6.71	0.0
`SiouxFalls_12`	11.46	0.0	10.92	11.47	0.0	12.69	11.72	0.0
`SiouxFalls_14`	64.69	0.0	45.87	64.79	0.0	119.34	64.79	0.0
`SiouxFalls_16`	10.11	0.0	9.97	10.15	0.0	10.33	10.18	0.0
`SiouxFalls_18`	10.70	0.0	10.37	10.93	0.0	11.16	10.87	0.0
`SiouxFalls`	19.95	0.1	18.10	20.77	0.1	21.68	20.52	0.1
`Friedrichshain Center`	42.47	7.7	39.57	43.34	0.0	47.61	43.11	0.1
`Prenzlauerberg Center`	59.90	70.3	56.98	61.07	0.1	67.21	60.81	0.1
`Tiergarten Center`	52.57	164.5	47.63	53.63	0.1	59.68	52.91	0.1
`Mitte Center`	62.36	30.8	58.76	63.59	0.2	71.08	63.11	0.3
`Anaheim`	12.46	204.8	12.26	12.49	0.5	12.91	12.48	0.5
`MPF Center`	65.88	1,988.0	60.94	67.63	2.8	75.11	66.57	3.7
`Barcelona`	6.87	6,174.8	6.54	11.75	1.9	6.54	11.75	1.9
`Winnipeg`	13.67	2,189.0	12.25	20.83	4.7	12.25	20.83	4.7
`ChicagoSketch`	14.24	2,004.9	14.09	14.31	1,154.6	14.50	14.30	2,465.1

Source: from the author (2015).

From the results in Table 2.2, three main observations can be made. First, there are small gaps between φ^l and $\Phi\{\varphi^l\}$, as well as between φ^u and $\Phi\{\varphi^u\}$, i.e. both piecewise-linear functions φ^l and φ^u have values that are, respectively, close to $\Phi\{\varphi^l\}$ and $\Phi\{\varphi^u\}$. The gap is the absolute difference between two values divided by the minimum of both values. In a small number of cases the gap is significant and we observe that, as expected, this occurs in instances with higher average or higher maximum utilization (ℓ_a/c_a), like `Barcelona` and `Winnipeg`. Second, we compare the results for models MM1 and LMM1. The gaps between φ^l and Φ, and between φ^u and Φ, are also small, which means that the piecewise functions have similar values to the original convex function Φ. However, for most of the instances, the computational times spent by MOSEK on the convex function are two to four orders of magnitude greater than the time spent by CPLEX on the piecewise-linear functions. The only case where solving the model with a piecewise linear function (computing φ^u with CPLEX) took longer than solving the model with the convex function Φ (using MOSEK) was for instance `ChicagoSketch`. However, CPLEX found good solutions quickly, and spent most of the time certifying optimality. For example, CPLEX found solutions with a gap of 3% with respect to the optimal solution in about 650 seconds, while MOSEK needed more than 1600 seconds to reach this gap.

The last important observation is that the MM1 model is a relaxation of MM2. Moreover, the shortest paths and even-split constraints (Eqs. 2.14) of model MM2 add a considerable number of variables and constraints to the model. Thus, evaluating MM2 with a convex function became impracticable in terms of computational time, and for this reason no corresponding results are reported. In the next experiment we evaluate both approximations for the full model (MM2).

2.4.2 Results for the tollbooth problem with piecewise-linear cost (LMM2)

This set of experiments tests the performance of CPLEX on LMM2, the model that includes shortest paths and even-split constraints. The main objective is show that even for the approximation in very small instances, the solver has a limited performance. We run the model considering both weight functions to calculate shortest paths (SPT and SPTF) and both piecewise-linear functions introduced in Section 2.2.3.

Table 2.3 present results for model LMM2 when the shortest path is calculated considering only the toll tariffs (SPT), and for tariffs plus the free flow time (SPTF), respectively. For each instance, we tested several scenarios of $\mathcal{K}$. For each scenario we present the objective function values of approximations φ^l and φ^u obtained by CPLEX, the corresponding $\Phi\{\varphi^l\}$ and $\Phi\{\varphi^u\}$ values (as described in the previous subsection), the gap returned by the solver for a time limit of 1800 seconds, and finally the running times in seconds. The null values (-) indicate that a feasible solution was not found within the time limit.

Table 2.3 illustrates the difficulty in solving these instances with CPLEX. For most of the instances no optimal solution was found within 30 minutes, and for many of them not even a feasible solution was found in this time limit. A small increase in the instance size implies in a large increase in the computational effort spent to solve the model. We observe that for SPT the solver has more difficulty in finding an initial solution, and the gap returned by the solver is slightly higher in comparison with SPTF. Furthermore, the computational time is slightly reduced for SPTF, and φ^l was computed slightly faster than was φ^u.

Table 2.3 – Computational results for LMM2 to SPT and SPTF

			Approx.		**Obj. Function**		**Solver *gap***		**Time(s)**	
Type	**Instance**	$\mathcal{K}$	φ^{l}	φ^{u}	$\mathbf{\Phi}\{\varphi^{\mathrm{l}}\}$	$\mathbf{\Phi}\{\varphi^{\mathrm{l}}\}$	φ^{l}	φ^{u}	φ^{l}	φ^{u}
SPT		2	6.47	6.70	6.52	6.52	0.00	0.00	91.7	8.1
		5	6.35	6.58	6.38	6.38	0.00	0.00	12.4	94.4
	`SiouxFalls_09`	10	6.27	6.47	6.33	6.33	0.00	0.00	4.1	35.2
		15	6.27	6.46	6.32	6.32	0.00	0.00	1.3	15.3
		20	6.27	6.46	6.32	6.32	0.00	0.00	0.4	3.5
		3	-	-	-	-	-	-	1,800.0	1,800.0
		7	-	-	-	-	-	-	1,800.0	1,800.0
	`SiouxFalls_10`	14	-	-	-	-	-	-	1,800.0	1,800.0
		21	6.70	6.90	6.76	6.78	0.59	0.42	1,800.0	1,800.0
		28	6.70	6.90	6.76	6.78	0.23	0.21	1,800.0	1,800.0
		3	-	-	-	-	-	-	1,800.0	1,800.0
		7	-	-	-	-	-	-	1,800.0	1,800.0
	`SiouxFalls_12`	15	-	-	-	-	-	-	1,800.0	1,800.0
		22	-	-	-	-	-	-	1,800.0	1,800.0
		30	-	-	-	-	-	-	1,800.0	1,800.0
		3	46.72	120.69	65.69	65.69	1.63	0.94	1,800.0	1,800.0
		7	46.24	120.08	65.13	65.13	0.75	0.60	1,800.0	1,800.0
	`SiouxFalls_14`	14	-	-	-	-	-	-	1,800.0	1,800.0
		21	46.14	-	64.96	-	0.49	-	1,800.0	1,800.0
		28	46.14	-	64.96	-	0.39	-	1,800.0	1,800.0
SPTF		2	6.28	6.47	6.33	6.33	0.00	0.00	0.5	0.3
		5	6.27	6.46	6.32	6.32	0.00	0.00	0.4	0.4
	`SiouxFalls_09`	10	6.27	6.46	6.32	6.32	0.00	0.00	0.2	573.3
		15	6.27	6.46	6.32	6.32	0.00	0.00	0.4	0.4
		20	6.27	6.46	6.32	6.32	0.00	0.00	0.4	0.4
		3	6.73	6.93	6.79	6.79	0.00	0.00	92.7	176.4
		7	6.70	6.90	6.76	6.78	0.00	0.15	203.2	1,800.0
	`SiouxFalls_10`	14	-	6.90	-	6.78	-	0.31	1,800.0	1,800.0
		21	6.70	6.90	6.76	6.78	0.32	0.51	1,800.0	1,800.0
		28	6.70	6.90	6.76	6.78	0.04	0.51	1,800.0	1,800.0
		3	11.19	13.09	11.67	11.82	0.00	1.87	1,748.4	1,800.0
		7	11.18	13.05	11.66	11.83	0.39	2.44	1,800.0	1,800.0
	`SiouxFalls_12`	15	-	-	-	-	-	-	1,800.0	1,800.0
		22	-	-	-	-	-	-	1,800.0	1,800.0
		30	11.18	-	11.66	-	1.64	-	1,800.0	1,800.0
		3	46.24	119.73	65.09	65.09	0.00	0.00	1,019.3	133.3
		7	46.14	119.65	64.96	64.99	0.48	0.25	1,800.0	1,800.0
	`SiouxFalls_14`	14	-	-	-	-	-	-	1,800.0	1,800.0
		21	-	-	-	-	-	-	1,800.0	1,800.0
		28	-	119.65	-	64.99	-	0.13	1,800.0	1,800.0

Source: from the author (2015).

Results for instances `SiouxFalls_06` and `SiouxFalls_08` were found for φ^l and φ^u in a less than one second, and for this reason they were omitted from the table. On the other hand, results were omitted for `SiouxFalls_16`, for both piecewise-linear functions and shortest-path evaluations SPT and SPTF, since the solver was unable to find any feasible solution within the time limit.

In summary, Table 2.2 shows that solving the simplified model MM1, i.e. MM2 without the shortest paths computation and even-load constraints, takes a considerable time, while their corresponding linearized versions φ^l and φ^u are calculated very quickly for almost all cases. Table 2.3, on the other hand, shows that the linearizations φ^l and φ^u of the full model MM2 takes a long time even for small instances. Thus, these results motivated us to propose a heuristic solution to solve the tollbooth problem, and the results of the proposed biased random-key genetic algorithm are presented in the next subsection.

2.4.3 Results for the biased random-key genetic algorithm

This section presents results for the biased random-key genetic algorithm applied on instances from class *S2*. We extended the experimental study performed by Buriol et al. (2010) in which results for only three of these instances were presented. Moreover, we provide an analysis of the best solution for each combination of instance, value of $\mathcal{K}$, and problem type (SPT or SPTF).

To tune the parameters, a set of experiments was performed. The experiment consisted of two steps. In the first step, we determined the fixed running time for each triplet: instance (`SiouxFalls`, `Prenzlauerberg Center`, and `Anaheim`), value of $\mathcal{K}$, and problem type (SPT or SPTF). To define the fixed time, we ran the BRKGA with local search using a set of predefined parameters: population size $p = 50$, elite set of size $p_e = 0.25p$, mutant set of size $p_m = 0.05p$, elite key inheritance probability $\rho_A = 0.7$, and a restart parameter $r = 10$. At every r generations we verify whether the best three individuals in the population have identical fitness (within 10^{-3} of each other). If they do, then the second and third best are replaced by two new randomly-generated solutions. The BRKGA was run for at least 500 and at most 2000 generations, stopping after 100 generations without improvement of the incumbent solution. The fixed time is defined to be the average of five independent runs.

The running time defined in the first step of the tuning phase is used in the second step to determine the best combination of parameter values. We ran the BRKGA for this fixed amount of time with parameters taken from the sets of values shown in the third column of Table 2.4. All combinations of parameter values were considered.

Table 2.4 – Parameter values in tuning experiment

Description	**Parameter**	**Values**
Population size	p	$\{50{,}100\}$
Elite size	p_e	$\{0.15p,\ 0.25p\}$
Mutation size	p_m	$\{0,\ 0.05p,\ 0.10p\}$
Inheritance probability	ρ_A	$\{0.5,\ 0.7\}$
Restart	r	$\{0,\ 10\}$
Local Search	q_{ls}	$\{0^*,\ 2,\ 5,\ 10\}$

*Indicates that no local search is applied.

Source: from the author (2015).

Given a set of triples, each consisting of an instance, a value of $\mathcal{K}$, and a problem type (SPT or SPTF), we run the BRKGA on each triple using all combinations of the parameters in Table 2.4. The relative gaps of the fitness values from each run to the best fitness over all runs for each triple is computed. We observe that using a local search in the BRKGA results in better solutions than using no local search. In the case of $q_{ls} = 0$, the average relative gap is 29.86, while for $q_{ls} = 2, 5$, and 10, the relative gap was 6.70, 5.96, and 5.88, respectively. Therefore, we analyze the remaining parameters considering only runs where $q_{ls} = 10$. Table 2.5 shows the average relative gaps for these remaining parameters. The best observed parameter values (in bold) were $p = 100$ for population size, $p_e = 0.15p$ for elite population size, $p_m = 0.05p$ for mutant population size, $\rho_A = 0.70$ for inheritance probability, and $r = 10$ for restart.

Table 2.5 – Average of relative gaps obtained for different parameters

Description	**Parameter**	**Value**	**Gap**
Population size	p	50	6.04
		100	**5.72**
Elite size	p_e	**0.15*p***	**5.82**
		0.25p	5.93
Mutation size	p_m	0	6.39
		0.05*p*	**5.49**
		0.10p	5.76
Inheritance probability	ρ_A	0.5	5.91
		0.7	**5.85**
Restart	r	0	6.23
		10	**5.53**

Source: from the author (2015).

Once the parameters were set, we ran the BRKGA with local search (BRKGA+LS) with a time limit of 3600 seconds (except for `ChicagoSketch`, the largest instance, for which we ran with a time limit of 7200 seconds). We allow the maximum number of generations to be 2000, and the maximum number of generations without improvement to be 100.

Table 2.6 shows averages over five runs of BRKGA+LS and a comparison between SPT and SPTF. For each value of $\mathcal{K}$, it lists the best solution value (Best Φ) over the five runs, average fitness value (Avg Φ), standard deviation (SD), and average running time in seconds.

The first observation is that as the value of $\mathcal{K}$ increases, the value of Φ tends to decrease and have less variance. In fact, in most cases, the best solutions were found for $\mathcal{K} \geq \frac{|A|}{2}$. With small $\mathcal{K}$ it is easy for flow to bypass tolled arcs, which impairs traffic engineering. On the other hand, the search space increases considerably for larger $\mathcal{K}$ values, making the problem hard to solve. Since there are $\binom{|A|}{\mathcal{K}}$ configurations for the location of $\mathcal{K}$ tolls and for each configuration each toll can have 20 different values, then the size of the solution space is $\sigma(\mathcal{K}) = \binom{|A|}{\mathcal{K}}20^{\mathcal{K}}$. Thus, the solution space size is much larger for $\mathcal{K} \geq \frac{|A|}{2}$ than for $\mathcal{K} < \frac{|A|}{2}$. Furthermore, even though the maximum of $\sigma(\mathcal{K})$ is achieved for a value of $\mathcal{K} < |A|$, in all of the instances, $\sigma(\mathcal{K}') > \sigma(\mathcal{K})$ for all $\mathcal{K}' > \mathcal{K}$, where $\mathcal{K}'$ is the largest $\mathcal{K}$ value tested. For example, for the `SiouxFalls` instance, for which the largest value of $\mathcal{K}$ tested was 70, $\sigma(\mathcal{K}) = \binom{76}{\mathcal{K}}20^{\mathcal{K}}$, which achieves a maximum for $\mathcal{K} = 73$.

In most entries of Table 2.6 the standard deviation is small, showing robustness of the algorithm. The table also shows that for small values of $\mathcal{K}$, SPTF has smaller Φ than SPT. This occurs because, for small values of $\mathcal{K}$, SPT has many zero-cost paths, making it difficult to influence flow distribution with tolls.

Table 2.6 – Detailed results of SPT and SPTF for BRKGA+LS

		SPT				**SPTF**			
Instance	$\mathcal{K}$	**Best Φ**	**Avg Φ**	**SD**	**Time (s)**	**Best Φ**	**Avg Φ**	**SD**	**Time (s)**
SiouxFalls	10	52.38	54.48	2.88	34.83	25.19	25.19	0.00	27.57
	20	32.14	38.01	6.08	59.52	22.72	22.86	0.10	30.26
	30	27.05	28.37	1.90	68.43	22.10	22.30	0.17	32.34
	50	21.59	21.90	0.27	41.81	21.64	21.83	0.18	48.74
	70	21.38	21.53	0.15	32.39	21.55	21.94	0.23	17.18
Friedrichshain Center	10	56.44	56.80	0.34	263.53	46.38	47.23	0.68	218.38
	50	46.59	48.32	2.44	526.92	43.45	43.52	0.06	226.39
	100	43.52	43.93	0.37	710.33	43.41	43.50	0.07	273.65
	300	42.90	43.45	0.33	395.85	43.38	43.59	0.14	209.36
	500	44.04	44.46	0.38	207.16	43.56	44.01	0.37	236.48
Prenzlauerberg Center	10	79.19	80.18	0.99	924.55	65.99	66.13	0.14	682.22
	50	68.21	69.95	1.80	1,780.96	61.72	62.61	0.75	890.95
	100	63.31	63.72	0.26	1,344.99	61.58	61.65	0.07	785.03
	450	61.82	62.18	0.26	1,015.67	61.63	61.81	0.16	735.60
	700	62.42	63.26	0.81	1,066.42	63.03	63.68	0.44	703.19
Tiergarten Center	10	61.97	62.00	0.07	418.51	53.24	53.28	0.06	401.85
	50	56.45	56.69	0.23	1,148.33	52.88	52.92	0.03	534.28
	100	54.14	54.79	0.47	1,499.94	52.92	53.04	0.13	505.92
	450	53.11	53.26	0.13	764.01	52.87	52.93	0.06	444.03
	700	53.60	53.98	0.25	578.06	52.97	53.10	0.12	331.22
Mitte Center	10	79.78	80.11	0.26	805.12	65.84	66.47	0.38	807.10
	50	69.74	70.58	0.67	1,880.73	63.86	64.03	0.21	1,096.38
	100	68.39	68.71	0.33	2,233.21	63.75	63.94	0.19	1,536.11
	400	63.94	64.11	0.15	2,323.10	63.90	64.17	0.22	955.07
	800	64.19	64.47	0.34	1,102.89	63.96	64.30	0.22	879.13
Anaheim	10	15.39	15.42	0.01	785.38	12.72	12.73	0.01	1,328.61
	50	14.01	14.05	0.04	2,611.93	12.58	12.60	0.02	2,058.35
	100	13.41	13.54	0.09	3,406.09	12.58	12.60	0.01	2,034.47
	500	12.73	12.89	0.11	3,602.11	12.62	12.68	0.05	3,601.32
	800	12.60	12.65	0.05	3,096.01	12.57	12.63	0.05	3,041.08
MPF Center	10	91.64	92.03	0.28	3,616.48	70.73	71.14	0.24	3,615.19
	100	82.28	82.62	0.51	3,616.98	66.64	66.73	0.07	3,611.59
	250	82.54	82.84	0.25	3,616.24	66.76	66.93	0.12	3,615.17
	1000	75.08	75.89	0.90	3,612.88	67.92	68.39	0.38	3,611.87
	2000	71.21	72.58	0.90	3,607.61	68.11	68.58	0.30	3,602.20
Barcelona	10	15.84	15.84	0.00	3,622.16	7.82	7.91	0.13	3,618.36
	100	9.41	9.48	0.05	3,622.80	7.25	7.26	0.01	3,613.41
	500	9.62	9.87	0.23	3,619.05	8.15	8.24	0.13	3,613.71
	1500	9.65	10.32	0.43	3,615.32	9.20	9.40	0.13	3,612.30
	2500	8.05	8.23	0.19	3,605.98	7.85	8.00	0.13	3,607.16
Winnipeg	10	32.34	35.22	2.09	3,627.08	17.45	17.59	0.10	3,625.19
	100	20.41	20.90	0.41	3,627.86	15.50	15.62	0.09	3,619.40
	500	26.76	31.68	4.59	3,629.67	19.45	19.69	0.17	3,617.72
	1500	20.34	21.70	1.02	3,616.06	18.96	19.92	0.87	3,618.39
	2800	16.67	16.72	0.06	3,607.96	16.04	16.49	0.29	3,606.33
ChicagoSketch	10	100.18	100.30	0.07	7,267.29	19.24	19.44	0.12	7,254.08
	100	22.14	22.58	0.41	7,257.85	16.62	16.70	0.05	7,268.57
	500	22.77	24.29	0.96	7,268.40	17.99	18.25	0.26	7,277.62
	1500	76.87	154.37	62.94	7,243.13	19.27	20.46	0.77	7,246.76
	2900	16.95	17.51	0.45	7,218.88	15.72	16.04	0.20	7,212.14

Source: from the author (2015).

Table 2.7 presents, for each instance, the shortest average user travel time using tolls obtained by BRKGA for the tollbooth problem in comparison with the optimal distribution flow obtained by solving linear program MM1. An optimal solution for MM1 is a lower bound for the tollbooth problem. The results show that with tolls it is possible to obtain a near-optimal flow distribution.

Table 2.7 – Approximation of the lower bound with tolls

Instance	**Lower Bound**	**BRKGA+LS**
`SiouxFalls`	19.95	21.38
`Friedrichshain Center`	42.47	42.90
`Prenzlauerberg Center`	59.90	61.57
`Tiergarten Center`	52.57	52.87
`Mitte Center`	62.36	63.59
`Anaheim`	12.46	12.57
`MPH Center`	65.88	66.64
`Barcelona`	6.87	7.25
`Winnipeg`	13.67	15.50
`ChicagoSketch`	14.24	15.72

Source: from the author (2015).

We next explore the main characteristics of the near-optimal solutions found by BRKGA+LS. For the best solution found in the five runs, Table 2.8 lists the average number of paths for each OD pair (#Path), the average number of hops among all OD shortest paths (#Hop), the average sum, over all OD pairs, of the tariffs on the shortest paths (#Toll), and the average number of distinct arcs used, over all OD pairs (#UArc).

Columns #Path in Table 2.8 show that when $\mathcal{K}$ increases, a strong reduction in the number of shortest paths is observed for SPT, while for SPTF, the reduction is not as pronounced. Again, this occurs because of the large number of zero-cost paths present in SPT when $\mathcal{K}$ is small. Of these, traffic flows on one or more paths of minimum hop count. On the other hand, for SPTF, the inclusion of free flow time to the arc weight leads to paths of distinct cost, with a few of minimum cost (in many cases a single minimum cost path).

Columns #Hop in Table 2.8 show the minimum hop count distance between OD vertices. For large values of $\mathcal{K}$ we observe that as the number of installed tolls increases, the hop count decreases in both SPT and SPTF. This occurs because with a large number of tolls it is possible to do better traffic engineering. For small values of $\mathcal{K}$ in SPT, the hop count is small because it corresponds to a minimum hop-count path among the zero-cost shortest paths.

Table 2.8 – Detailed results of best solution found by BRKGA+LS algorithm

		SPT				SPTF			
Instance	$\mathcal{K}$	**#Path**	**#Hop**	**#Toll**	**#UArc**	**#Path**	**#Hop**	**#Toll**	**#UArc**
SiouxFalls	0	1.97	2.51	-	4.97	1.05	2.14	-	3.24
	10	1.78	2.58	0.21	4.84	1.07	2.10	0.38	3.25
	30	1.32	2.48	1.12	4.00	1.13	2.25	1.09	3.46
	50	1.13	2.28	1.89	3.50	1.09	2.22	1.88	3.34
	70	1.10	2.34	2.96	3.45	1.06	2.20	2.71	3.30
Friedrichshain Center	0	1.96	9.36	-	11.94	1.52	12.24	-	12.82
	10	1.91	9.99	0.21	12.21	1.63	13.63	0.31	13.04
	100	1.42	11.43	1.76	12.58	1.48	12.49	2.14	12.48
	300	1.17	11.50	5.39	12.33	1.00	10.68	5.57	11.68
	500	1.03	10.35	10.39	11.20	1.00	10.44	10.49	11.44
Prenzlauerberg Center	0	2.75	14.72	-	17.11	1.79	18.67	-	18.00
	10	2.61	15.65	0.07	18.02	1.53	16.48	0.38	17.19
	100	1.60	16.41	2.08	17.33	1.30	16.06	2.20	17.02
	450	1.15	15.85	8.07	16.69	1.07	15.86	7.95	16.70
	700	1.12	15.00	15.14	15.97	1.01	14.57	14.69	15.62
Tiergarten Center	0	1.92	15.40	-	17.32	1.12	17.43	-	18.51
	10	2.07	17.20	0.00	18.57	1.07	15.76	0.42	17.23
	100	1.20	16.37	1.01	18.08	1.04	15.98	2.35	17.26
	450	1.03	15.74	8.14	16.98	1.00	16.11	8.95	17.11
	700	1.00	15.81	13.74	16.83	1.00	15.97	15.20	16.97
Mitte Center	0	2.79	14.95	-	17.75	1.33	17.49	-	17.81
	10	3.12	16.81	0.00	18.99	1.38	17.37	0.48	17.60
	100	1.34	15.83	0.45	17.61	1.14	16.13	2.53	16.84
	400	1.05	15.84	6.65	16.77	1.00	15.29	6.58	16.29
	800	1.02	15.12	13.21	16.11	1.02	15.48	14.21	16.32
Anaheim	0	8.71	15.64	-	21.45	1.35	15.67	-	17.46
	10	16.79	19.71	0.00	23.19	1.35	15.16	0.07	16.96
	100	3.29	18.31	0.02	20.61	1.39	15.52	0.74	16.51
	500	1.05	14.78	6.48	15.89	1.04	14.31	6.43	15.45
	800	1.01	14.32	11.85	15.41	1.00	14.50	12.20	15.50
MPF Center	0	5.09	24.35	-	27.54	2.28	31.51	-	29.17
	10	5.25	26.00	0.00	27.95	2.15	30.50	0.23	28.48
	250	3.54	27.98	0.54	29.08	1.34	27.47	2.29	27.37
	1000	1.10	24.98	8.63	25.87	1.21	26.63	9.40	26.33
	2000	1.09	23.79	20.41	24.21	1.00	24.00	20.90	25.00
Barcelona	0	7.37	15.85	-	20.82	1.16	18.73	-	20.73
	10	7.00	18.28	0.01	24.80	1.13	18.70	0.07	20.85
	500	5.02	21.25	0.14	25.03	1.14	19.01	0.69	20.58
	1500	1.38	19.51	8.36	20.35	1.03	18.37	7.21	19.35
	2500	1.08	15.87	16.45	16.77	1.01	16.12	16.69	17.10
Winnipeg	0	3.92	20.72	-	25.96	1.04	24.98	-	25.73
	10	4.22	21.96	0.00	27.15	1.05	24.80	0.15	25.36
	500	3.02	31.04	0.82	32.02	1.17	25.69	2.32	26.11
	1500	1.65	31.76	9.83	25.91	1.11	24.78	9.82	24.84
	2800	1.04	19.77	20.12	20.82	1.01	20.15	20.60	21.14
ChicagoSketch	0	20.04	14.86	-	23.69	1.01	12.30	-	13.32
	10	21.24	15.23	0.00	24.35	1.01	12.19	0.09	13.20
	500	9.88	15.72	0.49	22.49	1.00	12.33	0.95	13.35
	1500	2.64	19.38	4.82	17.57	1.00	12.52	4.97	13.53
	2900	1.16	12.23	12.91	13.22	1.00	11.54	12.23	12.55

Source: from the author (2015).

The columns #Toll in Table 2.8 show the average number of tolls that a user traverses on an OD shortest path. Clearly, this value increases with $\mathcal{K}$. Since an increase in $\mathcal{K}$ leads to a decrease in the number of shortest paths (column #Path), the number of distinct arcs (column #UArc) consequently decreases.

2.5 Concluding remarks

In this chapter we presented an extensive study of the tollbooth problem. Mathematical formulations for different versions of the tollbooth problem were presented, as well as linearizations that give lower and upper bounds for their objective functions. Computational tests were conducted taking into account the original and the linearized models, applied on two sets of synthetic and real-world instances. Moreover, a random-key genetic algorithm was run for this same set of instances.

When analyzing the results for the mathematical models, we concluded that the model MM2, which includes shortest paths and even-split constraints, has a large number of variables and constraints, making it difficult to be solved with general-purpose solvers, even when we limit ourselves to small instances. On the other hand, if shortest paths and even-split constraints are removed from the model, giving rise to a simplified version of the problem, the linearized versions of the problem can be solved efficiently with CPLEX. However, results obtained with the biased random-key genetic algorithm for the complete model shows it has a good tradeoff between computation time and solution quality on this problem.

Finally, considering that users naturally take the least costly path, toll setting can be used to better distribute the flow in the network and consequently reduce traffic congestion.

3 STACKELBERG NETWORK PRICING PROBLEM

Bilevel programming is a suitable framework to model asymmetric games, where one player (the leader) is the first to make a decision, taking into account the optimal reaction of the second player (the follower). In network optimization problems, especially in transportation networks, this framework is often used to model the reactions of the users in a network subjected to changes made by an agent that controls traffic. In a more general way, this framework is suitable for modeling problems with conflict of interest where an objective predominate over secondary objectives.

Bilevel programming problems are introduced by Heinrich von Stackelberg (STACKELBERG, 1934) when to investigating the problem known as *Stackelberg game*. From the combinatorial optimization community, bilevel programming first appears in Bracken and McGill (1973), although the term *bilevel* and *multilevel programming* are only introduced by Candler and Norton (1977). More recently, Colson et al. (2005), Colson et al. (2007) present a review of bilevel problems with applications and resolution methods. The book *Foundations of Bilevel Programming* (DEMPE, 2002) gives the main foundations, examples of applications and theoretical results for bilevel problems. The book *Metaheuristics for Bi-level Optimization* (TALBI, 2013) provide a background to implement metaheuristics for this type of problems with examples for many real-world applications.

In this chapter, we address the *Stackelberg network pricing problem*, a bilevel problem with application in the transportation networks. In this hierarchical problem, the two decision-making levels are conflicting. The lower level (followers), always take the decision that minimize your cost. The higher level (leader), considering the optimal decision on the lower level to make the decision seeking to maximize an objective. In analogy with transportation networks, the lower level decision can be a set of users traveling in a network choosing your routes based on the least cost path. The higher level can be an agent that controls the tariffs in a set of tolled arcs with the objective to maximize the revenue collected in these arcs. In this work, we apply a biased random-key genetic algorithm for large instances of this problem. The experimental results show that the algorithm found good solutions for large instances in a short time. This chapter is based on Stefanello et al. (2013).

3.1 Introduction

Transportation is an important component of the economy, a promoter of wealth, the development and the welfare of populations. Reducing the problems related to transportation networks has challenged not only traffic engineers but also researchers from several areas. Many rules and procedures are currently in use aiming at improving the traffic flow in a city or road. They can also attend other interests. Motivating the use of bicycle, for example, reduces traffic congestion and improves lifestyle. In spite of the world effort in reducing traffic flow, the high traffic still generating congestion problems in almost all cities and main roads. Another alternative to reduce some problems is applying tolls on some arcs of the network.

The main idea is that the deployment of tolls on certain roads can induce drivers to choose alternative routes, thus reducing congestion as the result of better traffic flow distribution. Naturally, tolls can increase the cost of a trip, but this can be compensated with less travel time, reduced fuel cost, and lower amounts of stress. In the 1950s, Beckmann et al. (1956) proposed the use of tolls with this objective. This idea has made its way into modern transportation networks. In 1975, Singapore implemented a program called *Electronic Road Pricing* or ERP. Several cities in Europe and the United States, such as in London and San Diego, have begun to charge toll on their transportation networks (BAI et al., 2010). Tolls are also applied in some small European towns, like Perugia (Italy), to reduce the number of people driving in downtown areas.

The optimization of transportation networks using tolls is addressed by many works. The goal of the minimum tollbooth problem (MINTB), first introduced by Hearn and Ramana (1998), is to minimize the number of toll locations to achieve system optimality. Yang and Zhang (2003) formulate second-best link-based pricing as a bi-level program and solve it with a genetic algorithm. In Bai et al. (2010) it is shown that the problem is *NP-hard* and a local search heuristic was proposed. In Stefanello et al. (2015c), an extension of Buriol et al. (2010), the authors deal with the problem of locating a fix number $\mathcal{K}$ of tolls, as well as defining their tariffs. For a complete of road pricing optimization problems, we refer the reader to the survey by Tsekeris and Voß (2008).

Another class of problems on flow networks is defined when only a given subset of the arcs can be tariffed. This is the case of the network pricing problem (NPP) introduced by Labbé et al. (1998), which is further explored in this work. In an NPP, an authority imposes charging tolls in a given set of arcs with the objective of maximizing the revenue, supposing that travellers always take the shortest cost path. The shortest path is computed considering the tolls and a fix link cost. In game theory, a similar problem is known as the Stackelberg game (STACKELBERG, 1952). In this game there is a leader and a follower. The leader plays first choosing the best strategy supposing that the follower reacts in an optimal way to its choice. Knowing the decision of the leader, the follower chooses a strategy considering its own benefit. Similar problems and applications are found in toll optimization systems (DEWEZ, 2004), long-distance freight transportation overseas (BROTCORNE et al., 2000), airline charging (CASTELLI et al., 2013), and in telecommunication networks (BASAR; SRIKANT, 2002; BOUHTOU et al., 2007).

A bilevel NPP was first introduced by Labbé et al. (1998) for a multicommodity network. The problem consists in determining the tariffs to tolls on a subset of arcs of a network, with the objective of maximizing the profit, given that users travel on shortest cost paths. In this problem, the authority is supposed to fix its toll prices first, and then the users choose their paths having the complete knowledge of all network costs. In Labbé et al. (1998) the general problem was proved to be NP-complete, while particular instances are polynomially solvable, as the single toll arc case. In the same work, it was also showed how the lower level optimization problem can be replaced by its primal and dual constraints and its optimality conditions, stating that the primal and dual objective functions must be equal, yielding a single level formulation. In Roch et al. (2005) the NPP with lower bound constraints on tolls was proved to be strongly NP-hard even for one single commodity. Other results can be found for instance in Bouhtou et al. (2007), Dewez (2004), Heilporn et al. (2010), and Brotcorne et al. (2012).

In this chapter, we approach the multicommodity network pricing problem named as Stackelberg network pricing problem. The objective is to determine the tariff of a given subset of arcs of the network maximizing the revenue computed by travelers that choose their shortest cost path routes. This work proposes a biased random-key genetic algorithm (BRKGA) to solve large scale instances from the bilevel multicommodity network pricing problem. We further present a set of experiments, providing some conclusions and research directions.

This chapter is organized as follows. In Section 3.2 we present an overview and a mathematical formulation for the Stackelberg network pricing problem. The biased random-key genetic algorithm to solve this problem is presented in Section 3.3. Computational results are reported in Section 3.4, and conclusions are drawn in Section 3.5.

3.2 The Stackelberg Network Pricing Problem

Consider a network represented by a directed graph $G = (V, A, c)$ where V represents the set of nodes (i.e., vertices or points of interest), and A the set of arcs (i.e., links or road segments). The set A is partitioned into two subsets, i.e., $A = A_1 \cup A_2$. Subset A_1 contains the $\mathcal{K} = |A_1|$ arcs that can be tariffed, and belongs to the leader while A_2 is owned by another agent in the network and the arc costs t_a are known *a priori*. Besides the tariffs, arcs from $\mathcal{K} = |A_1|$ also has a cost t_a. Thus, the arcs belonging to A_1 have cost $t_a + w_a$, while the arcs that belong to A_2 have cost t_a, where t_a is the fix cost of the arc and w_a is the tariff in the arc a.

In addition, let $K = \{(o(1), d(1)), (o(2), d(2)), \dots, (o(|K|), d(|K|)\} \subseteq V \times V$ denote the set of commodities or origin-destination (OD) pairs, where $o(k)$ and $d(k)$ represent, respectively, the origin and destination nodes for $k = 1, \dots, |K|$. Each commodity $k = 1, \dots, |K|$ has an associated demand of traffic flow d_k, i.e., for each OD pair $(o(k), d(k))$, there is an demand d_k that emanates from node $o(k)$ and terminates in node $d(k)$.

To ensure that the problem is bounded, it is assumed that for each commodity $k \in K$ there is an upper bound on the amount the customer is willing to pay, or there exists a path from origin to destination that uses only fixed cost arcs $a \in A_2$.

The multicommodity NPP has been formulated as follows (LABBÉ et al., 1998):

$$\max \sum_{k \in K} \sum_{a \in A_1} w_a x_a^k \tag{3.1}$$

$$\min \sum_{k \in K} \left\{ \sum_{a \in A_1} (t_a + w_a) x_a^k + \sum_{a \in A_2} t_a x_a^k \right\} \tag{3.2}$$

$$\sum_{a \in IN(v)} x_a^k - \sum_{a \in OUT(v)} x_a^k = \begin{cases} -d_k, & \text{if } v = d(k) \\ d_k, & \text{if } v = o(k) \\ 0, & \text{otherwise} \end{cases} \qquad \forall v \in V,\ \forall k \in K, \tag{3.3}$$

$$x_a^k, w_{a'} \in \Re^+ \qquad \forall a' \in A_1,\ \forall a \in A,\ \forall k \in K \qquad . \tag{3.4}$$

Here $IN(v)$ is the set of arcs entering v, and $OUT(v)$ is the set of arcs leaving v. In this model, the variables w_a represent the values of tolls on arcs $a \in A_1$ and the variables x_a^k represent the flow of commodity $k \in K$ on arc $a \in A$.

This model is a bilinear bilevel program since the upper level is linear in the tariff variables and the lower level is linear in the arc choice variables (HOESEL, 2008), resulting in a nonlinear formulation in the combination of these variables.

Observe that once the tariffs are defined, one can easily choose among all s-t paths of minimum total cost one path that maximizes the revenue. In other words, we assume that the follower always makes the best choice for the leader. A natural extension of the NPP is to allow negative tariffs. In this case, the values are incentives to travelers take this routes. In this work we limit only to non-negative tariffs.

A mixed integer linear formulation is provided for this problem in Labbé et al. (1998). This model was obtained by replacing the lower level problem by its optimality conditions and the flow variables in the upper level for proportion of the demand assigned to the tolled arcs. Thus, an arc formulation for NPP has been formulated as follows:

$$\max \sum_{k\in K}\sum_{a\in A_1} d_k w_a^k \tag{3.5}$$

subject to

$$\lambda_i^k - \lambda_j^k \le t_a + W_a \qquad \forall a=(i,j)\in A_1, \forall k\in K \tag{3.6}$$

$$\lambda_i^k - \lambda_j^k \le t_a \qquad \forall a=(i,j)\in A_2, \forall k\in K \tag{3.7}$$

$$\sum_{a\in A_1}\left(t_a x_a^k + w_a^k\right) + \sum_{a\in A_2} t_a x_a^k = \lambda_{d(k)}^k - \lambda_{o(k)}^k \qquad \forall k\in K \tag{3.8}$$

$$\sum_{a\in IN(v)} x_a^k - \sum_{a\in OUT(v)} x_a^k = \begin{cases} -1, & \text{if } v = d(k) \\ 1, & \text{if } v = o(k) \\ 0, & \text{otherwise} \end{cases} \qquad \forall v\in N, \forall k\in K \tag{3.9}$$

$$-Mx_a^k \le w_a^k \le Mx_a^k \qquad \forall a\in A_1, \forall k\in K \tag{3.10}$$

$$-M\left(1-x_a^k\right) \le w_a^k - W_a \le M\left(1-x_a^k\right) \qquad \forall a\in A_1, \forall k\in K \tag{3.11}$$

$$x_a^k\in\{0,1\}, w_a^k, W_a \in \Re^+ \qquad \forall a\in A_1, \forall k\in K \tag{3.12}$$

$$x_a^k \ge 0 \qquad \forall a\in A_2, \forall k\in K \tag{3.13}$$

$$\lambda_v^k \ge 0 \qquad \forall v\in V, \forall k\in K \tag{3.14}$$

Objective function (3.5) maximizes the revenue of the demand for each commodity that traverses the tariffed arcs. Constraint set (3.6) and (3.7) defines the weight distance from the vertices, since variables λ store the distance from vertice i to the destination of the commodity k. Variables W_a represent the value of the tariff for the arc a while w_a^k is used to calculate the tariff for each commodity. Constraint set (3.8) helps to define the tariffs of the tariffed arcs based on the vertices distances. Constraint set (3.9) guarantees flow conservation. Constraint set (3.10) and (3.11) set the tariffs only on arcs with flow, and the last constraint sets define the domain of the variables (3.12)–(3.14).

This model can be improved in the case of having many commodities to the same destination. The set of variables $\lambda_v^k, \forall\ v \in V$ and $\forall k \in K$ can be replaced by a set of variables $\lambda_v^q, \forall v \in V$ and $\forall q \in \mathcal{Q}$ where $\mathcal{Q}$ is the set of all destination nodes. For a large number of commodities $|\mathcal{Q}| \ll |K|$ this modification reduces the number of variables in comparison to the original model.

3.3 A biased random-key genetic algorithm

In this section we describe the biased random-key genetic algorithm (BRKGA) proposed to solve the Stackelberg network pricing problem. The general framework of this method is described in Section 1.2. Next, the encoding and decoding procedure are described, and the values of parameters, as well as the stopping criterion, are given in Section 3.4.

Solutions are encoded as an n-vector of random keys, where $n = |\mathcal{K}|$ is the cardinality of the tariffed arcs (A_1) in the network. Each random-key corresponds to a tariff that is decoded by $\lfloor \mathcal{X}_i * w_{max} \rfloor$, where $\mathcal{X}_i$ is the random-key associated with the tolled arc $i \in A_1$. Let d_k^0 the shortest distance from node s to node t of the commodity k when the tariffed arcs are defined to zero and d_k^∞ the shortest distance from s to t for commodity k when the tariffed arcs are defined to a sufficiently large value. Thus, a natural upper bound $w_{\max}$ on the value of a tariff on the network is given by

$$w_{max} = \max_{k \in K} \left\{ d_k^\infty - d_k^0 \right\}.$$

With the tariffs for each tolled arc defined, demands are routed forward to their destinations on shortest weight paths. Tariffed links have weights equal to their fixed costs plus their tariffs $(t_a + w_a)$, and untariffed links have only a fixed cost (t_a). For each commodity, all paths of minimum cost are evaluated, and the demand is sent by the higher cost path. In the case of a tie, the path with the higher number of tariffed arcs is selected. Finally, in the case of a second tie, the tie is break using the outgoing arc with the lowest index. Once the demands are routed, the revenue of each tolled arc can be computed for all commodities. The solution fitness value is then calculated and associated with the respective vector of random-key.

Two kinds of initial solutions are generated. Note that any set of positive tariffs produces feasible solutions. First, the model (3.5)-(3.14) is solved with constraint (3.12) relaxed. We observed that the values of tariffs of this solution were, in most cases, an upper approximation of the tariffs of the optimal solution. The values of tariffs resulted from the solver are recoded to the first random-key vector. The remaining solutions are randomly generated, i.e., each random-key is a random number in the interval $[0, 1]$.

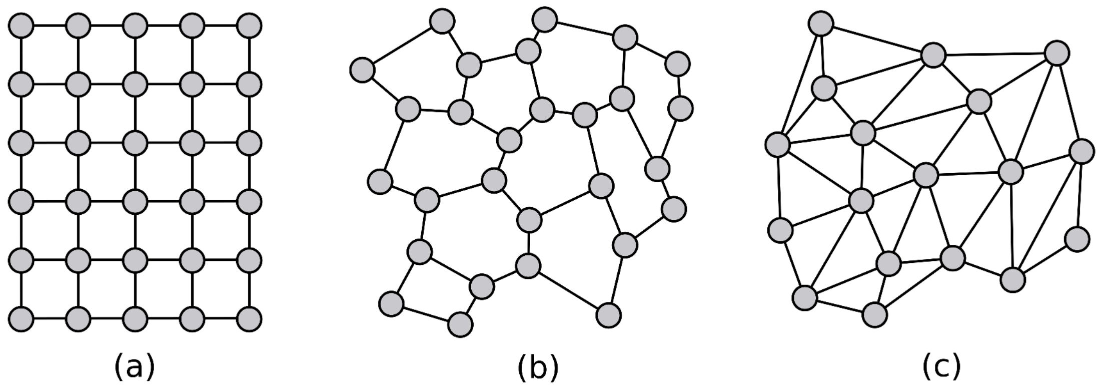
3.4 Computational results

In this section we present computational experiments with the models and algorithms presented in the previous sections. Initially, we describe the dataset used in the experiments. Then, we detail some experiments with CPLEX applied on the mathematical model (3.5)–(3.14) and a component based on the relaxation of this model added to a basic implementation of the BRKGA that help to improve the results. Finally, preliminary results for the BRKGA are reported with some considerations.

The experiments were done on a computer with an Intel Core i5 2300 processor running at 2.80 GHz, with 4 GB of DDR3 RAM of main memory, and Ubuntu 12.10 Linux operating system. The BRKGA was implemented in C++ and compiled with the g++ compiler, version 4.7.2. We used CPLEX 12.4[1] (API C++) with default configuration.

Three types of networks are used in the first experiment, as show in Figure 3.1:

Figure 3.1 – Network structures: (a) Grid network; (b) Voronoi network; (c) Delaunay network.



Source: from the author (2015).

For each edge from the original structure two directed arcs with opposite directions were created. A random weight $t_a \in [1, t_{max}]$ is assigned to each one. Distinct origin and destination nodes of each commodities are also randomly chosen, as well as the demand is randomly generated in the interval $[1, d_{max}]$ (by default $d_{max} = 20$).

[1]<www.cplex.com>

Tariffed arcs are selected based on the description provide on Brotcorne et al. (2000). However, some changes were applied to increase the number of travelers through tariffed arcs for increasing the difficulty to solve the problem. Shortest paths are then calculated. The frequency that each arc belongs to the shortest path is computed. The first $|A_1|/2$ arcs are selected considering the decreasing order of frequencies and the weights of these arcs are increased by a large value (usually a maximum integer representation). Their frequency count are also increased by a large value (it was used $|A|$). Next, considering the new values of weights, the demands are routed again, and the frequencies are updated. Finally in a decreasing order of frequencies, the tariffed arcs are selected. Once an arc is tariffed, a test is performed to check if there is at least one untariffed path from each commodity. If there is no such a path, the toll is removed and the next arc is selected to receive the toll. This process is repeated until $\mathcal{K}$ tariffs are assigned.

Sets of instances are created with 200, 500 and 1000 arcs (the exact number of arcs can vary according to the network structure). The fixed maximum weight of arcs t_{max} is defined to 10 and 30. The number of origin-destination pairs is selected from 50, 200 and 500, and the proportion of tariffed arcs from 10% and 20%, generating a total of 108 instances[2].

3.4.1 Results for the mathematical model

This subsection reports experiments with CPLEX running the model (3.5)–(3.14) considering the subset of instances with 200 arcs from Table 3.1.

The columns present the name of the instances, the CPLEX gap and the computational time (limited to 1h run). The name of each instance is composed by the network structure, #arcs, #o-d pairs, and the max arc weight t_{max}. A '-' in the table indicates the cases in which the computer memory was exceeded and then the run was interrupted.

Table 3.1 shows, as expected, that a higher number of tariffed arcs increases the difficulty of the solver to prove optimality. The same is observed for the number of commodities. This occurs because the variables of this model are directly related to these parameters. In general, as higher the number of variables, more difficult is to solve the problem.

[2]Available at <www.inf.ufrgs.br/~fstefanello>

Table 3.1 – Computational results for the mathematical model

	$\mathcal{K} = 10\%$		$\mathcal{K} = 20\%$	
Instance	**gap**	**Time(s)**	**gap**	**Time(s)**
`Grid-0200-0050-10`	0.00	9.94	3.46	3,600.23
`Grid-0200-0050-30`	0.00	2.69	0.00	126.57
`Grid-0200-0200-10`	-	-	200.09	3,600.66
`Grid-0200-0200-30`	54.24	3,600.32	286.44	3,600.98
`Grid-0200-0500-10`	451.26	3,601.28	366.22	3,600.13
`Grid-0200-0500-30`	734.92	3,600.06	1,974.75	3,600.08
`Delaunay-0200-0050-10`	0.00	1.44	0.00	2.88
`Delaunay-0200-0050-30`	0.00	11.33	0.00	105.05
`Delaunay-0200-0200-10`	0.00	104.75	13.41	3,606.50
`Delaunay-0200-0200-30`	1.74	3,601.21	11.37	3,605.06
`Delaunay-0200-0500-10`	118.84	3,602.50	731.74	3,600.06
`Delaunay-0200-0500-30`	-	-	604.50	3,601.28
`Voronoi-0200-0050-10`	0.00	1.57	0.00	169.75
`Voronoi-0200-0050-30`	0.00	6.67	0.00	1,099.20
`Voronoi-0200-0200-10`	5.08	3,600.49	117.19	3,600.79
`Voronoi-0200-0200-30`	3.99	3,600.39	92.02	3,600.58
`Voronoi-0200-0500-10`	-	-	-	-
`Voronoi-0200-0500-30`	-	-	-	-

Source: from the author (2015).

A relaxed version of the model was also solved. Defined the tariffs obtained by the solver, that can be fractional numbers, the demands are routed and the revenue is computed. Table 3.2 shows a comparison of the solution obtained by the relaxed model and 1000 randomly generated solutions for two values of $\mathcal{K}$.

In average, the solution value is about 40% less of the optimal values found in the previous experiment. Also, the relaxed solutions are in average, better than a randomly generated solution. In a set of 1000 random solutions for each instance, the average gap quality is around 64% less than the revenue of the relaxed solution. For the best of 1000 random solutions, the gap is still 20% less than the revenue of the relaxed solution.

Table 3.2 – Comparison of the solution obtained with the relaxed model and 1000 randomly generated solutions

	$\mathcal{K} = 10\%$			$\mathcal{K} = 20\%$		
	relaxed	**Random**		**relaxed**	**Random**	
Instance	**model**	**avg**	**max**	**model**	**avg**	**max**
`Grid-0200-0050-10`	2,227.0	928.9	2,191.0	4,094.1	825.3	1,981.0
`Grid-0200-0050-30`	4,508.1	1,199.5	3,229.0	10,704.0	2,719.5	7,367.0
`Grid-0200-0200-10`	5,537.4	1,512.2	5,043.0	8,470.5	1,916.3	6,026.0
`Grid-0200-0200-30`	13,129.1	5,294.6	13,176.0	25,584.2	8,114.9	20,491.0
`Grid-0200-0500-10`	10,586.6	5,363.2	13,535.0	17,619.1	5,195.5	12,702.0
`Grid-0200-0500-30`	27,783.2	10,326.9	25,983.0	41,060.2	14,674.6	32,904.0
`Delaunay-0200-0050-10`	652.0	234.3	511.0	1,090.0	325.6	773.0
`Delaunay-0200-0050-30`	1,290.3	461.5	965.0	2,526.0	536.9	1,198.0
`Delaunay-0200-0200-10`	1,666.0	702.7	1,486.0	2,179.4	845.2	1,706.0
`Delaunay-0200-0200-30`	4,560.0	1,413.1	3,075.0	8,293.9	3,143.2	6,409.0
`Delaunay-0200-0500-10`	3,722.6	1,985.1	3,618.0	6,125.5	2,646.3	5,049.0
`Delaunay-0200-0500-30`	8,500.8	4,327.0	8,873.0	14,621.5	6,787.2	11,437.0
`Voronoi-0200-0050-10`	1,263.8	400.6	986.0	2,799.0	469.1	1,345.0
`Voronoi-0200-0050-30`	2,896.1	832.1	2,099.0	7,251.5	1,342.3	3,389.0
`Voronoi-0200-0200-10`	3,621.4	1,431.4	2,881.0	7,423.3	3,257.3	5,888.0
`Voronoi-0200-0200-30`	11,523.0	4,751.9	11,726.0	23,367.5	6,647.6	14,742.0
`Voronoi-0200-0500-10`	7,248.7	2,757.8	6,088.0	21,236.2	9,992.7	20,699.0
`Voronoi-0200-0500-30`	22,606.0	10,611.1	22,064.0	42,007.3	14,405.6	29,920.0

Source: from the author (2015).

Analysing the value of tariffs of the relaxed model, we observed that, in most arcs, tariffs are higher or equal than the optimal tariffs values. Comparing the values of tariffs obtained from the relaxed model with the tariffs of the optimal solutions obtained in the previous experiment, we observe that 42.5% of the values of tariffs are equal, 43.7% are higher, and only 13.8% of the tariffs for the relaxed model are smaller that the optimal solution. Furthermore, the average of the difference between the values is 3.21 units, meaning that the obtained tariff values are close to the optimal values. Thus, these values can be used to provide good approximations for the tariff values. The time to compute this solution is less than 2 seconds for the instances from Table 3.1. This time represents less than 3% of the BRKGA time reported in the next subsection.

3.4.2 Results from the biased random-key genetic algorithm

This subsection presents results obtained with the biased random-key genetic algorithm applied to a set of large scale instances of the NPP. The experiments with the BRKGA were done with a population size of $p = 50$, an elite set of size $p_e = 0.25p$, a mutant set of size $p_m = 0.05p$, an elite key inheritance probability of $\rho_A = 0.7$, and the restart parameter equal to 50 (number of generations without improvement).

Table 3.3 shows the results of the BRKGA. The results are an average over ten runs with different random seeds and using as stopping criterion of 2000 generations. In this table we report the best know solution value (Best) found by the CPLEX in the previous experiment (optimal values are given in bold), or an extended execution of the BRKGA to 3000 generations. The columns Relax and Time(s) report the CPLEX results for the relaxed model (3.5)–(3.14). The column Relax shows the total revenue calculated with the tariffs obtained by the solver. Since the tolls values in the relaxed version can be non-integer, the revenue can also be fractional. The column Time(s) show the running time of the solver in seconds. The columns Avg, Max, SD and Time(s) report respectively the values of the average revenue, best revenue, standard deviation and the computational time in seconds for the BRKGA. The reported running time is not cumulative with the running time of the relaxed model. Finally, we report results for the cases where 10% and 20% of the arcs are tariffed.

Table 3.3 – Computational results for the BRKGA

	$\mathcal{K} = 10\%$							$\mathcal{K} = 20\%$						
Instance	**Best**	**Relax**	**Time(s)**	**Avg**	**Max**	**SD**	**Time(s)**	**Best**	**Relax**	**Time(s)**	**Avg**	**Max**	**SD**	**Time(s)**
`Delaunay-0200-0050-10`	**747**	652.0	0.1	747.0	**747**	0.0	39.4	**1473**	1,090.0	0.1	1,472.6	**1,473**	1.3	37.7
`Delaunay-0200-0050-30`	**1,814**	1,290.3	0.1	1,799.0	1,807	5.7	37.4	**3144**	2,526.0	0.1	2,985.3	**3,114**	82.4	38.1
`Delaunay-0200-0200-10`	**2,236**	1,666.0	0.5	2,236.0	**2,236**	0.0	86.4	3339	2,179.4	0.7	3,270.6	3,331	30.2	83.9
`Delaunay-0200-0200-30`	6,451	4,560.0	0.7	6,388.6	6,451	47.4	76.7	14001	8,293.9	0.8	13,612.4	13,876	148.1	75.4
`Delaunay-0200-0500-10`	6,473	3,722.6	2.7	6,473.0	6,473	0.0	169.7	9599	6,125.5	2.6	9,432.4	9,599	111.2	153.5
`Delaunay-0200-0500-30`	15,507	8,500.8	2.8	15,419.4	15,492	52.1	143.9	24540	14,621.5	4.3	23,799.0	24,540	457.2	142.4
`Delaunay-0500-0050-10`	1,227	1,008.0	0.4	1,222.1	1,227	4.5	105.4	2048	1,710.0	0.5	2,028.9	2,048	12.1	109.3
`Delaunay-0500-0050-30`	3,849	2,224.4	0.5	3,756.4	3,847	52.2	94.5	4774	3,457.0	0.6	4,611.0	4,774	146.3	96.6
`Delaunay-0500-0200-10`	3,089	2,160.5	3.1	3,058.5	3,089	12.8	241.4	6495	4,304.7	4.0	6,318.0	6,430	99.8	237.0
`Delaunay-0500-0200-30`	11,961	8,044.6	3.5	11,786.6	11,961	101.1	219.5	19756	12,595.3	3.9	19,199.4	19,663	208.0	220.4
`Delaunay-0500-0500-10`	7,354	4,446.9	9.3	7,256.6	7,343	69.1	354.6	13980	8,046.2	11.0	13,546.2	13,861	153.5	358.4
`Delaunay-0500-0500-30`	26,640	13,822.5	10.4	26,167.9	26,619	432.9	334.9	39004	21,382.8	10.8	37,988.9	38,890	586.6	327.4
`Delaunay-1000-0050-10`	1,696	1,239.4	1.5	1,688.3	1,696	6.3	232.2	2388	1,971.0	2.1	2,359.5	2,383	15.8	237.0
`Delaunay-1000-0050-30`	4,410	3,277.0	1.6	4,309.8	4,407	72.0	216.4	7580	6,050.0	3.1	7,417.1	7,515	69.4	218.8
`Delaunay-1000-0200-10`	5,601	3,763.5	7.0	5,536.4	5,600	51.7	654.0	9155	6,300.9	11.5	8,964.6	9,102	104.8	646.4
`Delaunay-1000-0200-30`	15,018	8,189.4	11.5	14,716.6	14,949	158.9	609.3	29476	19,014.3	13.2	28,696.3	29,389	478.2	612.4
`Delaunay-1000-0500-10`	16,612	9,064.0	24.2	16,290.5	16,486	222.2	943.3	25151	13,970.9	36.2	24,656.9	25,149	304.7	939.9
`Delaunay-1000-0500-30`	25,710	16,614.3	27.2	25,373.7	25,685	269.1	907.1	61565	34,601.9	35.9	60,098.7	61,015	551.0	895.9
`Grid-0200-0050-10`	**38,93**	2,227.0	0.2	3,893.0	**3,893**	0.0	51.2	**5525**	4,094.1	0.2	5,313.7	5,441	81.8	52.1
`Grid-0200-0050-30`	**65,43**	4,508.1	0.2	6,496.7	6,501	2.9	45.6	14151	10,704.0	0.2	13,694.5	14,012	162.9	49.2
`Grid-0200-0200-10`	10,454	5,537.4	1.0	10,407.9	10,454	39.5	109.6	15898	8,470.5	1.3	15,620.3	15,898	252.1	110.7
`Grid-0200-0200-30`	23,760	13,129.1	1.0	23,065.9	23,672	421.9	99.4	49638	25,584.2	1.3	48,745.1	49,638	1,054.4	101.8
`Grid-0200-0500-10`	18,945	10,586.6	2.9	18,825.5	18,945	82.3	206.1	34367	17,619.1	3.3	33,905.8	34,342	259.2	203.2
`Grid-0200-0500-30`	56,834	27,783.2	3.7	55,965.1	56,580	614.8	186.6	94813	41,060.2	4.4	91,049.2	93,937	1,869.4	192.4
`Grid-0500-0050-10`	6,779	3,834.0	1.1	6,711.6	6,779	54.6	133.8	10103	5,395.2	1.7	9,719.6	10,065	316.9	136.2
`Grid-0500-0050-30`	7,223	4,312.0	0.9	7,099.1	7,211	76.7	123.7	16499	13,211.4	1.3	15,890.0	16,370	363.3	126.1
`Grid-0500-0200-10`	11,227	6,321.8	4.8	11,067.3	11,213	72.4	344.9	23527	11,657.4	4.2	23,002.8	23,450	368.3	355.7
`Grid-0500-0200-30`	40,486	21,479.2	4.8	38,827.3	40,301	913.1	335.8	75548	32,123.7	5.9	73,920.6	75,464	1,241.8	332.3
`Grid-0500-0500-10`	17,662	8,457.0	14.2	17,491.9	17,636	109.5	539.1	54182	18,837.3	16.2	52,598.8	53,999	1,041.1	601.5
`Grid-0500-0500-30`	79,060	36,411.6	14.1	77,912.9	79,007	795.7	524.9	129323	57,971.3	21.7	125,633.8	128,804	2,106.5	534.4
`Grid-1000-0050-10`	5,612	4,040.0	4.1	5,508.6	5,608	55.3	276.2	11697	8,743.0	7.3	11,336.8	11,665	150.5	277.5
`Grid-1000-0050-30`	12,731	8,129.0	4.7	12,342.1	12,604	280.6	264.4	24309	20,187.3	9.5	22,152.5	23,716	796.9	300.7
`Grid-1000-0200-10`	20,265	8,740.1	15.5	20,008.7	20,237	158.4	865.8	37919	16,049.5	25.3	36,895.8	37,300	298.8	895.9
`Grid-1000-0200-30`	53,043	20,330.4	13.9	51,180.6	53,012	939.0	840.3	91484	47,980.6	24.0	87,838.7	91,253	2,780.3	828.7
`Grid-1000-0500-10`	40,893	13,270.9	54.0	40,353.4	40,844	406.6	1,497.0	71311	26,301.5	76.2	69,605.2	70,894	753.4	1,501.9
`Grid-1000-0500-30`	80,735	33,753.0	45.3	79,056.6	80,327	1,108.0	1,369.6	192204	71,360.8	66.3	185,104.5	190,730	3,046.9	1,363.6
`Voronoi-0200-0050-10`	**15,59**	1,263.8	0.2	1,559.0	**1,559**	0.0	58.1	**3643**	2,799.0	0.2	3,391.4	3,511	76.9	60.5
`Voronoi-0200-0050-30`	**42,59**	2,896.1	0.2	4,149.2	4,204	27.1	54.8	**8550**	7,251.5	0.2	7,980.6	8,333	163.5	57.9
`Voronoi-0200-0200-10`	5,563	3,621.4	1.0	5,518.9	5,563	40.7	130.6	11535	7,423.3	1.4	11,423.1	11,535	91.2	134.5
`Voronoi-0200-0200-30`	20,048	11,523.0	1.1	19,826.0	20,048	290.8	120.4	37669	23,367.5	1.5	37,009.2	37,665	506.6	119.0
`Voronoi-0200-0500-10`	11,002	7,248.7	3.4	10,785.7	11,002	183.4	195.2	42877	21,236.2	3.5	41,801.0	42,546	763.4	206.3
`Voronoi-0200-0500-30`	45,009	22,606.0	3.9	44,417.0	44,987	617.3	195.8	80054	42,007.3	4.3	78,602.0	79,849	638.9	198.3
`Voronoi-0500-0050-10`	4,064	2,949.2	1.3	4,001.2	4,064	29.5	137.2	8315	5,663.6	1.6	8,116.0	8,290	85.6	141.4
`Voronoi-0500-0050-30`	9,144	5,500.7	1.1	8,785.9	9,089	215.3	157.3	21317	19,580.0	1.7	21,100.9	21,313	156.0	164.0
`Voronoi-0500-0200-10`	7,940	4,599.3	5.3	7,828.0	7,940	84.6	416.4	18080	12,411.0	5.0	17,610.7	18,033	266.9	421.4
`Voronoi-0500-0200-30`	33,699	18,590.5	4.6	32,539.8	33,547	640.0	411.5	72276	44,523.4	6.6	70,345.8	71,915	1,226.7	417.1
`Voronoi-0500-0500-10`	20,377	10,262.8	14.3	19,927.3	20,244	252.7	681.1	48475	24,588.8	18.6	47,365.8	48,111	597.1	702.6
`Voronoi-0500-0500-30`	75,253	32,314.0	17.5	73,406.3	74,777	943.8	674.9	138913	55,682.3	23.0	135,419.4	137,742	1,753.8	674.1
`Voronoi-1000-0050-10`	4,795	3,492.3	2.6	4,737.8	4,784	22.7	320.5	10806	9,382.0	4.6	10,721.7	10,777	39.7	337.9
`Voronoi-1000-0050-30`	11,162	9,787.0	3.2	11,071.0	11,161	91.9	322.7	22135	18,066.0	7.2	21,183.6	22,119	650.4	335.5
`Voronoi-1000-0200-10`	12,014	7,579.0	15.6	11,496.6	11,984	261.1	985.8	31665	20,183.9	28.2	30,933.8	31,490	348.8	998.1
`Voronoi-1000-0200-30`	40,808	26,114.4	15.2	39,592.2	40,488	510.3	1,038.8	71007	44,108.7	21.5	68,606.4	70,347	821.8	1,050.3
`Voronoi-1000-0500-10`	32,399	15,755.9	48.6	31,902.9	32,299	354.3	1,869.8	83231	44,191.7	66.9	81,426.4	82,926	1,093.8	1,934.6
`Voronoi-1000-0500-30`	88,483	38,070.3	61.1	85,783.4	88,051	1,625.9	1,827.7	226550	146,376.1	65.2	218,943.5	223,702	2,974.4	1,890.9

Source: from the author (2015).

For the small instances, the BRKGA found the optimal solution or slightly less revenue than the optimal solution value. This indicates that at least for this set of instances, the proposed algorithm has a good performance, and we expect similar behavior for the other sets of instances. Related to the network structure, we observed that the grid networks have an average of standard deviation and running times slightly worse than the other structures. This occurs because in this kind of network the path length for each commodity tends to be higher than in Voronoi and Delaunay networks. By the same reason, in this kind of structures the revenue tends to be higher.

In the instances with weights t_a between $[1, 30]$ we observed that the standard deviation is worse than in the case of the weight between $[1, 10]$. Naturally this behavior is expected because in the first case a higher variation of the tariff values and revenue is observed.

3.5 Concluding remarks

In this chapter we presented a study of the Stackelberg network pricing problem. A biased random-key genetic algorithm is proposed to solve the problem, and a set of experiments was performed in a set of large scale network instances.

In the experiments, we observed that the relaxation of the arc formulation mathematical model solved by CPLEX provides a high-quality initial solution and a good approximation for the tariff values. Furthermore, CPLEX was not able to solve the mixed integer version of this model for the large scale networks instances, motivating the use of heuristics for solving the problem.

Finally, the BRKGA shows a good performance, reaching the optimal solution or a good approximation of the best revenue, even without include local search methods. We also evaluate some characteristics and behaviors of the proposed algorithm on the tested instances.

4 VMPLACEMENT PROBLEM

With the massive growth of data utilization over telecommunication networks, to provide a better use of the network resources has been a significant challenge for many service providers over the Internet. To apply techniques to minimize the traffic in a network can mean reduce operational costs or even become economically viable a new service. In this scenario, virtualization techniques become popular and cloud services continue to grow rapidly. To satisfy this demand, more and more data centers are being built across the world to supply the growing workload. However, the service providers have to match the requirements of different applications to the placement of virtual machines with the limited bandwidth links between geographically separated data centers while minimizing their cost.

In this chapter we approach the problem of allocating devices in the nodes of the network. These devices need to communicate with each other, consuming resources of the network. These resources are limited, and there is a cost for this communication that depends on the amount of flow between each pair of devices and the link used for this communication. Thus, the objective is to choose a placement for each device that minimize the communication cost. Additional requirements ensure that the minimum service quality is achieved. In analogy with an application on cloud computing over the telecommunication network, devices can be understood as virtual machines, while the nodes of the network are data centers. The objective is to place a set of virtual machines in a set of data centers that minimize the communication cost between the virtual machines, which, in general, means reduce the traffic over the entire network structure. To ensure that the bandwidth capacity, the latency requirements, and the data center capacity are respected are additional quality service requirements for this problem. To solve the virtual machine placement problem (*VMPlacement*), we use a Biased Random-Key Genetic Algorithm (BRKGA), as well as, a Greedy Randomized Adaptive Search Procedure (GRASP), both combined with a path-relinking strategy and an intensive local search procedure. An extensive set of experiments is provided to show the efficiency of our approach. This chapter is based on an extension of Stefanello et al. (2015a) and Stefanello et al. (2015b).

4.1 Introduction

Virtualization of physical servers have gained prominence in enterprise data centers. This is because virtualization offers virtually unlimited resources without any upfront capital investment and a simple pay-as-you-go charging model. Long term viability of virtualization depends, among other factors, on cost and performance. In order to attain performance guarantees, application providers can offer requirements for a number of virtual machines, bandwidth/latency requirements between virtual machines, and latency requirements between users of the service and virtual machines. Having all these performance guarantees for the application can help give an optimized service to the users. However, the service provider has to match the requirements of different applications to the placement of virtual machines with the limited bandwidth links between geographically separated data centers while minimizing its cost.

Unfortunately, today's public cloud platforms such as Amazon EC2[1] do not provide any performance guarantee, which in turn affects tenant cost. Specifically, the resource reservation model in today's clouds only provisions CPU and memory resources but ignores networking completely. Because of the largely oversubscribed nature of today's data center networks (e.g., Greenberg et al. (2009)), network bandwidth is a scarce resource shared across many tenants. In order to meet the reliability and the demand requirements, the data centers have to be placed all across the world. For instance, a teleconference call connects people from all over the world, and a data center within a reasonable distance to the end users is needed. For distributed data centers, networking cost is the major cost, which has not been accounted in the prior works on virtual machine placement to the best of our knowledge. With the limited bandwidth links between the data centers, networking intensive phases of applications collide and compete for the scarce network resources, which leads to their running times become unpredictable. The uncertainty in execution time further translates into unpredictable cost as tenants need to pay for the reserved virtual machines (VMs) for the entire duration of their jobs.

Placement of virtual machines within a data center have been widely explored Guo et al. (2010), Ballani et al. (2011), Xie and Hu (2012). These papers account for the networking needs in addition to the CPU and memory needs within a data center. For example, Guo et al. (2010) proposes bandwidth reservation between every pair of VMs. Ballani et al. (2011) proposes a simpler virtual cluster (VC) model where all virtual

[1]<http://aws.amazon.com/ec2/>

machines are connected to a virtual switch with links of bandwidth B. Xie and Hu (2012) extends these approaches to consider time-varying network requirement. However, all these works account for a single data center where the bandwidths are much larger as compared to the bandwidths across data centers. Instead, this work deals with the placement of virtual machines across geo-separated data centers.

In this work, we consider multiple data centers that are connected with limited bandwidth links. The latency between every pair of data centers is known. In order to meet the application's quality of service guarantees, there is a required minimum bandwidth and maximum latency between each pair of virtual machines. We assume that there are multiple users who would use these services, and users are connected to some data center. In order to meet the overall application performance, there is an additional requirement of maximum latency between users and the virtual machines. Intuitively, if there is a set of VMs needed by a user and the set does not have any requirement with any other user or VM, it can be placed in a single data center. However, a VM interacts with multiple VMs which may be needed by other users, thus increasing the set of options for placement. There is a cost of transferring data between data centers and the placement minimizes this cost thus preferring placement of all VMs in a single data center which may not be feasible due to the quality of service requirements.

This problem has similarity with the problem of *Virtual Network Embedding* (VNE), since the set of virtual machines and the relation between then can be considered as the *Virtual Network* (VN), while the set of data centers and the links can be mapped as the *Substrate Network* (SN). Fischer et al. (2013) describe a general framework for VNE. The VMPlacement problem can be described using this framework, but the problem has some differences from other works since we consider a complete graph connecting the data center, a quadratic cost function for the communication cost, and latency constraints that usually are ignored from other works.

This problem is a generalization of the NP-hard Generalized Quadratic Placement Problem (GQAP) given in Lee and Ma (2004). Solving this problem optimally is possible only in very small instances, which may not represent the size found in real-world applications. Thus, we propose heuristic methods based on local search for solving the Virtual Machine Placement Problem. We extend the BRKGA proposed in Stefanello et al. (2015a) and proposed a new GRASP algorithm, both coupled with a path-relinking and an extensive local search procedure. We test the performance of both algorithms in a dataset comprised of instances of sizes ranging from small to large. Both algorithms have similar performance, although a slight advantage for BRKGA was observed in small instances while GRASP has a slight advantage in larger instances. We show that the algorithms are able to quickly find feasible solutions and find high-quality final solutions, especially when the path-relinking procedure is used.

The rest of the chapter is organized as follows. In Section 4.2, we present mathematical models for the Virtual Machine Placement Problem in multiple data centers. Two metaheuristics are proposed in Section 4.3. Computational results are presented in Section 4.4. Finally, conclusions are drawn in Section 4.5.

4.2 Virtual Machine Placement Problem

In the Virtual Machine Placement Problem (VMPlacement), the objective is to place a set K of virtual machines (VM) in a set N of data centers (DC) in order to minimize the communication cost among virtual machines over the network.

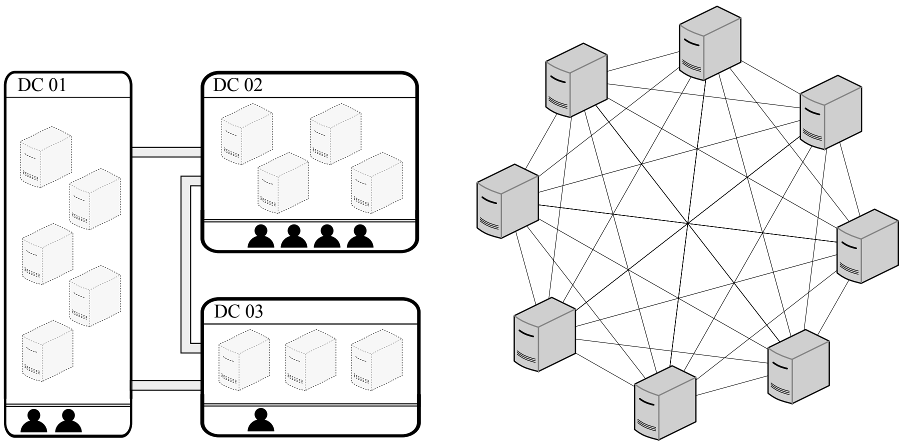
In this problem, each data center has a capacity a_i, which represents the number of virtual machines that can be placed in DC i. Also, between two data centers i and j, there are a bandwidth capacity (B_{ij}), a latency (L_{ij}), and a cost C_{ij} to transfer a data unit between the pair of data centers.

In order to meet the reliability and demand requirements of the applications, certain bandwidth and latency requirements can be imposed on the different VMs that are placed on the data centers. Each pair of virtual machines v and w has a required bandwidth (b_{vw}) whose sum overall VMs placed between DCs i and j cannot exceed B_{ij}. Furthermore, there is a required latency (l_{vw}), such that VMs v and w cannot be placed in data centers i and j if the required latency is greater than the respective data center latency.

Finally, there is a set U of users who access the system. Each user u is located at a data center $d(u)$ and has a required latency t_{uv} for each VM v.

Figure 4.1 shows a representation of the input data components: the data centers (Figure 4.2a), and the virtual machines (Figure 4.2b). The first component is composed by three data centers (rounded rectangles). Each data center has a number of users and a capacity (represented as a number of spots where VMs can be placed). The connection between each pair of DCs represents the bandwidth capacity, latency, and cost. The second component is composed by eight virtual machines, where each link represents the required bandwidth and the required latency.

Figure 4.1 – Input data representation



(a) Data centers. (b) Virtual machines.

Source: from the author (2015).

In the next subsections, we present three mathematical formulations for the VM-Placement problem. We first present a quadratic formulation, followed by two mixed integer linear formulations. Since VMPlacement is a generalization of the NP-hard Generalized Quadratic Assignment Problem, we extend the linear mathematical models proposed for the GQAP in Lee and Ma (2004) to VMPlacement. The models in Lee and Ma (2004) were also extended from the mixed integer linear programming formulation from Kaufman and Broeckx (1978) and Frieze and Yadegar (1983) to the Quadratic Assignment Problem, all of them based on the formulation for the QAP described in Koopmans and Beckmann (1957).

4.2.1 Quadratic mathematical model

A natural formulation for VMPlacement is based in a quadratic formulation as a generalization of GQAP. In what follows we summarize the parameters and present a quadratic mathematical model for VMPlacement (QMVMP) introduced in Stefanello et al. (2015a).

Parameters:

N :	set of data centers;
K :	set of virtual machines;
U :	set of users;
a_i :	capacity in number of VMs that DC i can host;
B_{ij} :	bandwidth between DCs i and j;
L_{ij} :	latency between DCs i and j;
C_{ij} :	cost of transferring a data unit between DCs i and j;
b_{vw} :	required bandwidth between VMs v and w;
l_{vw} :	required latency between VMs v and w;
$d(u)$:	DC which hosts user u;
t_{vu} :	required latency between user u and VM v;
c_{iv} :	cost of placing a VM v in a DC i.

Equations (4.1)-(4.7) present the quadratic mathematical model for the VMPlacement (QMVMP), where the binary decision variable $x_{iv} = 1$ if VM v is located into DC i, and $x_{iv} = 0$ otherwise.

$$\min \sum_{i \in N} \sum_{v \in K} c_{iv} x_{iv} + \sum_{i \in N} \sum_{j \in N} \sum_{v \in K} \sum_{w \in K} x_{iv} x_{jw} C_{ij} b_{vw} \tag{4.1}$$

subject to:

$$\sum_{v \in K} x_{iv} \leq a_i \qquad \forall\, i \in N, \tag{4.2}$$

$$\sum_{i \in N} x_{iv} = 1 \qquad \forall\, v \in K, \tag{4.3}$$

$$\sum_{v \in K} \sum_{w \in K} x_{iv} x_{jw} b_{vw} \leq B_{ij} \qquad \forall\, i, j \in N, \tag{4.4}$$

$$\sum_{i \in N} \sum_{j \in N} x_{iv} x_{jw} L_{ij} \leq l_{vw} \qquad \forall\, v, w \in K, \tag{4.5}$$

$$\sum_{i \in N} x_{iv} L_{i,d(u)} \leq t_{vu} \qquad \forall\, u \in U,\ \forall\, v \in K, \tag{4.6}$$

$$x_{iv} \in \{0, 1\} \qquad \forall\, i \in N,\ \forall\, v \in K. \tag{4.7}$$

Objective function (4.1) minimizes the cost of placing each pair of virtual machines v and w to DCs i and j. Constraints (4.2) require that the number of VMs in each DC must not exceed the DC capacity. Constraints (4.3) require that each VM must be assigned to exactly one DC. Constraints (4.4) require that the given bandwidth between each pair i

and j of DCs should not be surpassed by the total sum of bandwidth required among the virtual machines placed in these DCs. Constraints (4.5) assure that the latency required between each pair of VMs should be respected, i.e, if VMs v and w are placed respectively to DCs i and j, then the latency between DCs i and j should not exceed the required latency between VMs v and w. Constraints (4.6) require that the latency between a VM v and the DC where the user u is located be respected, i.e, a VM v can be only placed on a DC i if the latency between i and $d(u)$ is less than or equal to a given latency between the VM v and the user u. Finally, constraints (4.7) define the variables domain.

The performance of mixed integer linear programming solvers has improved considerably over the last few years. CPLEX[2] is a general-purpose black-box solver based on simplex and a branch-and-bound algorithm with the state-of-the-art exact algorithms for integer programming and has been successfully applied in many combinatorial optimization problems. To analyze the CPLEX performance and provide baseline results for comparison of heuristic methods, the following subsections present two linear mathematical models for VMPlacement problem.

4.2.2 Linear mathematical model I - LMVMP

Based on model $L3$ from Lee and Ma (2004) for GQAP, and from Frieze and Yadegar (1983) for QAP, we present a mixed-integer linear model for the VMPlacement. Let $y_{ivjw} = x_{iv}x_{jw}$, $\forall\ i, j = \{1, \ldots, N\}$ and $v, w = \{1, \ldots, K\}$, the mixed-integer linear mathematical model for VMPlacement named as LMVMP can be formulated as the following:

[2]<www.cplex.com>

$$\min \sum_{i \in N} \sum_{v \in K} c_{iv} x_{iv} + \sum_{i \in N} \sum_{j \in N} \sum_{v \in K} \sum_{w \in K} y_{ivjw} C_{ij} b_{vw} \tag{4.8}$$

subject to:

$$\sum_{v \in K} x_{iv} \leq a_i \qquad \forall\ i \in N, \tag{4.9}$$

$$\sum_{i \in N} x_{iv} = 1 \qquad \forall\ v \in K, \tag{4.10}$$

$$\sum_{i \in N} y_{ivjw} = x_{jw} \qquad \forall\ v, w \in K,\ \forall\ j \in N, \tag{4.11}$$

$$y_{ivjw} = y_{jwiv} \qquad \forall\ v, w \in K,\ \forall\ i, j \in N, \tag{4.12}$$

$$\sum_{v \in K} \sum_{w \in K} y_{ivjw} b_{vw} \leq B_{ij} \qquad \forall\ i, j \in N, \tag{4.13}$$

$$\sum_{i \in N} \sum_{j \in N} y_{ivjw} L_{ij} \leq l_{vw} \qquad \forall\ v, w \in K, \tag{4.14}$$

$$\sum_{i \in N} x_{iv} L_{i,d(u)} \leq t_{vu} \qquad \forall\ u \in U,\ \forall\ v \in K, \tag{4.15}$$

$$x_{iv} \in \{0, 1\} \qquad \forall\ i \in N,\ \forall\ v \in K, \tag{4.16}$$

$$0 \leq y_{ivjw} \leq 1 \qquad \forall\ i, j \in N,\ \forall\ v, w \in K. \tag{4.17}$$

The LMVMP is obtained by replacing the product $x_{iv} x_{jw}$ by y_{ivjw} from QMVMP. In addition four extra sets of constraints are considered. Constraints (4.11) and (4.12) define the relation between variables x and y. Constraints (4.12) impose the symmetry relation to variables y. Finally, constraints (4.17) define the domain of variables y.

We note that the model QMVMP has quadratic constraints, while LMVMP not. The objective function also changes from a quadratic function in QMVMP to linear in LMVMP. However, the mixed-integer linear problem LMVMP has a considerable higher number of variables, having variables y_{ivjw} in addition to the previous variables x_{iv}. Thus, the number of variables changes from $O(NK)$ in QMVMP to $O(N^2K^2)$ in LMVMP. We note that if the optimal solution of LMVMP is (x^*_{iv}, y^*_{ivjw}), then (x^*_{iv}) is the optimal solution for QMVMP. The proof that both models are equivalent can be obtained by extending the proof for QAP provided in Lee and Ma (2004).

4.2.3 Linear mathematical model II - LMVMP-II

Next we present a second linearization for VMPlacement problem (LMVMPII). This linear model is derived from Kaufman and Broeckx (1978) for QAP, which is probably the linearization for QAP with a lower number of variables and constraints. In Lee and Ma (2004) the authors extend the formulation for GQAP.

In this model, each binary decision variable x_{iv} is set to one when VM v is located into DC i, and zero otherwise. The auxiliary variables y_{iv} aggregate the cost for each placed VM v in a DC i, and n_{ijb} aggregate the bandwidth from VM v between data centers i and j.

$$\min \sum_{i \in N} \sum_{v \in K} c_{iv} x_{iv} + \sum_{i \in N} \sum_{v \in K} y_{iv} \tag{4.18}$$

subject to:

$$\sum_{v \in K} x_{iv} \le a_i \qquad \forall\, i \in N, \tag{4.19}$$

$$\sum_{i \in N} x_{iv} = 1 \qquad \forall\, v \in K, \tag{4.20}$$

$$\sum_{j \in N} \sum_{w \in K} C_{ij} b_{vw} x_{jw} - y_{iv} \le m_{iv}(1 - x_{iv}) \qquad \forall\, v \in K,\ \forall\, i \in N, \tag{4.21}$$

$$\sum_{w \in K} b_{vw} x_{jw} - n_{ijv} \le M(1 - x_{iv}) \qquad \forall\, v \in K,\ \forall\, i, j \in N, \tag{4.22}$$

$$\sum_{v \in K} n_{ijv} \le B_{ij} \qquad \forall\, i, j \in N, \tag{4.23}$$

$$L_{ij} x_{iv} \le l_{vw} + L_{ij}(2 - x_{iv} - x_{jw}) \qquad \forall\, v, w \in K, \forall\, i, j \in N, \tag{4.24}$$

$$\sum_{i \in N} x_{iv} L_{i,d(u)} \le t_{vu} \qquad \forall\, u \in U,\ \forall\, v \in K, \tag{4.25}$$

$$x_{iv} \in \{0, 1\} \qquad \forall\, i \in N,\ \forall\, v \in K, \tag{4.26}$$

$$y_{iv} \ge 0 \qquad \forall\, i \in N,\ \forall\, v \in K, \tag{4.27}$$

$$n_{ijv} \ge 0 \qquad \forall\, v, w \in K,\ \forall\, i, j \in N. \tag{4.28}$$

where

$$m_{iv} \ge \sum_{j \in N} \sum_{w \in K} C_{ij} b_{vw}, \quad \forall\, i \in N,\ \forall\, v \in K.$$

Constraints (4.21) impose the cost between the data centers i and j to the variables y_{ij}. Constraints (4.22) and (4.23) imposed the bandwidth constraints while constraints (4.24) imposed the latency constraints. The constraints (4.24) can be replaced by

$$x_{iv} + x_{jw} \leq 1 \quad \forall\, i, j \in N,\ \forall\, v, w \in K \text{ if } L_{ij} > l_{vw}. \tag{4.29}$$

We observe that CPLEX converts (4.24) into (4.29) in the pre-processing phase. Finally, constraints (4.26), (4.27), and (4.28) define the domain of variables.

This model is not used in practice since it uses big-M constraints (4.21), and the root-node bound is always zero. However, a stronger formulation can be obtained by adding the following cuts

$$y_{iv} \geq Y_{iv} x_{iv} \quad \forall\, i \in N,\ \forall\, v \in K, \tag{4.30}$$

where Y_{iv} is defined as the optimal value of the following generalized assignment model:

$$Y_{iv} = \min \sum_{j \in N} \sum_{w \in K} C_{ij} b_{vw} x_{jw} \tag{4.31}$$

subject to:

$$\sum_{w \in K} x_{jw} \leq a_j \qquad \forall\, j \in N, \tag{4.32}$$

$$\sum_{j \in N} x_{jw} = 1 \qquad \forall\, w \in K, \tag{4.33}$$

$$\sum_{j \in N} \sum_{w \in K} b_{vw} x_{jw} \leq B_{ij} \qquad \forall\, w \in K, \tag{4.34}$$

$$(4.24),\ (4.25) \tag{4.35}$$

$$x_{iv} = 1 \tag{4.36}$$

$$x_{jw} \in \{0, 1\} \qquad \forall\, j \in N,\ \forall\, w \in K. \tag{4.37}$$

These additional cuts were applied for the 3-dimensional assignment problem in Mittelmann and Salvagnin (2015). Note that these cuts can also be applied for GQAP suppressing the constraints (4.22)–(4.25) and (4.28) from LMVMPII, and (4.34)–(4.35) from the assignment model, which are the specific constraints for VMPlacement.

4.3 Heuristic Approaches

In this section we propose two metaheuristics approaches to solve the VMPlacement problem. Initially, the local search procedures are described in detail. Next, two path-relinking strategies are presented. Finally, in the following subsection is described the Greedy Randomized Adaptive Search Procedure (GRASP), followed by the Biased Random-Key Genetic Algorithm (BRKGA).

Place a virtual machine into a data center can violate some of the constraints. To deal with this situation, we use a penalization strategy to minimize the number of violated constraints. Thus, the cost of placing a VM v in DC i is calculated by the regular placement cost added by a sufficiently large number M for each violated constraint. This penalization strategy is applied whenever a solution is evaluated. In our experiments we use $M = 10^{10}$. Also, the notation for a solution S represents a vector of size $|K|$ with $S = \{n_1, \ldots, n_{|K|}\}$, with n_v representing the label (or index) of the data center where the virtual machine v is placed.

4.3.1 Local search procedures

Local search is a general approach for finding and improving solutions to hard combinatorial optimization problems. The most basic strategy of local search algorithms is to start from an initial solution and iteratively replace the current solution by a better neighbor solution, until no improvement can be reached. A neighbor solution can be obtained by applying moves defined by a neighborhood structure. In this work we define four neighborhood structures to obtain neighbor solutions, namely *shift*, *swap*, *chain2L*, and *chain3L*.

Shift. A shift operation moves a virtual machine from the current data center to a different data center. In a *shift search*, we select a virtual machine v in a circular order of their indexes (starting from index zero), and calculate the cost to move it from the current data center to each other data center. The virtual machine v is moved to the data center that produces the greatest improvement in the objective function or maintained in its original data center. In the next step, a new virtual machine is selected, and the evaluation is repeated. The procedure stops when no shift move can improve the solution. Note that the search is performed as a partial best improvement since the best movement is chosen for each virtual machine, instead choose the best move among all virtual machines.

Swap. A swap operation interchanges the positions of two virtual machines. In a *swap search*, we evaluate the cost of all swap moves between two virtual machines v and w in a circular order of their indexes (starting from index zero). When an improvement is reached, the virtual machine positions are interchanged, and the search continues by selecting a next pair of virtual machine. The procedure ends when no swap move can improve the solution. The evaluation of symmetrical movements is avoided.

Chain2L. A chain operation is a composition of two shift moves applied sequentially. In a *chain2L search*, the first shift move selects a virtual machine v from the data center i to move to a data center j. In the second shift move, a virtual machine w from j is moved to a data center k. We evaluate all compositions of two shift moves. When an improvement is reached, the moves are applied to the solution and the process is restarted with the new solution. Note that this neighborhood includes the *swap search* when k is equal to i.

Chain3L. The last neighborhood proposed extends the concept of the chain moves to a composition of three shift moves applied sequentially. A *chain3L search* involves four virtual machines and up to four data centers. This search also comprises three shift moves between two data centers, or a circle move among three data centers.

Given a solution S, the worst case time complexity to evaluate the cost to insert or remove a virtual machine v in S is $O(|K|)$. This complexity is reached because is necessary to evaluate whether the latency requirement is satisfied between v and $w \in S$. Also, is necessary evaluate the bandwidth cost between v and $w \in S$. Capacity constraints, uniqueness in the placement, and user latency requirements can be evaluated in constant time. Therefore, for the *shift search* a number of $|K|$ removals and $|K| * |N|$ insertions are evaluated, resulting in a complexity of $O(|K|^2|N|)$ (for the case where there is no improvement during the search). For the *swap search* the time complexity is $O(|K|^3)$, while for *chain2L search* is $O(|K|^3|N|)$. Finally, the *chain3L search* has a complexity of $O(|K|^4|N|)$.

Naturally, in all searches described above, the processing time can be reduced avoiding to evaluate moves that lead to a worse solution. Also, heuristic strategies do not need to explore the whole neighborhood, and only evaluate a subset of the neighbor solutions. This is done in the *chain3L search* prohibiting the insertion of a virtual machine in some data centers, and thus, reducing the search space.

Two heuristic strategies to reduce the amount of explored neighbor solutions in the *chain3L search* are proposed. The first reduce heuristic strategy is based in the cost Y_{iv} described in the Model (4.31)-(4.37) from Subsection 4.2.3. Given a parameter $\alpha_{3L} \in [0, 1]$, we allow to insert a virtual machine v in a data center i only if i belongs to the $\lfloor N * \alpha_{3L} \rfloor$ data centers with the lowest cost Y_{iv}. A low value of α_{3L} indicates a restricted search only in data centers with low values of Y_{iv}. A high value of α_{3L} indicates a more broad search. An analyzis for the instances described in Section 4.4.1 with their respective best known solutions corroborate for this heuristic strategy. In these solutions, more than 50% of the virtual machines are placed in the 30% data centers with lowest cost Y_{iv}, and approximately 75% are placed in the 50% data centers with the lowest cost.

The second reduce heuristic strategy is based on the communication cost assigned to the data centers. Given a solution $\mathcal{S}$, let C_i be the sum of the communication cost from i to all other data centers. We only evaluate compositions of movements that start from a virtual machine that belongs to the β_{3L} data center with the highest cost C_i. In summary, the idea is to limit the search to movements that reduce the communication cost of the most required data centers. Combining both reduction heuristics, the search space is considerably reduced, but the heuristics are still able to visit solutions in a promising search space. By default, we use $\alpha_{3L} = 0.3$ and $\beta_{3L} = 3$.

To take advantage of all these neighborhoods, two strategies are used. In the first (called `R`), every neighborhood is applied sequentially and only once, starting from the lowest to highest complexity neighborhood (*shift*, *swap*, *chain2L*, and *chain3L*). The second approach (called `V`), is based on the idea of Variable Neighborhood Descent (HANSEN et al., 2010), where the neighborhood searches are applied sequentially, but the search is restarted with the first neighborhood if an improvement is reached.

We name each local search strategy using two characters. The first character is a number that indicates the highest complex neighborhood applied. The second character indicates how these searches are integrated. For example, the name `3R` indicates that we consider local search procedures *shift search*, *swap search*, and *chain2L search* applied sequentially on mode `R`.

Note that using a penalization strategy described in the previous subsection, the local search is also applied to infeasible solutions. In this case, the local search also works as a repair procedure.

4.3.2 Path-relinking

Path-relinking (PR) is an approach to integrate intensification and diversification in the search. It consists in exploring trajectories that connect high-quality solutions. Starting from an initial solution, the scheme moves from one solution to another until the target solution is reached. The objective consists in finding a solution that is better than both the initial and target solutions.

The path-relinking was first suggested for tabu search in Glover (1989) and then formalized in Glover and Laguna (1993). Since then, this strategy was applied for a large number of combinatorial optimization problems and related studies as Glover (1997), Glover et al. (2000), Resende and Ribeiro (2005), Resende et al. (2010), Festa and Resende (2013), and Glover (2014), just to name a few.

The Algorithm 2 shows the pseudo-code for the path-relinking operator between solutions S and T, where S is the initial solution and T the guiding solution. In line 2 the incumbent solution is initialized. The loop between the lines 3 and 10 is repeated while the distance between both solutions is greater than an input parameter δ_{lim}. The distance $\Delta\{S,T\}$ is the minimum number of moves needed to transform S into T or vice-versa. We use $\delta_{lim} = \Delta\{S,T\}/2$ defined at the beginning of the procedure, since we observed that the probability to find an improved solution is greater in the first steps of the path. In line 4 a movement is applied to the solution S in direction to the solution T. Basically, we analyze all changes in S to T that reduce the distance $\Delta\{S,T\}$ by one, and apply in S the change with the least cost.

Several studies have experimentally found that it is convenient to add a local search exploration from some of the generated solutions in the path (MARTÍ et al., 2006). Since two consecutive solutions obtained by a relinking step are often very similar, it is generally not efficient to apply the local search at every step of the relinking process. Thus, in line 7 the local search is applied at each n iterations, where $n = (|K| * 0.5)/(4 + |K|/50)$. The parameter n is rounded up to the first even number. This ensures that the local search is applied to S and T each time, since, in line 10, the solutions are interchanged to use *back-and-forward* strategy (FESTA; RESENDE, 2013). Finally, in line 8 the incumbent solution is updated and returned in line 11.

Algorithm 2: Pseudo-code for a greedy path-relinking operator

```
1  Procedure PathRelinkingOperator($S$,$T$)
2      $S^* \leftarrow S$;
3      while $\Delta\{S,T\} \geq \delta_{lim}$ do
4          $S \leftarrow$ MakeMovement($S, T$);
5          $S' \leftarrow S$;
6          if $LSCondition$ then
7              $S' \leftarrow$LocalSearch($S'$);
8          if $f(S') < f(S^*)$ then
9              $S^* \leftarrow S'$;
10         Swap($S, T$);
11     return $S^*$;
```

Algorithm 3 and 4 describe two frameworks of how to administrate the elite set $\mathcal{E}$, and how solutions are selected for the path-relinking operator. In the first approach, the path-relinking operator is applied between a provided solution S and a selected solution T from the elite set. In the second approach, the path-relinking operator is applied multiple times, between a provided solution S and every solution from the elite set.

To simplify the notation, we name by path-relinking the general process to administrate the elite and make the path-relinking operator between two solutions. Regarding to the specific administration procedures, we name the fist approach by `PRS` and the second approach by `PRM`. The first two letters indicate that the algorithm is related to the path-relinking and the last letter indicates the framework for administrate the path-relinking operator. In the first, `S` indicates the application of a single path-relinking operator and in the second, `M` indicates the application of a multiple path-relinking operator.

In the beginning of the Algorithm 3, the elite set $\mathcal{E}$ is empty. The maximum size δ_{max} of the elite set is an input parameter. A solution S is added to $\mathcal{E}$ if it improves the best solution or is sufficiently different from each solution in $\mathcal{E}$ (lines 3, 6, and 11). We define that two solutions are sufficiently different if $\Delta\{S,T\} > \delta_{dis}$. We denote by $S \not\approx \mathcal{E}$ if $\Delta\{S,T'\} > \delta_{dis}, \forall\ T' \in \mathcal{E}$. In our case, we use $\delta_{dis} = \lfloor |K| * 0.1 \rfloor$. If the elite set has a minimum number of solutions ($\delta_{min} = 2$ in line 2), and S is sufficiently different, a solution T is selected for the path-relinking operator between S and T.

In line 4, solutions are ordered by a linear rank $r(T_i)$, $\forall\ i = 1 \ldots |\mathcal{E}|$, where the best solution has rank $r(T_1) = |\mathcal{E}|$ and the worst solution has rank $r(T_{|\mathcal{E}|}) = 1$. A solution $T \in \mathcal{E}$ is selected using the roulette wheel criterion, i.e, T is selected from the elite set with probability of $r(T)/\sum_{i \in r(T_i)}$. With this criterion, better solutions have more chance to be chosen. Since T is selected, the path-relinking operator is applied between S and T.

Lines 6 to 9 update the elite set after the path-relinking operator is applied. Line 6 evaluates whether S is sufficiently different or improves the best solution. If this is the case, S is added to $\mathcal{E}$. Line 8 is responsible for maintaining the cardinality of the elite set. T' is the most similar solution to S with worst solution than S, i.e, $T' \leftarrow \{T'' \in \mathcal{E} \mid f(T'') > f(S) \text{ and } \Delta\{T'', S\} \leq \Delta\{T'', S'\},\ \forall\ S' \in \mathcal{E}\}$. Finally, in line 13 the solution S is returned.

Algorithm 3: Pseudo-code for the PR administration framework `PRS`

```
1  Procedure PathRelinking($S$)
2      if $|\mathcal{E}| > \delta_{min}$ then
3          if $S \not\approx \mathcal{E}$ then
4              Select $T \in \mathcal{E}$ ;
5              $S \leftarrow$PathRelinkingOperator($S, T$);
6              if $S \not\approx \mathcal{E}$ then
7                  $\mathcal{E} \leftarrow \mathcal{E} \cup \{S\}$;
8              if $|\mathcal{E}| > \delta_{max}$ then
9                  Remove $T' \in \mathcal{E}$;
10     else
11         if $S \not\approx \mathcal{E}$ then
12             $\mathcal{E} \leftarrow \mathcal{E} \cup \{S\}$;
13     return $S$;
```

Another approach is outlined in Algorithm 4. The main difference from the previous approach is that when a solution S is added to $\mathcal{E}$, the path-relinking operator is applied between all pair of solutions from $\mathcal{E}$ to which the operator was not previously applied. This is done in the loop in lines 5 and 13. The flag $hasPairsST$ can be easily updated using memory structures, and it indicates whether there are S' and T' in the elite set that the path-relinking operator was not applied. In line 6 two solutions are selected. Pairs with better objective functions are selected first. The insertion operation (line 11) and the removal operation (line 13) follow the same rules than in the previous algorithm. In line 14 a solution is removed from the elite set in case it is full. We also use roulette wheel criterion by rank, where the best solution has rank equal to zero, and the worst solution has rank $|\mathcal{E}| - 1$. This ensures that the best solution are kept in the pool, and the worst solution has a high probability to be removed. Finally, in line 18 the solution S is returned.

Algorithm 4: Pseudo-code for the PR administration framework `PRM`

```
1  Procedure PathRelinking($S$)
2      if $|\mathcal{E}| > \delta_{min}$ then
3          if $S \not\approx \mathcal{E}$ then
4              $\mathcal{E} \leftarrow \mathcal{E} \cup \{S\}$;
5              while $hasPairsST$ do
6                  Select $S'$ and $T'$;
7                  $S'$ ←PathRelinkingOperator($S'$,$T'$);
8                  if $f(S') < f(S)$ then
9                      $S \leftarrow S'$;
10                 if $S' \not\approx \mathcal{E}$ then
11                     $\mathcal{E} \leftarrow \mathcal{E} \cup \{S'\}$;
12                 if $|\mathcal{E}| > \delta_{max}$ then
13                     Remove $T' \in \mathcal{E}$;
14             Remove $T' \in \mathcal{E}$;
15     else
16         if $S \not\approx \mathcal{E}$ then
17             $\mathcal{E} \leftarrow \mathcal{E} \cup \{S\}$;
18     return $S$;
```

Note that Algorithm 3 and 4 are designed in independent components, requiring only few configuration parameters (as local search strategy and elite size), and a procedure that provides different solutions. For this reason, these path-relinking strategy can be easily included in many different heuristic and metaheuristic frameworks. In the next subsections, we describe the GRASP and BRKGA metaheuristics which uses the path-relinking strategies.

4.3.3 Greedy Randomized Adaptive Search Procedure - GRASP

GRASP (Greedy Randomized Adaptive Search Procedure) is a multi-start metaheuristic for combinatorial optimization problems (FEO; RESENDE, 1989; FEO; RESENDE, 1995; FESTA; RESENDE, 2009a; FESTA; RESENDE, 2009b; RESENDE; RIBEIRO, 2010; MARTÍ et al., 2013; RESENDE; RIBEIRO, 2014). The algorithm is basically composed of two phases: construction and local search. The construction phase builds a feasible solution following a greedy randomized criterion, whose neighborhood is investigated until a local minimum is found during the local search phase. These phases are repeated over the iterations and the best local optimum found is returned as the heuristic solution. Its basic implementation is memoryless, because it does not make use of information collected in previous iterations. One way to add memory to GRASP is its hybridisation with path-relinking. A large review of this technique is presented in Festa and Resende (2013).

A high-level description of GRASP is presented in Algorithm 5. GRASP iterations are carried out in lines 3 to 8. In line 4, the procedure attempts to build a greedy randomized solution. A solution S is used as the starting solution for the local search in line 5. In line 6 the path-relinking is applied if a hybrid approach is used. If the local optimum S is better than the incumbent solution, then, in line 8 the incumbent solution is updated. In line 9, the best solution found overall GRASP iterations is returned as the GRASP solution.

Algorithm 5: Pseudo-code for GRASP

```
1 Procedure GRASP()
2     $S^* \leftarrow \emptyset$;
3     while stopping criterion is not satisfied do
4         $S \leftarrow$ GreedyRandomized();
5         $S \leftarrow$ LocalSearch($S$);
6         $S \leftarrow$ PathRelinking($S$);
7         if $f(S) < f(S^*)$ then
8             $S^* \leftarrow S$;
9     return $S^*$;
```

The Algorithm 6 shows a general pseudo-code for the greedy randomized constructive heuristic. The construction builds a solution S, one element at a time. In line 2, solution S is initialized empty. In line 3, the set of candidate list is initialized, as well as, the cost of placing each candidate element in a data center is calculated in line 4. The solution is built in the loop in lines 5 to 10. In line 6, a restricted candidate list (RCL) is build, and in line 7 a candidate is selected. In line 8 this element is added to the partial solution. In line 9, the candidate is removed from the set of candidates $\mathcal{C}$. In line 10 the placement costs for each candidate is re-evaluated. Finally, in line 11 the solution S is returned.

Algorithm 6: Pseudo-code for a greedy randomized constructive solution

```
Procedure GreedyRandomized()
    $S \leftarrow \emptyset$;
    Initialize set of candidates $\mathcal{C}$;
    Evaluate the incremental cost of candidates;
    while $\mathcal{C} \neq \emptyset$ do
        Build the $RCL$;
        Select $c \in RCL$;
        Add $c$ to solution: $S \leftarrow S \cup \{c\}$;
        Update candidates: $\mathcal{C} \leftarrow \mathcal{C} \backslash \{c\}$;
        Re-evaluate the incremental costs;
    return $S$;
```

Based on this framework, four constructive heuristic are developed to generate initial solutions in line 4 of GRASP procedure.

Random. In the first constructive heuristic, on each placement step a non-placed virtual machine is randomly selected and placed in a random data center. Note that this constructive approach is not a greedy heuristic. We decide to evaluate this approach since it is one of the simplest constructive methods, and we use as a baseline for comparing more sophisticated constructive methods.

Greedy. In the second constructive heuristic, on each placement step a non-placed virtual machine is randomly selected and placed in the data center that produces the least increase in the objective function. In this case, the RCL is composed only by one candidate that is the best data center for a specific randomly selected virtual machine.

DC-Greedy. In the third constructive heuristic, on each placement step a non-placed virtual machine is randomly selected and placed in one of the n data centers from the RCL. The RCL contains the candidates that produce lowest incremental placement cost. By default, we use $n = \max\{3, |N| * 0.2\}$. The value of n can variate on each iteration since we consider some additional conditions. First, candidates with the same cost to the first n candidates are also added to RCL. Second, if at least one candidate can be added to the solution keeping it feasible, then candidates that generate infeasibilities are ignored.

VM-Greedy. In the last constructive heuristic, on each placement step, all possible placements of virtual machines to data centers are evaluated, and one of the n best candidates is randomly selected. In this constructive heuristic, we use $n = \max\{5, |K| * 0.1\}$ for the RCL size, but the effective value of n can variate since we also include both additional conditions described in the previous constructive heuristic.

In these constructive heuristic, any random selection is based on uniform distribution.

Note that these constructive heuristic can generate infeasible solutions, and the repair process is made by the local search procedure.

4.3.4 Biased random-key genetic algorithm - BRKGA

In this section, we describe the biased random-key genetic algorithm (BRKGA) for the VMPlacement problem. The general framework is described in Section 1.2 while two decoders strategies and a hybrid BRKGA with path-relinking are described in the next subsections.

4.3.4.1 Decoders

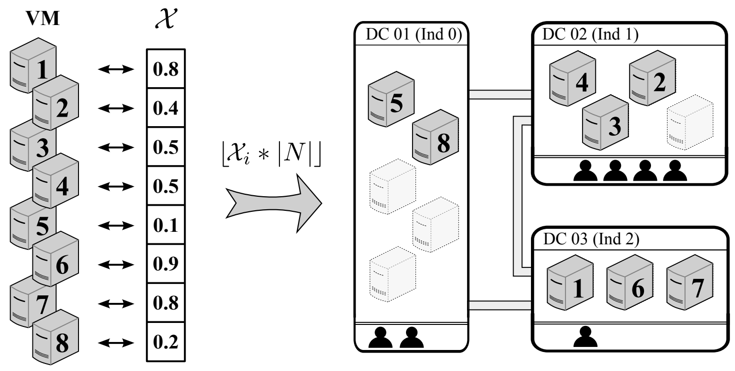
Solutions of the optimization problem are encoded as a vector $\mathcal{X}$ with $n = |K|$ random keys. To translate this vector into the solution of the VMPlacement problem, we propose two decoders, as described next.

Greedy Ordered Decoder - D1: In this decoder, the keys provide the order of placement. Following this order, a greedy strategy is used, placing each VM to the DC which produces the least increase in the objective function.

The decoder starts with a list whose each element is composed of the random key and the index of the virtual machine. The list is sorted by the keys. Now, the sorted list of index of virtual machines is used as an order in which virtual machines should be placed. Following this order, the next step is to place each virtual machine v to a DC. This is done by placing virtual machine v in DC i which produces the least increase in the objective function. Note that the cost to insert the VM v in each DC considers the previous virtual machines placed. When all VMs are placed, the decoder returns the fitness value for the respective vector $\mathcal{X}$ of random keys.

Location Decoder - D2: In this decoder, each key is decoded as the data center in which the virtual machine should be placed. Let k_i be the key corresponding to the VM of index i in $\mathcal{X}$, then this decoder simply places the VM of index i to DC $\lfloor k_i * |N| \rfloor$. Figure 4.2 shows an example of decoding for the decoder D2 for 3 data centers, 8 virtual machines and a vector $\mathcal{X}$ of random keys.

Figure 4.2 – Example of decoding for decoder D2



Source: from the author (2015).

Since both decoders are deterministic, we always obtain the same solution S' from a vector of random keys $\mathcal{X}$. In the case where a deterministic local search strategy is applied to S', a solution S'' can be obtained using the decoders and the local search, and thus, S'' can be associated to $\mathcal{X}$. However, depending on the decoder, its is possible to use a process called *recode* in $\mathcal{X}$, to obtain S'' only by using the decoder (without local search). In decoder D1, the recode can be computationally expensive and then we decided to maintain the local search as part of the decoder. In decoder D2, the recode can be applied by calculating the value of the key that will correspond to a data center in S''. Let i' the index of DC i, $lb = i'/|N|$ and $lu = (i+1)'/|N|$, a key that corresponds to DC can be given by $lb + (lu - lb)/2$ or any number in the interval $[lb, ub)$. Experiments including the recode process are presented in the Subsection 4.4.4. To simplify the notation, we denote by D3 the decoder D2 with recode process.

4.3.4.2 Hybrid BRKGA and path-relinking

Path-relinking is an approach to integrate intensification and diversification in the search and can be incorporated on many metaheuristic frameworks. Path-relinking is frequently used with GRASP algorithms (LAGUNA; MARTÍ, 1999; OLIVEIRA et al., 2004; MATEUS et al., 2010; FESTA; RESENDE, 2013), but references can be found for other metaheuristics as Scatter Search (GLOVER et al., 2003), Tabu Search (ARMENTANO et al., 2011), VNS (PÉREZ et al., 2007). Regarding GAs, we find several papers where the path-relinking technique is applied, like for example Basseur et al. (2005), Zhang and Lai (2006), and Vallada and Ruiz (2010), just to name a few.

In this work, we propose a new approach that hybridise the path-relinking with BRKGA algorithm. In our approach, we include the procedure 3 or 4 at n generations of BRKGA. By default we use $n = 1$, i.e., we apply the path-relinking each time a new generation is produced. Since both path-relinking procedures require a solution S, we randomly select S by uniform distribution among all elite solutions of the BRKGA population. In the case that the path-relinking returns a solution S' better than S, then the corresponding vector $\mathcal{X}$ of S is recoded to be adjusted to S'.

In Section 4.4, experiments with both path-relinking strategies in comparison with the standard approach are presented in detail. Since the path-relinking uses the recode process, experiments are performed only with decoder D3.

4.4 Computational results

The experiments were conducted on a computer with an AMD FX-8150 Eight-Core 3.6 GHz CPUs, with at least 32 GB of RAM running GNU/Linux, except for experiments with CPLEX which we use a computer with quad-core Intel Xeon E5530 2.4 GHz CPUs, with at least 48 GB of RAM running GNU/Linux. Algorithms are implemented in `C++`, with optimization flag `-O3`. For BRKGA we use the API described in Toso and Resende (2014). The commercial solver IBM ILOG CPLEX Optimizer version 12.6.0.0 (C++ API) was used to evaluate the mathematical models. All experiments used a single thread but multiple experiments were run in parallel.

Experiments were conducted with two main objectives. The first was to investigate the CPLEX performance considering the different mathematical models described in Section 4.2. The second was to evaluate the performance of the heuristic approaches described in Section 4.3 and some hybridisation variants, including metaheuristic approaches, local search strategies, and path-relinking procedures.

Initially we describe the method to generate the dataset used in the experiments. In the following we report results for CPLEX, for each metaheuristic approach, and then we report the best results of each method. Finally, we use our best approaches on a set of instances of GQAP, comparing with the state-of-the-art results for the problem described in Mateus et al. (2010).

4.4.1 Data set

In this subsection we present an instance generator that we proposed and implemented to generate the data set used in the computational experiments reported by this work. For generating each instance the generator receives as input four parameters: $|N|$, $|K|$, $|U|$, and P (the latter represents the percentage of the overall data center occupation).

To ensure the generator creates instances that admit feasible solutions, we generate the data for each instance based on n sets of pre-placed virtual machines to data centers (by default $n = 3$). Given a capacity of each data center, each set of pre-placed $s \in \mathcal{S}$ is generated by randomly placing each virtual machine to a data center, with probability proportional to the data center capacity, ensuring the capacity is not violated. Biased on these pre-placements, we generate the remaining data respecting the constraints of the problem, ensuring at least n feasible solutions for each instance.

Considering a random numbers generator using uniform distribution, and M' be a sufficiently large number, we generate the parameter data for each instance using the following steps.

Data center capacity: The total number of available virtual machines is given by $n' = \max\left(|N|, \left\lceil \frac{|K|}{|P|} \right\rceil\right)$. Thus, to define the values of a_i for each DC i, we start with all $a_i = 0$ and select n' times a random data center i, and increase a_i by one. We also ensure that each data center has a capacity a_i greater or equal to one. At this step, the n pre-placements described before are generated.

Required virtual machine bandwidth: For each pair of virtual machine $v \in K$ and $w \in K$ the bandwidth b_{vw} is a random number in the interval $[0 : 9]$. This matrix is symmetric, i.e, $b_{vw} = b_{wv}$, and $b_{vv} = 0$.

Data center bandwidth: Having defined the bandwidth between each pair of virtual machines in the previous step, we generate the values of data center bandwidth based in the pre-placements $\mathcal{S}$. Let b^s_{ij} be the sum of bandwidth between all virtual machines pre-placed to i and j in $s \in \mathcal{S}$. For each pair of data centers ij, we associate a bandwidth $B_{ij} = \max\{b^s_{ij}\}$, $\forall s \in S$. This matrix also is symmetric, i.e $B_{ij} = B_{ji}$, with $B_{ii} = M'$.

Data center latency: For each pair of data center ij, the latency L_{ij} is a random number selected the interval $[5 : 20]$. This matrix is symmetric, i.e, $L_{ij} = L_{ji}$, with $L_{ii} = 0$.

Required virtual machine latency: Let l^s_{vm} be the latency L_{ij} between the data centers ij where v is placed in i and w is placed in j in the pre-placement $s \in \mathcal{S}$.

We randomly select $n = |K| * 2$ distinct pairs vw to associate a required latency $l_{vw} = \max\{l^s_{vm}\}$, $\forall s \in \mathcal{S}$. The remaining latency l_{vw} is defined as M', indicating that no latency is required. We also ensure that $l_{vw} = l_{wv}$, and $l_{vv} = 0$.

Users in data centers: Users are allocated at random to data centers chosen with probability proportional to their capacity. More than one user can be located at the same data center.

Required user latency: For each user, we randomly select a virtual machine v to define a required latency. Let $i(s)$ be the data center where v is placed in $s \in \mathcal{S}$, thus the required latency between u and v is given by $t_{vu} = \max\{L_{d(u),i(s)}\}, \forall s \in \mathcal{S}$. The remaining user required latency t is set to M'.

Transferring data center cost: For each pair of data center ij, the cost C_{ij} is a random number in the interval $[10.00 : 100.00]$. This matrix is symmetric, i.e $C_{ij} = C_{ji}$, and $C_{ii} = 0$.

The parameters $|N|$ and $|K|$ can be used to define the instances sizes, while parameters $|U|$ and P can be used to adjust how the problem should be restricted. Finally, we generate a sets of instances by combining values from $N = \{5, 10, 25\}$, $K = \{15, 20, 25, 50, 100, 150, 200\}$, $U = \{K_i * 0.5, K_i, K_i * 1.5\}$, and $P = \{70, 90\}$. Tables 4.1 and 4.3 contains all instances we generated, and the values of $|N|$, $|K|$, $|U|$ and P are encoded in the name of instance in this respective order. All instances and their best known solutions are available at <www.inf.ufsm.br/~stefanello/instances/>.

Part of the objective function from Equation (4.1) is in fact not used for these instances since we are not considering the fix cost $c_{iv} \in \mathbb{R}$ of placement or installation a VM to a DC. Thus, without loss of generality, for these instances of the VMPlacement problem we may assume $c_{iv} = 0$, $\forall\ i \in N$ and $v \in K$. However, these values can be different for GQAP instances, or a more general case of VMPlacement instances that consider installation cost.

4.4.2 CPLEX results

In this section, we investigate the CPLEX performance for the both linear mathematical models described in Section 4.2. We seek to carry out an empirical study to analyze which size of instances the CPLEX can handle and prove optimality, and which model provides better integer solutions and lower bounds when CPLEX ends by a time limit. The objective is also to obtain baseline results for comparison of heuristic methods. We first analyze the results for the small size instances, followed by the set of the medium size instances and large size instances. We run CPLEX for QMVMP model. Since the performance of linear models was considered better than for the quadratic model, we report only results for the mixed-integer linear model.

4.4.2.1 Results for small size instances

In the first experiment we evaluated the performance of CPLEX with the mathematical models described in Section 4.2. We used the standard CPLEX solvers for models LMVMP and LMVMPII. The running time limit was set to three hours per run ($10,800$ seconds) and the number of threads was set to one. The remaining parameters were maintained on the default values.

In Fischetti and Monaci (2014) the authors exploited erraticism in search and how to take advantage of this behavior. Also, the authors show that CPLEX can have a wide contrast in its behavior due randomized initial conditions. Thus, to better evaluate the CPLEX performance we perform ten runs for each instance, each one with a different random seed defined by the CPLEX parameter `RandomSeed`.

Table 4.1 shows CPLEX results. The first column shows the name of the instances. The second column (BKS) shows the objective function of the best known solution value for each instance (optimal solutions are showed in boldface). The next two columns show the average running times of CPLEX to solve each instance for each mathematical model. Numbers in parenthesis denote the number of runs that the solver did not prove optimality within the time limit. The next two columns show the average running times that CPLEX spent to find the BKS value. The average is over ten runs or the number of times that CPLEX found BKS (indicated by the number in parenthesis). A signal '$-$' indicates that no run found a solution as good as BKS within the time limit. Finally, the last two columns show the best running times over all runs that CPLEX spent to find the BKS.

Table 4.1 – CPLEX detailed results for small instances

		AVG Time(s)		AVG BKS Time(s)		MIN BKS Time(s)	
Instance	**BKS**	**LMVMP**	**LMVMPII**	**LMVMP**	**LMVMPII**	**LMVMP**	**LMVMPII**
05_015_007_70	**25,844.02**	2.9	6.2	2.3	2.2	1.1	0.4
05_015_007_90	**23,557.30**	26.0	61.9	17.4	15.4	10.2	1.4
05_015_015_70	**10,904.78**	0.4	1.0	0.3	0.4	0.2	0.3
05_015_015_90	**24,354.96**	4.2	6.9	3.1	1.8	2.3	0.8
05_015_022_70	**14,163.60**	0.2	0.7	0.1	0.1	0.1	0.0
05_015_022_90	**32,318.02**	22.4	21.3	21.2	5.2	12.7	0.9
05_020_010_70	**38,572.62**	292.8	701.7	232.4	138.1	6.4	1.2
05_020_010_90	**64,710.80**	68.3	107.3	58.5	24.7	31.2	1.9
05_020_020_70	**55,288.76**	134.5	437.6	100.9	212.4	10.5	37.8
05_020_020_90	**57,574.90**	1.7	2.1	1.3	0.4	0.8	0.4
05_020_030_70	**28,433.34**	11.1	17.4	7.1	3.7	2.8	0.4
05_020_030_90	**66,088.70**	1.9	2.6	1.8	0.7	1.2	0.3
05_025_012_70	**43,300.76**	1271.5	4,025.5	1,063.8	862.6	195.0	2.2
05_025_012_90	100,865.02	10,800 (10)	10,800 (10)	9,902.6 (4)	6,165.6 (8)	9,746.0	28.7
05_025_025_70	**42,890.40**	58.4	126.7	52.5	39.0	26.9	1.3
05_025_025_90	103,791.96	10,800 (10)	10,800 (10)	9,764.3 (1)	–	9,764.3	–
05_025_037_70	**97,335.12**	161.4	592.8	117.0	161.2	19.6	2.8
05_025_037_90	**73,363.56**	10,262 (7)	10,800 (10)	8,993.2	3,782.8 (9)	5,916.3	33.8

Source: from the author (2015).

We can draw three main observations from this experiment. First, for both models, the solver tends to increase significantly the time spent to prove optimality for instances with 5 or more data centers, and 25 or more virtual machines. This indicates a baseline for the size of instances that this version of CPLEX can handle and prove optimality using these models in this computer. The second observation is that CPLEX performance for both models was relatively similar. CPLEX was able to prove or not the optimality in the same instances, except for instance `05_025_037_90` which CPLEX proved optimality in three runs for LMVMP, while for LMVMPII CPLEX has an average gap of 8%. However, considering the time that each model spent to find a solution with objective function equal to BKS, in most cases CPLEX with LMVMPII spent less time than considering the model LMVMP. Finally, the last observation is about the variance of time to find the BKS. With the randomized initial conditions, CPLEX presented a large variance in its behavior. For example, for the instance `05_025_012_70` and LMVMPII, in the best case CPLEX found BKS in 2.2 seconds, while in the worst case it took $3,458.01$ seconds to find the same solution. For this instance, the time to prove optimality also have large variance, alternating between a minimum of $3,685.67$ and a maximum of $4,828.05$.

We also evaluate the model LMVMPII without including the additional set of cuts (4.30). In comparison with the model with the cuts, we observed a higher decrease in the performance of CPLEX. For six instances CPLEX was not able to prove optimality in any of the ten runs. For these cases, the gap of CPLEX is still high. For example, for the instance `05_025_012_90`, the average gap was 53%, confirming that the relaxation quality of this model is low. Furthermore, the time to find BKS increased around 8 times, and the number of nodes explored within the time limit or to prove optimality increased around 7 times in comparison with the model that includes the set of cuts (4.30).

4.4.2.2 Results for median and large size instances

In this subsection we present results for CPLEX when applied to a set of median and large instances. In this experiment, we evaluate CPLEX for each model and each instance with one run for a time limit of one day ($86,400$ seconds).

Table 4.2 shows for each instance the BKS obtained over all experiments, including the reported in this work for heuristic methods and additional non-reported experiments used to evaluate the previous version of the algorithms. The column FO shows, for each model, the objective function value of the best integer solution found by CPLEX. Column CPLEX GAP shows the gap returned by CPLEX. The last column shows the gap from BKS to the best integer solution returned by CPLEX.

The first observation from this experiment is that for instances with 10 data centers, CPLEX was able to start solve the models, but still with a high gap even after 24 hours of computation. However, we observed that the gap returned by CPLEX as well as the gap to BKS are lower for LMVMPII in comparison with the values of LMVMP. The average gap returned by CPLEX with LMVMPII was around half of the average gap for LMVMP. The average gap to BKS was 2.19 for LMVMPII, while for LMVMP the gap was 4.96.

The second observation is that for instances with 25 data centers and LMVMP model, CPLEX spent the whole time in the presolve phase without solving the root relaxation node. Instead for LMVMPII, CPLEX was able to start a node exploration and found feasible solutions. For instances with 25 data centers, CPLEX explored an average of approximately 34800 nodes for instances with 100 VMs, 5700 nodes for instances with 150 VMs, and 1000 nodes for instances with 200 VMs. Even CPLEX maintaining a high gap, the gap from BKS was relatively small, considering the size of the instance and the difficulty to find feasible solutions (STEFANELLO et al., 2015a).

Table 4.2 – CPLEX detailed results for median and large instances

		FO		CPLEX GAP		BKS GAP	
Instance	**BKS**	**LMVMP**	**LMVMPII**	**LMVMP**	**LMVMPII**	**LMVMP**	**LMVMPII**
10_025_012_70	114,582.50	116,264.44	115,734.58	42.37	24.29	1.47	1.01
10_025_012_90	84,461.30	88,087.12	85,814.72	53.92	28.31	4.29	1.60
10_025_025_70	90,997.90	93,729.48	92,407.94	29.59	19.30	3.00	1.55
10_025_025_90	124,763.66	125,365.26	125,335.92	31.18	16.56	0.48	0.46
10_025_037_70	100,801.80	104,350.38	100,801.80	24.16	15.04	3.52	0.00
10_025_037_90	106,617.94	107,558.00	107,471.04	24.06	20.06	0.88	0.80
10_050_025_70	414,535.12	442,548.40	435,929.26	83.22	39.98	6.76	5.16
10_050_025_90	458,879.74	480,146.00	477,439.06	75.60	30.79	4.63	4.04
10_050_050_70	360,102.12	374,071.02	366,972.40	68.85	43.23	3.88	1.91
10_050_050_90	400,233.16	420,173.88	416,615.52	72.88	36.32	4.98	4.09
10_050_075_70	349,135.78	362,853.92	353,979.20	54.03	35.20	3.93	1.39
10_050_075_90	498,190.58	513,161.02	510,062.50	54.45	25.23	3.01	2.38
10_100_050_70	1,647,975.00	1,884,262.62	1,732,985.94	91.55	42.59	14.34	5.16
10_100_050_90	1,792,257.68	1,916,126.76	1,826,484.56	89.56	34.79	6.91	1.91
10_100_100_70	1,463,498.00	1,546,897.78	1,500,763.50	86.22	47.12	5.70	2.55
10_100_100_90	2,126,993.26	2,256,408.46	2,169,460.06	82.73	31.03	6.08	2.00
10_100_150_70	1,563,152.40	1,702,573.74	1,586,082.86	78.44	45.02	8.92	1.47
10_100_150_90	1,847,076.66	1,968,341.96	1,883,289.84	70.74	33.59	6.57	1.96
Average	-	-	-	61.86	31.58	4.96	2.19
25_100_050_70	1,887,688.24	-	1,976,649.48	-	44.15	-	4.71
25_100_050_90	2,116,849.00	-	2,184,190.20	-	34.89	-	3.18
25_100_100_70	1,953,155.20	-	2,056,428.04	-	44.55	-	5.29
25_100_100_90	2,021,228.76	-	2,091,723.56	-	35.72	-	3.49
25_100_150_70	1,967,364.52	-	2,061,181.36	-	43.91	-	4.77
25_100_150_90	2,160,014.54	-	2,235,936.02	-	35.35	-	3.51
25_150_075_70	4,603,163.50	-	4,768,514.02	-	42.99	-	3.59
25_150_075_90	4,618,491.80	-	4,786,179.50	-	36.54	-	3.63
25_150_150_70	3,882,650.94	-	4,085,463.10	-	46.34	-	5.22
25_150_150_90	4,706,129.66	-	4,899,667.66	-	36.82	-	4.11
25_150_225_70	4,340,090.18	-	4,530,866.46	-	46.00	-	4.40
25_150_225_90	4,523,393.44	-	4,708,994.42	-	35.51	-	4.10
25_200_100_70	6,937,008.98	-	7,539,073.36	-	50.79	-	8.68
25_200_100_90	9,034,147.98	-	9,391,760.06	-	34.93	-	3.96
25_200_200_70	7,146,330.32	-	7,579,453.48	-	49.13	-	6.06
25_200_200_90	8,578,620.94	-	8,909,349.84	-	36.53	-	3.86
25_200_300_70	7,638,447.16	-	8,033,640.92	-	42.32	-	5.17
25_200_300_90	8,195,152.30	-	8,579,652.20	-	38.94	-	4.69
Average	-	-	-	-	40.86	-	4.58

Source: from the author (2015).

4.4.3 GRASP results

This subsection presents results and analyzes feasibility for different constructive heuristic combined with different local search procedures embedded in the GRASP framework. Also, we report experiments with the path-relinking procedure.

The main goals of the following experiments are show how difficult is to find feasible solutions using a constructive heuristic, and explore the effect of embedding a local search procedure to obtain a feasible solution. Finally, the last main objective is to evaluate the impact of each component added to the framework, i.e., constructive heuristic, local search method, and path-relinking strategy.

In the first part of our experiment, we evaluate the performance of each constructive heuristic described in Subsection 4.3.3, embedded in the GRASP algorithm. We performed one run for each instance, combination of constructive heuristic, and local search method. The stopping criterion of each run was a time limit of $|K| * |N| * \theta$ seconds, where $\theta = 0.8$. We use this stopping criterion in all experiments in order to provide a fair comparison for all strategies evaluated.

Table 4.3 shows the average of the percentage of feasible solutions found on each run of the algorithm for all instances and combinations of local search (rows on first column) and constructive heuristic (columns 2 to 5).

Table 4.3 – Percentage of feasible solutions found by GRASP for different constructive heuristic and local search strategy

LSType	*Random*	*Greedy*	*DC-Greedy*	*VM-Greedy*	**Average**
`NoLS`	0.00	1.02	1.96	0.57	0.89
`1R`	23.70	22.12	27.88	21.96	23.91
`2R`	75.50	76.25	78.28	76.61	76.66
`2V`	77.26	78.02	79.72	78.41	78.35
`3R`	81.95	81.90	83.35	82.29	82.37
`3V`	82.93	82.66	84.06	83.20	83.21
`4R`	82.75	82.53	84.16	82.95	83.10
`4V`	84.08	83.61	84.98	84.19	84.22
Average	63.52	63.51	**65.55**	63.77	64.09

Source: from the author (2015).

The first observation from Table 4.3 is that the probability of finding a feasible solution using only a constructive heuristic without local search is low (row `NoLS`). In average, the percentage of feasible solutions was less than 1%, with an average of less than 2% for the constructive heuristic *DC-Greedy*. However, the main observation is that the constructive heuristic has a lower impact on the percentage of feasibility solution in comparison with the impact of any local search method. When a local search procedure is considered (also used as repair procedure), the percentage of feasible solutions increase considerable. For example, using `2R` or `2V` the percentage of feasible solutions is higher than 75% or higher than 82% for `3V`. This occurs even with the random constructive heuristic, showing that the local search has a performance that is not high-dependent of the initial solution. We also analyze the quality of the feasible solutions found on each strategy. The conclusions are similar to the percentage of feasible solutions, where the solution quality are similar for all constructive heuristic and tends to be better when we consider a more complex neighborhood search. Thus, in the remaining experiments we adopted the *DC-Greedy* constructive heuristic as default since it presents the highest percentage of feasible solutions.

Table 4.4 shows the percentage of feasible solutions for each instance for the constructive heuristic *DC-Greedy*.

Even with a significant increase in the percentage of feasible solutions found when the local search is applied, some instances still have a low percentage of feasible solutions. This shows that the set of instances used to evaluate the algorithms are diverse and not trivial to solve. We also observed that, as expected, most instances with a higher percentage of data center occupation has a low percentage of feasible solutions in comparison with the respective instance with a low percentage of occupation.

Table 4.4 – Percentage of feasible solutions found by GRASP with the *DC-Greedy* constructive heuristic

Instance	NoLS	1R	2R	2V	3R	3V	4R	4V
10_025_012_70	10.90	60.50	89.92	90.88	95.62	95.75	96.06	96.24
10_025_012_90	1.27	24.57	90.13	92.25	94.79	95.68	95.09	96.18
10_025_025_70	25.75	92.23	99.09	99.44	99.70	99.78	99.80	99.86
10_025_025_90	0.08	6.09	46.20	52.23	58.34	62.96	59.35	64.72
10_025_037_70	2.91	59.80	76.71	81.60	84.88	86.55	86.02	88.67
10_025_037_90	0.06	4.93	37.30	44.24	50.14	57.58	54.19	63.35
10_050_025_70	4.38	42.79	94.06	94.45	95.43	95.80	95.73	96.17
10_050_025_90	0.13	21.10	88.04	88.80	92.41	92.63	92.59	93.11
10_050_050_70	0.34	23.77	69.28	71.09	76.80	76.97	78.49	78.78
10_050_050_90	0.05	13.24	54.16	56.53	64.63	65.21	65.01	65.81
10_050_075_70	1.50	49.55	68.47	72.63	76.83	77.26	78.12	78.96
10_050_075_90	0.07	10.02	67.46	68.59	71.43	72.29	73.02	74.02
10_100_050_70	3.34	65.63	98.34	98.49	98.91	98.96	98.79	98.62
10_100_050_90	0.01	3.07	80.76	80.60	86.31	86.54	86.63	86.99
10_100_100_70	1.66	39.19	86.97	87.45	91.79	91.28	93.50	93.38
10_100_100_90	0.00	7.45	83.73	84.02	86.42	86.81	87.11	86.66
10_100_150_70	0.82	41.94	82.79	83.35	85.92	86.01	86.27	86.89
10_100_150_90	0.00	1.46	73.12	73.33	73.85	74.18	75.44	75.63
25_100_050_70	8.10	77.11	96.44	95.89	97.79	98.03	98.31	98.46
25_100_050_90	0.00	13.16	82.81	84.37	88.04	87.90	87.78	89.16
25_100_100_70	0.56	40.15	86.63	88.12	90.76	91.93	91.77	91.82
25_100_100_90	0.00	3.08	64.58	66.90	72.86	74.69	73.30	75.48
25_100_150_70	1.23	46.98	92.22	93.80	96.07	96.42	96.78	97.31
25_100_150_90	0.00	2.78	64.08	66.30	72.35	73.97	72.45	75.14
25_150_075_70	5.84	60.39	94.84	95.35	96.05	96.03	96.81	97.12
25_150_075_90	0.01	6.60	84.01	84.39	87.36	87.73	87.88	88.13
25_150_150_70	0.77	31.90	79.71	79.68	83.50	84.00	84.65	85.29
25_150_150_90	0.00	5.23	63.12	64.75	68.89	70.49	70.48	69.27
25_150_225_70	0.53	31.81	79.31	80.36	83.05	83.15	82.89	83.33
25_150_225_90	0.00	1.59	48.00	47.46	56.87	57.90	58.14	56.01
25_200_100_70	0.21	31.49	83.76	85.27	88.58	86.11	90.90	90.86
25_200_100_90	0.00	21.90	94.83	95.36	97.16	97.76	96.91	97.36
25_200_200_70	0.07	22.82	72.57	75.00	81.10	80.57	81.96	81.38
25_200_200_90	0.00	21.36	93.03	93.51	95.24	95.41	95.02	95.36
25_200_300_70	0.00	13.36	64.51	66.92	72.57	72.53	73.12	73.55
25_200_300_90	0.00	4.49	86.92	86.54	88.24	89.12	89.31	90.08
Average	1.96	27.88	78.28	79.72	83.35	84.06	84.16	84.98

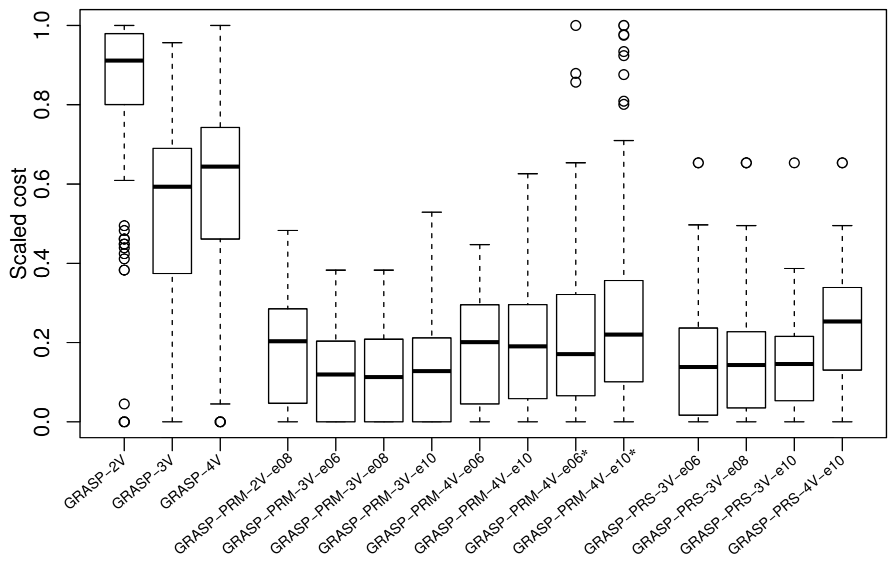
Source: from the author (2015).

In the next set of experiments, the main objective is to evaluate the path-relinking component in the GRASP procedure. For each experiment, we performed five independent runs totalling an amount of 180 samples for each experiment. We also use $\theta = 0.8$ as stopping criterion The local search strategy used in the path-relinking is always the same used after the construction phase in GRASP procedure. Experiments are chosen to analyze the performance for different combination of algorithms and to show the impact of each component.

To compare the results with respect to the best solution found on each run, we first scale the objective function value to the range $[0, 1]$ since each instance can have very different values. The scale is a simple transformation where for each instance, the largest cost over all analyzed experiments is scaled to 1 and the lowest is scaled to 0.

Figure 4.3 shows the distribution of the scaled cost for each algorithm. The box plots show the location of the minimum value, lower quartile, median, upper quartile, and maximum value of each experiment. The dots are the outliers. Experiments are represented on the horizontal axis and are labelled as composition of main algorithm name, path-relinking type, local search strategy, and size of elite set parameter. Experiments ended with "$*$" use a full exploration on the neighborhood *chain3L*, i.e, $\alpha_{3L} = 1$ and $\beta_{3L} = |N|$. Also, note that the first three experiments are a simple GRASP procedure and does not include the path-relinking component.

Figure 4.3 – Dispersion of scaled cost for each algorithm



Source: from the author (2015).

The first observation is about the relation between the intensification of the neighborhood exploration and restarting the search. On the one hand, from Table 4.3 and 4.4 we observe that the local search mode `V` overcomes in most cases the mode `R` in the percentage of feasible solutions. Since both strategies are executed for the same time, is natural that the local search mode `R` has a higher number or iterations than mode `V`, which has a more intensive exploration in high-quality solutions. This is an indication that is preferable to invest in the exploration of the neighborhood of a high-quality solution instead of making a full restart to a new point of the search space. One the other hand, even with a higher ability of the local search `4V` in producing feasible solutions and exploring a large neighborhood, when we consider the best solution found in the experiments, the local search `3V` produces better results than the search `4V`. This indicates that for the proposed neighborhood strategies, there is a trade-off between intensification of the neighborhood exploration and time spent on each local search strategy. This can be used to guide the search and select the best strategies for a general proposed method.

The second observation is that the path-relinking component improves significantly the results in comparison with the case without this strategy (case GRASP-2V, GRASP-3V, and GRASP-4V). We also observe that both path-relinking strategies have similar performance. Furthermore, the results have a low influence by the values of elite size in the tested interval.

To confirm the results presented in Figure 4.3, we use the R package to test the normality of these distributions using the Shapiro-Wilk test and apply the Mann-Whitney-Wilcoxon U test. For all tests, we assume a confidence interval of 99%. Shapiro-Wilk tests indicate that no cost 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 as null hypothesis that the location statistics are equal in both distributions. We also use a p-value correction procedure based on false discovery rate (FDR) to minimize the number of false positives.

Table 4.5 shows U test results for each pair of algorithms. 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 cost of the corresponding algorithm. The upper-right diagonal elements are the differences in location statistics for each pair of algorithms. A negative difference indicates that the "row algorithm" has its location statistics lower (better) than the "column algorithm", and the positive difference is the opposite. The bottom-left diagonal elements are the p-values of each test. Math signals indicate when $p<0.01$ for a U test between "row algorithm" and "column algorithm" for the respective signal alternative hypothesis. The case "less" indicates that the "row algorithm" overcome the "column algorithm", or the opposite in the case "greater".

Table 4.5 – Values of medians, p-values, and difference in median location for cost distributions using a confidence interval of 99% for GRASP algorithm

	GRASP			**GRASP-PRM**								**GRASP-PRS**			
	2V	3V	4V	2V-e08	3V-e06	3V-e08	3V-e10	4V-e06	4V-e10	5V-e06	5V-e10	3V-e06	3V-e08	3V-e10	4V-e10
2V	**0.911**	0.311	0.265	0.683	0.755	0.756	0.751	0.683	0.679	0.676	0.642	0.730	0.728	0.734	0.647
3V	<	**0.593**	-0.048	0.379	0.452	0.456	0.448	0.387	0.379	0.360	0.332	0.424	0.422	0.427	0.342
4V	<	>	**0.644**	0.426	0.496	0.498	0.489	0.434	0.421	0.416	0.389	0.468	0.465	0.476	0.393
PRM-2V-e08	<	<	<	**0.203**	0.075	0.081	0.069	-0.001	0.003	-0.014	-0.051	0.056	0.053	0.061	-0.035
PRM-3V-e06	<	<	<	<	**0.119**	0.002	-0.006	-0.077	-0.075	-0.085	-0.121	-0.022	-0.029	-0.017	-0.118
PRM-3V-e08	<	<	<	<	0.839	**0.113**	-0.010	-0.077	-0.082	-0.086	-0.125	-0.027	-0.028	-0.022	-0.119
PRM-3V-e10	<	<	<	<	0.568	0.353	**0.128**	-0.072	-0.075	-0.082	-0.116	-0.020	-0.017	-0.012	-0.110
PRM-4V-e06	<	<	<	0.930	>	>	>	**0.201**	-0.007	-0.014	-0.049	0.051	0.054	0.058	-0.039
PRM-4V-e10	<	<	<	0.846	>	>	>	0.582	**0.190**	-0.015	-0.045	0.056	0.055	0.058	-0.037
PRM-5V-e06	<	<	<	0.409	>	>	>	0.376	0.294	**0.171**	-0.041	0.063	0.058	0.067	-0.031
PRM-5V-e10	<	<	<	>	>	>	>	>	>	>	**0.220**	0.100	0.093	0.096	0.006
PRS-3V-e06	<	<	<	<	0.075	0.040	0.089	<	<	<	<	**0.139**	-0.004	0.003	-0.097
PRS-3V-e08	<	<	<	<	0.016	0.010	0.091	<	<	<	<	0.737	**0.144**	0.009	-0.097
PRS-3V-e10	<	<	<	<	0.065	0.066	0.243	<	<	<	<	0.774	0.384	**0.146**	-0.102
PRS-4V-e10	<	<	<	>	>	>	>	>	>	0.056	0.774	>	>	>	**0.253**

Source: from the author (2015).

Table 4.5 confirms that the difference between the results of the standard GRASP procedure and the procedure that includes the path-relinking component are statistically significant for confidence interval of 99%. Tests between cases with local search `3V` and different elite sizes indicate that the difference are not significant, even for both path-relinking strategies. However analyzing the values of median and median location, we conclude that there is a very low advantage for the `PRM` strategy with elite size equal to 8.

A report for the solution quality obtained with this strategy is showed in the Subsection 4.4.5. Next we report results for the BRKGA.

4.4.4 BRKGA results

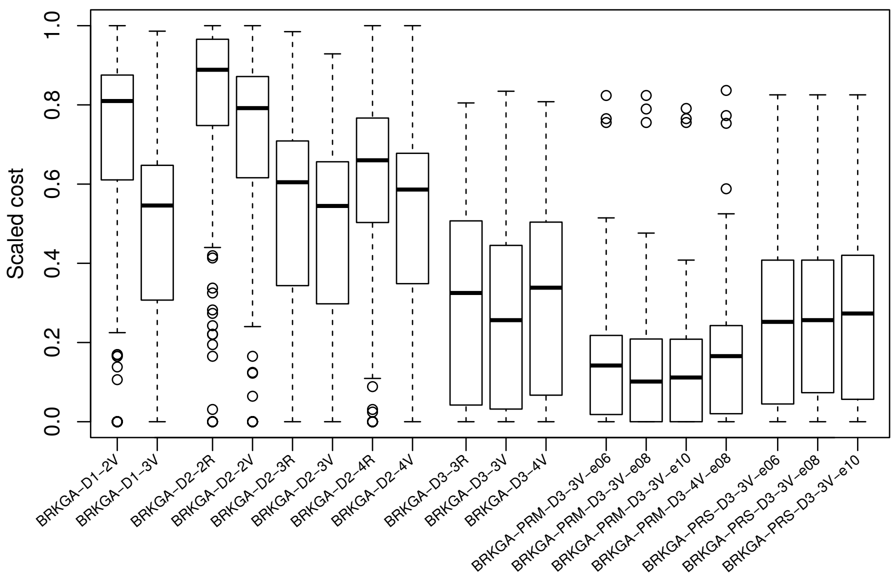
This subsection presents results for the biased random-key genetic algorithm. The objective is analyze different procedures, decoders, the recode process, and the path-relinking procedure introduced in the algorithm. The experiments follow the rules and the objectives explained in the previous subsection.

BRKGA parameters was set to the same parameters as described in Stefanello et al. (2015a) which are similar to the values suggested by Gonçalves and Resende (2011), i.e., the elite size $p_e = 0.24p$, the set of mutants $p_m = 0.12p$, and the probability of inheriting $\rho_A = 0.6$. The restart parameter was disable and the population size $p = 75$. The stopping criterion for all runs was a time set to $|K| * |N| * \theta$ seconds (where $\theta = 0.8$).

We run experiments that include different local searches, decoders and path-relinking procedures. In each experiment, we performed five independent runs with different random seed for each instance, given a total of 180 runs. To compare the results we scale the values of the objective function for the best solution found on each run to the range $[0, 1]$, as described in the previous subsection.

Figure 4.4 shows the box plot for the distribution of the scaled cost for each algorithm. Algorithms are represented on the horizontal axis and are labelled as a composition of main algorithm name, path-relinking strategy, decoder method, local search strategy, and size of elite set parameter.

Figure 4.4 – Dispersion of scaled cost for each algorithm



Source: from the author (2015).

The first observation is that the recode process helps to improve the results, since all experiments with decoder D3 have medians lower than the respective correspondent algorithm with decoders D1 and D2. Note for example that without the recode, one modification in a random key can influence in one or more placements (since the local search can apply many changes). Using the recode, when one key is changed, the modification is in only one placement. This well-defined behavior helps to keep a more accurate information of each individual. The second observation is that for the local search strategies, the same conclusions from the GRASP can be taken for BRKGA. Experiments that include mode `V` overcomes the mode `R`, as well as, using local search `3V` provides better results than the other local search strategies. Finally, strategies that include the path-relinking procedure are more effective to produce better results than the standard counterpart. However, differently as observed in GRASP, the difference between `PRM` and `PRS` is significant. We attribute the worst performance of `PRS` due to the small number of applications of the path-relinking since it is limited to the number of generations. Also, after some iterations of BRKGA, the solutions in the elite set of BRKGA tend to be similar, and fail in the similarity check with the solution in the elite set of path-relinking, and thus, the path-relinking operator is not performed.

Table 4.6 shows U test results for each pair of algorithms. We present the values of medians, p-values, and the difference in median location for the scaled cost distributions using a confidence interval of 99% for all experiments. The organization of this table follows the description of Table 4.5 presented in the previous subsection. For layout reasons we omit in this table the data for experiments with local search `2R`, `3R`, and `4R`, since the mode `V` for the respective neighborhood level always overcome the mode `R`. We also omit the decoder D3 in the label of the experiments with BRKGA-PR, since we perform experiments only with this decoder.

Table 4.6 – Values of medians, p-values, and difference in median location for cost distributions using a confidence interval of 99% for BRKGA algorithm

	BRKGA							**BRKGA-PRM**				**BRKGA-PRS**		
	D1-2V	D1-3V	D2-2V	D2-3V	D2-4V	D3-3V	D3-4V	3V-e06	3V-e08	3V-e10	4V-e08	3V-e06	3V-e08	3V-e10
D1-2V	**0.810**	0.231	0.002	0.224	0.197	0.468	0.413	0.623	0.641	0.646	0.591	0.487	0.476	0.471
D1-3V	<	**0.546**	-0.223	0.001	-0.026	0.228	0.170	0.382	0.403	0.407	0.366	0.251	0.242	0.239
D2-2V	0.888	>	**0.792**	0.222	0.182	0.454	0.397	0.604	0.624	0.626	0.579	0.473	0.462	0.458
D2-3V	<	0.920	<	**0.545**	-0.033	0.229	0.173	0.390	0.410	0.415	0.367	0.249	0.244	0.239
D2-4V	<	0.017	<	>	**0.586**	0.250	0.194	0.414	0.433	0.439	0.391	0.272	0.264	0.260
D3-3V	<	<	<	<	<	**0.256**	-0.056	0.166	0.190	0.185	0.130	0.028	0.013	0.017
D3-4V	<	<	<	<	<	>	**0.338**	0.212	0.243	0.241	0.188	0.083	0.067	0.069
PRM-3V-e06	<	<	<	<	<	<	<	**0.142**	0.018	0.021	-0.025	-0.138	-0.148	-0.153
PRM-3V-e08	<	<	<	<	<	<	<	0.081	**0.101**	0.001	-0.040	-0.159	-0.170	-0.174
PRM-3V-e10	<	<	<	<	<	<	<	0.038	0.888	**0.112**	-0.043	-0.163	-0.177	-0.176
PRM-4V-e08	<	<	<	<	<	<	<	0.040	>	>	**0.166**	-0.106	-0.116	-0.119
PRS-3V-e06	<	<	<	<	<	0.011	<	>	>	>	>	**0.252**	-0.014	-0.014
PRS-3V-e08	<	<	<	<	<	0.166	<	>	>	>	>	0.102	**0.256**	-0.002
PRS-3V-e10	<	<	<	<	<	0.066	<	>	>	>	>	0.112	0.804	**0.273**

Source: from the author (2015).

This table confirms the considerations described above showing that the best approaches consider the path-relinking component (`PRM`). Also, as mentioned for GRASP in the previous subsection, the experiments for different elite sizes indicate that the differences are not statistical significant for a confidence interval of 99%. However, analyzing the values of the median, and median location we conclude that using the size of elite set equal to 8, the procedure tends to produce better results.

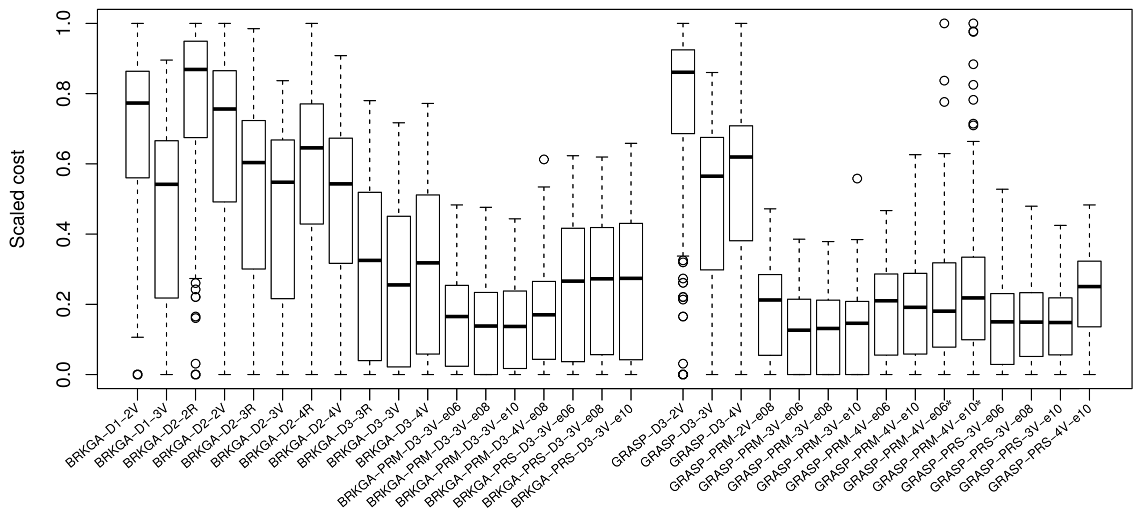
In Stefanello et al. (2015a) this problem was first introduced, and the first version of BRKGA is presented. From the original approach, three main improvements are made. First, we added a new neighborhood search. In the original approach, we described a local search that includes only shift and swap moves (namely LSW that correspond to `2V` in this work). We observe from the results in Figure 4.4 and in the Table 4.6 that the inclusion of this new neighborhood strategy (*chain search*) helps to improve significantly the results (comparison between BRKGA-D2-2V and BRKGA-D2-3V). Second, the inclusion of the recode procedure in the decoder D2 (defined as decoder D3), also improves significantly the results (comparison between BRKGA-D2-2V and BRKGA-D3-2V, as well as, BRKGA-D2-3V and BRKGA-D3-3V). Finally, we included a path-relinking component that in the best of our knowledge, is the first experiment that consider this hybrid approach between BRKGA and path-relinking.

A final observation about running times of BRKGA is that results are reported using single-thread in order to provide a fair comparison with the GRASP and CPLEX results. However, the BRKGA API provides an efficient multi-thread decoding Toso and Resende (2014), that could be used to reduce substantially the running times when multiple processors are available.

4.4.5 Additional comparison

In this subsection we report an overview over the best strategies for BRKGA and GRASP, providing addition comparisons and informations for both algorithms. Figure 4.5 shows an overview for the experiments for both strategies reported in the previous subsections.

Figure 4.5 – Dispersion of scaled cost for each algorithm



Source: from the author (2015).

As depicted in Figure 4.5 and reported in subsections 4.4.3 and 4.4.4, the best approach for both algorithms is consider the local search `3V` and include the path-relinking component with elite size equal to 8. Thus, Table 4.7 presents supplementary information for the experiments with both strategies. The first column shows the instance. The second column shows the best-known solution value (BKS) for each instance. The next columns show the minimum, average and maximum of percentage gap for both algorithms. The percentage gap is calculated by the formula (FO-BKS)*100/BKS, were FO is the respective value of the objective function for the considered algorithm. Finally, columns Time shows the minimum and average time in seconds in which each algorithm obtained the last improvement in the objective function.

Table 4.7 – Comparison of percentage gap and last improve time for BRKGA-PR and GRASP-PR

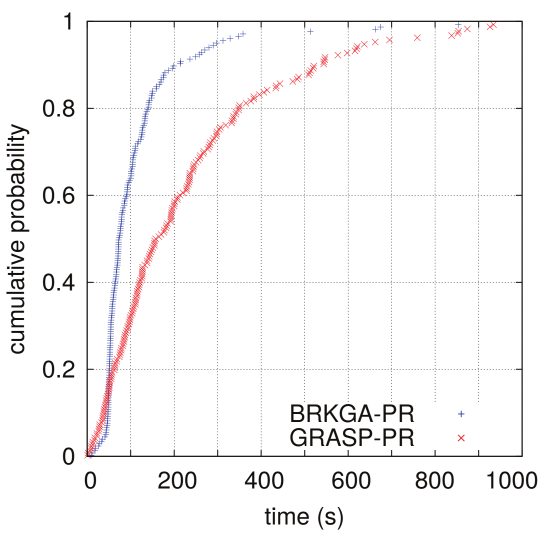
		BRKGA-PR					**GRASP-PR**				
		GAP (%)			**Time (s)**		**GAP (%)**			**Time (s)**	
Instance	**BKS**	**min**	**Avg**	**Max**	**Min**	**Avg**	**min**	**Avg**	**Max**	**Min**	**Avg**
10_025_012_70	114,582.50	0.00	0.00	0.00	1.6	22.7	0.00	0.00	0.00	1.3	38.7
10_025_012_90	84,461.30	0.00	0.32	0.54	1.8	35.9	0.00	0.00	0.00	16.1	49.2
10_025_025_70	90,997.90	0.00	0.00	0.00	0.5	0.8	0.00	0.00	0.00	48.5	111.3
10_025_025_90	124,763.66	0.00	0.15	0.18	1.9	45.1	0.00	0.04	0.18	19.1	59.8
10_025_037_70	100,801.80	0.00	0.00	0.00	0.7	0.8	0.00	0.00	0.00	0.4	1.0
10_025_037_90	106,617.94	0.00	0.00	0.00	0.7	13.3	0.00	0.00	0.00	1.1	2.3
10_050_025_70	414,535.12	0.11	0.26	0.47	12.5	86.5	0.00	0.08	0.13	47.9	190.9
10_050_025_90	458,879.74	0.00	0.02	0.07	38.3	76.0	0.00	0.09	0.16	87.5	193.6
10_050_050_70	360,102.12	0.00	0.10	0.49	11.4	36.0	0.00	0.00	0.00	4.5	92.4
10_050_050_90	400,233.16	0.00	0.37	0.64	21.2	32.9	0.00	0.28	0.62	32.9	185.6
10_050_075_70	349,135.78	0.00	0.00	0.00	4.7	29.8	0.00	0.00	0.00	7.7	62.1
10_050_075_90	498,190.58	0.39	0.58	0.87	47.9	153.7	0.00	0.26	0.47	18.1	147.8
10_100_050_70	1,647,975.00	0.00	1.98	3.32	82.8	395.4	1.12	2.26	3.37	457.5	652.2
10_100_050_90	1,792,257.68	0.01	0.18	0.31	98.1	380.7	0.14	0.31	0.55	292.5	493.0
10_100_100_70	1,463,498.00	0.00	0.08	0.37	50.9	271.7	0.00	0.03	0.14	518.3	624.6
10_100_100_90	2,126,993.26	0.00	0.23	0.48	433.7	656.5	0.12	0.28	0.53	60.6	483.5
10_100_150_70	1,563,152.40	0.00	0.13	0.40	190.1	437.5	0.06	0.30	0.59	198.7	537.4
10_100_150_90	1,847,076.66	0.01	0.12	0.32	137.2	429.0	0.08	0.26	0.39	252.6	575.8
Average		0.03	0.25	0.47	63.1	172.5	0.08	0.23	0.40	114.7	250.0
25_100_050_70	1,887,688.24	0.18	0.43	0.78	160.3	1,128.9	0.12	0.30	0.47	953.2	1,464.6
25_100_050_90	2,116,849.00	0.03	0.52	0.81	904.3	1,615.4	0.26	0.37	0.46	741.2	1,326.4
25_100_100_70	1,953,155.20	0.36	0.62	0.83	284.5	770.2	0.00	0.35	0.75	253.9	1,122.9
25_100_100_90	2,021,228.76	0.35	0.72	1.24	976.0	1,516.7	0.44	0.55	0.65	847.5	1,207.5
25_100_150_70	1,967,364.52	0.30	0.57	0.82	276.5	1,141.6	0.00	0.42	0.70	160.5	1,377.1
25_100_150_90	2,160,014.54	0.56	0.80	1.08	776.3	1,509.1	0.44	0.62	0.77	528.9	1,156.7
25_150_075_70	4,603,163.50	0.20	0.37	0.49	531.1	1,877.3	0.24	0.33	0.41	992.6	1,567.2
25_150_075_90	4,618,491.80	0.21	0.33	0.49	1,223.7	1,807.9	0.21	0.31	0.40	1,986.0	2,548.8
25_150_150_70	3,882,650.94	0.19	0.36	0.54	788.5	2,183.6	0.19	0.34	0.49	1,560.7	2,156.1
25_150_150_90	4,706,129.66	0.41	0.53	0.68	597.5	1,984.0	0.32	0.46	0.58	889.2	2,067.9
25_150_225_70	4,340,090.18	0.13	0.34	0.53	1,657.6	2,347.1	0.47	0.59	0.80	1,648.5	2,321.8
25_150_225_90	4,523,393.44	0.44	0.54	0.69	271.5	1,684.5	0.17	0.37	0.53	543.2	1,607.5
25_200_100_70	6,937,008.98	0.22	0.28	0.35	3,123.8	3,938.7	0.07	0.26	0.44	1,855.8	2,841.7
25_200_100_90	9,034,147.98	0.15	0.29	0.48	2,017.7	3,352.2	0.15	0.29	0.44	1,371.5	2,688.1
25_200_200_70	7,146,330.32	0.13	0.24	0.44	1,271.1	2,948.6	0.10	0.35	0.52	1,718.1	3,201.1
25_200_200_90	8,578,620.94	0.00	0.15	0.26	1,170.0	2,669.4	0.18	0.22	0.30	326.3	1,681.0
25_200_300_70	7,638,447.16	0.00	0.25	0.35	1,763.6	3,216.9	0.19	0.28	0.39	819.1	2,745.9
25_200_300_90	8,195,152.30	0.05	0.18	0.27	1,523.7	2,689.1	0.13	0.27	0.37	913.1	2,796.6
Average		0.22	0.42	0.62	1,073.2	2,132.3	0.20	0.37	0.53	1,006.1	1,993.3

Source: from the author (2015).

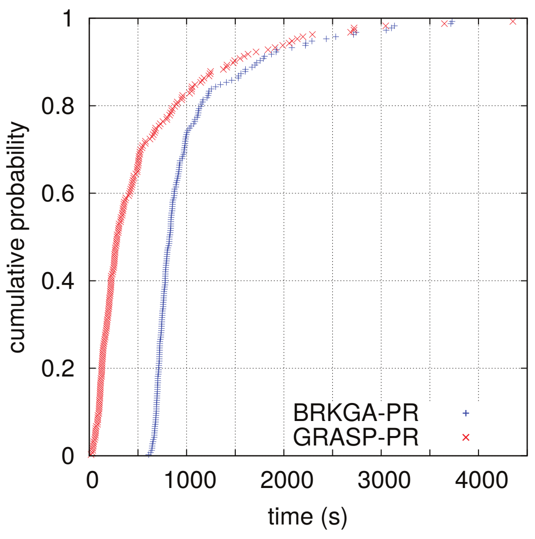
From this table, we observe that the percentage gap is relatively small for both algorithms, showing that the approaches are efficient to find good quality solutions. For instances with 10 data centers, in most cases both algorithms found the BKS. Also, considering that each run was stopped by time limit ($|N| * |K| * 0.8$), from columns Time we observe that a significant parcel of the execution was spent without improvements. This indicates that the algorithm has a fast convergence, and will probably provide a good solution if stopped before the used time limit. Also, we observe that BRKGA-PR is slightly faster than GRASP-PR for the instance set with 10 data centers. However, the opposite happens for the instances with 25 data centers.

Finally, the last experiment uses the Time-To-Target (TTT) plots to display the running time distribution for the algorithm to find a solution at least as good as a given target value. TTT plots were proposed by Aiex et al. (2007) and have been advocated by Hoos and Stützle (1998b), Hoos and Stützle (1998a) as a way to characterize the running times of stochastic algorithms for combinatorial optimization problems. The experiment consists in performing 200 runs of BRKGA-PR and GRASP-PR algorithms for two instances until a target is reached. The target was set to the worst solution found for both strategies on the previous experiment, i.e, 1,468,973 for `010_100_100_70` and 9,077,672 for `025_200_100_90`. The instances chosen was one at each group of data centers. We select the instance with the lowest difference of average percentage gap between both approaches (excluding equal values).

Figure 4.6 – Cumulative probability distribution



(a) TTT plot for `10_100_100_70`



(b) TTT plot for `25_200_100_90`

Source: from the author (2015).

Figures 4.7a and 4.7b illustrate the cumulative probability plot obtained by using BRKGA-PR and GRASP-PR for two instances. For the instance `10_100_100_70`, we observe that BRKGA-PR has a higher probability to find a solution at least as good as the target in less time than GRASP-PR. For instance `25_150_225_70` the probability is the opposite.

In summary, both approaches have good performance according to specific scenarios. The main performance of BRKGA-PR is in small and median instances while GRASP-PR has a better performance in large instances. The path-relinking component provides a higher improvement for both strategies. We attribute this performance to a rugged landscape for this problem and the fact that is hard to find feasible solutions. Since in general the path-relinking is performed between two feasible solutions, this component explores a search space where solutions are feasible or has a lower number of infeasibilities. Thus, when the local search is applied to a solution in the path, the search tends to be faster and lead to a new minimum local, that can be better than the initial and the guided solution.

Finally, we evaluate our algorithm considering a set of instances which sizes were not address before in the literature for this problem. We test in instances consider up to 25 data centers and 200 virtual machines. Even that in real world application this size can be considered small, previous works on a related problem in the literature has to address instances up to 50 facilities (virtual machines). Thus, we consider a significant step since we develop solutions for instances five times larger the previously considered size. Furthermore, we provide a new database for a future comparison of new algorithms.

In the next subsection we evaluate our proposed method in an adapted set of instances from a similar problem and compare our approach with a method described in the literature.

4.4.6 Results for the Generalized Quadratic Assignment Problem

The VMPlacement is a generalization of GQAP, our proposed method can be easily adapted for GQAP. Since we evaluate the infeasibilities in the objective function, we only need to change the cost evaluation function. However, for convenience, we adapted the instances of GQAP for VMPlacement problem, just given a sufficiently large value for the bandwidth capacity between each pair of data centers and required latency between each pair of virtual machines. Finally, we also define an empty set of users.

The main objective of this experiment is to provide a brief comparison with a method described in the literature to show that our approach has a competitive performance.

We first perform an experiment with CPLEX to provide a baseline comparison for exact methods. After, we compare our approaches to a GRASP with path-relinking proposed by Mateus et al. (2010).

We use a set of instances proposed by Cordeau et al. (2006). These instances has 20 to 50 facilities (virtual machines) and 6 to 20 locations (data centers). Instances are listed in Table 4.8 and are labelled respectively by the number of facilities (virtual machines), locations (data centers), and a parameter that controls the tightness of the capacity constraints. The higher the value of the parameter, the higher is the tightness of the capacity constraints. Since this set comprises the largest instance set available for GQAP, and we observe a good performance of our approach we evaluate our strategies only in this set of instances. These experiments are carried out on a computer with an Intel(R) Core(TM) i7 CPU 930 2.80 GHz with 12GB of main memory.

Cordeau et al. (2006) evaluate the CPLEX (version 8.1) exploring the emphasizes parameter on the branch-and-bound tree exploration for a time limit of two hours. CPLEX was able to prove optimality only for the instance `30-08-55`. Pessoa et al. (2010) proposed exact algorithms that combine branch-and-bound with a new Lagrangean decomposition and the Reformulation-Linearization Technique and prove the optimality for another 13 instances of the previously described instance set.

In the first experiment we evaluate the performance of CPLEX with LMVMP and LMVMPII described in Section 4.2. We run CPLEX for each instance with a time limit of one day (86,400 seconds). We also set the tree memory parameter (`TreLim`) to 5000 to stop the execution when this memory limit is exceeded. The remaining parameters are kept to their default values.

Table 4.8 shows the results for each instance listed in the first column. The second column shows the best-known solutions for each instance (optimal values are given in boldface). The next two sets of columns show the results for each linear mathematical model. Column Nodes shows the number of nodes solved by CPLEX. Column Integer Sol shows the objective function of best solution found during the execution. Column gap shows the percentage gap between the lower bound and the best integer solution found by CPLEX. Column Time shows the running time in seconds to CPLEX prove the optimality or achieve a stopping criterion. Instances that CPLEX found the optimal solution or prove the optimality are highlighted in boldface.

Table 4.8 – CPLEX results for GQAP instances

		LMVMP				LMVMPII			
Instance	**BKS**	**Nodes**	**Integer Sol**	**gap** (%)	**Time** (s)	**Nodes**	**Integer Sol**	**gap** (%)	**Time** (s)
20-15-35	**1,471,896**	2,990	**1,471,896**	**0.00**	3,788.3	30,709,073	**1,471,896**	7.95	41,981.1
20-15-55	**1,723,638**	10,839	**1,723,638**	**0.00**	9,149.2	30,269,870	**1,723,638**	6.58	39,265.6
20-15-75	**1,953,188**	5,725	**1,953,188**	**0.00**	4,546.3	1,867,883	**1,953,188**	**0.00**	3,979.6
30-06-95	**5,160,920**	24,889	**5,160,920**	**0.00**	9,471.3	16,820,694	**5,160,920**	**0.00**	35,086.4
30-07-75	**4,383,923**	47,884	**4,383,923**	**0.00**	22,429.5	26,644,434	**4,383,923**	4.50	63,524.3
30-08-55	**3,501,695**	793	**3,501,695**	**0.00**	370.1	14,090,023	**3,501,695**	**0.00**	36,009.5
30-10-65	**3,620,959**	41,020	**3,620,959**	**0.00**	65,805.4	19,972,814	**3,620,959**	13.32	58,100.3
30-20-35	**3,379,359**	1,872	3,605,129	23.96	86,400.0	14,857,869	**3,379,359**	35.36	50,023.8
30-20-55	**3,593,105**	1,680	3,865,716	31.23	86,400.0	14,117,019	**3,593,105**	30.27	54,253.0
30-20-75	**4,050,938**	1,946	4,245,753	26.43	86,400.0	13,553,603	**4,050,938**	21.13	59,363.0
30-20-95	**5,710,645**	3,575	5,840,934	12.96	86,400.0	6,869,516	**5,710,645**	2.71	86,400.2
35-15-35	**4,456,670**	3,290	**4,456,670**	13.23	86,400.0	15,797,188	4,457,348	34.76	40,190.5
35-15-55	**4,639,128**	3,078	4,723,959	19.34	86,400.0	13,903,550	**4,639,128**	27.52	40,295.3
35-15-75	6,301,723	1,139	6,395,402	27.83	86,400.0	13,348,410	6,327,723	28.02	59,428.9
35-15-95	**6,670,264**	1,546	7,370,866	32.09	86,400.0	9,664,857	6,689,421	20.38	86,400.0
40-07-75	7,405,793	34,769	**7,405,793**	3.60	86,400.0	20,466,362	**7,405,793**	12.94	35,228.8
40-09-95	7,667,719	8,471	7,941,330	18.26	86,400.0	16,718,721	7,694,904	18.61	38,337.7
40-10-65	7,265,559	6,790	7,305,381	10.80	86,400.0	16,475,064	**7,265,559**	22.79	49,526.1
50-10-65	10,513,029	5,467	**10,513,029**	4.15	86,400.0	14,617,831	**10,513,029**	19.39	52,238.3
50-10-75	11,217,503	2,209	11,415,840	19.89	86,400.0	14,018,304	11,251,072	24.42	54,332.5
50-10-95	12,845,598	1,960	13,242,115	18.57	86,400.0	13,205,512	**12,845,598**	19.25	57,175.5

Source: from the author (2015).

Observing the results in Pessoa et al. (2010) and comparing with the results in Table 4.8, we note that the branch-and-bound approach in Pessoa et al. (2010) tends to be more efficient that CPLEX. In general, the running time tends to be small than the required by CPLEX. However, both methods are not strictly comparable since the experiments use different stop criterion, and are reported over different computers. In the end, the main observation from these results is that even with a high increase in the computational power and the development of many algorithms and improvements over each CPLEX version, this set of instances is still hard to be solved by exact methods.

In a second experiment, we compare our strategies similarity as described for GRASP-PR proposed by Mateus et al. (2010), state-of-art algorithm for GQAP. For each instance we performed 200 independent runs. Each run stopped when a solution value as good as the best-known solution was found (column BKS in Table 4.8).

Two main differences are observed between our approaches and GRASP-PR proposed by Mateus et al. (2010). First, we performed a wide exploration with the local search strategy. Mateus et al. (2010) uses an approximate local search using shift and swap moves. In our approach, we explore all neighbors using three neighborhood structures. Second, we use a penalization strategy to deal with infeasible solutions. This allows passing by through infeasible solutions to reach new feasible solutions, which can be difficult to be reached only exploring a feasible search space.

Table 4.9 shows a comparison of GRASP-PR from Mateus et al. (2010) and our best two approaches described in the previous subsection. The first column shows the name of the instance. The next columns show some information for each experiment. Columns Min, Avg, Max, Sd give the minimum, average, and maximum times, as well as the standard deviation of these runs to find a solution with values equal to BKS. Finally, column 0.95 shows the time in which 95% of the runs find the BKS.

The results described in Mateus et al. (2010) are reported over a computer Dell PE1950 with dual quad core 2.66 GHz Intel Xeon processors. Unfortunately, there is no specific information about the processor model, and then we suppose the model Intel Xeon X5355 2.66 GHz, based on server model, the number of cores e clock frequency. We run our algorithms for this set of experiments on a computer with an Intel(R) Core(TM) i7 CPU 930 2.80 GHz. The machine that we use is approximately 1.2 times faster than the one used in Mateus et al. (2010) (based on Single Thread Rating data from <wwww.cpubenchmark.net>). The execution times for the two implementations are not strictly comparable since the languages and compilers used are different. However, at a higher level, this comparison provides an idea of the behavior of the algorithms.

Table 4.9 – Comparison algorithms for GQAP

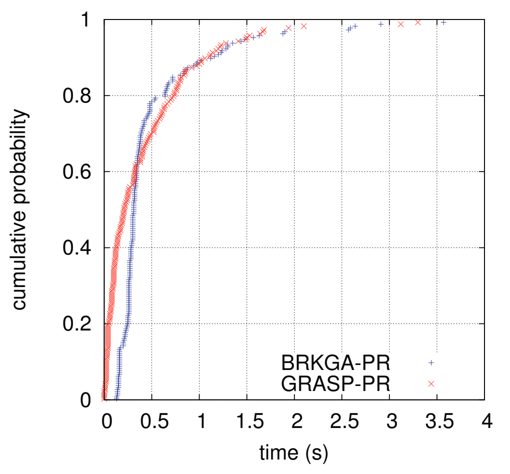
	Mateus et al. (2010)					**BRKGA-PR**					**GRASP-PR**				
Instance	Min	Avg	Max	Sd	0.95	Min	Avg	Max	Sd	0.95	Min	Avg	Max	Sd	0.95
20-15-35	0.16	7.05	38.87	6.47	21.04	0.06	0.13	0.28	0.04	0.17	0.01	0.08	0.57	0.09	0.22
20-15-55	0.24	2.87	14.42	2.18	7.69	0.06	0.12	0.17	0.03	0.16	0.01	0.13	0.75	0.14	0.36
20-15-75	0.26	2.01	12.82	1.72	5.25	0.08	0.39	3.44	0.50	1.32	0.01	0.41	3.30	0.51	1.41
30-06-95	0.55	2.59	23.81	2.22	6.44	0.16	0.41	2.23	0.25	0.88	0.01	0.48	1.78	0.36	1.16
30-07-75	0.50	7.80	38.47	5.47	18.18	0.15	0.66	5.22	0.63	1.86	0.01	0.48	4.10	0.49	1.28
30-08-55	0.18	1.61	4.89	0.95	3.60	0.13	0.27	0.40	0.06	0.35	0.01	0.03	0.24	0.03	0.10
30-10-65	2.75	121.94	1,032.80	146.06	514.82	0.18	1.58	12.27	1.83	5.35	0.01	1.59	8.83	1.53	4.41
30-20-35	1.08	79.03	4,441.40	312.62	166.21	0.24	0.73	1.73	0.29	1.27	0.01	1.81	8.60	1.60	5.14
30-20-55	1.28	25.16	150.11	21.19	66.82	0.25	1.35	9.84	1.37	4.05	0.01	1.73	7.43	1.37	4.58
30-20-75	2.11	41.43	759.81	68.39	148.43	0.29	0.71	1.47	0.23	1.15	0.01	1.01	4.84	0.90	2.67
30-20-95	833.99	543,019.01	2,533,608.00	747,962.39	2,186,440.80	8.82	514.61	2,780.71	491.24	1,556.91	1.40	1,880.75	7,363.47	1,643.47	5,563.42
35-15-35	8.41	306.11	1,717.94	242.49	775.25	0.24	5.62	27.82	5.64	17.71	0.09	12.26	53.11	11.26	35.16
35-15-55	4.33	21.13	75.69	11.95	42.47	0.32	0.97	4.10	0.54	1.71	0.01	0.38	1.56	0.31	0.97
35-15-75	5.18	68.23	621.83	74.17	183.19	0.35	1.09	3.30	0.48	1.99	0.07	1.66	6.85	1.39	4.63
35-15-95	6.61	1,454.00	19,171.48	3,057.43	6,949.08	3.75	152.66	693.83	130.20	406.25	0.44	126.32	822.98	132.03	386.36
40-07-75	4.53	59.37	377.06	51.21	159.00	0.26	0.64	2.46	0.27	1.02	0.01	0.57	3.29	0.58	1.64
40-09-95	6.18	417.00	5,017.56	610.28	1,490.31	1.65	58.30	398.04	58.47	148.19	0.27	16.87	99.82	16.26	48.26
40-10-65	0.84	17.87	115.06	15.88	52.73	0.37	0.79	1.12	0.21	1.03	0.01	0.14	0.71	0.13	0.38
50-10-65	2.52	24.56	84.64	16.34	64.04	0.67	1.42	2.06	0.34	1.88	0.01	0.12	0.58	0.09	0.28
50-10-75	22.79	1,352.41	24,507.34	3,085.42	4,404.50	1.70	299.78	1,595.95	311.53	965.27	0.19	38.58	195.59	40.51	120.84
50-10-95	9.97	89.36	1,059.59	91.95	200.20	1.98	22.36	145.65	25.65	73.16	0.14	9.38	39.38	7.72	25.55
Average	43.55	26,053.36	123,470.17	35,989.85	104,843.81	1.03	50.69	271.05	49.04	151.98	0.13	99.75	410.85	88.61	295.66
Median	2.52	41.43	377.06	51.21	148.43	0.26	0.97	3.44	0.50	1.86	0.01	1.01	4.84	0.90	2.67

Source: from the author (2015).

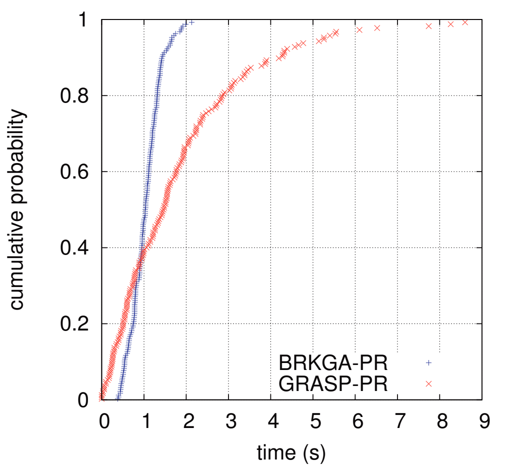
Even considering that the running time is over different hardware, the data from Table 4.9 shows that our version of GRASP-PR, as well as the BRKGA-PR approach has a significant lower running time in comparison with the results reported in Mateus et al. (2010). The average time reduction for each instance for the minimum time to solve is near to two orders of magnitude. This evidence that our proposed strategies has competitive performance since they are able to find the BKS in all runs in running times significantly shorter than the reported in Mateus et al. (2010). We also observe that our strategies are robust in the sense the algorithms were developed for a specific problem and they also perform well in a general application without any modification in the algorithm.

Figure 4.7 shows the Time-To-Target plots that show the running time distribution for BRKGA-PR and GRASP-PR to find a target solution defined as BKS. We choose the same instances used to show the time-to-target for GRASP-PR in Mateus et al. (2010). We observe that the behaviour, in general, is similar for BRKGA-PR and GRASP-PR, but the performance of one algorithm can be better than the other depending of the instance.

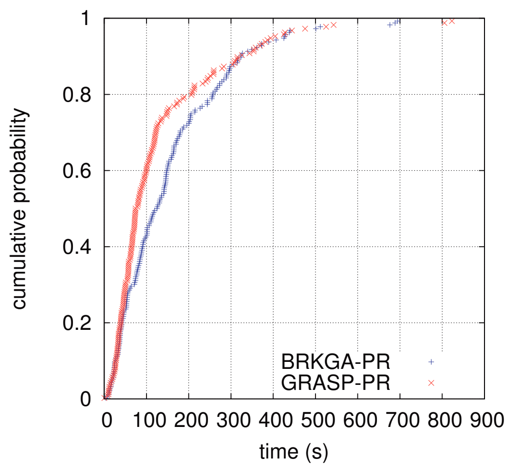
Figure 4.7 – Cumulative probability distribution for BRKGA-PR and GRASP-PR running times for instances `20-15-75`, `30-20-35`, `35-15-95`, and `40-09-95`



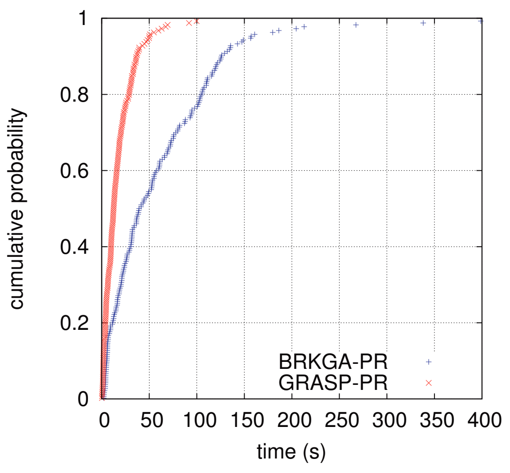
(a) TTT plot for `20-15-75`.



(b) TTT plot for `30-20-35`.



(c) TTT plot for `35-15-95`.



(d) TTT plot for `40-09-95`.

Source: from the author (2015).

4.5 Concluding remarks

In this chapter, we presented the problem of minimizing the cost of placement virtual machines across geo-separated data centers. A quadratic and two linear mathematical formulation was presented. Moreover, we present several heuristic for solving this problem. In the experiments, we evaluate the performance of CPLEX using the proposed mathematical formulations, and the proposed heuristic methods.

The results of CPLEX show that by adding the set of cuts the solver improves significantly the quality of the lower bounds for the LMVMPII model. We also observed that for this model, CPLEX can handle larger instances than considering LMVMP model, obtaining better lower bounds and feasible solutions in less computational time. However, as an exact method, CPLEX is limited to solve small instances, showing that heuristic methods are required on larger size instances.

Two metaheuristics approaches are used to solve the problem, the GRASP, and BRKGA. In both methods, we use a path-relinking procedure as an intensification approach. Also, we use a local search method that include many neighborhood strategies. Both strategies had similar performance, with the best performance achieved using local search `3V` and path-relinking `PRM`. A slight advantage for BRKGA was observed in small instances while GRASP has a slight advantage in larger instances. The same performance was observed when our strategies were applied to GQAP instances. In this case, our methods outperform the previous state-of-the-art results.

Finally, considering the high cost involved in this kind of problem and the difficulty to obtain feasible solutions when considering several constraints and limited resources, the proposed algorithms are good alternative to reduces costs, while maintaining the reliability and the demand requirements of data centers.

5 CONCLUDING REMARKS

Network structures are often used to describe different combinatorial optimization problems. In this thesis, we proposed algorithms to solve three optimization problems to control and optimize the flow in transportation and telecommunication networks. For transportation networks, the first problem considers to install tollbooths to minimize the average user travel time. The second problem considers maximizing the revenue collected in a subset of tolled arcs. The third problem, an application of telecommunication networks and cloud computing, considers minimizing the communication cost over the network.

Chapter 2 presented the *Tollbooth problem*. We formalized the problem mathematically and proposed two piecewise linear functions to approximate the convex cost function used to evaluate the congestion cost. Furthermore, we performed a large study with a previously biased random-key genetic algorithm, besides a extensive computational experiments were performed, including exact and heuristic methods to provide a vast analysis of the obtained solutions. For the exact methods, the approach based on the commercial solver CPLEX has lower performance, even for small instances. On the other hand, the biased random-key genetic algorithm shows a good performance with a good tradeoff between computational time and solution quality.

In Chapter 3 we investigated the *Stackelberg network pricing problem*. For this bilevel problem, we used CPLEX to solve a relaxed model to provide a good quality initial solution. However, the integer model is limited to solve small size instances. The biased random-key genetic algorithm shows good performance, even without a local search method.

Chapter 4 considered the problem of placement of virtual machines across geo-separated data centers (*VMPlacement problem*). To the best of our knowledge, we are the first to introduce this problem that is a generalization of the classic generalized quadratic assignment problem (GQAP). We formalized the problem mathematically using a quadratic model and two mixed integer linear models extended from the GQAP. Also, we proposed a local search method with an intensive exploration of the neighborhood, and a path-relinking as an intensification method was incorporated in two metaheuristic approaches. The first is a greedy randomized adaptive search procedure, and the second is a biased random-key genetic algorithm. An extensive set of experiments was performed to evaluate different configurations of the algorithm. We tested our algorithms on a set of synthetic instances where we show that the exact approach has poor performance, except for instances with up to 25 virtual machines. We also evaluate the proposed algorithms in a set of instances from the literature for GQAP, and our heuristic approach overcomes the state-of-the-art in the tested instances.

BRKGA was widely used during this research, and some considerations can be made about this technique. First, the algorithm is a very flexible approach, which can be easily adapted for many combinatorial optimization problem, requiring only specify how to encode and decode a solution. Also, the algorithm requires the configuration of a few parameters, which, in general, are easy to be configured and have a set of pre-defined values. Finally, as experienced in our tested problems and from many related works from literature, BRKGA has the capability to produce high-quality solutions for many combinatorial optimization problems.

Summarizing, our major contributions are:

- The proposition of a new problem of placement of virtual machines across geo-separated data centers (VMPlacement), with a formalization through mathematical models and the proposition of methods to solve it. Since this problem is a generalization of the classical generalized quadratic assignment problem, the algorithms developed for this problem can be extended to this family of related problems;
- A study of mathematical formulations for all described problems and an evaluation of the performance of a general purpose commercial solver as an exact method. In all problems, the solver has a low performance even for small instances of the problem, indicating that heuristic approaches are more suitable to solve these problems;
- The evaluation of biased random-key genetic algorithm (BRKGA) as an efficient tool for solving different kinds of combinatorial optimization problems, especially over network structures.

We also would like to report some ongoing studies and strategies that did not obtain significant results or are directions for future works, since they are not explored in depth in this work. One of the objective during this research was to develop hybridization methods between heuristic methods and mathematical programming methods, also named *matheuritics*. Approaches as fix-and-optimize (GINTNER et al., 2005), RINS (DANNA et al., 2005), and local-branching (FISCHETTI; LODI, 2003) are examples of some methods that have been successfully applied in many combinatorial optimization problems. In our research, we investigated methods to improve heuristic results using mathematical programming. Following we summarize some strategies that we develop for each problem and some empirical observations:

- For the problems discussed in this thesis, a common aspect is that CPLEX was able to solve the mathematical formulation only for small size instances. The models analysed in this thesis require a large number of variables, or have constraints that lead to a low relaxation quality. This is the main reason that leads to a low performance of CPLEX. With a higher number of variables, the solver tends to increase the number of nodes to explore. With a low relaxation quality, the solver tends to explore more nodes in the branch-and-bound tree since with a low relaxation quality, the bounds are worst, and fewer cuts based on bounds are applied. We tried to improve all these formulations, using alternative formulations with a lower number of variables, or reformulating them to improve the relaxation, or adding constraints to generate additional cuts. However when successful modifications were included, it was not enough to obtain a significant improvement in the performance of the solver. With a low performance of the solver, a low performance of hybrid mathematical programming methods also is expected, justifying the low performance of our developed approaches.
- For the tollbooth problem, we observed a higher number of variables and a low relaxation quality, mostly due to the utilization of the Big-M in the formulation. An additional difficult was the nonlinear objective function of the problem, which was approximated by piecewise linear functions used by the solver. For the computational results, we observed a very weak performance of the solver even for approximated results and, for this reason, no additional experiments with hybrid approach was performed.
- For the Stackelberg network pricing problem, we proposes a modification in the mathematical model to reduce the number of variables for the case where there are demands to the same destination. Given the values of tariffs for each tariffed arc, we calculate the first k shortest path for each commodity. Thus, we allow the demand to be sent only through these arcs or arcs that belongs to the untolled least cost paths. The flow variables for the remaining arcs are eliminated or fixed to zero. This processes is iterated while modifications in the values of tariffs are found. This was a promising technique since the number of variables was significantly reduced for small values of k and the model is solved, in general, in a few seconds. Naturally, small values of k tend to limit the search space, while higher values of k tend to increase the number of variables, and consequently the time spent by the solver. We expect

that this technique can be successful used as a local search strategy. However, to calculate the k shortest paths for each commodity and to build the mathematical model for medium and large size instances can take a significant amount of time. Thus, maybe applying this heavy local search strategy can be better suitable to a non-population-based algorithm.

- For the VMPlacement problem, both proposed linear mathematical models have a low relaxation quality for medium and large size instances. In the experiments, we do not find simple rules to eliminate variables with a good efficiency, i.e., remove or fix a subset of variables and keep a higher probability of finding the best solution. However, we try some alternatives. Knowing that the solver has a good performance for instances with up to 25 or 30 virtual machines, we randomly select n data centers such that the sum of placed virtual machines in theses data centers is near to this limit. The variables for this subset of virtual machines are kept in the model. The remaining variables are removed (or fixed to the current value of the current solution). An additional cut based on the *local branching* inequality (FISCHETTI; LODI, 2003) was used to limit the search space for a maximum number of changes on the current solution, reducing the time spent by the solver considerably. Improvements from the resolution of this restricted model correspond to change virtual machines between the data centers by exploring a large neighborhood search space. However, to solve this model may require a long time and, for this reason, the performance was worst that the best strategy reported. Also, for instances with a high number of available virtual machines, the number of variables in the model can still be significantly large.

Research on network problems has a high importance, since network structure are constantly changing by considering new network sizes or requirements. In our research, new strategies and algorithms were proposed to provide solutions and deal with problems and new scenarios of some network optimization problems.

REFERENCES

AIEX, R. M.; RESENDE, M. G. C.; RIBEIRO, C. C. TTT plots: A perl program to create time-to-target plots. **Optimization Letters**, Springer-Verlag, vol. 1, no. 4, p. 355-366, 2007.

ANBIL, R. et al. Recent Advances in Crew-Pairing Optimization at American Airlines. **Interfaces**, Institute for Operations Research and the Management Sciences (INFORMS), vol. 21, no. 1, 1991.

ANDRADE, C. E. **Evolutionary Algorithms for some Problems in Telecommunications**. 226 f. Thesis (Ph.D Thesis) — Universidade Estadual de Campinas, 2015.

ANDRADE, C. E. et al. Evolutionary algorithms for overlapping correlation clustering. In: CONFERENCE ON GENETIC AND EVOLUTIONARY COMPUTATION, 14, New York, 2014. **Proceedings...** New York: ACM Press, 2014. p. 405-412.

ANDRADE, C. E. et al. A biased random-key genetic algorithm for wireless backhaul network design. **Applied Soft Computing**, vol. 33, 2015.

ANDRADE, C. E. et al. Biased Random-Key Genetic Algorithms for the Winner Determination Problem in Combinatorial Auctions. **Evolutionary Computation**, vol. 23, no. 2, 2015.

ARMENTANO, V. A.; SHIGUEMOTO, A.; LØKKETANGEN, A. Tabu search with path relinking for an integrated production–distribution problem. **Computers & Operations Research**, vol. 38, no. 8, 2011.

BAI, L.; HEARN, D. W.; LAWPHONGPANICH, S. Decomposition techniques for the minimum toll revenue problem. **Networks**, Wiley Online Library, vol. 44, no. 2, 2004.

BAI, L.; HEARN, D. W.; LAWPHONGPANICH, S. A heuristic method for the minimum toll booth problem. **Journal of Global Optimization**, Springer US, vol. 48, no. 4, 2010.

BALLANI, H. et al. Towards predictable datacenter networks. **ACM SIGCOMM Computer Communication Review**, ACM Press, New York, vol. 41, no. 4, 2011.

BARNHART, C. et al. Branch-and-Price: Column Generation for Solving Huge Integer Programs. **Operations Research**, vol. 46, no. 3, p. 316-329, 1998.

BASAR, T.; SRIKANT, R. A Stackelberg Network Game with a Large Number of Followers. **Journal of Optimization Theory and Applications**, Springer Netherlands, vol. 115, no. 3, p. 479-490, 2002.

BASSEUR, M.; SEYNHAEVE, F.; TALBI, E.-G. Path Relinking in Pareto Multi-objective Genetic Algorithms. In: Coello Coello, C.; Hernández Aguirre, A.; ZITZLER, E. (Ed.). **Evolutionary Multi-Criterion Optimization SE - 9**. Springer Berlin Heidelberg, vol. 3410, p. 120-134. 2005.

BEAN, J. C. Genetic Algorithms and Random Keys for Sequencing and Optimization. **INFORMS Journal on Computing**, vol. 6, no. 2, p. 154-160, 1994.

BECKMANN, M. J.; MCGUIRE, C. B.; WINSTEN, C. B. **Studies in the economics of transportation**. Published for the Cowles Commission for Research in Economics by Yale University Press. 1956.

BLUM, C.; ROLI, A. Metaheuristics in combinatorial optimization. **ACM Computing Surveys**, ACM, New York, NY, USA, vol. 35, no. 3, 2003.

BOUHTOU, M. et al. Tariff Optimization in Networks. **INFORMS Journal on Computing**, vol. 19, no. 3, 2007.

BRACKEN, J.; MCGILL, J. T. Mathematical Programs with Optimization Problems in the Constraints. **Operations Research**, vol. 21, no. 1, 1973.

BROSTRÖM, P.; HOLMBERG, K. Multiobjective design of survivable IP networks. **Annals of Operations Research**, Springer Netherlands, vol. 147, no. 1, 2006.

BROTCORNE, L. et al. A Tabu search algorithm for the network pricing problem. **Computers & Operations Research**, vol. 39, no. 11, 2012.

BROTCORNE, L. et al. A Bilevel Model and Solution Algorithm for a Freight Tariff-Setting Problem. **Transportation Science**, vol. 34, no. 3, 2000.

Bureau of Public Roads. **Traffic Assignment Manual**. Washington, DC: US Department of Commerce, Urban Planning Division, 1964.

BURIOL, L. S. et al. A biased random-key genetic algorithm for road congestion minimization. **Optimization Letters**, Springer Berlin / Heidelberg, vol. 4, no. 4, 2010.

BURIOL, L. S.; RESENDE, M. G. C.; THORUP, M. Speeding Up Dynamic Shortest-Path Algorithms. **INFORMS Journal on Computing**, vol. 20, no. 2, 2008.

CANDLER, W.; NORTON, R. **Multilevel programming**. Washington D.C. 1977.

CASTELLI, L.; LABBÉ, M.; VIOLIN, A. A Network Pricing Formulation for the revenue maximization of European Air Navigation Service Providers. **Transportation Research Part C: Emerging Technologies**, Elsevier Ltd, vol. 33, 2013.

ČERNÝ, V. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. **Journal of Optimization Theory and Applications**, Kluwer Academic Publishers-Plenum Publishers, vol. 45, no. 1, p. 41-51, 1985.

COLSON, B.; MARCOTTE, P.; SAVARD, G. Bilevel programming: A survey. **4OR**, vol. 3, no. 2, 2005.

COLSON, B.; MARCOTTE, P.; SAVARD, G. An overview of bilevel optimization. **Annals of Operations Research**, vol. 153, no. 1, 2007.

CORDEAU, J. F. et al. A memetic heuristic for the generalized quadratic assignment problem. **INFORMS Journal on Computing**, vol. 18, no. 4, p. 433-443, 2006.

DANNA, E.; ROTHBERG, E.; PAPE, C. L. Exploring relaxation induced neighborhoods to improve MIP solutions. **Mathematical Programming**, Springer-Verlag, vol. 102, no. 1, 2005.

DAVIDSON, K. B. A flow travel time relationship for use in transportation planning. In: AUSTRALIAN ROAD RESEARCH BOARD CONFERENCE, Sydney, 1966. **Proceedings...** Sydney: Australian Road Research Board (ARRB), vol. 3, no. 1, 1966. p. 183-194.

DEMPE, S. **Foundations of bilevel programming**. Springer US, 309 p. 2002.

DEWEZ, S. **On the toll setting problem**. 176 f. Thesis (PhD) — Université Libre de Bruxelles, 2004.

DIAL, R. B. Minimal-revenue congestion pricing part I: A fast algorithm for the single-origin case. **Transportation Research Part B: Methodological**, vol. 33, no. 3, 1999.

DIAL, R. B. Minimal-revenue congestion pricing Part II: An efficient algorithm for the general case. **Transportation Research Part B: Methodological**, vol. 34, no. 8, 2000.

DORIGO, M.; MANIEZZO, V.; COLORNI, A. Ant system: Optimization by a colony of cooperating agents. **IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics**, vol. 26, no. 1, p. 29-41, 1996.

EKSTRÖM, J.; SUMALEE, A.; LO, H. K. Optimizing toll locations and levels using a mixed integer linear approximation approach. **Transportation Research Part B: Methodological**, vol. 46, no. 7, 2012.

FEO, T. A.; RESENDE, M. G. C. A probabilistic heuristic for a computationally difficult set covering problem. **Operations Research Letters**, vol. 8, no. 2, p. 67-71, 1989.

FEO, T. A.; RESENDE, M. G. C. Greedy Randomized Adaptive Search Procedures. **Journal of Global Optimization**, Kluwer Academic Publishers, vol. 6, no. 2, p. 109-133, 1995.

FESTA, P.; RESENDE, M. G. C. An annotated bibliography of GRASP - Part I: Algorithms. **International Transactions in Operational Research**, vol. 16, no. 1, 2009.

FESTA, P.; RESENDE, M. G. C. An annotated bibliography of GRASP-Part II: Applications. **International Transactions in Operational Research**, vol. 16, no. 2, 2009.

FESTA, P.; RESENDE, M. G. C. Hybridizations of GRASP with Path-Relinking. In: TALBI, E.-G. (Ed.). **Hybrid Metaheuristics**. Springer Berlin Heidelberg, p. 135-155. 2013, (Studies in Computational Intelligence, vol. 434).

FISCHER, A. et al. Virtual Network Embedding: A Survey. **IEEE Communications Surveys & Tutorials**, vol. 15, no. 4, 2013.

FISCHETTI, M.; LODI, A. Local branching. **Mathematical Programming**, Springer, vol. 98, no. 1-3, 2003.

FISCHETTI, M.; MONACI, M. Exploiting Erraticism in Search. **Operations Research**, vol. 62, no. 1, p. 114-122, 2014.

FORTZ, B.; THORUP, M. Increasing Internet Capacity Using Local Search. **Computational Optimization and Applications**, vol. 29, no. 1, 2004.

FRIEZE, A.; YADEGAR, J. On the quadratic assignment problem. **Discrete Applied Mathematics**, vol. 5, no. 1, 1983.

GAREY, M. R.; JOHNSON, D. S. **Computers and Intractability: A Guide to the Theory of NP-Completeness**. New York, NY, USA: W. H. Freeman & Co. 1979.

GENDREAU, M.; POTVIN, J.-Y. **Handbook of Metaheuristics**. Springer US, 648 p. 2010.

GINTNER, V.; KLIEWER, N.; SUHL, L. Solving large multiple-depot multiple-vehicle-type bus scheduling problems in practice. **OR Spectrum**, vol. 27, no. 4, 2005.

GLOVER, F. Future paths for integer programming and links to artificial intelligence. **Computers & Operations Research**, vol. 13, no. 5, p. 533-549, 1986.

GLOVER, F. Tabu Search—Part I. **ORSA Journal on Computing**, vol. 1, no. 3, p. 190-206, 1989.

GLOVER, F. Tabu Search and Adaptive Memory Programming - Advances, Applications and Challenges. In: BARR, R.; HELGASON, R.; KENNINGTON, J. (Ed.). **Interfaces in Computer Science and Operations Research**. Springer US, p. 1-75. 1997, (Operations Research/Computer Science Interfaces Series, vol. 7).

GLOVER, F. Exterior Path Relinking for Zero-One Optimization. **International Journal of Applied Metaheuristic Computing**, IGI Global, vol. 5, no. 3, 2014.

GLOVER, F.; LAGUNA, M. Tabu Search. In: REEVES, C. R. (Ed.). **Modern Heuristic Techniques for Combinatorial Problems**. Oxford, England: Blackwell Scientific Publishing, p. 70-150. 1993.

GLOVER, F.; LAGUNA, M.; MARTÍ, R. Fundamentals of scatter search and path relinking. **Control and Cybernetics**, Vol. 29, n, no. 3, p. 653-684, 2000.

GLOVER, F.; LAGUNA, M.; MARTÍ, R. Scatter Search and Path Relinking: Advances and Applications. In: GLOVER, F.; KOCHENBERGER, G. (Ed.). **Handbook of Metaheuristics**. Springer US, p. 1-35. 2003, (International Series in Operations Research & Management Science, vol. 57).

GOLDBERG, D. E. **Genetic algorithms in search, optimization, and machine learning**. 1. ed. [S.l.]: Addison-Wesley Publishing Company, 412 p. 1989.

GOMORY, R. E. Outline of an algorithm for integer solutions to linear programs. **Bulletin of the American Mathematical Society**, vol. 64, no. 5, 1958.

GONÇALVES, J. F.; RESENDE, M. G. C. Biased random-key genetic algorithms for combinatorial optimization. **Journal of Heuristics**, vol. 17, no. 5, 2011.

GONÇALVES, J. F.; RESENDE, M. G. C. A biased random key genetic algorithm for 2D and 3D bin packing problems. **International Journal of Production Economics**, vol. 145, p. 500-510, 2013.

GONÇALVES, J. F.; RESENDE, M. G. C.; TOSO, R. F. **Biased and unbiased random-key genetic algorithms: An experimental analysis**. Florham Park, New Jersey, 12 p. 2012.

GREENBERG, A. et al. VL2: A Scalable and Flexible Data Center Network. **ACM SIGCOMM Computer Communication Review**, ACM, New York, NY, USA, vol. 39, no. 4, p. 51, 2009.

GUO, C. et al. SecondNet: A Data Center Network Virtualization Architecture with Bandwidth Guarantees. In: INTERNATIONAL CONFERENCE ON - CO-NEXT, 10, New York, 2010. **Proceedings...** New York: ACM, 2010. p. 12.

HANSEN, P.; MLADENOVIĆ, N.; Moreno Pérez, J. A. Variable neighbourhood search: Methods and applications. **Annals of Operations Research**, vol. 175, no. 1, p. 367-407, 2010.

HEARN, D. W.; RAMANA, M. V. Solving congestion toll pricing models. **Equilibrium and Advanced Transportation Modelling**, Kluwer Academic, p. 109-124, 1998.

HEARN, D. W.; YILDRIM, M. B. A toll pricing framework for pricing assignment problems and elastic demands. In: GENDREAU, M.; MARCOTTE, P. (Ed.). **Transportation and network analysis: current trends**. Springer. 2002, (Applied Optimization, vol. 63).

HEILPORN, G. et al. A polyhedral study of the network pricing problem with connected toll arcs. **Networks**, Wiley Subscription Services, Inc., A Wiley Company, vol. 55, no. 3, p. 234-246, 2010.

HOESEL, S. van. An overview of Stackelberg pricing in networks. **European Journal of Operational Research**, vol. 189, no. 3, 2008.

HOOS, H. H.; STÜTZLE, T. Evaluating Las Vegas Algorithms: Pitfalls and Remedies. In: CONFERENCE ON UNCERTAINTY IN ARTIFICIAL INTELLIGENCE, 14, San Francisco, 1998. **Proceedings...** San Francisco: Morgan Kaufmann Pub., 1998. p. 238-245.

HOOS, H. H.; STÜTZLE, T. **On the empirical evaluation of Las Vegas algorithms**. [S.l.], 7 p. 1998.

KAUFMAN, L.; BROECKX, F. An algorithm for the quadratic assignment problem using Bender's decomposition. **European Journal of Operational Research**, vol. 2, no. 3, 1978.

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by Simulated Annealing. **Science**, vol. 220, no. 4598, p. pp. 671-680, 1983.

KOOPMANS, T. C.; BECKMANN, M. J. Assignment problems and the location of economic activities. **Econometrica**, vol. 25, no. 1, p. 53-76, 1957.

LABBÉ, M.; MARCOTTE, P.; SAVARD, G. A Bilevel Model of Taxation and Its Application to Optimal Highway Pricing. **Management Science**, vol. 44, no. 12-Part-1, 1998.

LAGUNA, M.; MARTÍ, R. GRASP and Path Relinking for 2-Layer Straight Line Crossing Minimization. **INFORMS Journal on Computing**, vol. 11, no. 1, p. 44-52, 1999.

LAWLER, E. L.; WOOD, D. E. Branch-and-Bound Methods: A Survey. **Operations Research**, vol. 14, no. 4, 1966.

LEE, C. G.; MA, Z. **The generalized quadratic assignment problem**. Toronto, Ontario, M5S 3G8, Canada, 20 p. 2004.

MARTÍ, R.; LAGUNA, M.; GLOVER, F. Principles of scatter search. **European Journal of Operational Research**, vol. 169, no. 2, 2006.

MARTÍ, R.; RESENDE, M. G. C.; RIBEIRO, C. C. Multi-start methods for combinatorial optimization. **European Journal of Operational Research**, vol. 226, no. 1, p. 1-8, 2013.

MARTINEZ, C. et al. BRKGA Algorithm for the Capacitated Arc Routing Problem. **Electronic Notes in Theoretical Computer Science**, vol. 281, 2011.

MATEUS, G. R.; RESENDE, M. G. C.; SILVA, R. M. A. GRASP with path-relinking for the generalized quadratic assignment problem. **Journal of Heuristics**, vol. 17, no. 5, 2010.

MITTELMANN, H.; SALVAGNIN, D. On solving a hard quadratic 3-dimensional assignment problem. **Mathematical Programming Computation**, Springer Berlin Heidelberg, p. 1-16, 2015.

MLADENOVIĆ, N.; HANSEN, P. Variable neighborhood search. **Computers & Operations Research**, vol. 24, no. 11, p. 1097-1100, 1997.

OLIVEIRA, C. A. S.; PARDALOS, P. M.; RESENDE, M. G. C. GRASP with Path-Relinking for the Quadratic Assignment Problem. In: RIBEIRO, C. C.; MARTINS, S. L. (Ed.). **Experimental and Efficient Algorithms**. Springer Berlin Heidelberg, p. 356-368. 2004, (Lecture Notes in Computer Science, vol. 3059).

PÉREZ, M. P.; RODRÍGUEZ, F. A.; MORENO-VEGA, J. M. A hybrid VNS–path relinking for the p-hub median problem. **IMA Journal of Management Mathematics**, vol. 18, no. 2, 2007.

PESSOA, A. A. et al. Algorithms for the generalized quadratic assignment problem combining Lagrangean decomposition and the Reformulation-Linearization Technique. **European Journal of Operational Research**, Elsevier B.V., vol. 206, no. 1, 2010.

POP, P. C. **Generalized network design problems: modeling and optimization**. Berlin: De Gruyter, 203 p. 2012.

RESENDE, M. G. C. Biased random-key genetic algorithms with applications in telecommunications. **TOP**, vol. 20, no. 1, 2012.

RESENDE, M. G. C. Introdução aos algoritmos genéticos de chaves aleatórias viciadas. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, 13, Natal - RN, 2013. **Anais...** Natal - RN: Sociedade Brasileira de Pesquisa Operacional, 2013. p. 12.

RESENDE, M. G. C.; RIBEIRO, C. C. GRASP with Path-Relinking: Recent Advances and Applications. In: **Metaheuristics: Progress as Real Problem Solvers**. New York: Springer-Verlag, vol. 1, p. 29-63. 2005.

RESENDE, M. G. C.; RIBEIRO, C. C. Greedy Randomized Adaptive Search Procedures: Advances, Hybridizations, and Applications. In: GENDREAU, M.; POTVIN, J.-Y. (Ed.). **Handbook of Metaheuristics**. Springer US, p. 283-319. 2010, (International Series in Operations Research & Management Science, vol. 146).

RESENDE, M. G. C.; RIBEIRO, C. C. GRASP: Greedy Randomized Adaptive Search Procedures. In: BURKE, E. K.; KENDALL, G. (Ed.). **Search Methodologies**. Springer US, p. 287-312. 2014.

RESENDE, M. G. C. et al. Scatter search and path-relinking: Fundamentals, advances, and applications. In: **Handbook of Metaheuristics**. [s.n.], p. 87-107. 2010.

ROCH, S.; SAVARD, G.; MARCOTTE, P. An approximation algorithm for Stackelberg network pricing. **Networks**, vol. 46, no. 1, 2005.

RUIZ, E. et al. A biased random-key genetic algorithm for the capacitated minimum spanning tree problem. **Computers & Operations Research**, vol. 57, 2015.

SCHRANK, D.; LOMAX, T.; EISELE, B. **2011 Urban Mobility Report**. [S.l.]. 2011.

STACKELBERG, H. V. **Marktform und Gleichgewicht**. Berlin, Germany: Julius Springer, Vienna, Austria. 1934.

STACKELBERG, H. von. **The theory of the market economy**. Oxford, England: Oxford University Press. 1952.

STEFANELLO, F. et al. A Biased Random-key Genetic Algorithm for Placement of Virtual Machines across Geo-Separated Data Centers. In: CONFERENCE ON GENETIC AND EVOLUTIONARY COMPUTATION, 15, Madrid, 2015. **Proceedings...** Madrid: ACM, 2015. p. 1-8.

STEFANELLO, F. et al. A New Linear Model for Placement of Virtual Machines across Geo-Separated Data Centers. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, 47, Porto de Galinhas, PE, 2015. **Anais...** Porto de Galinhas, PE: Sociedade Brasileira de Pesquisa Operacional, 2015. p. 1-11.

STEFANELLO, F. et al. On the minimization of traffic congestion in road networks with tolls. **Annals of Operations Research**, 2015.

STEFANELLO, F.; BURIOL, L. S.; RESENDE, M. G. C. A biased random-key genetic algorithm for a network pricing problem. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, 45, Natal - RN, 2013. **Anais...** Natal - RN: Sociedade Brasileira de Pesquisa Operacional, 2013. p. 12.

STEFANELLO, F. et al. Routing in Road Networks: the toll booth problem. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, 44, Rio de Janeiro - RJ, 2012. **Anais...** Rio de Janeiro - RJ: Sociedade Brasileira de Pesquisa Operacional, 2012. p. 12.

TALBI, E.-G. **Metaheuristics: From Design to Implementation**. [S.l.]: Wiley Publishing. 2009.

TALBI, E.-G. (Ed.). **Metaheuristics for Bi-level Optimization**. Berlin, Heidelberg: Springer Berlin Heidelberg, 288 p. (Studies in Computational Intelligence, vol. 482). 2013.

TOSO, R. F.; RESENDE, M. G. C. A C++ application programming interface for biased random-key genetic algorithms. **Optimization Methods and Software**, vol. 30, no. 1, p. 1-15, 2014.

TSEKERIS, T.; VOSS, S. Design and evaluation of road pricing: state-of-the-art and methodological advances. **NETNOMICS: Economic Research and Electronic Networking**, Springer Netherlands, vol. 10, no. 1, 2008.

VALLADA, E.; RUIZ, R. Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. **Omega**, vol. 38, no. 1-2, 2010.

VAZIRANI, V. V. **Approximation Algorithms**. Berlin, Heidelberg: Springer Berlin Heidelberg, 378 p. 2003.

WARDROP, J. G. Some theoretical aspects of road traffic research. **Proceedings of the Institution of Civil Engineers, Part II**, Thomas Telford Ltd., vol. 1, no. 36, p. 325-378, 1952.

WEN, W. A dynamic and automatic traffic light control expert system for solving the road congestion problem. **Expert Systems with Applications**, vol. 34, no. 4, 2008.

WILLIAMSON, D. P.; SHMOYS, D. B. **The Design of Approximation Algorithms**. 1st. ed. Cambridge University Press, 504 p. 2011.

XIE, D.; HU, Y. C. The Only Constant is Change: Incorporating Time-Varying Network Reservations in Data Centers. In: SIGCOMM, 12, New York, 2012. **Proceedings...** New York: ACM, (SIGCOMM '12), 2012. p. 199-210.

YANG, H.; ZHANG, X. Optimal Toll Design in Second-Best Link-Based Congestion Pricing. **Transportation Research Record**, vol. 1857, no. 1, 2003.

ZHANG, G.; LAI, K. Combining path relinking and genetic algorithms for the multiple-level warehouse layout problem. **European Journal of Operational Research**, vol. 169, no. 2, 2006.