Universidade Estadual de Campinas
Instituto de Computação

Rafael Kendy Arakaki

Algorithms and mathematical formulations for arc routing problems

Algoritmos e formulações matemáticas para problemas de roteamento em arcos

CAMPINAS
2020

Rafael Kendy Arakaki

Algorithms and mathematical formulations for arc routing problems

Algoritmos e formulações matemáticas para problemas de roteamento em arcos

Tese apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação.

Dissertation presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Doctor in Computer Science.

Supervisor/Orientador: Prof. Dr. Fábio Luiz Usberti

Este exemplar corresponde à versão final da Tese defendida por Rafael Kendy Arakaki e orientada pelo Prof. Dr. Fábio Luiz Usberti.

CAMPINAS
2020

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

Ar12a

Arakaki, Rafael Kendy, 1993-
Algorithms and mathematical formulations for arc routing problems / Rafael Kendy Arakaki. – Campinas, SP : [s.n.], 2020.

Orientador: Fábio Luiz Usberti.
Tese (doutorado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Problema de roteamento em arcos. 2. Programação linear inteira. 3. Meta-heurística. 4. Otimização combinatória. 5. Pesquisa operacional. I. Usberti, Fábio Luiz, 1982-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Algoritmos e formulações matemáticas para problemas de roteamento em arcos
Palavras-chave em inglês:
Arc routing problems
Integer linear programming
Metaheuristic
Combinatorial optimization
Operations research
Área de concentração: Ciência da Computação
Titulação: Doutor em Ciência da Computação
Banca examinadora:
Fábio Luiz Usberti [Orientador]
Franklina Maria Bragion de Toledo
Mário César San Felice
Christiano Lyra Filho
Paulo Morelato França
Data de defesa: 05-03-2020
Programa de Pós-Graduação: Ciência da Computação

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0002-5931-7710
- Currículo Lattes do autor: http://lattes.cnpq.br/0035579465475655

Universidade Estadual de Campinas
Instituto de Computação

Rafael Kendy Arakaki

Algorithms and mathematical formulations for arc routing problems

Algoritmos e formulações matemáticas para problemas de roteamento em arcos

Banca Examinadora:

- Prof. Dr. Fábio Luiz Usberti
  IC/UNICAMP
- Profa. Dra. Franklina Maria Bragion de Toledo
  ICMC/USP
- Prof. Dr. Mário César San Felice
  DC/UFSCar
- Prof. Dr. Paulo Morelato França
  FEEC/UNICAMP
- Prof. Dr. Christiano Lyra Filho
  FEEC/UNICAMP

A ata da defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Campinas, 05 de março de 2020

Em toda parte, só se aprende com quem se gosta.

(Johann Wolfgang von Goethe)

Agradecimentos

Agradeço a Deus pela vida. Agradeço às pessoas que apoiaram para que esta tese pudesse ser escrita. Agradeço aos meus pais, Paulo e Elaine, às minhas tias, Amélia e Alice (*in memoriam*), aos meus irmãos, Paulo e Matheus, à minha avó Regina, aos demais membros da família e aos meus amigos de Campo Grande e de Campinas. Agradeço ao meu orientador por toda a mentoria e exemplo tanto acadêmico quanto pessoal. Agradeço aos professores e funcionários do IC que permitem funcionar esse ambiente de excelência em aprendizagem e pesquisa.

Agradecimento às agências de fomento. Agradeço pelo financiamento desta tese à Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), processo nº 2016/00315-0, e à CAPES. O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001. As opiniões, hipóteses e conclusões ou recomendações expressas neste material são de responsabilidade do(s) autor(es) e não necessariamente refletem a visão da FAPESP.

Resumo

Problemas de roteamento em arcos têm por objetivo determinar rotas de custo mínimo que visitam um subconjunto de arcos de um grafo, com uma ou mais restrições adicionais. Esta tese estuda três problemas NP-difíceis de roteamento em arcos: (1) o problema de roteamento em arcos capacitado (CARP); (2) o problema de roteamento em arcos capacitado e aberto (OCARP); e (3) o problema do carteiro chinês com cobertura (CCPP). Apresentamos formulações matemáticas e métodos exatos e heurísticos para tratar computacionalmente esses problemas: (i) uma heurística construtiva gulosa e randomizada é proposta para o CARP; (ii) uma metaheurística de algoritmos genéticos híbrido e dois métodos de limitantes inferiores por programação linear inteira, um branch-and-cut e um baseado em redes de fluxos, são propostos para o OCARP; e (iii) um método exato branch-and-cut com desigualdades válidas e uma heurística construtiva são propostos para o CCPP.

Extensivos experimentos computacionais utilizando instâncias de benchmark foram executados para demonstrar o desempenho dos métodos propostos em relação aos métodos da literatura, considerando tanto a qualidade das soluções obtidas quanto o tempo de processamento. Nossos resultados mostram que os métodos propostos são estado da arte.

Os problemas estudados apresentam aplicações práticas relevantes: o CARP tem aplicações em coleta de lixo urbano e remoção de neve de estradas; o OCARP tem aplicações em roteamento de leituristas e na definição de caminhos de corte em chapas metálicas; e o CCPP tem aplicações em roteamento de leituristas com o uso de tecnologia wireless. A solução desses problemas remete à diminuição de custos logísticos, melhorando a competitividade das empresas.

Abstract

Arc routing problems aim to find minimum cost routes that visit a subset of arcs of a graph, with one or more side constraints. This thesis studies three NP-hard arc routing problems: (1) the capacitated arc routing problem (CARP); (2) the open capacitated arc routing problem (OCARP); and (3) the covering Chinese postman problem (CCPP). We present mathematical formulations and heuristic and exact methods to computationally solve these problems: (i) a greedy and randomized constructive heuristic is proposed for the CARP; (ii) a hybrid genetic algorithm metaheuristic and two linear integer programming lower bound methods, one based on branch-and-cut and one based on flow networks, are proposed for the OCARP; and (iii) an exact branch-and-cut method with valid inequalities and a constructive heuristic are proposed for the CCPP.

Extensive computational experiments using benchmark instances were performed to demonstrate the performance of the proposed methods in comparison to the previous methods, regarding both quality of solutions and processing time. Our results show that the proposed methods are state-of-the-art.

The studied problems have many relevant practical applications: the CARP has applications on urban waste collection and snow removal; the OCARP has applications on the routing of meter readers and the cutting of metal sheets; and last, the CCPP has applications on automated meter readers routing. The solution of these problems leads to the reduction of logistics costs, improving businesses competitiveness.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Arc routing problems

The father of arc routing is considered to be the famous Swiss mathematician Leonhard Euler with his proposal of the Königsberg bridges problem in the 18th century. Afterwards, a good number of 19th century mathematicians became interested in arc routing problems, more specifically in graph traversing feasibility problems [25].

However, the study of modern arc routing, including algorithms and the use of computers, started in the 1960s with the study of the Chinese postman problem [30]. This section presents a description and a literature review of the most important arc routing problems in the context of this thesis.

1.1.1 Chinese postman problem (CPP)

The famous Königsberg bridges problem considers the question "Given an undirected and connected graph, is there a closed walk traversing all edges exactly once each?". And then the options are true or false, there is no optimization in it. Meigu Guan (Mei-Ko-Kwan) introduced the optimization to the problem. The first paper in English describing it was published in 1962 [30] where the problem is stated as: "A mailman has to cover his assigned segment before returning to the post office. The problem is to find the shortest walking distance for the mailman". This problem is widely known as the Chinese postman problem (CPP).

A more formal description of the Chinese postman problem is given next. Given an undirected graph $G(V, E)$ with non-negative costs on the edges, the objective is to find a minimum cost tour such that all the edges are visited at least once.

Eiselt et al. [24] presents an efficient polynomial algorithm for the CPP composed of two phases. The first phase determines a minimum cost set of edges that need to be duplicated so the graph becomes Eulerian (i.e. all nodes having even degree). The second phase is to construct a tour in this augmented graph visiting each edge once, which is an easy task for Eulerian graphs. Edmonds and Johnson [23] described an efficient polynomial algorithm for the first phase by solving a minimum cost matching problem.

Although the CPP is an easy problem, there is still ongoing research involving the CPP with a wide variation of side constraints. Papadimitriou [34] shows that if the

graph is mixed (containing both directed and undirected edges) solving the first phase is an NP-hard problem. Corberán et al. [21] presented a survey on the Windy CPP, a generalization of the mixed CPP where the cost of traversing an undirected edge can be different depending on the direction the edge is traversed. A recent survey of CPP variations, algorithms and complexity studies can be found in [43].

1.1.2 Rural postman problem (RPP)

The rural postman problem (RPP) [33] is a generalization of the CPP where given an undirected graph $G(V, E)$ just a subset of edges $E_R \subseteq E$ needs to be visited. In other words, the objective is to find a minimum cost tour such that each edge $e \in E_R$ is visited at least once. The problem is NP-hard, unless the graph induced by E_R is connected.

Ghiani and Laporte [27] survey the RPP, while Mourão et al. [22] provides an updated bibliography for the most recent papers addressing the problem.

1.1.3 Capacitated arc routing (CARP)

The capacitated arc routing problem [28] (CARP) is defined on an undirected graph $G(V, E)$. Each edge $(i, j) \in E$ has a cost $c_{ij} \geq 0$ and a demand $d_{ij} \geq 0$ associated to it. If an edge (i, j) has positive demand $d_{ij} > 0$ then it is called a required edge. Let $E_R \subseteq E$ be the set of required edges. A fleet of M identical vehicles is available, each with capacity D, starting from a special node (v_0) called depot. While traversing the graph, a vehicle might (i) service an edge, which deducts its capacity by d_{ij} and increases the solution cost by c_{ij} or (ii) deadhead an edge, which only increases the solution cost by c_{ij}. A vehicle route is defined by a tour starting and ending at the depot.

The literature contains numerous real-world applications of the CARP, for instance, waste collection, street sweeping, snow plowing, etc. An interested reader can found more details of CARP applications in [10, 22].

Exact and lower bound methods

Golden and Wong [28] presented the first mathematical formulation for the CARP. This formulation contains an exponential number of variables and constraints, and probably because of its implementation complexity, it has not been used to solve the problem.

Belenguer and Benavent [11] presented a two-index formulation for the CARP with a polynomial number of variables. This formulation was used in a branch-and-cut algorithm that the authors proposed a few years later [15]. The authors noted this method as effective only for small and medium instances where a small number of vehicles was employed.

Belenguer and Benavent [15] also proposed the so called one-index formulation. In this formulation, just one set of variables $x_e \in \mathbb{Z}_{\geq 0}, \forall e \in E$ is used to represent the aggregated deadheading (not servicing) number of visits performed by all vehicles. A downside of this formulation is that its solutions are not necessarily feasible for the CARP. Nonetheless, the lower bounds obtained are valid. In other words, it is a relaxed formulation for the CARP.

A cutting plane algorithm for the one-index formulation was developed by Belenguer and Benavent [5]. This method was able to compute very good lower bounds for the problem in a reasonably small processing time, improving substantially the best results in comparison to previous methods.

Martinetlli et al. [20] presented a branch-and-cut lower bounding method for the CARP, also using the one-index formulation. Their method employed exact separation procedures for the valid inequalities previously presented by Belenguer and Benavent [5]. Additionaly, they developed an dual-ascent heuristic aiming a more efficient method for separating the *capacity constraints*. This method obtained quite impressive results for very large benchmark instances.

Many other exact and lower bounding methods were proposed for the CARP, based on integer programming, combinatorial methods, etc. An interested reader can find a survey on exact and lower bounding methods for CARP in [13].

Heuristic methods

Many heuristics methods have been proposed for the CARP. In general, heuristic methods for the CARP can be classified in two categories: (i) constructive heuristics; and (ii) metaheuristic methods. Constructive heuristics start building a solution from scratch and progressively include elements into the solution until feasibility is obtained. Metaheuristics are methods that aim for sampling a good set of solutions from a solution space that is too big to be completely sampled. Metaheuristics may employ constructive heuristics, local search, stochastic optimization and population-based evolutionary methods in order to obtain a good trade-off between solutions quality and the computational effort to find the solutions.

The first constructive heuristic proposed for CARP was the *construct-strike* presented by Golden and Wong in the seminal paper [28]. Other remarkable constructive heuristics are *path-scanning* and *augment-merge* [29], *Improved merge* [14] and *path-scanning with ellipse rule* [38].

Regarding the metaheuristics, many have been proposed for the CARP and they are currently the state-of-the-art methods for finding good enough (i.e. not necessarily optimal) solutions for the CARP in a limited processing time.

In 2000, Hertz et al. [31] published the CARPET, a tabu search method for the CARP with three local search operators. Polacek et al. [36] published a variable neighborhood search (VNS) method with only two kinds of moves. This method obtained good solutions while spending relatively short processing times.

Usberti et al. [41] proposed a greedy randomized adaptive search procedure (GRASP) method which obtained excellent results at the cost of a high processing time. In their metaheuristic, the solutions are constructed using a procedure adapted from the PS-Ellipse heuristic from Santos et al. [38], and then improved by a set of specialized local search moves including some path-relinking moves able to explore the unfeasible regions of the solution space.

Lacomme et al. [16] proposed a good performing genetic algorithm (GA) with local searches for the CARP. In this GA, the solutions are represented by a simple permutation

of required edges, and a special shortest path algorithm is employed to transform each permutation in a CARP solution. This method improved many best known solution costs for benchmark instances in its publication date.

Many others metaheuristics have been proposed for the CARP. We refer the interested reader for the survey [37].

1.1.4 Open capacitated arc routing (OCARP)

The open capacitated arc routing problem (OCARP) was proposed by Usberti et al. [40]. This problem is a variation of the CARP where no depot is considered and the routes may start and end in any node. In other words, the routes may be open or closed walks in the graph. In comparison to the CARP, the literature of OCARP is quite modest.

In the seminal paper [40] the authors proposed an integer linear programming (ILP) formulation for the OCARP, a set of benchmark instances and obtained the first upper and lower bounds for the problem. In a sequence work, Usberti [28] proposed a branch-and-bound algorithm with improved results in comparison to the ILP formulation regarding solutions costs and lower bounds. This algorithm formulated the OCARP as a special case of the Capacity and Degree Constrained Minimum Spanning Forest Problem (CD-CMSFP). Both the ILP formulation method and the branch-and-bound algorithm were able to solve only small sized instances ($|E_R| \leq 50$) and a few of medium sized instances ($50 \leq |E_R| \leq 100$).

In relation to heuristic methods, Usberti et al. [40] proposed a reactive path-scanning with ellipse rule (RPS) for the OCARP. This heuristic consisted of a bias-randomized greedy heuristic where parameters were adjusted for each instance. The RPS was effective in its objective of generating non-trivial feasible solutions with little computational effort. One year later, Usberti [28] presented a metaheuristic method of greedy randomized adaptive search procedure (GRASP) with path-relinking (PR) that largely improved the RPS results. However, the author noted that in comparison to the CARP literature, the methods for OCARP could still improve more: "The OCARP average gap is 3.94% using the GRASP with PR upper bounds and the branch-and-bound lower bounds. This is relatively high comparing to the CARP average gap of 0.79%. Therefore, a large room for improvements is still available, leaving for future researches to focus on the design of algorithms that can tighten the bounds here introduced".

1.2 Contributions

The main contributions of this thesis are:

- A constructive heuristic for the CARP, inspired by the work of Santos et al. [38].
- A hybrid genetic algorithm (HGA) metaheuristic for the OCARP, which obtained substantially better results than previous works.
- A feasibilization procedure, employed by the HGA, with potential to be applied to other routing problems with limited fleet and limited capacities.

- A one-index relaxed formulation for the OCARP, inspired by a formulation originally designed for the CARP [15]. In addition, we propose a branch-and-cut method for solving it.
- A parameterized flow-based relaxed formulation $RFB(k)$ for the OCARP.
- A graph augmentation procedure, employed by the $RFB(k)$, that is able to control the flow-based formulation tightness. As far as we know this approach is novel in the literature of mathematical formulations for routing problems and we believe it has potential to be applied to other problems.
- We present a new arc routing problem called the covering Chinese postman problem (CCPP), inspired by applications on the routing of automated meter readers.
- We model the CCPP by integer programming and we propose a branch-and-cut method and a constructive heuristic for it.

1.3 Structure of the thesis

This thesis is a compilation of research papers written by the author in collaboration with other researchers during the doctoral program. Chapter 2 presents a published paper about a constructive heuristic called PS-Efficiency designed for the capacitated arc routing problem (CARP). Chapter 3 shows a published paper describing a hybrid genetic algorithm (HGA) for the open capacitated arc routing problem (OCARP). Chapter 4 contains a preprint paper describing two lower bounding procedures based on relaxed mathematical formulations for the open capacitated arc routing problem (OCARP). Chapter 5 consists of a paper presented in a conference where a new arc routing problem is proposed, called covering Chinese postman problem. In addition, a branch-and-cut method and a constructive heuristic are proposed for solving it. Chapter 6 contains a discussion summarizing and making relations between the papers described in Chapters 2 to 5. Chapter 7 contains the final comments.

Chapter 2

An efficiency-based path-scanning heuristic for the capacitated arc routing problem

---

The paper presented next is a full article published in the Computers and Operations Research in 2019 and it is co-authored with Fábio Luiz Usberti [6] (DOI: `https://doi.org/10.1016/j.cor.2018.11.018`). The original publication is available at `www.sciencedirect.com` and the copyright is owned by Elsevier Ltd. In this text we present an improved constructive heuristic with a greedy rule called PS-efficiency. This heuristic was based on previous path-scanning heuristics which are well known in the literature of arc routing problems. Computational experiments show the improved performance of PS-efficiency regarding solution quality and parameterization robustness.

The capacitated arc routing problem (CARP) is an important combinatorial optimization problem that has been extensively studied in the last decades. The objective is to optimize routes that service demands located on the edges of a graph, given a fleet of homogeneous vehicles with limited capacity that starts and ends its routes at a specific node (depot). This work proposes a new path-scanning heuristic for the CARP which introduces the concept of efficiency rule. Given the current vehicle location, its traversed distance and the amount of serviced demand, the efficiency rule selects the most promising edges to service next. Computational experiments conducted on a set of benchmark instances reveal that the proposed heuristic substantially outperformed all previous path-scanning heuristics from literature.

2.1 Introduction

Given scattered demands in the edges of a network that must be serviced by a fleet of identical vehicles with limited capacity all gathered in a central depot. How to service these demands within the shortest distance and without exhausting the vehicles capacities? This question is addressed by the *capacitated arc routing problem* (CARP) proposed

by Golden and Wong [18] in 1981. Since then CARP attracted the attention of many researchers mainly due to its optimization difficulty and numerous applications.

The CARP has been extensively studied and we refer the reader to references [24, 29, 4] for a comprehensive survey. Applications of CARP include but are not restricted to waste collection [17], meter reading [14] and snow plows [23]. Moreover, many real-world complex operations encouraged the emergence of CARP variants, where additional constraints and special characteristics are modeled. Examples of CARP variants include *CARP with time windows* [39, 1], *periodic CARP* [41], *open CARP* [37, 2], and others [25]. The improvement in methods for CARP can often be extended to these variants.

CARP is an NP-hard problem for which many exact and heuristic methodologies have been proposed [24]. Optimal solutions have been reported only for relatively small instances involving around 100 edges with positive demand [7], while the best results for larger instances are provided by heuristic methods [11].

Constructive heuristics start building a solution from scratch and progressively include elements into the solution until feasibility is obtained. The first constructive heuristic proposed for CARP was the *construct-strike* presented by Golden and Wong in the seminal paper [18]. Two years later, Golden et al. [19] proposed two other constructive heuristics called *path-scanning* and *augment-merge*. Since then, constructive heuristics with better performance have been proposed, such as *improved merge* [3] and *path-scanning with ellipse rule* [31].

Our contribution. This work proposes a constructive heuristic called *path-scanning with efficiency rule* (PS-Efficiency) for CARP, which extends the ideas of the *path-scanning with ellipse rule* (PS-Ellipse) proposed by Santos et al. [31]. In PS-Ellipse, an ellipse rule is triggered when the current vehicle capacity is below a given static threshold. Once activated, the rule restricts the next candidate edges to be within the ellipse containing the depot and the current node as focal points. The PS-Efficiency improves these ideas by: (i) proposing a dynamic threshold for activating the rule; (ii) the set of candidate edges are selected based on a route efficiency index with respect to the ratio of demand over distance.

Computational experiments conducted on a set of benchmark instances were performed to compare the proposed method to other path-scanning heuristics from literature. Experiments reveal that PS-Efficiency substantially outperformed PS-Ellipse [31].

This paper is organized as follows. The CARP is formally defined in Section 2.2. Section 2.3 presents a review of constructive heuristics for CARP. Section 2.4 describes the PS-Efficiency heuristic. Section 2.5 shows the computational experiments and the performance analysis of the methodologies. Section 2.6 provides the final remarks.

2.2 Problem Definition

The Capacitated Arc Routing Problem (CARP) [18] can be formally defined as follows. Let $G(V, E)$ be an undirected graph with a non-negative cost or length $c(v_i, v_j)$ and a non-negative demand $d(v_i, v_j)$ assigned to each edge $(v_i, v_j) \in E$. Let $E_R \subseteq E$ be the set of edges with positive demand, called *required edges*. A fleet of identical vehicles with limited

capacity D is used to service all required edges. While traversing an edge (v_i, v_j), a vehicle might (i) service (v_i, v_j), which deducts its capacity by $d(v_i, v_j)$ and increases the solution cost by $c(v_i, v_j)$ or (ii) deadhead (v_i, v_j), which only increases the solution cost by $c(v_i, v_j)$.

A route is a trajectory of a vehicle in G, represented by the sequence of traversed edges, which starts and finishes at a distinguished node v_0 called *depot*. A set of routes is defined feasible if the sum of demands serviced in each route does not exceed D and each required edge is serviced by exactly one vehicle. The objective is to find a feasible set of routes with minimum cost.

2.3 Constructive heuristics for CARP

The literature of CARP heuristics can be generally classified in three major categories: (i) constructive heuristics, which start from an empty solution and iteratively grow it into a feasible solution; (ii) local search methods, that start from a feasible solution and attempt to improve it by exploring a neighbourhood; (iii) metaheuristics, which usually comprise a combination of constructive heuristics, local search methods and likely some additional optimization techniques, such as short and long-term memory, evolutionary framework, diversification and intensification. Constructive heuristics are usually simple and fast. On the other hand, metaheuristics have higher complexity and require more processing time while often providing the best solutions. The decision of which method is more suitable for a given application depends on the available computing time and the characteristics of the instances.

Constructive heuristics remain an important area of research for CARP. As described by Santos et al. [31], there are several reasons for investigating these heuristics: (i) fast processing time, which is crucial for many real-time applications; (ii) warm starting solutions for metaheuristics; (iii) less fine-tuning of parameters; (iv) more easily adapted to the numerous CARP variants. For example, PS-Ellipse was used to construct initial solutions for at least three metaheuristics [32, 38, 33] and inspired heuristics for at least four variants of CARP [37, 20, 15, 8].

The first constructive heuristics for CARP have been proposed by Golden and Wong [18] and Golden et al. [19]: *path-scanning* (PS), *augment-merge* (AM) and *construct-strike* (CS). The CS heuristic have not attracted much attention (except for *modified-CS* from Pearn [27]), possibly because of its complexity. On the other hand, PS and AM have been extensively used as benchmark heuristics for CARP and have given rise to new improved methods based on their classical versions. Special attention will be given to the review of path-scanning heuristics in Section 2.3.1.

Belenguer et al. [3] proposed the *improved merge* (IM) heuristic based on AM, achieving very good results for large instances (more than 100 required edges) with fairly little computational cost. Ulusoy [36] proposed an innovative *route-first cluster-second* method where first a giant tour including all required edges is built which is then partitioned into feasible tours by a *Split* algorithm. Later, Prins et al. [30] proposed improving methods for Ulusoy's algorithm. Other constructive heuristics for CARP include *parallel-insert* [10], *augment-insert* [26] and the four heuristics proposed by Wøhlk [40] in her thesis:

modified path-scanning, *double outer scan*, *node duplication heuristic* and *A-ALG*.

2.3.1 Path-scanning heuristics

The original *path-scanning* (PS) proposed by Golden et al. [19] is a greedy heuristic that starts with the vehicle at the depot node, then progressively services the nearest unserviced edge. The vehicle stops and returns to the depot when the remaining vehicle capacity is not sufficient to service more edges. Then, a new vehicle is employed to service the remaining edges and the process is repeated. Often there are multiple required edges with minimum distance. In this case the heuristic employs five different criteria to decide which unserviced edge $(v_i, v_j) \in E_R$ will be serviced next: (1) minimize $c(v_i, v_j)/d(v_i, v_j)$; (2) maximize $c(v_i, v_j)/d(v_i, v_j)$; (3) minimize cost of node v_j back to depot node; (4) maximize cost of node v_j back to depot node; (5) use criterion 4 if vehicle has more than half capacity, otherwise use criterion 3. An instance is solved five times, using the five different criteria, then the best solution is returned. Using the five different criteria often generates five very different solutions, therefore increasing the chance of finding a good solution.

A modified version of PS was proposed by Pearn [27] by selecting one of the original five criteria at random as tie-breaking for each time multiple unserviced edges with minimum distance are identified. This heuristic will be called *path-scanning with random criteria* (PS-RC) from now on. One of the main advantages of PS-RC is to produce much more than five different solutions because of its stochastic behavior. Results with number of executions $k = 30$ were published and obtained better performance than the original path-scanning for all instances experimented.

Later, Belenguer et al. [3] proposed another modified version of *path-scanning*, here called *path-scanning with random edge* (PS-RE), which simplifies the tie-breaking decision by just randomly selecting one of the tied edges. The authors compared the performance of PS-RE and PS-RC on benchmark instances and found the performance of both heuristics for $k = 20$ and $k = 50$ to be fairly similar.

Finally, Santos et al. [31] proposed the *path-scanning with ellipse rule* (PS-Ellipse) that works similarly to PS-RE but also considers an "ellipse rule". This rule triggers when the vehicle remaining capacity is low enough and restricts the vehicle to service only a subset of edges that are within an ellipse. The PS-Ellipse produced far better solutions than its precedents PS-RE and PS-RC with a minor increase in computational time. A detailed description of PS-Ellipse is provided next.

Let $ned = |E_R|$ be the number of required edges, $td = \sum_{(v_i,v_j)\in E_R} d(v_i, v_j)$ be the total demand, $tc = \sum_{(v_i,v_j)\in E_R} c(v_i, v_j)$ be the total cost of required edges, α a real parameter, (v_g, v_h) the last serviced edge in the route, $SP(v_p, v_q)$ the shortest path cost from node v_p to node v_q, rvc the remaining vehicle capacity and v_0 the depot node. The heuristic starts with an empty vehicle at the depot and iteratively inserts in the route the closest unserviced edge (v_i, v_j) that does not violate the rvc. Tie-breaking rule selects one of the tied edges at random. The *ellipse rule* is activated right after rvc satisfies the *triggering criteria* (1). After the *ellipse rule* is triggered, the vehicle can only service edges that are inside the ellipse with focal points v_h and v_0. An edge $(v_i, v_j) \in E_R$ is inside the ellipse

if inequation (2) holds. The vehicle returns to the depot when its capacity is insufficient to service any edge inside the ellipse. A new empty vehicle starts at depot node and the process is repeated until all required edges are serviced.

$$rvc \leq \alpha \times td/ned \tag{1}$$

$$SP(v_h, v_i) + c(v_i, v_j) + SP(v_j, v_0) \leq tc/ned + SP(v_h, v_0) \tag{2}$$

The rationale behind PS-Ellipse is that if the remaining vehicle capacity is low then only edges that are close to the path from the current node to the depot should be considered for servicing. The authors performed computational experiments comparing PS-RC, PS-RE and PS-Ellipse on a set of benchmark instances. The results revealed that PS-Ellipse reduced the average deviation from lower bounds by about 44% in comparison to PS-RC and PS-RE.

2.4 Path-scanning with efficiency rule

The *path-scanning with efficiency rule* (PS-Efficiency) is a heuristic based on the *path-scanning with ellipse rule* (PS-Ellipse) proposed by Santos et al. [31] described in Section 2.3.1. It constructs a set of feasible routes in a greedy fashion, where each route is created starting from the depot and then sequentially selecting the nearest unserviced edge $(v_i, v_j) \in E_R$ for which the demand $d(v_i, v_j)$ does not exceed the remaining vehicle capacity. Similarly to PS-Ellipse, when the vehicle reaches a certain load threshold it activates an *efficiency rule* which restricts the vehicle to service only edges considered "cost-efficient".

2.4.1 Terminology and definitions

The terminology adopted by the algorithm is given next: $ned(E')$ is the number of required edges in the set E', $td(E') = \sum_{(v_i,v_j)\in E'} d(v_i, v_j)$ is the total demand of set E', $tc(E') = \sum_{(v_i,v_j)\in E'} c(v_i, v_j)$ is the total cost of edges in set E', rvc is the remaining vehicle capacity, α is a real parameter, (v_g, v_h) is the last serviced edge in the route and v_0 is the depot.

The function $near(v_h)$ is given by (3) and defines the set of unserviced edges close enough to v_h. The function $SP(v_i, v_j)$ gives the shortest path cost from node v_i to node v_j.

$$near(v_h) = \{(v_i, v_j) \in E_R | min\{SP(v_h, v_i), SP(v_h, v_j)\} \leq tc(E_R)/ned(E_R) \\ \textbf{and } (v_i, v_j) \text{ is unserviced}\} \tag{3}$$

Let a route R be defined by the sequence $((u_1, v_1), (u_2, v_2), ...(u_n, v_n))$, where each (u_i, v_i) is a serviced edge. The route efficiency index $eff(R)$ is given by equation (4), where the numerator is the demand serviced and the denominator is the distance traversed

including the return to the depot. In summary, $eff(R)$ is the ratio of the demand serviced by distance traversed of route R.

$$eff(R) = \frac{\sum_{i=1}^{n} d(u_i, v_i)}{SP(v_0, u_1) + \sum_{i=1}^{n} c(u_i, v_i) + \sum_{i=1}^{n-1} SP(v_i, u_{i+1}) + SP(v_n, v_0)} \tag{4}$$

2.4.2 Efficiency rule

The load threshold that activates the *efficiency rule* is given by the *triggering criteria* (5). If $near(v_h)$ is empty, the algorithm adopts inequation (1) instead of (5).

$$rvc \leq \alpha \times td(near(v_h))/ned(near(v_h)) \tag{5}$$

Once the vehicle achieves the load threshold, the vehicle is constrained to service only edges $(v_i, v_j) \in E_R$ satisfying the *efficiency rule* (6).

$$\frac{d(v_i, v_j)}{SP(v_h, v_i) + c(v_i, v_j) + SP(v_j, v_0) - SP(v_h, v_0)} \geq eff(R) \tag{6}$$

The *efficiency rule* only allows edges to be serviced by the route if their demand compensates for the increase of the route cost; we call these edges cost-efficient. More precisely, the route efficiency index $eff(R)$ is not allowed to decrease once the *efficiency rule* is activated. Figure 2.1 gives an example with three required edges (v_i, v_j), (v_k, v_l) and (v_m, v_n). In this example, only the edges (v_i, v_j) and (v_m, v_n) are satisfying the *efficiency rule*.

After the determination of the cost-efficient edges, the algorithm selects the closest one to the vehicle that does not exceed the rvc to be serviced next. Ties are broken randomly. If the vehicle capacity is insufficient to service any cost-efficient edge then the vehicle returns to the depot. If there are still unserviced edges, a new vehicle is employed and the process is repeated. The PS-Efficiency pseudocode is given by Algorithm 1.

A comparison between PS-Ellipse and PS-Efficiency is provided next:

1. Comparing the *triggering criteria* (1) and (5), the latter considers only unserviced edges that are close to the current node while the former considers all edges. The reasoning for adopting the dynamic criteria (5) instead of the static criteria (1) is that edges which are distant from the current node are often less relevant to decision making.

2. Comparing the *restricting rules* (2) and (6), the *efficiency rule* (6) considers dynamic information about the current route expressed by the efficiency index $eff(R)$. On the other hand, *ellipse rule* (2) relies solely on static information of the instance. The rationale of the *efficiency rule* is that optimal CARP solutions are likely composed of routes with high efficiency indices.

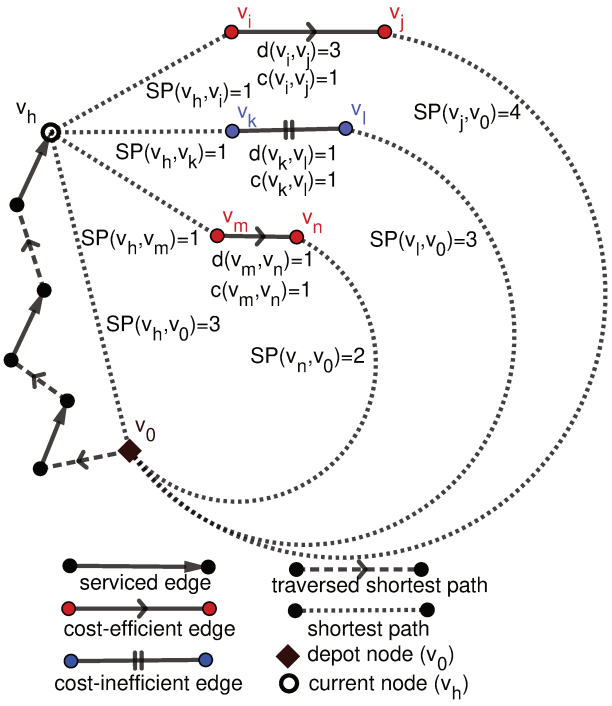

Figure 2.1: Example of efficiency rule for a route R with $eff(R) = 1$.

Regarding computational complexity, PS-Efficiency can be implemented in the same worst-case time complexity as PS-Ellipse. From Algorithm 1, the most expensive computational step relies is the determination of the set F of candidate edges, which has $O(|E_R|)$ linear complexity on the number of required edges. That step is present in both PS-Ellipse and PS-Efficiency, and since it is repeated $|E_R|$ times for each solution and k solutions are generated, the whole procedure has $O(k|E_R|^2)$ time complexity.

Algorithm 1: PS-Efficiency

Input: $G(V, E)$: instance graph; D: vehicle capacity; **c**: cost vector; **d**: demand vector; **SP**: matrix of shortest path costs; α: real parameter; k: number of iterations;
Output: *bestSol*: best solution found;
begin
  $bestSol \leftarrow \emptyset$;
  for $it = 1$ to k **do**
    $sol \leftarrow \emptyset$; $R \leftarrow \{v_0\}$; $rvc \leftarrow D$; $v_h \leftarrow v_0$; $sumDist \leftarrow 0$; $rule \leftarrow$ false;
    for $n = 1$ to ned **do**
      if $near(v_h) \neq \emptyset$ **and** inequation (5) is satisfied **then**
        $rule \leftarrow$ true;
      else if $near(v_h) = \emptyset$ **and** inequation (1) is satisfied **then**
        $rule \leftarrow$ true;
      if $rule$ is false **then**
        Determine set of candidate edges $F \subseteq E_R$;
      else
        Determine set of candidate edges $F \subseteq E_R$ satisfying (6);
      if $F = \emptyset$ **then**
        $sol \leftarrow sol \cup (R \cup \{v_0\})$; // end of route
        $R \leftarrow \emptyset$; $rvc \leftarrow D$; $v_h \leftarrow v_0$;
        $sumDist \leftarrow 0$; $rule \leftarrow$ false;
      else
        Select (v_i, v_j) randomly from F and add it to R; // edge is serviced
        $sumDist \leftarrow sumDist + SP(v_h, v_i) + c(v_i, v_j)$;
        $rvc \leftarrow rvc - d(v_i, v_j)$; $v_h \leftarrow v_j$;
        $eff(R) \leftarrow (D - rvc)/(sumDist + SP(v_h, v_0))$;
    if $(cost(sol) < cost(bestSol))$ **or** $bestSol = \emptyset$ **then**
      // If current solution is the better so far
      $bestSol \leftarrow sol$;
  return $(bestSol)$;

2.5 Computational experiments

2.5.1 Settings and instances

The heuristics in Table 2.1 were implemented in C++ using the LEMON [12] library for graph algorithms and data structures. The experiments were executed in an Intel Core i7-6700K 4.0 GHz with 8 GB of RAM and Linux 64-bit operating system. Full experimental data, instances and the algorithms source codes are available on-line[1].

Table 2.2 presents the data concerning the instances benchmark considered in the experiments, which includes 23 *gdb*[19], 34 *val* [5], 24 *egl* [22], 25 *C* [6], 25 *D* [6], 25 *E* [6], 25 *F* [6], and 10 *egl-large* [9] instances, totaling 191 instances.

2.5.2 Comparative performance analysis

The results presented in Table 2.3 compare the PS-Efficiency with three path-scanning methods from literature: PS-RC [27], PS-RE [3] and PS-Ellipse [31]. Each heuristic was evaluated considering multiple values of number of iterations k. Regarding PS-Ellipse

[1] `http://www.ic.unicamp.br/~fusberti/problems/carp`

Table 2.1: Path-scanning algorithms.

Algorithm	Triggering criteria	Restricting rule	Tie-breaking
PS-RC (Section 2.3.1)	-	-	random criteria
PS-RE (Section 2.3.1)	-	-	random edge
PS-Ellipse (Section 2.3.1)	static (1)	*ellipse rule* (2)	random edge
PS-Efficiency (Section 2.4)	dynamic (5)	*efficiency rule* (6)	random edge
PS-Alt1 (Section 2.5.3)	static (1)	*efficiency rule* (6)	random edge
PS-Alt2 (Section 2.5.3)	dynamic (5)	*ellipse rule* (2)	random edge

Table 2.2: Benchmark instances for CARP.

group	$\|V\|$	$\|E\|$	$\|E_R\|$
gdb	7-27	11-55	11-55
val	24-50	34-97	34-97
egl	77-140	98-190	51-190
C	32-97	42-140	32-107
D	32-97	42-140	32-107
E	26-97	35-142	28-107
F	26-97	35-142	28-107
egl-large	255	375	347-375

and PS-Efficiency, multiple values of parameter α were considered. The results are shown in percentage of the average deviation from lower bounds *Gap(%)*, defined as: *Gap(%)*= $100(UB - LB)/LB$, where UB is the solution cost and LB is the lower bound given by [7]. Comparing PS-Efficiency and PS-Ellipse for each k and α, the best deviations are highlighted in bold. Over all heuristics and parameters, the best results were underlined.

Overall comparison. Table 2.3 shows that the heuristics PS-Efficiency and PS-Ellipse clearly outperformed by a great margin the PS-RC and PS-RE heuristics. For all sets of instances, the average deviations obtained by PS-Efficiency and PS-Ellipse with $k = 1000$ are better than those obtained by PS-RC and PS-RE with $k = 20000$.

The best overall value of α for PS-Ellipse was $\alpha = 1.5$ and for PS-Efficiency was $\alpha = 3.0$. Considering $k = 20000$, the heuristics PS-RE, PS-Ellipse($\alpha = 1.5$) and PS-Efficiency($\alpha = 3.0$) reduced the PS-RC overall *Gap(%)* by 0.70%, 42.23% and 53.08%, respectively.

PS-Efficiency vs. PS-Ellipse. In comparison to PS-Ellipse ($\alpha = 1.5$), the PS-Efficiency ($\alpha = 3.0$) reduced the overall *Gap(%)* by 14.22%, 16.12% and 18.79% for k 1000, 10000 and 20000, respectively.

The most significant reduction of deviations from PS-Ellipse ($\alpha = 1.5$) to PS-Efficiency ($\alpha = 3.0$), considering the results with $k = 20000$, was reported for the F instances where the *Gap(%)* was reduced from 6.04% to 4.18% (reduction of 30.7%) and the least significant reduction was observed for the *gdb* instances from 0.88% to 0.84% (reduction of 4.5%).

Consistency Analysis. The results of PS-Efficiency and PS-Ellipse for each value of k and α show that PS-Efficiency performed better for every set of instances with the only exceptions being the *gdb* instances with $\alpha = 1.0$ and $k \in \{1000, 10000\}$; and *val* instances

with $\alpha = 1.0$ and $k = 1000$. Therefore, the performance of PS-Efficiency was consistently better for almost all instances and parameters considered.

Robustness Analysis. Table 2.3 shows PS-Efficiency as much more parameter-robust than PS-Ellipse for all sets of instances. Let a variation factor for each heuristic be given by: $100 \times (max_{GAP} - min_{GAP})/min_{GAP}$, where max_{GAP} is the largest deviation obtained in the experiment considering all values of α and min_{GAP} the smallest. The overall variation factor with $k = 20000$ for PS-Ellipse is 63.10% while the same factor for PS-Efficiency is 23.35%. In special, the set of instances *egl* with $k = 20000$ presents the highest difference between variation factors, where PS-Ellipse obtained 230.92% and PS-Efficiency obtained 13.69%.

2.5.3 Effect of triggering criteria and restricting rule

PS-Ellipse and PS-Efficiency are two path-scanning heuristics distinguished by their (i) *triggering criteria* that determine the conditions to activate a (ii) *restricting rule* which restrain the set of edges that can be serviced. To evaluate the effectiveness of these features we have compared PS-Ellipse and PS-Efficiency with two new path-scanning heuristics, called PS-Alt1 and PS-Alt2. These new heuristics implement alternative combinations of the *triggering criteria* and *restricting rules* as shown by Table 2.1.

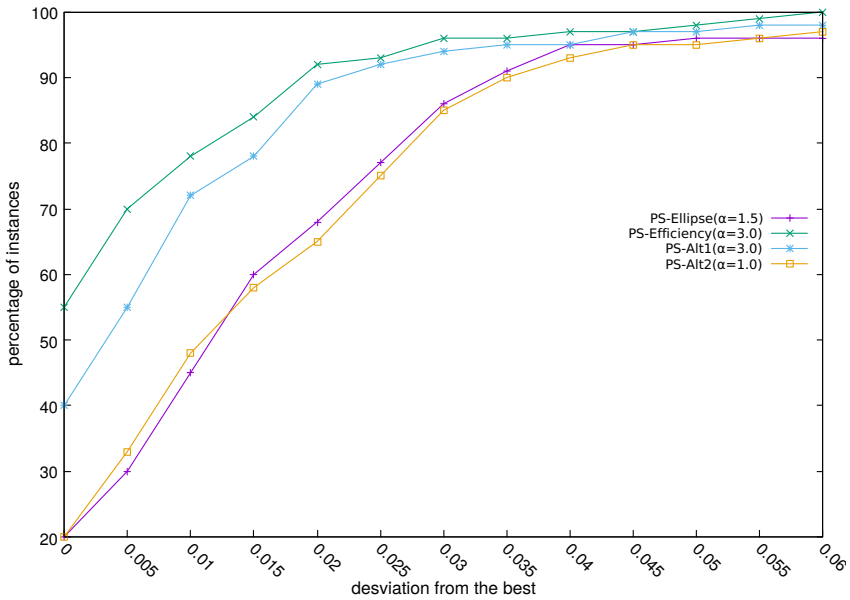

Figure 2.2: Comparative experiment of path-scanning algorithms.

The experiment aims to show the impact of each feature embedded in PS-Efficiency. To achieve that, the experiment is executed for the four path-scanning heuristics: PS-Ellipse, PS-Efficiency, PS-Alt1 and PS-Alt2. Each heuristic was executed for all instances in Table 2.2 with $k = 20000$ and using its overall best value of parameter α. The performance profiles introduced by Dolan and Moré [13] are adopted as comparison method.

Figure 2.2 shows the results. The experiment works by first computing a deviation factor from the best using the following formula for each heuristic and each instance: $(UB - UB_{best})/UB_{best}$, where UB is the solution cost obtained by the heuristic and

UB_{best} is the best solution cost obtained among all heuristics. From that, it is calculated a percentage of the instances (y-axis) that fall below certain thresholds of deviation factors from the best (x-axis) for each method.

Figure 2.2 shows that PS-Efficiency obtained the best solutions for 55% of the instances while PS-Alt1 obtained the best solutions for 40% of instances. Tied in the last place, PS-Ellipse and PS-Alt2 obtained the best solutions for 20% of the instances.

Considering that PS-Efficiency and PS-Alt1 both use the same *restricting rule* and diverge only by their *triggering criteria*, it is possible to infer from the results that the *efficiency rule* is a key factor for the better performance of these methods in comparison to PS-Ellipse and PS-Alt2.

Observing the performance of PS-Alt2 in Figure 2.2, it is revealed that the dynamic *triggering criteria* was ineffective when used in combination with the *ellipse rule*. This indicates that there is a synergy between the dynamic *triggering criteria* and the *efficiency rule*, thus both features should be employed together so as to achieve the best outcome.

2.5.4 Computational times

Table 2.4 shows the CPU average processing times required by the heuristics for each set of instances. Each method was executed for $k = 20000$ iterations and for six different values of the parameter $\alpha \in \{1.0, 1.5, 2.0, 2.5, 3.0, 3.5\}$. The PS-RC and PS-RE are shown having the overall smallest processing times, while PS-Ellipse consumed less computing time than PS-Efficiency. This can be explained by the fact that the dynamic information used by PS-Efficiency requires often updates, such as identifying nearby unserviced edges and computing the route efficiency index. On the other hand, PS-Ellipse uses only static information from the graph thereby requiring less computation. The PS-Alt1 and PS-Alt2 processing times are positioned between PS-Ellipse and PS-Efficiency as expected since they combine features of both methods.

Another important pattern is that the processing time of PS-Efficiency is approximately 1.5 times the processing time of PS-Ellipse for all sets of instances. In other words, the observed time increase seems bounded by a constant factor. This was expected since PS-Efficiency has the same worst-case asymptotic time complexity as PS-Ellipse.

2.5.5 Comparison to metaheuristic approaches

Constructive heuristics are often incorporated by metaheuristics because they are fast procedures that produce feasible solutions with potentially good quality and high variability. For example: TSA2 (*tabu search algorithm*) [9], GRASP [38] and HMA [11] used the path-scanning heuristics PS, PS-Ellipse and PS-RE respectively to warm start its initial solutions.

The CARP literature contains some reports on the effects of incorporating constructive heuristics into metaheuristics frameworks. Lacomme et al. [21] proposed a *memetic algorithm* (MA) and reported that initializing the MA initial population with three different constructive heuristics influenced positively for the MA final solution quality. Brandão and Eglese [9] proposed two versions of a deterministic *tabu search algorithm*: TSA1 and

TSA2, with the difference being that TSA2 is the TSA1 executed 5 times starting with 5 different initial solutions obtained by 5 different constructive heuristics. Brandão and Eglese argued that "the diversity provided by the different starting solutions was found to be useful in ultimately finding high quality solutions". Polacek et al. [28] proposed a *variable neighbourhood search* (VNS) and reported results on initializing their method using different solutions obtained by a constructive heuristic. It was reported that the initial solution cost did not influenced much in terms of final solution quality but influenced greatly for shortening the VNS processing time.

Moreover, some recent metaheuristics such as HGA [2] and MAENS [34, 16] employed constructive heuristics as internal components of its local search methods. In these local search methods, a subset of the routes of a feasible solution is selected and reconstructed using constructive heuristics and, if the reconstructed routes have lower costs than the original ones, then the solution is updated. Authors of MAENS stated that their local search method is likely to generate high-quality solutions due the adoption of constructive heuristics (PS and Ulusoy's) that are known to produce relatively good solutions.

In summary, good constructive heuristics are commonly incorporated by CARP metaheuristics. Nevertheless, a comparison between constructive heuristics and metaheuristics is interesting considering the trade-off involving solution costs and processing times. Table 2.5 presents results comparing the PS-Efficiency ($\alpha = 3.0$) to two high performance metaheuristics (MAENS [34] and HMA [11]). Table 2.5 upper segment presents the average deviation from lower bounds *Gap(%)*, calculated as in Section 2.5.2. Table 2.5 lower segment shows the average processing time *CPU(s)* for each method.

The processing times for the metaheuristics were obtained from Chen et al. [11]. Following the practice of literature [11, 32, 38], a CPU time conversion factor, based on CPU frequency, was adopted. Therefore, the processing times reported for the metaheuristics [11] were scaled by a factor of 0.7. It is worth mentioning that differently from other methods, HMA is stopped when its best incumbent cost achieves a known lower bound (obtained from literature). This characteristic may substantially favor the HMA processing time in comparison to the other methods.

Table 2.5 shows that metaheuristics obtained better solutions but at the cost of higher processing times. For example, the overall *Gap(%)* for PS-Efficiency ($k = 20000$) is 4.71% while for the metaheuristics vary from 0.19% to 0.55%. On the other hand, the average processing times for PS-Efficiency is 10.14s while for the metaheuristics vary from 204.88s to 230.30s.

Table 2.3: Computational experiment results.

group	k	PS-RC(k)	PS-RE(k)	PS-Ellipse(k, α)						PS-Efficiency(k, α)					
				α : 1.0	1.5	2.0	2.5	3.0	3.5	α : 1.0	1.5	2.0	2.5	3.0	3.5
gdb	1000	3.95	3.77	**1.63**	1.50	1.98	1.92	4.83	7.82	1.70	**1.44**	**1.19**	**1.24**	**1.83**	**1.85**
	10000	2.65	2.15	**1.10**	1.02	1.34	1.22	3.52	6.44	1.16	**0.74**	**0.78**	**0.80**	**1.03**	**1.16**
	20000	2.21	2.06	1.04	0.88	1.25	1.22	3.46	6.18	**1.02**	**0.74**	**0.75**	**0.71**	**0.84**	**1.05**
val	1000	8.42	8.35	**6.12**	5.38	5.10	5.26	6.03	8.17	6.20	**5.29**	**4.85**	**4.83**	**4.65**	**4.36**
	10000	5.85	6.29	4.37	3.70	3.42	3.53	4.23	6.31	**4.28**	**3.65**	**3.33**	**3.27**	**2.75**	**3.15**
	20000	5.60	5.81	3.82	3.38	3.12	3.26	3.98	5.88	**3.79**	**3.18**	**2.98**	**2.80**	**2.53**	**2.78**
egl	1000	16.64	16.43	8.59	9.02	12.04	15.05	19.96	25.32	**8.50**	**7.49**	**7.51**	**7.74**	**7.73**	**8.03**
	10000	15.32	15.29	7.30	7.67	10.22	13.32	18.62	23.67	**7.07**	**6.49**	**6.41**	**6.59**	**6.80**	**6.47**
	20000	14.92	15.00	7.05	7.51	10.01	12.62	18.20	23.33	**6.89**	**6.31**	**6.06**	**6.28**	**6.41**	**6.25**
C	1000	16.08	16.27	10.40	9.28	9.50	10.03	10.45	12.65	**9.91**	**9.27**	**8.90**	**8.61**	**8.88**	**8.61**
	10000	13.39	12.99	7.40	7.27	7.28	7.29	8.64	9.99	**7.44**	**6.83**	**6.96**	**6.34**	**6.69**	**6.61**
	20000	12.76	12.22	7.26	6.71	6.64	6.60	8.20	9.77	**6.74**	**6.27**	**6.36**	**5.96**	**6.09**	**6.34**
D	1000	12.15	12.30	10.16	9.40	8.48	7.74	7.68	7.20	**9.55**	**8.13**	**7.26**	**7.18**	**6.49**	**6.86**
	10000	9.23	9.45	6.74	6.28	5.91	5.66	5.35	4.97	**6.34**	**5.74**	**4.75**	**4.86**	**4.93**	**4.48**
	20000	8.77	8.85	6.26	5.71	5.33	5.03	4.92	4.45	**5.73**	**5.15**	**4.40**	**4.20**	**4.13**	**4.30**
E	1000	16.10	16.18	10.90	10.06	9.91	10.05	11.56	12.62	**9.57**	**9.54**	**8.26**	**8.28**	**8.20**	**8.27**
	10000	13.09	12.60	8.04	7.68	7.72	7.98	9.25	10.25	**7.45**	**6.93**	**6.29**	**5.97**	**6.00**	**5.82**
	20000	12.57	11.96	7.34	7.16	6.70	7.38	8.55	9.77	**6.96**	**6.49**	**5.55**	**5.62**	**5.53**	**5.57**
F	1000	12.34	11.53	10.04	8.92	8.96	8.47	8.64	7.75	**9.48**	**8.27**	**8.50**	**7.78**	**7.40**	**6.66**
	10000	9.20	9.07	7.43	6.53	6.00	6.50	6.05	5.81	**6.64**	**6.40**	**5.80**	**5.47**	**4.77**	**4.64**
	20000	8.56	8.47	6.83	6.04	5.54	5.84	5.48	5.27	**6.00**	**5.69**	**5.29**	**4.98**	**4.18**	**4.44**
egl-large	1000	28.14	27.61	17.78	16.97	16.98	17.91	18.25	19.53	**17.06**	**16.45**	**16.42**	**16.07**	**16.59**	**16.26**
	10000	25.50	26.37	16.23	15.65	16.03	16.49	16.79	17.81	**16.12**	**15.47**	**14.76**	**15.19**	**15.22**	**15.08**
	20000	25.09	26.12	16.01	15.16	15.45	16.09	16.10	17.27	**15.65**	**15.05**	**14.69**	**14.50**	**14.23**	**14.95**
overall	1000	12.96	12.82	8.73	8.09	8.37	8.75	10.14	11.86	**8.31**	**7.53**	**7.12**	**6.99**	**6.94**	**6.84**
	10000	10.50	10.45	6.55	6.20	6.42	6.90	8.23	9.87	**6.28**	**5.75**	**5.38**	**5.27**	**5.20**	**5.12**
	20000	10.04	9.97	6.15	5.80	5.94	6.41	7.81	9.46	**5.81**	**5.33**	**4.98**	**4.85**	**4.71**	**4.89**

In **bold**: the best results for each α comparing PS-Efficiency and PS-Ellipse.
In underline: the best results regarding all algorithms.

Table 2.4: Average processing times for $k = 20000$.

group	PS-RC	PS-RE	PS-Ellipse	PS-Efficiency	PS-Alt1	PS-Alt2
gdb	0.76	0.70	0.74	1.19	0.91	1.08
val	2.16	2.04	2.18	3.08	2.52	3.04
egl	6.02	5.94	7.70	11.82	9.53	10.12
C	2.34	2.27	2.63	3.90	3.03	3.60
D	2.20	2.12	2.27	3.15	2.55	3.12
E	2.20	2.16	2.45	3.69	2.88	3.38
F	2.08	1.98	2.06	2.91	2.38	2.88
egl-large	59.34	58.04	71.05	97.52	81.04	87.02
overall	5.49	5.34	6.36	9.07	7.41	8.20

Average processing time in seconds (s).

Table 2.5: Comparison of constructive heuristics and metaheuristics.

group	PS-Efficiency ($\alpha = 3.0$)			Metaheuristics	
	$k = 1000$	$k = 10000$	$k = 20000$	MAENS	HMA
***Gap(%)* (average deviation from lower bounds, in percentage)**					
gdb	1.83	1.03	0.84	0.01	0.00
val	4.65	2.75	2.53	0.22	0.06
egl	7.73	6.80	6.41	0.92	0.39
C	8.88	6.69	6.09	0.46	0.07
D	6.49	4.93	4.13	0.26	0.09
E	8.20	6.00	5.53	0.61	0.05
F	7.40	4.77	4.18	0.30	0.03
egl-large	16.59	15.22	14.23	3.54	1.97
overall	6.94	5.20	4.71	0.55	0.19
***CPU(s)* (average processing time, in seconds)**					
gdb	0.06	0.63	1.26	3.13	0.83*
val	0.21	2.10	3.29	33.85	18.70*
egl	0.69	6.75	13.49	348.94	452.22*
C	0.21	2.13	4.20	115.85	37.95*
D	0.16	1.64	3.36	153.67	24.59*
E	0.20	2.02	4.04	112.62	81.58*
F	0.16	1.55	3.10	116.79	5.91*
egl-large	5.53	55.28	110.40	1706.11	2872.80*
overall	0.51	5.09	10.14	204.88	230.30*

*: HMA is stopped when it achieves a best known lower bound (obtained from literature).

2.6 Conclusion

This paper proposed a constructive heuristic for the capacitated arc routing problem called *path-scanning with efficiency rule* (PS-Efficiency). This heuristic uses a dynamically activated rule which restricts the search towards promising candidate edges to be serviced next by the current route. The *efficiency rule* is based on how each edge would affect the current route efficiency calculated as the ratio of serviced demand by traversed distance.

Computational experiments conducted on a set of benchmark instances revealed that the proposed heuristic outperformed all previous path-scanning heuristics by a considerable margin and was found to be more parameter-robust. Moreover, the impact of the *efficiency rule* was investigated and found to be a key factor for the high performance of the PS-Efficiency heuristic.

Future research can focus on assessing the effectiveness of PS-Efficiency when applied to other routing problems, such as the vehicle routing problem [35]. Another possible line of research is to investigate the speed-ups that can be achieved by implementing a parallelized PS-Efficiency.

2.7 Acknowledgment

This work was supported by grants #2016/00315-0, São Paulo Research Foundation (FAPESP) and 307472/2015-9, National Counsel of Technological and Scientific Development (CNPq).

References

[1] H Murat Afsar. A branch-and-price algorithm for capacitated arc routing problem with flexible time windows. *Electronic Notes in Discrete Mathematics*, 36:319–326, 2010.

[2] Rafael Kendy Arakaki and Fábio Luiz Usberti. Hybrid genetic algorithm for the open capacitated arc routing problem. *Computers & Operations Research*, 90:221–231, 2018.

[3] José-Manuel Belenguer, Enrique Benavent, Philippe Lacomme, and Christian Prins. Lower and upper bounds for the mixed capacitated arc routing problem. *Computers & Operations Research*, 33(12):3363–3383, 2006.

[4] José Manuel Belenguer, Enrique Benavent, and Stefan Irnich. The capacitated arc routing problem: Exact algorithms. In Ángel Corberán and Gilbert Laporte, editors, *Arc Routing: Problems, Methods, and Applications*, chapter 9, pages 183–221. SIAM Publications, Philadelphia, PA, 2014.

[5] Enrique Benavent, Vicente Campos, Angel Corberán, and Enrique Mota. The capacitated arc routing problem: lower bounds. *Networks*, 22:669–690, 1991.

[6] Patrick Beullens, Luc Muyldermans, Dirk Cattrysse, and Dirk Van Oudheusden. A guided local search heuristic for the capacitated arc routing problem. *European Journal of Operational Research*, 147:629–643, 2002.

[7] Claudia Bode and Stefan Irnich. The shortest-path problem with resource constraints with (k,2)-loop elimination and its application to the capacitated arc-routing problem. *European Journal of Operational Research*, 238(2):415 – 426, 2014.

[8] Adamo Bosco, Demetrio Laganà, Roberto Musmanno, and Francesca Vocaturo. Modeling and solving the mixed capacitated general routing problem. *Optimization Letters*, 7(7):1451–1469, 2013.

[9] José Brandão and Richard Eglese. A deterministic tabu search algorithm for the capacitated arc routing problem. *Computers & Operations Research*, 35:1112–1126, 2008.

[10] Luc Chapleau, Jacques A Ferland, Guy Lapalme, and Jean-Marc Rousseau. A parallel insert method for the capacitated arc routing problem. *Operations Research Letters*, 3(2):95–99, 1984.

[11] Yuning Chen, Jin-Kao Hao, and Fred Glover. A hybrid metaheuristic approach for the capacitated arc routing problem. *European Journal of Operational Research*, 253(1):25–39, 2016.

[12] Balázs Dezs, Alpár Jüttner, and Péter Kovács. Lemon - an open source c++ graph template library. *Electronic Notes in Theoretical Computer Science*, 264(5):23–45, 2011.

[13] Elizabeth D Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.

[14] Richard Eglese, Bruce Golden, and Edward Wasil. Route optimization for meter reading and salt spreading. In Ángel Corberán and Gilbert Laporte, editors, *Arc Routing: Problems, Methods, and Applications*, chapter 13, pages 303–320. SIAM Publications, 2014.

[15] Alireza Eydi and Leila Javazi. Model and solution approach for multi objective-multi commodity capacitated arc routing problem with fuzzy demand. *Journal of Industrial and Systems Engineering*, 5(4):208–229, 2012.

[16] H. Fu, Y. Mei, K. Tang, and Y. Zhu. Memetic algorithm with heuristic candidate list strategy for capacitated arc routing problem. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1–8, Barcelona, Spain, July 2010.

[17] Gianpaolo Ghiani, Cândida Mourão, Leonor Pinto, and Daniele Vigo. Routing in waste collection applications. In Ángel Corberán and Gilbert Laporte, editors, *Arc Routing: Problems, Methods, and Applications*, chapter 15, pages 351–370. SIAM Publications, Philadelphia, PA, 2014.

[18] B. L. Golden and R. T. Wong. Capacitated arc routing problems. *Networks*, 11:305–315, 1981.

[19] L. Bruce Golden, S. James DeArmon, and K. Edward Baker. Computational experiments with algorithms for a class of routing problems. *Computers & Operations Research*, 10:47–59, 1982.

[20] Lucio Grandinetti, Francesca Guerriero, Demetrio Laganà, and Ornella Pisacane. An optimization-based heuristic for the multi-objective undirected capacitated arc routing problem. *Computers & Operations Research*, 39(10):2300–2309, 2012.

[21] Philippe Lacomme, Christian Prins, and Wahiba Ramdane-Cherif. Competitive memetic algorithms for arc routing problems. *Annals of Operations Research*, 131(1-4):159–185, 2004.

[22] YO Leon Li and W. Richard Eglese. An interactive algorithm for vehicle routeing for winter-gritting. *Journal of the Operational Research Society*, pages 217–228, 1996.

[23] Gang Liu, Yongfeng Ge, Tony Z Qiu, and Hamid R Soleymani. Optimization of snow plowing cost and time in an urban environment: A case study for the city of Edmonton. *Canadian Journal of Civil Engineering*, 41(7):667–675, 2014.

[24] M. Cândida Mourão and Leonor S. Pinto. An updated annotated bibliography on arc routing problems. *Networks*, 70(3):144–194, 2017.

[25] Luc Muyldermans and Gu Pang. Variants of the capacitated arc routing problem. In Ángel Corberán and Gilbert Laporte, editors, *Arc Routing: Problems, Methods, and Applications*, chapter 10, pages 223–253. SIAM Publications, Philadelphia, PA, 2014.

[26] W. L. Pearn. Augment-insert algorithms for the capacitated arc routing problem. *Computers & Operations Research*, 18:189–198, 1991.

[27] Wen Lea Pearn. Approximate solutions for the capacitated arc routing problem. *Computers & Operations Research*, 16(6):589–600, 1989.

[28] Michael Polacek, Karl F Doerner, Richard F Hartl, and Vittorio Maniezzo. A variable neighborhood search for the capacitated arc routing problem with intermediate facilities. *Journal of Heuristics*, 14(5):405–423, 2008.

[29] Christian Prins. The capacitated arc routing problem: Heuristics. In Ángel Corberán and Gilbert Laporte, editors, *Arc Routing: Problems, Methods, and Applications*, chapter 7, pages 131–157. SIAM Publications, Philadelphia, PA, 2014.

[30] Christian Prins, Nacima Labadi, and Mohamed Reghioui. Tour splitting algorithms for vehicle routing problems. *International Journal of Production Research*, 47(2):507–535, 2009.

[31] Luís Santos, João Coutinho-Rodrigues, and John R Current. An improved heuristic for the capacitated arc routing problem. *Computers & Operations Research*, 36(9):2632–2637, 2009.

[32] Luís Santos, João Coutinho-Rodrigues, and John R Current. An improved ant colony optimization based algorithm for the capacitated arc routing problem. *Transportation Research Part B: Methodological*, 44(2):246–266, 2010.

[33] Ronghua Shang, Hongna Ma, Jia Wang, Licheng Jiao, and Rustam Stolkin. Immune clonal selection algorithm for capacitated arc routing problem. *Soft Computing*, 20(6):2177–2204, 2016.

[34] Ke Tang, Yi Mei, and Xin Yao. Memetic algorithm with extended neighborhood search for capacitated arc routing problems. *IEEE Transactions on Evolutionary Computation*, 13(5):1151–1166, 2009.

[35] P. Toth and D. Vigo. *Vehicle Routing: Problems, Methods, and Applications*. SIAM Publications, Philadelphia, PA, 2014.

[36] Gündüz Ulusoy. The fleet size and mix problem for capacitated arc routing. *European Journal of Operational Research*, 22:329–337, 1984.

[37] Fábio Luiz Usberti, Paulo Morelato França, and André Luiz Morelato França. The open capacitated arc routing problem. *Computers & Operations Research*, 38(11):1543 – 1555, 2011.

[38] Fábio Luiz Usberti, Paulo Morelato França, and André Luiz Morelato França. Grasp with evolutionary path-relinking for the capacitated arc routing problem. *Computers & Operations Research*, 40(12):3206–3217, 2013.

[39] Pieter Vansteenwegen, Wouter Souffriau, and Kenneth Sörensen. Solving the mobile mapping van problem: A hybrid metaheuristic for capacitated arc routing with soft time windows. *Computers & operations research*, 37(11):1870–1876, 2010.

[40] Sanne Wøhlk. *Contributions to arc routing*. PhD thesis, University of Southern Denmark, Aarhus, Denmark, 2005.

[41] Yuzhou Zhang, Yi Mei, Ke Tang, and Keqin Jiang. Memetic algorithm with route decomposing for periodic capacitated arc routing problem. *Applied Soft Computing*, 52:1130–1142, 2017.

Chapter 3

Hybrid genetic algorithm for the open capacitated arc routing problem

---

The paper presented next is a full article published in the Computers and Operations Research in 2018 and it is co-authored with Fábio Luiz Usberti [1] (DOI: `https://doi.org/10.1016/j.cor.2017.09.020`). The original publication is available at `www.sciencedirect.com` and the copyright is owned by Elsevier Ltd. In this paper we present a hybrid genetic algorithm for the OCARP which obtained substantially better results in comparison to previous methods. In special the proposed method was able to generate good quality solutions with variability for instances where previous approaches presented issues in finding feasible solutions.

The Open Capacitated Arc Routing Problem (OCARP) is an NP-hard arc routing problem where, given an undirected graph, the objective is to find the least cost set of routes that services all edges with positive demand (required edges). The routes are subjected to capacity constraints in relation to edge demands. The OCARP differs from the Capacitated Arc Routing Problem (CARP) since OCARP does not consider a depot and routes are not constrained to form cycles. A hybrid genetic algorithm with feasibilization and local search procedures is proposed for the OCARP. Computational experiments conducted on a set of benchmark instances reveal that the proposed hybrid genetic algorithm achieved the best upper bounds for almost all instances.

3.1 Introduction

The objective of arc routing problems consists of determining the least cost traversal of a given subset of edges in a graph, with one or more collateral constraints. Many real world applications are modeled as arc routing problems, such as street sweeping, garbage collection, mail delivery, meter reading, etc. Estimated expenditure on these services exceeds billions of dollars per year in the United States, thus revealing the economical importance of such problems [9, 4, 8, 5].

The Capacitated Arc Routing Problem (CARP) [11] is a combinatorial optimization

problem defined on an undirected graph $G(V, E)$ with non-negative costs and demands associated to the edges. A fleet of identical vehicles with limited capacity is considered. The set of vehicles must service all edges with positive demand (*required edges*). The objective is to find a minimum cost set of routes that start and finish in a distinguished node, called depot. The CARP has been extensively studied over the last decades and we refer the reader to [22, 20] for a comprehensive survey.

A CARP variation consists in allowing both closed and open routes. An open route can use different nodes to start (*source*) and end (*sink*) the route, while a closed route is a cycle in which the source and sink nodes are necessarily the same. This problem in which routes are not constrained to form cycles is called the Open Capacitated Arc Routing Problem (OCARP) [29]. There are at least two problems that can be formulated as OCARP instances, the Meter Reader Routing Problem [26, 3, 32] and the Cutting Path Determination Problem [19, 25].

The OCARP has been proved *NP-hard* [30]. Attempts were made to solve OCARP to optimality, including a branch-and-bound algorithm [29] and an integer linear programming formulation [30]. These methods solved to optimality only small-sized instances (up to 27 nodes and 55 required edges). Thus it is of interest the development of heuristics that achieve good solutions in reasonable time in practice, which usually involve larger instances.

This work proposes a Hybrid Genetic Algorithm (HGA) for the OCARP. Computational experiments were performed to evaluate the proposed method. The results are compared to other heuristic methods from literature hence revealing the HGA good performance.

This paper is organized as follows. The OCARP is formally defined in Section 3.2. Section 3.3 presents a review of previous works for OCARP and applications. Section 3.4 describes the proposed HGA for OCARP. Section 3.5 contains computational experiments on a set of benchmark instances. Section 3.6 provides the final remarks.

3.2 Problem Definition

The Open Capacitated Arc Routing Problem (OCARP) proposed by Usberti et al. [30] is defined as follows. Let $G(V, E)$ be an undirected connected graph where non-negative costs c_{ij} and non-negative demands d_{ij} are assigned to each edge $(i, j) \in E$. If an edge (i, j) has positive demand $d_{ij} > 0$ then it is called a required edge. Let $E_R \subseteq E$ be the set of required edges. A fleet of M identical vehicles is available, each with capacity D. While traversing the graph, a vehicle might (i) service an edge, which deducts its capacity by the edge demand and increases the solution cost by the edge cost or (ii) deadhead an edge, which only increases the solution cost by the edge cost. A vehicle route is defined by a sequence of directed edges (arcs) traversed by the vehicle, and here both open and closed routes are considered, i.e., an OCARP route may start and end at distinct nodes.

A feasible OCARP solution is composed by at most M routes, which collectively service all required edges and do not exceed the vehicles capacity D (Figure 3.1). OCARP aims to find a feasible set of routes with minimum cost.

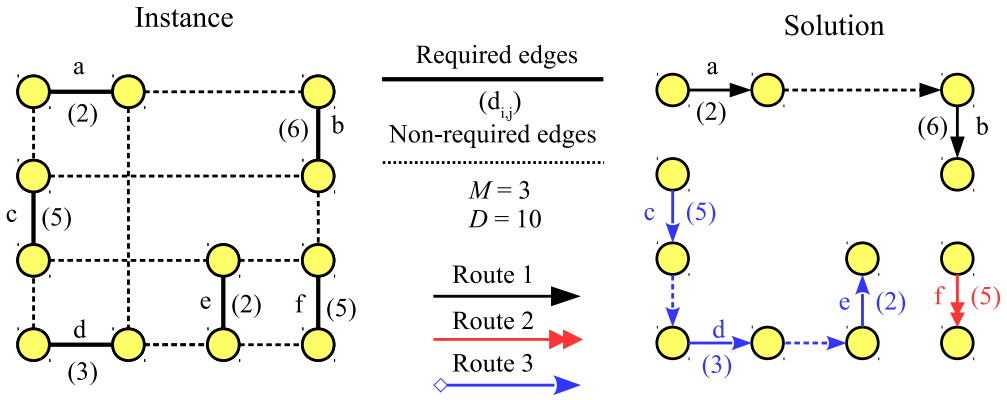

Figure 3.1: OCARP: an instance and a feasible solution.

The OCARP is a variant of the CARP that considers two important differences. First, OCARP allows routes to be open while CARP requires that all routes must start and end at a specific node (depot). Second, many CARP heuristics considers the number of vehicles M as decision variable, while in OCARP the number of vehicles is a fixed parameter. Instances with a very tight number of vehicles can be harder to solve or even to obtain good solutions, as will be discussed in Section 3.5.

3.3 Previous Work

The OCARP was introduced by Usberti et al. [30] and the authors presented an integer linear programming formulation to the problem. This formulation has an exponential number of variables and constraints, which could be the reason why the authors reported difficulties to solve medium and large instances. In the following work, Usberti et al. [29] proposed a branch-and-bound algorithm that improved the known lower bounds. This algorithm exploited the relationship between OCARP and the Capacity and Degree Constrained Minimum Spanning Forest Problem (CDCMSFP).

With respect to heuristic approaches, Usberti et al. [30] proposed a reactive path-scanning with ellipse rule (RPS) that obtained the first known solutions for OCARP. Afterwards Usberti [28] proposed a greedy randomized adaptive search procedure (GRASP) with path-relinking that improved the known best solutions. Nevertheless, the largest instances still presented significant optimality gaps (greater than 30%).

Fung et al. [10] presented a memetic algorithm for another problem, which was also called open capacitated arc routing problem. Their problem differs from the original OCARP on three points: (i) routes starts from a depot but need not to return to the depot; (ii) the number of vehicles is a decision variable with an associated cost; (iii) the graph and the required edges are directed.

3.3.1 Applications

Meter Reader Routing Problem

The Meter Reader Routing Problem (MRRP) considers the routing of employees responsible for metering electric, water and gas consumption data within urban areas. The employees are taken by car to the starting point of their routes and after finishing their routes they take public transport. The MRRP can be modeled as an OCARP, since the walking routes performed by the employees start and end at possibly different nodes. The objective is to minimize the routes travel-time and also consider working shifts. Stern and Dror [26] consider a route-first cluster-second heuristic where initially a single non-capacitated route cover all required edges. The route is then partitioned, and each part is designated to a single employee. Bodin and Levy [3] consider an arc partitioning algorithm that was later applied by Wunderlich et al. [32] to route employees for the Southern California Gas Company (SOCAL) from Los Angeles, USA.

Cutting Path Determination Problem

The Cutting Path Determination Problem (CPDP) concerns in finding the trajectories for a set of blowtorches to perform a cut pattern on a steel plate within minimum time. Moreira et al. [21] investigated a CPDP version where the problem was transformed into a dynamic rural postman problem. Rodrigues et al. [25] modeled another CPDP version as rural postman problem and proposed a heuristic method for the problem. Instances of CPDP can be formulated as OCARP instances by a polynomial transformation algorithm explained in [30].

3.4 Hybrid Genetic Algorithm

Genetic algorithms are metaheuristics inspired by the theory of evolution, using concepts such as natural selection, reproduction, genetic heritage and mutation [15]. Genetic algorithms with local search are called hybrid genetic algorithms. Several routing problems have been successfully addressed by hybrid genetic algorithms, some examples are: VRP (Vehicle Routing Problem) [21], CARP (Capacitated Arc Routing Problem) [16] and PCARP (Periodic Capacitated Arc Routing Problem) [17]. This paper proposes a hybrid genetic algorithm for the OCARP with the intent to overcome the feasibility issues presented by previous approaches [30, 28].

3.4.1 Algorithm Overview

The proposed HGA for OCARP is composed of the traditional genetic algorithm components (population initialization, selection and crossover) in addition to the following procedures: (i) feasibilization procedure, (ii) local search, (iii) population restart. The feasibilization procedure is responsible for obtaining a feasible solution from a chromosome. The local search explores the neighborhood of a feasible solution in an attempt to

improve the solution. The population restart avoids premature convergence of the population. The HGA does not employ mutation given the difficulty of finding and maintaining feasible OCARP solutions. Algorithm 2 presents the pseudocode.

Algorithm 2: HGA

Input: $G(V, E)$: instance graph; M: number of vehicles; D: vehicle capacity; **d**: demand vector; P: population size parameter; Q: inter-route local search parameter; W: local search filter parameter; K: restart interval parameter;
Output: *bestSol*: best solution found;
Objective: Search heuristically for a good OCARP solution within time limit.

```
begin
    SP ← distance_matrix_initialization(G); // Section 3.4.2
    (POP, bestSol) ← initial_population(SP, M, D, d, P, Q); // Section 3.4.5
    while time limit not exceeded do
        parents_selection_breeding(POP); // Section 3.4.6
        for pair of selected parents (parent1, parent2) do
            C ← crossover(parent1, parent2); // Section 3.4.7
            sol ← feasibilization(C, SP, M, D, d); // Section 3.4.8
            if sol is feasible then
                (sol', C') ← local_search(sol, SP, M, D, d, Q); // Section 3.4.9
                New individual added to population;
                if (cost(sol') < cost(bestSol)) then
                    bestSol ← sol';
        POP ← update_population(POP, SP, M, D, d, K, Q); // Section 3.4.10
    return (bestSol);
```

3.4.2 Distance Matrix Initialization

The first step of HGA is to initialize the matrix of distances between required arcs. Let E_R be the set of required edges and A_R be the set of required arcs, where for each required edge $\{i, j\} \in E_R$ there are two corresponding required arcs $(i, j), (j, i) \in A_R$. A matrix SP of dimensions $|A_R| \times |A_R|$ is computed such that each entry $SP[e, f]$ is the shortest path cost from the ending node of arc $e \in A_R$ to the starting node of arc $f \in A_R$. For sparse graphs (where $|E|$ is much less than $|V|^2$) the SP can be computed within $O(|V|^3)$ time and $O(|V|^2)$ space by using the Floyd-Warshall algorithm [6]. The SP allows HGA to retrieve the distances between required arcs in $O(1)$ time throughout the optimization process.

3.4.3 Chromosome Encoding

The genetic algorithm uses a non-capacitated route encoding for the chromosome. The chromosome is composed by a sequence of $|E_R|$ required arcs representing a single vehicle route of unlimited capacity that services all required edges $(i, j) \in E_R$ (Figure 3.2).

The non-required edges are implicitly inserted in the shortest path of successive pairs of required arcs. An OCARP solution associated to a chromosome is obtained only after the feasibilization procedure (Section 3.4.8).

This chromosome codification is an example of an *Indirect Solution Representation* (ISR) [23] with the benefit that any ISR can be optimally solved to obtain the best OCARP solution associated with it. This will be further discussed in Section 3.4.8.

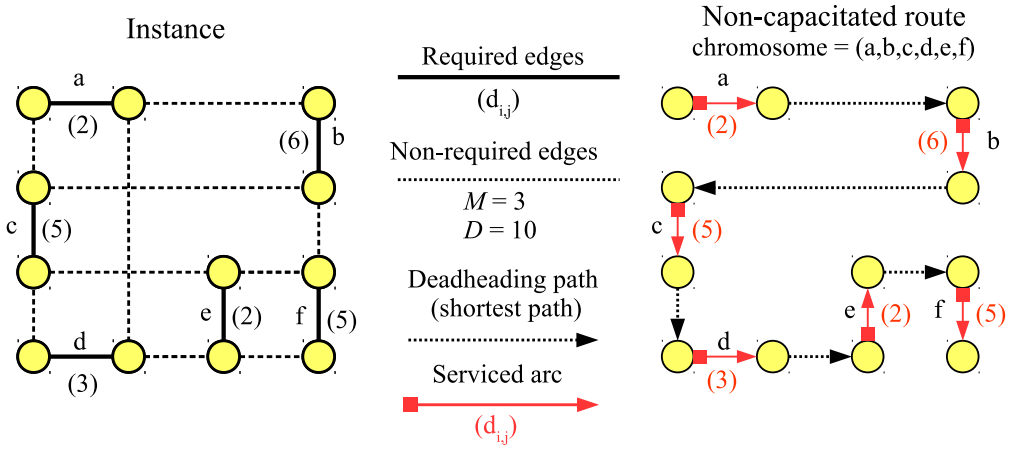

Figure 3.2: An instance and a corresponding chromosome.

3.4.4 Fitness

The fitness of a solution *sol* is defined as $1/(cost(sol) - LB_0)$, where LB_0 is a trivial lower bound: the sum of cost of all required edges. If any solution *sol* is found where $cost(sol) = LB_0$ then the HGA is halted since an optimal solution was found.

3.4.5 Initial Population

This procedure aims the initialization of P feasible individuals to fill the initial population of the HGA, where P is the population size. The construction of an individual has two steps. The first step constructs a non-capacitated route and the second step splits the non-capacitated route into capacitated sub-routes in an attempt to generate an OCARP solution.

First-step A non-capacitated route is constructed by a nearest neighbor heuristic. The route is initialized with a random required arc. The route is then augmented by iteratively adding the closest required arc, considering both route endpoints, until all E_R required edges are covered. Tie-breaking rule is to choose one of the closest required arcs at random. The non-capacitated route is then improved through 2-opt local search [13] until a local minimum is found. The nearest neighbor heuristic time complexity is $O(|E_R|^2)$ and each 2-opt iteration is $O(|E_R|^2)$. Therefore, the construction of a non-capacitated route is bounded by $O(c|E_R|^2)$, where c is an upper bound for the number of 2-opt iterations. Note that c is not actually employed by HGA but rather used exclusively for the analysis of complexity.

Second-step The feasibilization procedure (Section 3.4.8) attempt to split the non-capacitated route into M capacitated sub-routes. If the feasibilization is successful, the solution is inserted in the population. Otherwise, the infeasible individual is discarded.

The whole process is repeated until the population contains P feasible individuals. If initial population procedure is unable to generate P feasible individuals within time limit, the HGA is halted and the best feasible solution (if any) is returned.

3.4.6 Parents Selection for Breeding

At each generation, $P - 1$ crossovers are performed and each crossover requires the selection of two parent chromosomes to generate a single offspring.

The selection of the parents is made by the Stochastic Universal Selection [1]. In this technique a wheel is created and partitioned in sections, one associated with each individual and that have size proportional to the fitness of the individual. Then a single pointer is spun at random, and all parents are selected by selecting them at evenly spaced intervals starting from the random pointer. This method has the advantage to select the parents in a more predictable behavior than the Roulette Wheel Selection, reducing the bias of the selection operation [1].

3.4.7 Crossover

The $C1$ crossover operator, as described by Reeves [24], was adopted. First a point is chosen at random. The sequence of genes from the first parent to the left of the crossover point is copied to the offspring. The remaining genes are copied to the offspring in the same order they appear in the second parent. Figure 3.3 gives an example with $E_R = \{(a, b), (c, d), (e, f)(g, h), (i, j), (k, l)\}$.

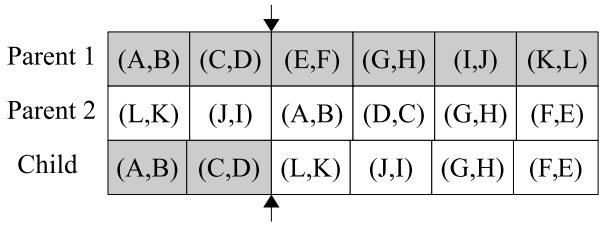

Figure 3.3: $C1$ Crossover operation.

3.4.8 Feasibilization

The feasibilization procedure starts with a non-capacitated route given by a chromosome C and tries to split it into capacitated sub-routes, using a *Split* algorithm. The *Split* algorithm was first proposed by Ulusoy [27] and makes an optimal decomposition of the non-capacitated route into a set of capacitated sub-routes. Since the OCARP considers a limited fleet, our algorithm adapted the *Split* to find the optimal set composed of up to M capacitated sub-routes, if there is at least one such decomposition. Therefore, the

output of *Split* is a set of M minimum-cost sub-routes, subject to vehicle capacity and that cover all required edges.

Algorithm 3 shows the *Split* algorithm while Figure 3.4 exemplifies the process. The *Split* algorithm starts by creating an auxiliary graph H. Figure 3.4(i) presents the instance and a chromosome C that is composed by a sequence of required arcs. Figure 3.4(ii) shows the cost of shortest path between each successive pair of required arcs. Figure 3.4(iii) shows the auxiliary graph H associated to C. The graph H is a directed acyclic graph (DAG) where each vertex represents a required arc from C and each arc represents a capacitated sub-route of C that is feasible with respect to the vehicle capacity. The cost of each arc of H is given by the cost of each corresponding sub-route.

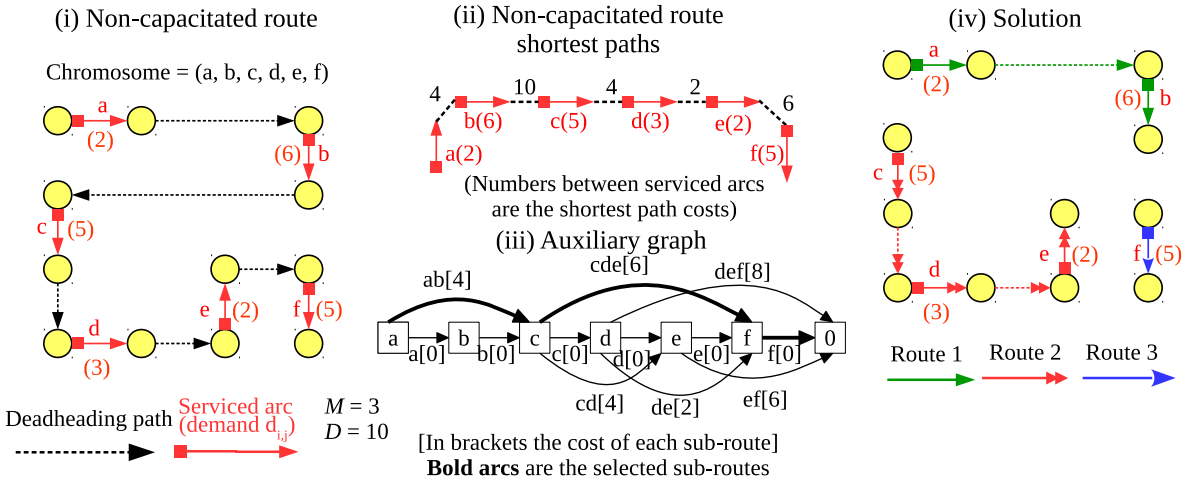

Figure 3.4: Operation of *Split* algorithm.

The *Split* algorithm finds the shortest path in H starting from the first vertex, in topological order, to the last vertex composed by at most M arcs. Since graph H is a DAG, the shortest path can be obtained by Algorithm 4, which is a straightforward adaptation of Bellman-Ford algorithm [6]. The OCARP solution is the set of routes associated to the arcs selected in the shortest path of H, as shown in Figure 3.4(iv).

If the *Split* algorithm fails to find a shortest path with at most M arcs, then another feasibilization procedure is attempted as explained next. Considering all capacitated sub-routes generated by the *Split* algorithm (arcs of H), a sub-route with the least remainder capacity is inserted into a new partial solution (example in Figure 3.5). Tie-breaking rule is to choose one sub-route at random. Then, a non-capacitated route is created containing all required edges still uncovered by the partial solution. This non-capacitated route is constructed by the same process employed in population initialization's first step. Finally, the *Split* algorithm is applied on the new non-capacitated route, this time using one less vehicle. If the algorithm find a feasible solution, the corresponding capacitated sub-routes join the partial solution into a complete OCARP solution. Otherwise, the

Algorithm 3: Split

Input: C: chromosome; **SP**: distance matrix; M: number of vehicles; D: vehicle capacity; **d**: demand vector;
Output: *sol*: solution; $H(V_H, A_H)$: auxiliary graph;
Objective: Find the optimal splitting of the non-capacitated route associated to chromosome C.

```
begin
    // Graph H(V_H, A_H) initialization
    Let C be defined as a sequence (a_1, ..., a_|C|) of required arcs;
    V_H ← ∅; A_H ← ∅; cost[] ← ∅;
    for i ← 1 to |C| do
        V_H ← V_H ∪ {a_i};
    V_H ← V_H ∪ {0}; // sink node
    for i ← 1 to |C| do
        // sub-routes attending exactly one required arc
        sum_demand ←d[a_i];
        sum_deadheading ← 0;
        A_H ← A_H ∪ {(a_i, a_{i+1})};
        cost[(a_i, a_{i+1})] ← 0;
        for j ← i + 1 to |C| do
            // sub-routes attending more than one required arc
            sum_demand ← sum_demand + d[a_j];
            sum_deadheading ← sum_deadheading + SP[a_{j-1}][a_j];
            if sum_demand ≤ D then
                A_H ← A_H ∪ {(a_i, a_{j+1})};
                cost[(a_i, a_{j+1})] ← sum_deadheading;
    DAG_shortest_path(V_H, A_H, cost, M, a_1); // Algorithm 4
    if exists shortest path with at most M arcs then
        sol ← OCARP solution associated to the found shortest path;
        return (sol, ∅);
    else
        return (∅, H);
```

process is repeated up to M iterations. If the *Split* algorithm does not find a feasible solution within M iterations, the chromosome C is declared infeasible and discarded. The feasibilization procedure pseudocode is shown in Algorithm 5.

Algorithm 4: DAG shortest path with at most M arcs.

Input: $H(V_H, A_H)$: directed acyclic graph; **cost**: cost vector for arcs $\in A_H$; M: number of vehicles; v_1: first vertex in topological order of $H(V_H, A_H)$;
Output: **dst**: distances; **pred**: predecessors;
Objective: Find shortest path in H from v_1 to other vertices using at most M arcs.
begin
  Initialize $\mathbf{dst}[v] \leftarrow \infty$ and $\mathbf{pred}[v] \leftarrow NIL$ for each $v \in V_H$.
  Initialize $\mathbf{dst}[v_1] \leftarrow 0$;
  for $k \leftarrow 1$ *to* M **do**
    for $v_i \in V_H$ in reverse topological order **do**
      for $(v_i, v_j) \in A_H$ **do**
        if $\boldsymbol{dst}[v_i] + \boldsymbol{cost}[(v_i, v_j)] < \boldsymbol{dst}[v_j]$ **then**
          $\mathbf{dst}[v_j] \leftarrow \mathbf{dst}[v_i] + \mathbf{cost}[(v_i, v_j)]$;
          $\mathbf{pred}[v_j] \leftarrow v_i$;

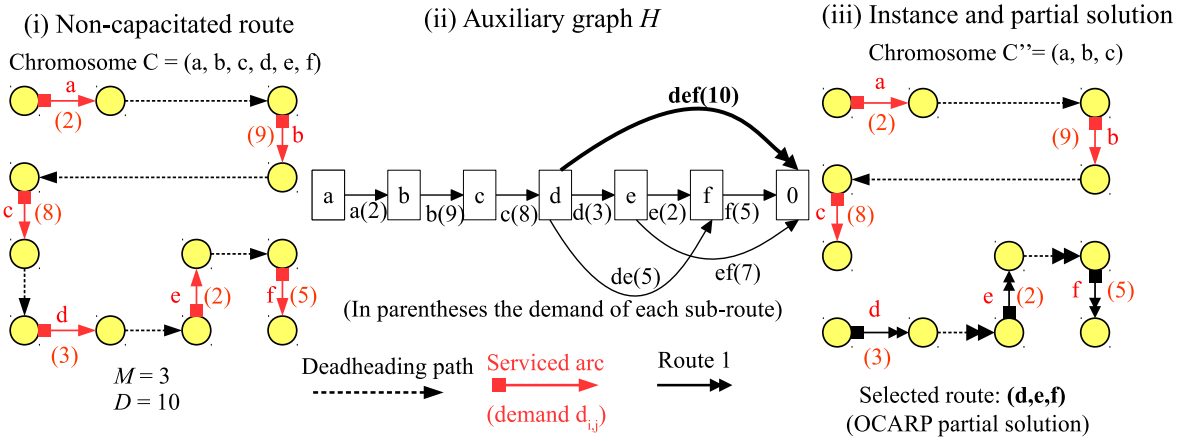

Figure 3.5: Feasibilization procedure operation: strategy to improve feasibility.

Algorithm 5: Feasibilization procedure

Input: C: chromosome; **SP**: distance matrix; M: number of vehicles; D: vehicle capacity; **d**: demand vector;
Output: *sol*: solution;
Objective: Generates an OCARP solution based on a non-capacitated route defined by C, using the Split algorithm and a feasibilization strategy.

```
begin
    C' ← C; sol ← ∅; found ← false;
    for k ← 0 to M − 1 do
        M' ← M − k;
        (sol', H) ← Split(C', SP, M', D, d); // Algorithm 3
        if sol' is feasible then
            sol ← sol ∪ sol';
            found ← true;
            k ← M; // Feasible solution found
        else
            Let R_max be a sub-route from H that services most demand;
            sol ← sol ∪ R_max; // Route R_max is added to partial solution.
            Create a non-capacitated route C'' using population initialization first step
              servicing only the required edges that are not serviced in partial solution sol;
            C' ← C'';
    if found is true then
        return sol; // Feasibitilization success
    else
        return ∅; // Feasibilization failed
```

The feasibilization procedure goal is to intensify the search for feasible solutions even when the *Split* algorithm alone is not enough to achieve feasible solutions. Additional experiments have shown that the *Split* algorithm alone has a high ratio of failure for large instances with very tight number of vehicles (Section 3.5.1).

A complexity analysis of Algorithms 3, 4 and 5 is described below. Algorithm 4 runs in $O(M|E_R|^2)$ since $|A_H|$ is bounded by $|E_R|^2$. The *Split* algorithm (Algorithm 3) first creates the auxiliary graph H in $O(|E_R|^2)$ and then calls Algorithm 4. Thus Algorithm 3 has complexity $O(M|E_R|^2)$. The feasibilization procedure (Algorithm 5) main loop is executed M times in the worst case (in practice it can be much less if the instance is not tight on the number of vehicles). At each iteration, it executes the *Split* algorithm and may call the creation of a new partial solution through the population initialization's first step (Section 3.4.5). The *Split* algorithm is $O(M|E_R|^2)$, while the population initialization's first step is $O(c|E_R|^2)$, where c is an upper bound of 2-opt local search iterations. Therefore, the whole feasibilization procedure worst case complexity is bounded by $O(M(M+c)|E_R|^2)$.

3.4.9 Local Search

Local search algorithms search for better solutions that are similar (neighboring) to the current solution. Local search is often the most computational expensive routine of metaheuristics, therefore it seems wise to apply local search only to solutions that are promising. This work uses a statistical filter for local search, proposed by Usberti et al.

[31], able to classify bad solutions within a certain confidence interval. A variable representing the improvement ratio between initial and local search solution costs is considered. The improvement ratio of a solution s is defined as $improvement(s) = cost(s)/cost(s')$, where s' is the local minimum solution obtained through s after local search. The filter starts by applying local search to the first W solutions $s_1, s_2, ..., s_W$ and storing a sampling of $improvement(s_i)$, $\forall i \in \{1, 2, ..., W\}$. From the sampling it is computed μ as average and ρ as standard deviation. Then this information is used to decide whether a solution is allowed to local search or not. A solution s is considered good enough for local search if it satisfies the following condition: $cost(s)/cost(s^*) \leq (\mu + 2\rho)$, where s^* is the incumbent solution. The filter gives a confidence interval of approximately 95% probability to reject solutions that cannot be improved by local search to overcome the incumbent solution s^*, assuming that $improvement(s)$ is an independent random variable with normal distribution.

The local search pseudocode is shown by Algorithm 6. The local search is composed by the following procedures: (i) the inter-route local search that reconstructs pair of routes, (ii) the intra-route local search that executes 2-opt for each route, and (iii) the *Split* local search that first executes the $Split^{-1}$ algorithm to make a new chromosome for the current solution and then executes *Split* algorithm which explores the solutions neighborhood by optimally splitting the chromosome.

Algorithm 6: Local Search

Input: *sol*: initial solution; **SP**: distance matrix; M: number of vehicles; D: vehicle capacity; **d**: demand vector; Q: local search set size parameter;
Output: *sol*: local search solution; C: chromosome of local search solution;
Objective: Improve an OCARP feasible solution.

```
begin
    intraCost ← ∞;
    while cost(sol) < intraCost do
        sol ← inter-route_local_search(sol, SP, M, D, d, Q); // Algorithm 7
        sol ← intra-route_local_search(sol, SP, M);
        intraCost ← cost(sol);
        C ← Split^-1(sol, SP, M); // Algorithm 8
        sol ← Split(C, SP, M, D, d); // Algorithm 3
    return (sol, C);
```

Inter-route The inter-route local search uses a deconstruct/reconstruct approach applied to pairs of nearby routes (Algorithm 7). Consider a set $S_{(u,v)}$ composed by the Q nearest routes to the required edge $(u, v) \in E_R$. Each pair of routes from set $S_{(u,v)}$ is submitted to the reconstruction phase, which starts by creating a new pair of routes servicing the same required edges. The new pair of routes is created using the same process employed by population initialization but servicing only the required edges from the original pair and using two vehicles. The new pair of routes replaces the original pair if a lower cost is achieved. The inter-route local search is applied for each set $S_{(u,v)}$, where $(u, v) \in E_R$. An example of the inter-route local search is presented in Figure 3.6, where the set of routes $S_a = \{r_1, r_3, r_4\}$ is reconstructed.

Intra-route The intra-route local search consists of the 2-opt local search [13] applied to each route individually. Each 2-opt local search is executed until a local minimum is achieved. Considering that each route is a sequence of required arcs linked by deadheading shortest paths, the 2-opt operator consists of removing two shortest path links and replacing them by two other links with lower costs and that maintains the route integrity.

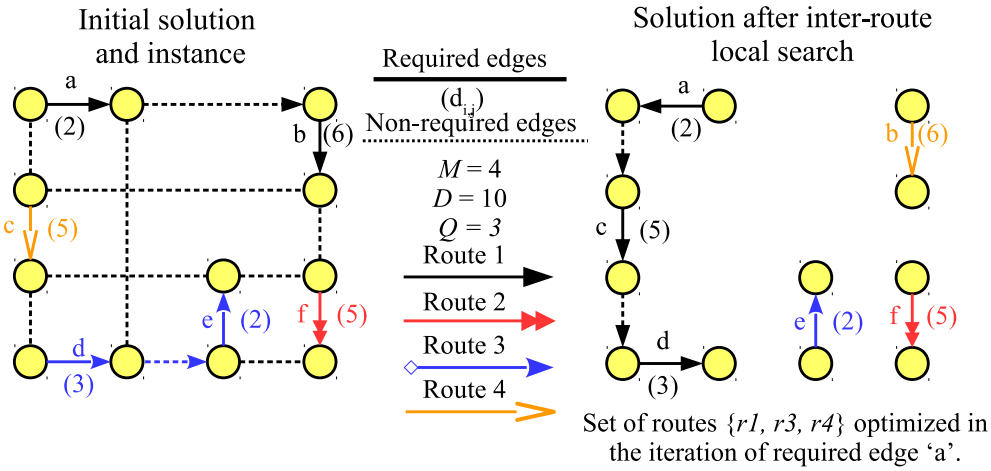

Figure 3.6: Example of inter-route local search optimization.

Split local search The $Split^{-1}$ takes a solution s and generates an associated chromosome. For this, the $Split^{-1}$ (Algorithm 8) operates on an undirected graph $I(V, E \cup E_s)$ where each edge $(u, v) \in E_s$ represents one route of s, where u is the starting vertex of the first required arc of that route and v is the ending vertex of the last required arc of the same route. On graph I a single non-capacitated route servicing the edges of E_s is constructed applying the first step of population initialization. The resulting non-capacitated route on graph I is transformed into a non-capacitated route on original graph G by replacing each edge of E_s by its corresponding route in s. The new non-capacitated route is then optimally split by the *Split* algorithm. It is worth noticing that the solution obtained by $Split(Split^{-1}(s))$ in this case is guaranteed to be feasible and to have a cost at least as good as s since the routes of s is a viable solution for the *Split* algorithm.

A complexity analysis of the local search procedure is provided. Each iteration of inter-route main loop creates and sorts a set $S_{(u,v)}$ in $O(|E_R| + MlogM)$, followed by the creation of $O(Q^2)$ new pairs of routes in the reconstruction phase (where Q is a fixed parameter). Each pair of routes is created and feasibilized in $O(c|E_R|^2)$, considering that c is an upper bound for 2-opt iterations. The complexity of each inter-route iteration is $O(|E_R|+MlogM+c|E_R|^2)$. A non-trivial OCARP problem has $M < |E_R|$, so the sum can be expressed simply as $O(c|E_R|^2)$. Therefore, the inter-route local search (Algorithm 7) complexity, including all $|E_R|$ main loop iterations, is $O(c|E_R|^3)$. The intra-route consists of the 2-opt local search operations on each route. Since there are $|E_R|$ required arcs distributed among the routes, the local search second step is bounded by $O(c|E_R|^2)$. The $Split^{-1}$ (Algorithm 8) is bounded by $O(cM^2)$ and the *Split* algorithm is $O(M|E_R|^2)$. The complexity of the entire local search procedure (Algorithm 6) is bounded by $O(c|E_R|^3)$

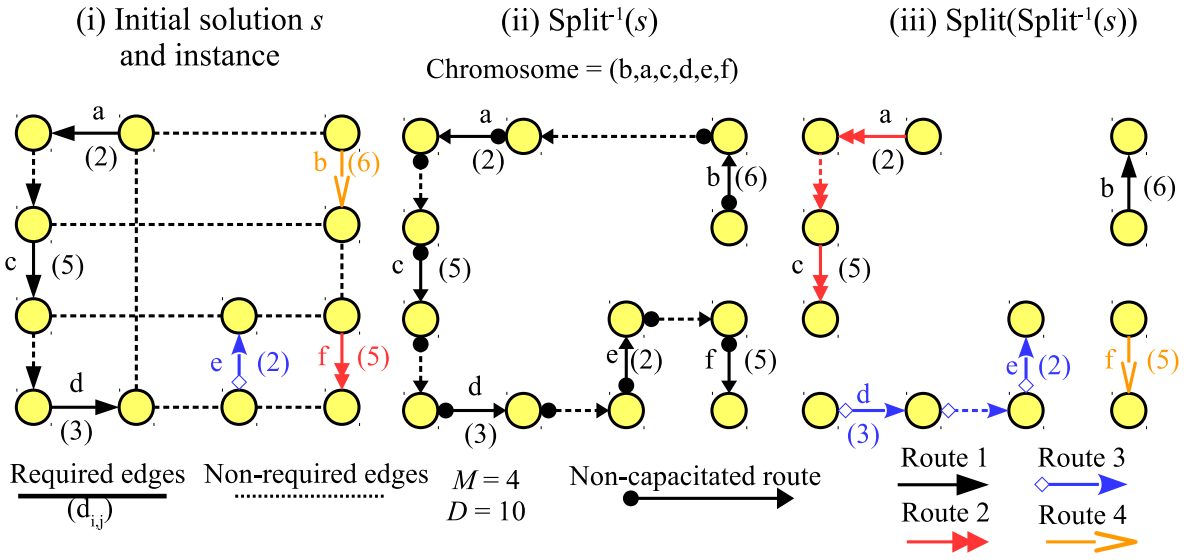

Figure 3.7: Example of $Split^{-1}$ followed by *Split* execution.

for each iteration.

Algorithm 7: Inter-route local search

Input: *sol*: initial solution; **SP**: distance matrix; M: number of vehicles; D: vehicle capacity; **d**: demand vector; Q: set size parameter;
Output: *sol*: local search solution;
Objective: Improve an OCARP feasible solution by reconstructing pair of routes;
begin
  Let *sol* be an array of routes $(r_1, r_2, ..., r_{|M|})$;
  for $(u, v) \in E_R$ **do**
    Let $\mathbf{dist}_{(\mathbf{u},\mathbf{v})}[1..M]$ be a vector with M empty positions;
    for $i \leftarrow 1$ *to* M **do**
      $distance \leftarrow +\infty$;
      for each required arc $(a, b) \neq (u, v)$ in the route r_i of solution *sol* **do**
        $distance \leftarrow \min\{distance, \mathbf{SP}[(u, v)][(a, b)], \mathbf{SP}[(u, v)][(b, a)], \mathbf{SP}[(v, u)][(a, b)], \mathbf{SP}[(v, u)][(b, a)]\}$;
      $\mathbf{dist}_{(\mathbf{u},\mathbf{v})}[i] \leftarrow (distance, i)$;
    Sort vector $\mathbf{dist}_{(\mathbf{u},\mathbf{v})}[1..M]$ by the distances in non-decreasing order;
    Let $S_{(u,v)}$ be the set of the first Q routes of *sol* in the order defined by $\mathbf{dist}_{(\mathbf{u},\mathbf{v})}[1..M]$;
    for each pair of routes r_i and r_j in $S_{(u,v)}$ **do**
      $ncr\_pair \leftarrow$ Create a non-capacitated route using initial population first step servicing the required edges in r_i and r_j;
      $sol\_pair \leftarrow$ feasibilization(ncr_pair, **SP**, 2, D, **d**); // Algorithm 5
      if routes of sol_pair have lower cost than r_i and r_j added **then**
        Replace routes r_i and r_j in *sol* by the routes of sol_pair;
  return *sol*;

Algorithm 8: Split^{-1}

Input: s: solution; **SP**: distance matrix; M: number of vehicles;
Output: C: chromosome representing s;
Objective: Create a chromosome representing an OCARP solution.
begin
  Initialize graph $I(V, E \cup E_s)$;
  $E_s \leftarrow \emptyset$;
  for $i \leftarrow 1$ *to* M **do**
    Add to E_s the edge (u, v) where u is the starting vertex of the first required arc of the i-th route of s and v the ending vertex of the last required arc of the same route;
  Create a non-capacitated route $ncr(s)$ on I using the first step method from population initialization servicing the edges of E_s;
  Create the chromosome (non-capacitated route) C on G associated to $ncr(s)$ by replacing each edge of E_s by its corresponding route of s;
  return C;

3.4.10 Population Update

The new population is updated for the next generation by including the best individual and the offspring. For each infeasible offspring, an individual from the current population is chosen at random to be inserted into the new population. Therefore, the population is always composed of P feasible individuals.

To maintain diversity, the HGA employs a population restart procedure executed every K generations. This procedure was necessary to avoid the population premature convergence, specially since HGA does not apply mutation. At each restart half of the population is replaced by new solutions. The discarded individuals are chosen at random, but the best solution is never selected. The new individuals are created by the same method employed by population initialization.

3.5 Computational Experiments

The computational experiments were conducted on a benchmark of CARP instances, which includes 23 *gdb*[12], 34 *val* [2], 24 *egl* [18], 32 *A* [14], and 24 *B* [14] instances, totaling 137 instances. The depot was considered a common node while the rest of the data left intact. Full experimental data, instances and HGA source code are available on-line[1].

The computational tests considered three classes of instances regarding the number of vehicles: $M = M^*$, $M = M^* + 1$ and $M = M^* + 2$, where M^* is the minimum number of vehicles required for a feasible solution. Consequently, 137 CARP instances and three different numbers of vehicles summed up 411 OCARP instances. Table 3.1 summarizes the data for the OCARP instances.

The experiment compared the results obtained by the HGA (Hybrid Genetic Algorithm) with two methods from literature: RPS (Reactive Path-scanning with ellipse rule) heuristic [30] and GRASP (Greedy Randomized Adaptive Search Procedure with path-relinking) metaheuristic [28].

[1] `http://www.ic.unicamp.br/~fusberti/problems/ocarp`

The RPS, GRASP and HGA were executed in an Intel Xeon X3430 2.4 GHz with 8 GB of RAM and Linux 64-bit operating system. HGA was implemented in C++ and uses the LEMON [7] library for graph algorithms. The RPS and GRASP were implemented in C and its source codes were provided by the authors. The parameters of HGA are given by Table 3.2, these values were selected after previous empirical tests.

Table 3.3 contains the overall results for each group-class of instances. The $Gap(\%) = 100*(UB-LB)/LB$ is the average deviation from lower bound, where upper bounds (UB) are provided by the heuristics and the lower bounds (LB) were reported by [29]. The column $Feas(\%)$ measures the percentage ratio of feasible solutions, while $FeasDiff(\%)$ measures the percentage of feasible and different solutions. Both ratios were calculated for the set of solutions obtained until the time limit or until one million solutions were generated. The field $CPU(s)$ shows the average processing time to attain the best solution (in seconds). All instances were processed for one hour by each heuristic.

Table 3.1: Benchmark instances for OCARP.

Instance set	$\|V\|$	$\|E\|$	$\|E_R\|$	M^*
gdb	7-27	11-55	11-55	3-10
val	24-50	34-97	34-97	2-10
egl	77-140	98-190	51-190	5-35
A	10-40	15-69	11-63	1-25
B	13-40	15-69	11-53	2-21

Table 3.2: Hybrid genetic algorithm parameters

Parameter	Description	Value
P	Population size	20
K	Population restart interval (generations)	150
W	Local search filter sampling size	100
Q	Inter-route local search set size	4

Table 3.3 shows that HGA obtained significantly better overall solutions than RPS and GRASP with less processing time. The experiment also confirmed that the value of the parameter M has meaningful impact on results, since all algorithms have a better performance as the value of M is increased from M^* to $M^* + 1$ and $M^* + 2$.

The results in Table 3.3 for *gdb* and *val* report that HGA can attain equal or better solutions with less processing time for small instances. The results with *egl* show the superior performance of HGA for larger instances. Especially for *egl* instances with $M = M^*$ class, which are the hardest instances, the deviation from lower bound $Gap(\%)$ was substantially reduced from 15.94% (GRASP) to 11.83% (HGA). For A and B instances the HGA shows a slightly worse but competitive performance in comparison to GRASP. The overall $Gap(\%)$ show that HGA performed better than GRASP and RPS.

From Table 3.3 it is worth noticing the difference between $Feas(\%)$ and $FeasDiff(\%)$ for each heuristic. The RPS has very similar $Feas(\%)$ and $FeasDiff(\%)$, explained by

Table 3.3: Computational experiment overall results.

		$Gap(\%)$			$Feas(\%)$			$FeasDiff(\%)$			$CPU(s)$		
group	class	RPS	GRASP	HGA	RPS	GRASP	HGA	RPS	GRASP	HGA	RPS	GRASP	HGA
gdb	M^*	0.09	**0.07**	**0.07**	77.45	85.50	98.55	77.38	78.50	97.38	172.73	0.3	< 0.1
	M^*+1	**0.05**	**0.05**	**0.05**	98.47	98.91	100.00	98.47	96.44	99.69	11.5	0.1	< 0.1
	M^*+2	0.05	**0.02**	**0.02**	99.95	100.00	100.00	99.95	97.73	99.65	14.5	0.1	< 0.1
	overall	0.06	**0.05**	**0.05**	91.96	94.81	99.52	91.93	90.89	98.91	66.2	0.2	< 0.1
val	M^*	3.83	3.19	**3.18**	83.81	87.58	99.22	83.73	22.47	87.66	658.4	170.1	32.1
	M^*+1	3.29	**2.24**	**2.24**	97.96	99.01	100.00	97.95	46.93	93.64	818.5	32.1	3.0
	M^*+2	3.30	**1.46**	**1.46**	99.70	100.00	100.00	99.69	46.82	93.39	791.4	13.2	1.1
	overall	3.48	**2.29**	**2.29**	93.82	95.53	99.74	93.79	38.74	91.56	756.1	71.8	12.0
egl	M^*	31.74	15.94	**11.83**	49.22	60.14	95.26	49.13	3.07	79.04	2066.4	1142.5	1424.0
	M^*+1	20.69	7.20	**6.75**	81.10	87.69	99.99	80.87	12.95	77.80	1844.4	923.6	951.4
	M^*+2	18.49	5.33	**5.19**	91.07	94.48	100.00	90.90	18.27	76.06	2057.9	1037.3	528.8
	overall	23.64	9.49	**7.92**	73.79	80.77	98.41	73.63	11.43	77.63	1989.6	1034.5	968.1
A	M^*	4.90	**3.54**	3.58	65.07	69.35	94.81	58.51	30.51	64.50	492.87	35.10	95.44
	M^*+1	3.20	**1.72**	**1.72**	84.48	87.47	97.66	78.72	60.64	86.76	439.74	16.16	23.30
	M^*+2	2.98	**1.14**	**1.14**	90.55	94.05	98.85	84.83	76.69	93.50	410.49	0.28	1.63
	overall	3.69	**2.13**	2.14	80.04	83.62	97.11	74.02	55.95	81.58	447.7	17.2	40.1
B	M^*	3.66	**2.91**	2.94	55.60	62.61	89.00	50.58	26.43	40.79	479.06	94.94	92.56
	M^*+1	1.34	**0.73**	**0.73**	79.74	87.74	97.70	76.77	63.66	75.85	316.74	0.73	1.33
	M^*+2	1.08	**0.23**	**0.23**	88.32	94.40	99.45	85.35	78.18	84.99	268.49	0.26	1.98
	overall	2.03	**1.29**	1.30	74.56	81.59	95.38	70.90	56.09	67.21	354.8	32.0	32.0
overall	M^*	8.31	4.93	**4.22**	67.14	73.60	95.56	64.66	30.54	73.97	753.44	267.25	295.95
	M^*+1	5.43	2.36	**2.27**	88.67	92.29	99.04	86.75	55.09	87.06	268.49	0.26	1.98
	M^*+2	4.95	1.61	**1.58**	94.06	96.64	99.63	92.16	62.57	89.88	702.28	185.12	93.63
	overall	6.23	2.96	**2.69**	83.29	87.51	98.08	81.19	49.40	83.64	714.0	208.7	187.6

In **bold**: best results regarding $Gap(\%)$.
$Gap(\%)$: average deviation from lower bound (%).
$Feas(\%)$: average ratio of feasible solutions (%).
$FeasDiff(\%)$: average ratio of feasible and different solutions (%).
$CPU(s)$: average time for the heuristic to attain its best solution (in seconds).

the fact that RPS does not employ local search which could in turn reduce solution variability. On the other hand, GRASP relies heavily on local search methods and therefore has $FeasDiff(\%)$ substantially lower than $Feas(\%)$. In overall, HGA has $FeasDiff(\%)$ higher than GRASP for all instances and is very competitive with RPS, where the HGA is better for $M = M^*$ and $M = M^* + 1$ while RPS is better for $M = M^* + 2$.

Table 3.4 presents the detailed results for the *egl* instances and class $M = M^*$. The number of required edges ($|E_R|$), number of vehicles (M^*), lower bound (LB), upper bound (UB), deviation from lower bound $Gap(\%)$ and the ratio of unique feasible solutions $FeasDiff(\%)$. In bold are the best upper bounds and in italics the instances for which HGA produced a major $Gap(\%)$ improvement ($> 5\%$).

Table 3.4 shows that HGA obtained the best upper bounds for all instances and improved the best known upper bounds for 16 instances. In special, HGA produced a major $Gap(\%)$ improvement over the highlighted (in italic) instances. The highlighted instances

Table 3.4: Detailed results for *egl* with $M = M^*$ instances.

				UB			$Gap(\%)$			$FeasDiff(\%)$		
instance	$\|E_R\|$	M^*	LB	RPS	GRASP	HGA	RPS	GRASP	HGA	RPS	GRASP	HGA
egl-e1-A	51	5	1673	1802	**1775**	**1775**	7.71	6.10	6.10	66.61	0.76	69.11
egl-e1-B	51	7	1591	1823	**1749**	**1749**	14.58	9.93	9.93	59.16	0.35	81.10
egl-e1-C	51	10	1523	1723	**1652**	**1652**	13.13	8.47	8.47	66.63	0.16	67.51
egl-e2-A	72	7	2019	2302	**2173**	**2173**	14.02	7.63	7.63	63.44	1.01	90.75
egl-e2-B	72	10	1944	2249	2079	**2062**	15.69	6.94	6.07	64.46	0.88	84.56
egl-e2-C	72	14	1900	2424	**2084**	**2084**	27.58	9.68	9.68	39.25	0.16	74.92
egl-e3-A	87	8	2277	2856	2541	**2533**	25.43	11.59	11.24	52.36	2.90	95.92
egl-e3-B	87	12	2221	2759	2438	**2409**	24.22	9.77	8.46	58.88	2.40	93.53
egl-e3-C	87	17	2188	2804	2362	**2357**	28.15	7.95	7.72	47.55	0.90	81.14
egl-e4-A	98	9	2453	3095	2656	**2631**	26.17	8.28	7.26	52.51	2.78	94.61
egl-e4-B	98	14	2453	3198	2720	**2708**	30.37	10.88	10.40	46.43	1.04	91.50
egl-e4-C	98	*19*	*2453*	*3789*	*3208*	***2812***	*54.46*	*30.78*	*14.64*	*0.14*	*0.04*	*22.48*
egl-s1-A	75	7	1584	1942	**1799**	**1799**	22.60	13.57	13.57	77.55	1.64	69.45
egl-s1-B	75	10	1475	1859	**1745**	**1745**	26.03	18.31	18.31	80.89	1.79	60.97
egl-s1-C	75	14	1415	2080	1891	**1874**	47.00	33.64	32.44	42.67	0.10	84.74
egl-s2-A	147	14	3228	4303	**3693**	**3693**	33.30	14.41	14.41	68.29	8.45	98.07
egl-s2-B	147	*20*	*3176*	*4979*	*4360*	***3775***	*56.77*	*37.28*	*18.86*	*3.65*	*0.55*	*87.78*
egl-s2-C	147	*27*	*3174*	*4812*	*3856*	***3691***	*51.61*	*21.49*	*16.29*	*7.73*	*0.78*	*84.06*
egl-s3-A	159	15	3393	4298	3824	**3808**	26.67	12.70	12.23	82.65	18.39	97.06
egl-s3-B	159	22	3379	4403	3742	**3677**	30.30	10.74	8.82	55.03	7.25	94.47
egl-s3-C	159	29	3379	4782	3758	**3733**	41.52	11.22	10.48	22.45	3.65	79.15
egl-s4-A	190	19	4186	5174	4522	**4486**	23.60	8.03	7.17	69.91	12.97	97.16
egl-s4-B	190	27	4186	5403	4497	**4393**	29.07	7.43	4.95	50.80	4.74	88.10
egl-s4-C	190	*35*	*4186*	*8023*	*6937*	***4971***	*91.66*	*65.72*	*18.75*	*0.01*	*0.01*	*8.70*

In **bold**: best results regarding UB.
In *italics*: instances that HGA produced a major $Gap(\%)$ improvement ($> 5\%$).
LB: lower bound reported by Usberti et al. [29]. UB: best solution cost;
$Gap(\%)$: deviation from lower bound (%).
$FeasDiff(\%)$: ratio of feasible and different solutions (%).

are also the most difficult ones regarding feasibility. Instance *egl-s4-C*, for example, has UB reduced from 6937 (GRASP) to 4971 (HGA), while $FeasDiff(\%)$ was increased from 0.01 (GRASP) to 8.70 (HGA). In conclusion, the improvement can be credited to HGA effectiveness of generating feasible solutions with reasonably variability when the number of vehicles is tight.

3.5.1 Feasibilization Procedure Evaluation

This section aims to evaluate the importance of the feasibilization procedure (Section 3.4.8) for the HGA performance. Let HGA′ be the HGA where the feasibilization procedure (Algorithm 5) is replaced by a single call of the *Split* algorithm (Algorithm 3) and, if *Split* fails, the solution is discarded. An experiment comparing HGA and HGA′ was executed using the same experimental settings in Section 3.5.

In Table 3.5 $n\_infeasible$ is the number of instances for which no feasible solution was found within the time limit. Column $Feas(\%)$ measures the percentage ratio of feasible solutions, while $FeasDiff(\%)$ measures the percentage ratio of feasible and different solutions. Both ratios were calculated for the set of solutions obtained until the time

limit or until one million solutions were generated.

The results in Table 3.5 show that HGA′ could not find feasible solutions for 17 instances with class $M = M^*$ and 4 instances for classes $M = M^* + 1$ and $M = M^* + 2$ combined. Table 3.5 shows that $Feas(\%)$ and $FeasDiff(\%)$ from HGA′ are substantially lower than the ratios from HGA, especially for *egl* instances with $M = M^*$ where a difference of $FeasDiff(\%)$ from 18.70% to 79.04% is reported. On the other hand, the $FeasDiff(\%)$ difference for the *egl* instances with $M = M^* + 2$ varies only from 66.00% to 76.06%.

As conclusion the feasibilization procedure is a key feature to make HGA competitive on instances with very tight number of vehicles. The strategy employed by feasibilization procedure is generic and its potential to be applied on more complex routing problems with either (i) small fixed number of vehicles, or (ii) very onerous vehicle cost, is an interesting research topic.

Table 3.5: Feasibilization procedure experiment results

		$n\_infeasible$		$Feas(\%)$		$FeasDiff(\%)$	
group	class	HGA′	HGA	HGA′	HGA	HGA′	HGA
gdb	M^*	**0**	**0**	71.04	98.55	70.69	97.38
	$M^* + 1$	**0**	**0**	96.71	99.01	96.40	96.44
	$M^* + 2$	**0**	**0**	100.00	100.00	99.65	99.65
	overall	**0**	**0**	89.25	99.52	88.91	98.91
val	M^*	**0**	**0**	88.40	99.22	75.88	87.66
	$M^* + 1$	**0**	**0**	99.02	100.00	92.17	93.64
	$M^* + 2$	**0**	**0**	99.02	100.00	93.61	93.39
	overall	**0**	**0**	95.81	99.74	87.22	91.56
egl	M^*	7	**0**	34.55	95.26	18.70	79.04
	$M^* + 1$	1	**0**	73.24	99.99	52.45	77.80
	$M^* + 2$	0	**0**	89.77	100.00	66.00	76.06
	overall	8	**0**	65.85	98.41	45.71	77.63
A	M^*	4	**0**	68.93	94.81	45.40	64.50
	$M^* + 1$	1	**0**	84.71	97.66	74.24	86.76
	$M^* + 2$	1	**0**	91.22	98.85	88.07	93.50
	overall	6	**0**	81.62	97.11	69.24	81.58
B	M^*	6	**0**	62.87	89.00	28.03	40.79
	$M^* + 1$	1	**0**	81.41	97.70	66.06	75.85
	$M^* + 2$	0	**0**	91.23	99.45	81.20	84.99
	overall	7	**0**	78.50	95.38	58.43	67.21
overall	M^*	17	**0**	66.77	94.67	49.12	73.97
	$M^* + 1$	3	**0**	87.60	99.04	76.99	87.06
	$M^* + 2$	1	**0**	94.58	99.63	86.22	89.88
	overall	21	**0**	82.98	97.78	70.78	83.64

$n\_infeasible$: number of instances with no feasible solutions.
$Feas(\%)$: average ratio of feasible solutions.
$FeasDiff(\%)$: average ratio of feasible and different solutions.

3.6 Conclusion

This paper proposed a hybrid genetic algorithm (HGA) for the open capacitated arc routing problem. The HGA is composed by the genetic algorithm combined with local search and feasibilization procedures. The key features of HGA are: (i) solutions are represented by a non-capacitated route; (ii) *Split* algorithm, which optimally splits a non-capacitated route into feasible routes; (iii) feasibilization procedure to tackle instances with a tight number of vehicles; (iv) inter-route local search with a deconstruct/reconstruct approach; (v) statistical filtering of solutions for local search.

The computational experiments, using a set of benchmark instances from literature, has shown that HGA outperformed state-of-the-art methods from literature for almost all instances. A major difficulty reported by previous approaches was the obtainability of feasible solutions for instances with very tight number of vehicles. The HGA performance depended on the feasibilization procedure to overcome the feasibility issue. Future research should focus on assessing the effectiveness of HGA methods if applied to more complex routing problems. The design of better lower bounding procedures in order to reduce the optimality gaps presented is also an interesting field of research.

3.7 Acknowledgment

This work was supported by grants #2016/00315-0, São Paulo Research Foundation (FAPESP) and 307472/2015-9, National Counsel of Technological and Scientific Development (CNPq). We also thank to two anonymous referees for their valuable comments.

References

[1] E. James Baker. Reducing bias and inefficiency in the selection algorithm. *Proceedings of the second international conference on genetic algorithms*, pages 14–21, 1986.

[2] Enrique Benavent, Vicente Campos, Angel Corberán, and Enrique Mota. The capacitated arc routing problem: lower bounds. *Networks*, 22:669–690, 1991.

[3] L. Bodin and L. Levy. The arc partitioning problem. *European Journal of Operational Research*, 53:393–401, 1991.

[4] A. Corberán and C. Prins. Recent results on arc routing problems: An annotated bibliography. *Networks*, 56(1):50–69, 2010.

[5] Ángel Corberán and Gilbert Laporte. *Arc Routing: Problems, Methods, and Applications*. SIAM, Philadelphia, PA, USA, 2014.

[6] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2nd edition, 2001.

[7] Balázs Dezs, Alpár Jüttner, and Péter Kovács. Lemon - an open source c++ graph template library. *Electronic Notes in Theoretical Computer Science*, 264(5):23–45, 2011.

[8] Moshe Dror. *Arc routing: theory, solutions and applications*. Springer Science & Business Media, New York, NY, USA, 2012.

[9] H. A. Eiselt, Michel Gendreau, and Gilbert Laporte. Arc Routing Problems, Part I: The Chinese Postman Problem. *Operations Research*, 43(2):231–242, 1995.

[10] Richard YK Fung, Ran Liu, and Zhibin Jiang. A memetic algorithm for the open capacitated arc routing problem. *Transportation Research Part E: Logistics and Transportation Review*, 50:53–67, 2013.

[11] B. L. Golden and R. T. Wong. Capacitated arc routing problems. *Networks*, 11:305–315, 1981.

[12] L. Bruce Golden, S. James DeArmon, and K. Edward Baker. Computational experiments with algorithms for a class of routing problems. *Computers and Operations Research*, 10:47–59, 1982.

[13] GW Groves and JH Van Vuuren. Efficient heuristics for the rural postman problem. *ORiON*, 21(1):33–51, 2005.

[14] Alain Hertz. *Recent trends in arc routing. In: Sharda R., Voß S., Golumbic M. C., Hartman IBA, editors. Graph theory, combinatorics and algorithms.* Springer US, 2005.

[15] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.* MIT Press, Cambridge, MA, USA, 1992.

[16] Philippe Lacomme, Christian Prins, and Wahiba Ramdane-Chérif. A genetic algorithm for the capacitated arc routing problem and its extensions. In *Applications of evolutionary computing*, pages 473–483. Springer, 2001.

[17] Philippe Lacomme, Christian Prins, and Wahiba Ramdane-Chérif. Evolutionary algorithms for periodic arc routing problems. *European Journal of Operational Research*, 165(2):535–553, 2005.

[18] YO Leon Li and W. Richard Eglese. An interactive algorithm for vehicle routeing for winter-gritting. *Journal of the Operational Research Society*, pages 217–228, 1996.

[19] L. M. Moreira, J. F. Oliveira, A. M. Gomes, and J. S. Ferreira. Heuristics for a dynamic rural postman problem. *Computers and Operations Research*, 34:3281–3294, 2007.

[20] Luc Muyldermans and Gu Pang. Variants of the capacitated arc routing problem. In Ángel Corberán and Gilbert Laporte, editors, *Arc Routing: Problems, Methods, and Applications*, chapter 10, pages 223–253. SIAM, 2014.

[21] Christian Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers and Operations Research*, 31(12):1985–2002, 2004.

[22] Christian Prins. The capacitated arc routing problem: Heuristics. In Ángel Corberán and Gilbert Laporte, editors, *Arc Routing: Problems, Methods, and Applications*, chapter 7, pages 131–157. SIAM, 2014.

[23] Christian Prins, Philippe Lacomme, and Caroline Prodhon. Order-first split-second methods for vehicle routing problems: A review. *Transportation Research Part C: Emerging Technologies*, 40:179–200, 2014.

[24] R. Colin Reeves. Feature article-genetic algorithms for the operations researcher. *INFORMS journal on computing*, 9:231–250, 1996.

[25] Ana Maria Rodrigues and José Soeiro Ferreira. Cutting path as a rural postman problem: solutions by memetic algorithms. *International Journal of Combinatorial Optimization Problems and Informatics*, 3(1):31–46, 2012.

[26] H. I. Stern and M. Dror. Routing electric meter readers. *Computers and Operations Research*, 6:209–223, 1979.

[27] Gündüz Ulusoy. The fleet size and mix problem for capacitated arc routing. *European Journal of Operational Research*, 22(3):329–337, 1985.

[28] Fábio Luiz Usberti. *Heuristic and exact approaches for the open capacitated arc routing problem*. PhD thesis, Universidade Estadual de Campinas, 2012.

[29] Fábio Luiz Usberti, Paulo Morelato França, and André Luiz Morelato França. *Branch-and-bound algorithm for an arc routing problem*. Annals XLIV SBPO, Rio de Janeiro, 2012.

[30] Fábio Luiz Usberti, Paulo Morelato França, and André Luiz Morelato França. The open capacitated arc routing problem. *Computers and Operations Research*, 38(11):1543 – 1555, 2011.

[31] Fábio Luiz Usberti, Paulo Morelato França, and André Luiz Morelato França. Grasp with evolutionary path-relinking for the capacitated arc routing problem. *Computers and Operations Research*, 40(12):3206–3217, 2013.

[32] J. Wunderlich, M. Collette, L. Levy, and L. Bodin. Scheduling meter readers for southern california gas company. *Interfaces*, 22:22–30, 1992.

Chapter 4

Lower bounding methods for the open capacitated arc routing problem: a branch-and-cut algorithm and a parameterized flow-based formulation

---

The text presented below is a preprint submitted to Computers and Operations Research in 2019 and it is co-authored with Fábio Luiz Usberti. In this manuscript we present two methods for obtaining lower bounds for the OCARP: (i) a branch-and-cut integer programming formulation with two sets of valid inequalities, and (ii) a parameterized flow-based integer linear programming formulation where the parameter can control how tight or relaxed are the lower bounds provided by the formulation. We show extensive computational experiments and also properties regarding relations between these formulations.

The *open capacitated arc routing problem* (OCARP) is a combinatorial optimization problem. Consider an undirected graph with demands scattered over the edges and a limited fleet of vehicles that are required to service the demands and which can start their routes in any node and also end in any node. Vehicles have limited capacity in relation to the amount of demands they can service. The objective is to find a set of routes that collectively service all demands within the minimum cost. The OCARP has been proved NP-hard. State-of-the-art solution methods developed for OCARP are heuristics, which can obtain non-trivial solutions within reasonable computational time. Nonetheless these solutions are unattached to any guarantee with respect to their quality. To address this matter, this work focuses on lower bounding procedures to assess the quality of heuristic solutions for OCARP. Two lower bounding methods are proposed: a branch-and-cut algorithm (B&C) and a flow formulation called $RFB(k)$. The B&C algorithm considers an one-index formulation with two sets of valid inequalities and their corresponding exact separation algorithms. The $RFB(k)$ is a relaxed model for OCARP parameterized by an integer k which expresses how tight ($k \to \infty$) or relaxed ($k \to 0$) the model is. We show $RFB(k = 0)$ is tighter than the one-index formulation and

$RFB(k = \infty)$ computes an optimal lower bound. Extensive computational experiments conducted on five sets of benchmark instances revealed the good performance of proposed methods which improved by a large margin the previous results from literature and proved optimality for more than 70% of the instances.

4.1 Introduction

The capacitated arc routing problem (CARP), proposed by Golden and Wong [15], aims to compute routes for a fleet of vehicles that are required to service some demands scattered over the edges of a network, where each vehicle has a limited capacity of servicing. The objective is to build a minimum cost set of routes such that all demands are serviced and the vehicles capacities are respected. In the CARP, all vehicles should start and end their routes in a central node (*depot*).

This work addresses the *open capacitated arc routing problem* (OCARP) [29]: a variation of CARP which does not consider a depot node. Routes are allowed to start and end at any node. Likewise the CARP [15], the OCARP has been proved *NP-hard* [29]. At least two problems that can be modelled by the OCARP can be found in the literature: the meter reader routing problem [27, 8] and the cutting path determination problem [21, 25, 26]. Procedures for representing these problems in the form of OCARP instances are discussed in [29].

The literature contains two exact methods for the OCARP: (i) an integer linear programming formulation with an exponential number of variables and constraints [29], and (ii) a branch-and-bound algorithm which exploited three different relaxed formulations for the problem [28]. Computational experiments revealed that both methods were able to solve only small-sized instances. This result is understandable since for the CARP, a similar but much more studied problem, the proposed exact methods seem also limited to medium-sized instances [6].

Regarding heuristic methods, two metaheuristics were proposed for the OCARP: (i) a *greedy randomized adaptive search procedure* (GRASP) [28], and (ii) a *hybrid genetic algorithm* (HGA) [3]. Both methods performed much better than the exact methods in terms of solution cost and processing time, especially for large instances. However, heuristic methods commonly does not provide any lower bounds and therefore cannot estimate the quality of obtained solutions. The development of lower bounding procedures comes as a tool to evaluate the performance of heuristic methods and to better assess heuristic solutions impact on real-world applications, which usually involve large instances.

This work proposes two lower bounding methods for the OCARP: (i) a *branch-and-cut* algorithm (B&C) and (ii) a parameterized flow-based formulation called $RFB(k)$. The B&C was based on the branch-and-cut algorithm originally proposed for CARP by Belenguer and Benavent [5] and further improved by works [1] and [20]. The $RFB(k)$ is a relaxed formulation which can be more or less tight based on an integer parameter $k \in \mathbb{Z}_{\geq 0}$, where $k = 0$ is the most relaxed model and the $k = \infty$ version is the tightest model which we show to be a valid formulation for OCARP.

Extensive computational experiments over benchmark instances revealed that the pro-

posed methods consistently outperformed with a large margin all the existing lower bounding procedures for the OCARP. Computational experiments involving the parametrization of $RFB(k)$ were also investigated.

This article is organized as follows. Section 4.2 defines the OCARP. Section 4.3 presents a literature review. Section 4.4 describes the proposed branch-and-cut algorithm. Section 4.5 presents the parameterized flow-based formulation. Section 4.6 shows the computational experiments and discusses the methodologies performance. Section 4.7 contains some concluding comments.

4.2 Problem Definition

The open capacitated arc routing problem (OCARP) proposed by Usberti et al. [29] is next defined. Let $G(V, E)$ be undirected connected graph with costs $c_{ij} \geq 0$ and demands $d_{ij} \geq 0$ assigned to each edge $(i, j) \in E$. The set of edges with positive demand $E_R = \{(i, j) \in E : d_{ij} > 0\}$ is called the set of required edges. A group of M homogeneous vehicles, each with limited capacity D, is considered to service the demands. In order for an edge $(i, j) \in E$ be *serviced*, it needs to be visited by some vehicle, which increases the solution cost by c_{ij} and decreases the assigned vehicle capacity by d_{ij}. A vehicle is also allowed to visit an edge $(i, j) \in E$ without servicing it, which is called *deadheading*, and this only increases the solution cost by c_{ij} with no effect for the vehicle capacity. A vehicle route is defined by a path of edges in G traversed by the vehicle. Routes are allowed to start in any node and also end in any node.

In summary, feasible OCARP solutions is a set of up to M routes such that all required edges are serviced and the vehicles limited capacities are not violated (Figure 4.1). The OCARP aims to find the lowest cost feasible solution.

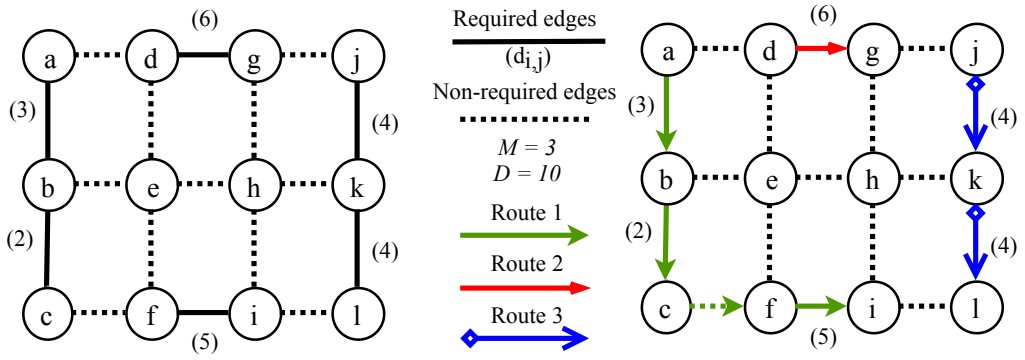

Figure 4.1: OCARP: an instance and a feasible solution.

The CARP and OCARP differ because the OCARP allows open or closed routes, while the CARP allows only closed routes that start and end in a special node (*depot*). Moreover, many heuristic methods proposed for CARP interpret that the number of vehicles (M) is not restricted to any upper bound, while for OCARP the value of M is necessarily limited, otherwise the OCARP would be trivially solvable [29].

4.3 Previous Work

The first mathematical model proposed for the capacitated arc routing problem (CARP) was an integer linear programming (ILP) formulation given by Golden and Wong [15] in 1981. This formulation required an exponential number of constraints and presented a very weak linear relaxation lower bound. Two decades later, Belenguer and Benavent [5] proposed a polynomial-size formulation with $O(|E|)$ integer variables that produced good lower bounds while spending a moderate computational time. This formulation inspired our branch-and-cut algorithm for the OCARP described in Section 4.4.

The literature related to exact, heuristic and lower bounding methods for the CARP is quite extensive. For a comprehensive survey we refer the interested reader for the references [10, 24, 2, 6, 22] .

The open capacitated arc routing problem (OCARP) was proposed by Usberti et al. [29]. In the seminal paper the authors proposed an integer linear programming (ILP) formulation. In a following work, Usberti [28] proposed a branch-and-bound algorithm whose results were competitive to the ILP formulation regarding solutions costs and lower bounds. This algorithm formulated the OCARP as a special case of the Capacity and Degree Constrained Minimum Spanning Forest Problem (CDCMSFP). This method main feature was to solve three relaxed versions of CDCMSFP in order to obtain valid lower bounds for OCARP. Regarding the performance, these exact methods were able to solve only small sized instances ($|E_R| \leq 50$) and a few of the medium sized instances ($50 \leq |E_R| \leq 100$).

In relation to heuristic methods, Usberti et al. [29] proposed a reactive path-scanning with ellipse rule (RPS) for the OCARP. This heuristic consisted of a bias-randomized greedy heuristic with some parameters that were automatically adjusted depending on the instance characteristics. The RPS obtained non-trivial feasible solutions with little computational effort. One year later, Usberti [28] presented a metaheuristic method of greedy randomized adaptive search procedure (GRASP) with path-relinking that improved with a large margin the previous method. After these methods were evaluated, the set of largest instances *egl* still exhibited an average optimality gap of 9.49% and the hardest instance exhibited a gap of 86.07%.

More recently Arakaki et al. [3] presented a hybrid genetic algorithm (HGA) for the OCARP. This method consisted of a genetic algorithm hybridized with local search procedures and a feasibilization procedure. The authors stated that the feasibilization procedure was especially important to find solutions when the number of vehicles is very tight: in these instances each vehicle needs to occupy its limited capacity wisely otherwise some demands will not be able to be serviced. The HGA was shown to surpass previous methods in terms of optimality gap. For the *egl* instances it exhibited an average resulting gap of 7.92% and the hardest instance exhibited a gap of 32.44%.

It is also worth to note that Fung et al. [13] presented a different problem also called open capacitated arc routing problem. Their version of OCARP is focused in an application for waste collection. Their problem definition is very different from ours since: (i) a directed complete graph is considered and demands are located in some of the directed arcs; (ii) routes start in a depot but can end in any node; (iii) each vehicle has a

limited capacity for demand serving and also a limited span for travelling distances; and (iv) a solution can employ any number of vehicles where each vehicle used sums a fixed cost in the objective function.

4.4 Branch-and-cut for the OCARP

The branch-and-cut algorithm proposed for the OCARP is presented in this section. The model is presented in Section 4.4.1. Separation algorithms for the two sets of valid inequalities, called *odd edge cut-set constraints* and *capacity constraints*, are described in Sections 4.4.2 and 4.4.3 respectively.

4.4.1 One-index formulation for the OCARP

Inspired by the work of Belenguer and Benavent [5] for the CARP, here we propose the *one-index formulation* for the OCARP. This relaxed formulation considers only one class of variables x_e that represents the number of times an edge e is deadheaded (i.e. when a vehicle visit an edge without servicing it) aggregating all vehicles. Since solutions for this formulation have all vehicles aggregated they do not necessarily correspond to a feasible solution for OCARP. Details regarding this difficulty in the case of CARP are discussed in [5] but applies by the same reasons for the OCARP. Nonetheless, these issues does not prevent the formulation to compute good lower bounds at moderate computational cost.

Let $G^+(V^+, E^+)$ be a supergraph of $G(V, E)$ where $V^+ = V \cup \{v^+\}$ and v^+ is a dummy node and $E^+ = E \cup \{(v^+, v_i) : v_i \in V\}$ includes a set of dummy edges connecting v^+ to every node in G^+ with zero cost and zero demand. An example of G^+ is given by Figure 4.2. From that, one can observe that open routes in G can be represented in G^+ by closed routes that visit v^+ exactly once. In this formulation, v^+ will represent a special node that all routes should be connected to, visiting it exactly once.

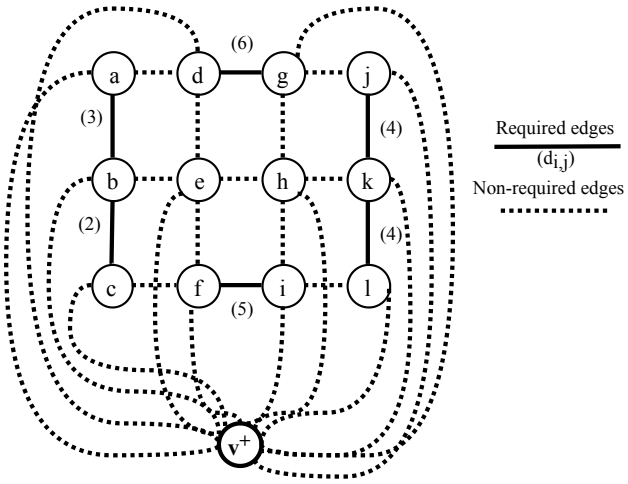

Figure 4.2: Graph G^+ example for the instance in Figure 4.1.

Let $S \subseteq V^+ \setminus \{v^+\}$ be a set of nodes excluding v^+. Let the cut-set $\delta(S) = \{(i, j) \in E^+ : i \in S$ **and** $j \notin S\}$ be the set of edges that have one endpoint in S and the other

outside S (i.e. cut edges of S). Also, let that $\delta_R(S) = \delta(S) \cap E_R$ be the set of cut edges of S with positive demand (*required edges*). Similarly, we define $E^+(S) = \{(i,j) \in E^+ : i \in S \textbf{ and } j \in S\}$ as the sets of edges such that both nodes are in S and $E_R^+(S) = E^+(S) \cap E_R$ as the set of *required edges* with both nodes in S.

Consider a set S such that $|\delta_R(S)|$ is odd. One can observe at least one edge in $\delta(S)$ must be deadheaded such that the vehicles collectively can service the required edges in $\delta_R(S)$ considering that they need to enter and leave S. This gives the reasoning for the *odd edge cut-set constraints*:

$$\sum_{e \in \delta(S)} x_e \geq 1 \qquad \forall (S \subseteq V^+ \setminus \{v^+\} \text{ where } |\delta_R(S)| \text{ is odd}) \tag{1}$$

Now consider a set S with multiple required edges in $E_R^+(S) \cup \delta_R(S)$. A lower bound on the number of vehicles needed to service these demands is $k(S) = \lceil \sum_{(i,j) \in E_R^+(S) \cup \delta_R(S)} d_{ij}/D \rceil$ because of vehicles limited capacities. Since each vehicle must enter and leave S at least once, at least $2k(S) - |\delta_R(S)|$ edges must be deadheaded. From that, we can define the *capacity constraints*:

$$\sum_{e \in \delta(S)} x_e \geq 2k(S) - |\delta_R(S)| \qquad \forall S \subseteq V^+ \setminus \{v^+\} \tag{2}$$

Considering that the left-hand side from constraints (1) and (2) are the same, a combination of them in just one set of constraints is possible. This combination considers the greater right-hand side computed from both. To achieve that, we define $\alpha(S)$ as following:

$$\alpha(S) = \begin{cases} max\{2k(S) - |\delta_R(S)|, 1\} & \text{if } |\delta_R(S)| \text{ is odd,} \\ max\{2k(S) - |\delta_R(S)|, 0\} & \text{if } |\delta_R(S)| \text{ is even.} \end{cases} \tag{3}$$

Finally, the one-index formulation for OCARP is defined:

$$MIN \quad \sum_{(i,j) \in E^+} c_{ij} x_{ij} \tag{4}$$

s.t.

$$\sum_{e \in \delta(v^+)} x_e \leq 2M \tag{5}$$

$$\sum_{e \in \delta(S)} x_e \geq \alpha(S) \qquad \forall S \subseteq V^+ \setminus \{v^+\} \tag{6}$$

$$x_e \in \mathbb{Z}_{\geq 0}, \qquad \forall e \in E^+ \tag{7}$$

Function (4) minimizes the cost of deadheading edges. Note that the real cost also involves the cost of visiting each *required edge* once, but this is a fixed cost for every feasible solution. Constraint (5) define the maximum number of routes allowed for the solution. The constraints (6) combine both the *odd edge cut-set constraints* and *capacity constraints*.

4.4.2 Separation algorithm for odd edge cut-set constraints

This section provides an exact separation algorithm for the *odd edge cut-set constraints* that aim to identify a set of constraints violated by a linear solution. This procedure uses the *odd minimum cut-set algorithm* from Padberg and Rao [23].

Given a linear solution x^* of the formulation, let each edge $e \in E^+$ have capacity x_e^*; moreover, label as *odd* all nodes in G^+ that have an odd number of incident *required edges*. An *odd cut* $\delta(S)$ is an edge cut-set that contains an odd number of odd nodes pertaining to $S \subseteq V^+ \setminus \{v^+\}$.

From these capacities on the edges, the odd minimum cut-set algorithm computes a *Gomory-Hu tree* [17] where the odd nodes are called terminals. This algorithm creates a tree such that each node correspond to a subset of nodes of G^+ containing exactly one odd node. This tree is called *maximum flow tree*, from which one can obtain a maximum flow (or minimum cut) of any pair of odd nodes by searching the least cost edge that separate the two odd nodes in the tree. In order to identify a violated *odd edge cut-set constraints* one just need to search for the edges weighted less than one that represents a cut $\delta(S)$ where $|\delta_R(S)|$ is odd. Even considering that regarding the *odd edge cut-set constraints* we would need to check only for cuts where $|\delta_R(S)|$ is odd, our algorithm strategy is to check all edges of the tree and add the combined constraints (4.6) whenever they are found to be violated. More specifically, after the *Gomory-Hu tree* is computed, each cut corresponding to an edge of the tree will be checked to violate or not the constraints (4.6) and each violating constraint (4.6) found will be added to the model.

The separation algorithm executes up to $|V| - 1$ times a maximum flow (or minimum cut) algorithm. Our method adopted the Preflow algorithm [14] which has a worst case time complexity of $O(|V|^2\sqrt{|E|})$. Therefore, the whole procedure complexity is $O(|V|^3\sqrt{|E|})$.

4.4.3 Separation algorithm for capacity constraints

Inspired by the work of Ahr [1], Martinelli et al. [20] proposed an exact separation method for the *capacity constraints* in the case of CARP. This method can be used for our OCARP one-index formulation. Here we describe the separation algorithm for the *capacity constraints*.

Given a linear solution x^*, the separation algorithm should find a set $S \subseteq V^+ \setminus \{v^+\}$ such that the capacity constraint is violated. More specifically, this algorithm executes a *mixed integer linear programming* (MILP) formulation that searches for the set S that has a capacity constraint that is most violated by x^* (i.e. difference between right-hand side and left-hand side is the greatest).

The MILP formulation is composed by five sets of variables: (i) $h_e \in \{0,1\}, \forall e \in E^+$ is 1 when edge e is a cut edge, i.e. one endpoint is in the cut S and the other is outside; (ii) $f_e \in \{0,1\}, \forall e \in E^+$ is 1 when both incident nodes are within S, and 0 otherwise; (iii) $s_i \in [0,1], \forall i \in V$ is 1 if node $i \in S$ and 0 otherwise; (iv) $k \in \mathbb{Z}_0^+$ is the minimum number of vehicles to service the demands in the cut S; and (v) $\gamma \in [0, 1-\beta]$ is a slack variable, where $\beta \ll 1$ is a fixed small value greater than zero. From these variables, the MILP *capacity constraints* separation formulation is defined:

$$MAX \quad 2k - \sum_{e \in E_R} h_e - \sum_{(i,j) \in E^+} x^*_{ij} h_e \tag{8}$$

s.t.

$$h_e - s_i + s_j \geq 0 \qquad \forall e = (i,j) \in E^+ \tag{9}$$

$$h_e + s_i - s_j \geq 0 \qquad \forall e = (i,j) \in E^+ \tag{10}$$

$$-h_e + s_i + s_j \geq 0 \qquad \forall e = (i,j) \in E^+ \tag{11}$$

$$s_i - f_e \geq 0 \qquad \forall e = (i,j) \in E^+ \tag{12}$$

$$s_j - f_e \geq 0 \qquad \forall e = (i,j) \in E^+ \tag{13}$$

$$s_i + s_j - f_e \leq 1 \qquad \forall e = (i,j) \in E^+ \tag{14}$$

$$\sum_{e \in \delta(\{i\})} (h_e + f_e) - s_i \geq 0 \qquad \forall i \in V^+ \tag{15}$$

$$h_e + f_e \leq 1 \qquad \forall e = (i,j) \in E^+ \tag{16}$$

$$s_{v^+} = 0 \tag{17}$$

$$k = \sum_{e \in E^+} \frac{d_e(h_e + f_e)}{Q} + \gamma \tag{18}$$

$$h_e, f_e \in \{0,1\} \qquad \forall e \in E^+ \tag{19}$$

$$s_i \in [0,1] \qquad \forall i \in V^+ \setminus \{v^+\} \tag{20}$$

$$k \in \mathbb{Z}^+_{\geq 0} \tag{21}$$

$$\gamma \in [0, 1-\beta] \tag{22}$$

Function (8) maximizes the capacity constraint violation. Constraints (9), (10) and (11) couple the variables s_i and h_e. Similarly, constraints (12), (13) and (14) couple variables s_i and f_e. Constraints (15) establish that if node i is inside the cut then at least one incident edge must be assigned as inside the cut or cut edge. Constraints (16) forbid an edge to be inside the cut and be a cut edge at the same time. Constraint (17) prevents the depot to be included in the cut. Constraint (18) defines a lower bound for the number of vehicles needed to service edge demands in the cut, using γ as slack variable for a ceiling function. Constraints (19), (20), (21) and (22) define the variables domains. It is interesting to note that s_i variables do not need to be defined integral since their coupling with h_e and f_e already guarantee this.

4.5 Parameterized flow-based formulation

The parameterized flow-based formulation $RFB(k)$ is executed over a directed graph $G^k(V \cup V' \cup \{v_0\}, A \cup A'_R \cup A'_{in} \cup A^k_{out} \cup A_{v_0})$ based on the original undirected graph $G(V,E)$ and an integer parameter $k \geq 0$. The augmented graph G^k includes all nodes and edges of G such that each undirected edge $\{v_i, v_j\} \in E$ is represented by two opposite directed arcs $(v_i, v_j), (v_j, v_i) \in A$ with the same cost. The sets V', A'_R, A'_{in}, A^k_{out} contain artificial nodes, artificial required arcs and artificial arcs that go in and out the artificial nodes,

respectively. These sets are described as following.

For each required edge $(v_i, v_j) \in E_R$ the following elements are created: (i) two artificial nodes $v'_i, v'_j \in V'$; (ii) a pair of artificial required arcs $(v'_i, v'_j), (v'_j, v'_i) \in A'_R$ with the same cost as $(v_i, v_j) \in E_R$; (iii) two artificial directed arcs $(v_i, v'_i), (v_j, v'_j) \in A'_{in}$ with zero cost that goes from original nodes to corresponding artificial nodes. At this step we have arcs going from original nodes in V to artificial nodes in V' but not the other way around. We call this partial graph $G_p(V \cup V', A \cup A'_R \cup A'_{in})$.

The set of arcs A^k_{out}, which contains all artificial arcs exiting artificial nodes, is parameterized by an integer $k \in \mathbb{Z}_{\geq 0}$. To construct this set, for each artificial node $v'_i \in V'$, such that $\exists (v'_i, v'_j) \in A'_R$, we execute a *breath first search* in the partial graph G_p starting at the associated original node $v_i \in V$ and limited to a depth of k. This procedure will generate a search tree identifying all nodes reachable from $v_i \in V$ by using up to k edges. This search tree will identify two types of nodes: (i) artificial nodes and (ii) original nodes. First, for each artificial node v'_a found, where $v'_a \neq v'_i$ and $v'_a \neq v'_j$, we create an artificial arc $(v'_i, v'_a) \in A^k_{out}$ with cost equal to the shortest path cost from $v_i \in V$ to $v_a \in V$ in G. Second, for every original node v_o found to be at exactly k edges of distance from v_i we create an artificial edge $(v'_i, v_o) \in A^k_{out}$ with cost equal to the shortest path cost from v_i to v_o in G. After defining A^k_{out}, we call this partial graph $G^k_p(V \cup V', A \cup A'_R \cup A'_{in} \cup A^k_{out})$. It is worth to note that by the way A^k_{out} is defined, the graph G^k_p has shortest paths between any pair of artificial nodes $v'_a, v'_b \in V'$ with cost equal to the shortest path costs between the corresponding original nodes $v_a, v_b \in V$ in G, for any $k \in \{0, 1, ..., \infty\}$.

Finally, we consider a special node v_0 that is connected to all artificial nodes in V' in both directions by a set of arcs $A_{v_0} = \{(v_0, v'_i), (v'_i, v_0) : v'_i \in V'\}$ with zero cost. These procedures conclude the computation of the augmented graph G^k. An example is given by Figures 4.3 and 4.4 for $G^{k=0}$ and $G^{k=2}$, respectively, for the OCARP instance given by Figure 4.1.

The $RFB(k)$ represents a network flow problem in which demands are located on the artificial nodes of each artificial required arc. Each arc $(v'_i, v'_j) \in A'_R$ has one arbitrary node chosen, say v'_i, to have demand d'_i equal to the demand of the required edge $(i, j) \in E_R$, while the demand d'_j of the other node is zero. The network flow represents the aggregated remaining vehicle capacities, starting from v_0, serving the node demands and going back to v_0. Node v_0 is the source of all flows. Up to M arcs with up to D units of demand each leave v_0, each one representing the starting of one vehicle route.

Two sets of variables are defined for the model: (i) x_{ij} represents the number of times arc (i, j) is visited by any route, and (ii) y_{ij} represents the amount of flow traversing arc (i, j). Functions $\delta^+(i)$ and $\delta^-(i)$ are defined as the sets of directed arcs which leave and enter node v_i respectively. From that, the MILP flow-based formulation for the OCARP is defined:

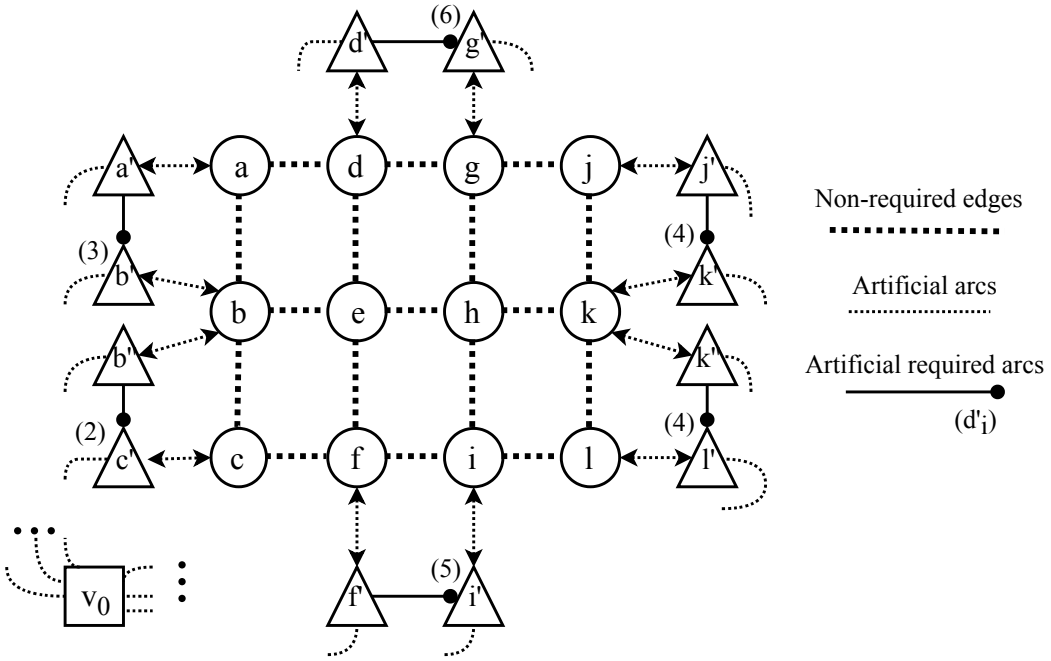

Figure 4.3: Example of $G^{k=0}$ for the instance in Figure 4.1. Arcs connecting v_0 are partially hidden.

$$MIN \quad \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{1}$$

s.a.

$$\sum_{j\in\delta^+(i)} x_{ij} = 1 \qquad (i \in V') \tag{2}$$

$$\sum_{j\in\delta^-(i)} x_{ji} = 1 \qquad (i \in V') \tag{3}$$

$$x_{ij} + x_{ji} = 1 \qquad ((i,j),(j,i) \in A'_R) \tag{4}$$

$$\sum_{j\in\delta^+(i)} x_{ij} - \sum_{j\in\delta^-(i)} x_{ji} = 0 \qquad (i \in V \cup \{v_0\}) \tag{5}$$

$$\sum_{j\in\delta^+(0)} x_{0j} \leq M \tag{6}$$

$$\sum_{j\in\delta^-(i)} y_{ji} - \sum_{j\in\delta^+(i)} y_{ij} = d_i \qquad (i \in V \cup V') \tag{7}$$

$$y_{ij} \leq (D - d_i)x_{ij} \qquad ((i,j) \in A \cup A'_R \cup A'_{in} \cup A^k_{out} \cup A_{v_0}) \tag{8}$$

$$x_{ij} \in \mathbb{Z}_{\geq 0}, \qquad ((i,j) \in A \cup A'_R \cup A'_{in} \cup A^k_{out} \cup A_{v_0}) \tag{9}$$

$$y_{ij} \in \mathbb{R}_{\geq 0}, \qquad ((i,j) \in A \cup A'_R \cup A'_{in} \cup A^k_{out} \cup A_{v_0}) \tag{10}$$

The objective function (1) minimizes the cost of the arcs in which the flows will stream on. Constraints (2) and (3) force artificial nodes to be visited exactly once. Constraints

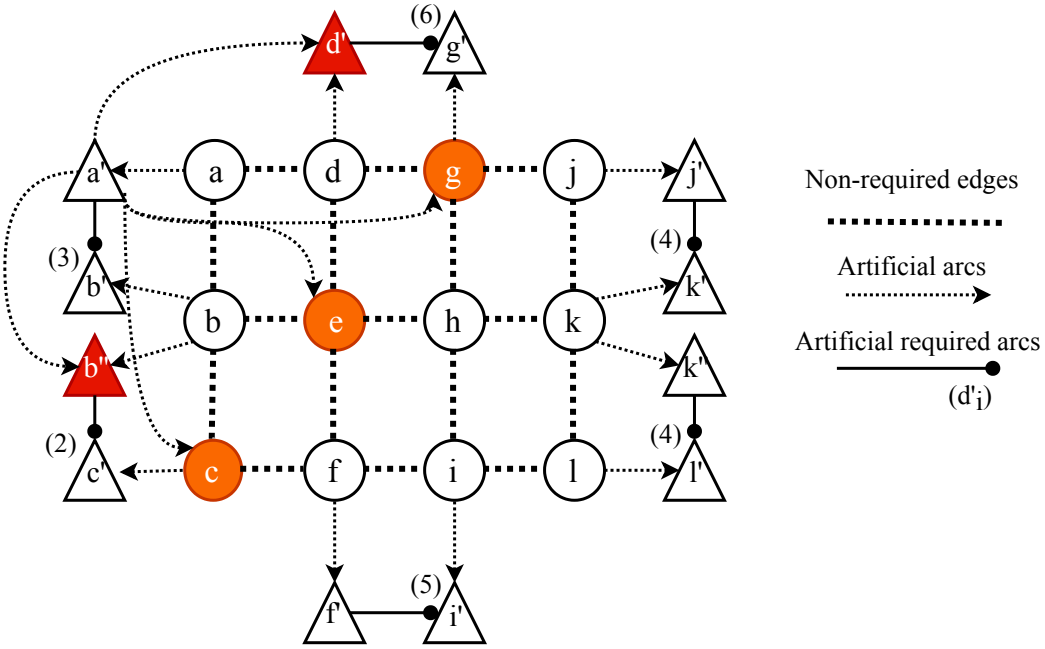

Figure 4.4: Example of $G^{k=2}$ showing arcs in A^k_{out} exiting from node a' (for the instance in Figure 4.1). Arcs leaving other artificial nodes and arcs connecting v_0 are omitted.

(4) assure that artificial required arcs are visited. Constraints (5) represent the routes continuity. Constraint (6) limits the number of flows starting from the origin v_0. Constraints (7) establish the balance flow for each node including the demand of artificial nodes. Constraints (8) couple visiting variables x_{ij} with flow variables y_{ij}. Constraints (9) and (10) describe the variables domains.

The $RFB(k)$ is a relaxed formulation since its optimal solutions are not necessarily feasible solutions for the OCARP. That happens because the flow variables y_{ij} represent aggregated capacities from multiple vehicles. This could represent a scenario where a single demand is split between two or more vehicles, which is not allowed for the OCARP. Nonetheless, this issue does not prevent $RFB(k)$ to obtain valid lower bounds.

One important feature of $RFB(k)$ is that parameter k can be used to set how much relaxed or tight will be the formulation. As greater is k, more tight is the formulation (i.e. its optimal solution cost becomes closer to the OCARP optimal solution cost). In fact, $RFB(k = 0)$ is the most relaxed version, while $RFB(k = \infty)$ is an exact formulation for the OCARP, as shown by Corollary 1.

Property 1. *Every route R of an optimal OCARP solution can be described as an alternating sequence of required edges $R = ((u_1, v_1), (u_2, v_2), ..., (u_n, v_n))$, where $\forall i, (u_i, v_i) \in E_R$ are the edges serviced by the route, and shortest paths between subsequent arcs endpoints (i.e. from v_i to u_{i+1}) are considered.*

Proof. Let R be a route from an optimal OCARP solution. The route R necessarily: (i) starts in a required edge endpoint that will be first serviced; and (ii) ends in a required edge endpoint that will be last serviced. Otherwise the solution would not be optimal since OCARP allows open routes. An exception is when non-required zero cost edges are

considered in the start and end of R but those can be ignored. Route R starts visiting and servicing the first required edge and successively goes to the next required edge that will be serviced. Since the solution is optimal, the route will go necessarily through a shortest path to reach the next required edge endpoint. This process is repeated until the last edge is serviced and the route ends. □

Property 2. *Any OCARP optimal solution represents a feasible solution for* $RFB(k = \infty)$ *with the same cost.*

Proof. Let Ω be an OCARP optimal solution. We show next how to construct a solution Ω' for $RFB(k = \infty)$ related to Ω and with the same cost. Each route R of Ω can be described as a sequence of required edges being serviced with shortest paths between every pair of consecutive required edges (Property 1). We show next how to transform a route R from Ω in an equivalent route R' in $RFB(k = \infty)$. Route R' starts in v_0, then go to the artificial node associated to the first required edge endpoint in R. The first artificial required arc is visited and serviced, then R' goes to the next artificial arc corresponding to next required arc in R, using arcs in $A_{out}^{k=\infty}$ which correspond to all shortest paths between artificial nodes. Route R' is ended when the last required arc is visited, returning to v_0. Since each arc in $A_{out}^{k=\infty}$ has the same cost as its corresponding shortest path in G, it follows that R and R' have the same cost. Therefore, Ω and Ω' have the same cost.

Regarding the flow constraints, an amount of D units of flow exits v_0 for each starting route and is used to service the demands of each route. Since Ω is feasible and each route capacity is limited to D units, the flow constraints will be satisfied. □

Property 3. $RFB(k = \infty)$ *solutions represent feasible solutions for the OCARP with the same cost.*

Proof. It is worth to notice that an $RFB(k = \infty)$ solution cannot visit any of the original nodes since its routes transit through a complete graph of the artificial nodes which are disconnected to the original nodes. Let Ω' be a solution for $RFB(k = \infty)$. We show next how to construct a solution Ω for OCARP related to Ω' and with the same cost. For this, we show first that each route R' of Ω' can be transformed in a route R of Ω represented by the form given in Property 1. Route R' starts at v_0, then visits a sequence of artificial nodes while servicing the artificial required arcs, until the route ends and returns to v_0. Since each arc in $A_{out}^{k=\infty}$ represents a shortest path in G, route R' represents a path in G. In other words, R' is an alternating sequence of required edges with shortest path between every subsequent pair. Moreover, R and R' have the same cost. By extension, Ω and Ω' have the same cost.

With respect to the flow constraints, Constraints (2-4) state each artificial node is visited by one entering arc and one exiting arc, so each artificial node is visited by exactly one route. Constraints (7-8) ensure a flow, represented by y variables, starting from v_0 and going through the sequence of arcs denoted by the x variables. Considering that each artificial node is visited by just one route, and Constraints (8) exclude flows from arcs that are not visited, then the set of nodes incident by each flow are disjoint. Therefore up to M flows of D units each, representing the vehicles, will leave v_0 and service the edges isolated from each other. Thus, each route R' of respects the capacity constraints, and so the constructed solution Ω is feasible for OCARP. □

Corollary 1. *$RFB(k = \infty)$ is an exact formulation for the OCARP.*

Proof. Since every OCARP optimal solution can be transformed in a solution for $RFB(k = \infty)$ with the same cost (Property 2) and any $RFB(k = \infty)$ solution can be transformed in a solution for OCARP with the same cost (Property 3), it follows that any optimal solution for $RFB(k = \infty)$ represents an optimal solution for the OCARP. □

Property 4. *Any solution of $RFB(k = 0)$ represents a solution for the one-index formulation (Section 4.4.1) with the same cost.*

Proof. Let Ω be a feasible solution for $RFB(k = 0)$. We show that Ω can be transformed in a solution Ω' for the one-index formulation with the same cost. By observing Figures 4.2 and 4.3 it is not difficult to see that $G^{k=0}$ and G^+ are very similar. For example, the one-index formulation could be executed with the graph $G^{k=0}$ as input graph and it would result in the same optimal solution cost as G^+, since the zero-cost arcs linking artificial nodes and corresponding original nodes in $G^{k=0}$ does not interfere in the optimal solution cost.

Let a solution Ω' be computed, starting from Ω, by replacing artificial arcs and artificial nodes visited in $G^{k=0}$ by their equivalent original edges and nodes: (i) consider v_0 and v^+ equivalent nodes; (ii) consider each artificial node equivalent to their corresponding original node; (iii) artificial arcs $(v_i, v'_i), (v'_i, v_i) : v_i \in V, v'_i \in V'$ in A'_{in} and A'_{out} are ignored. Since only zero cost arcs are removed, the costs of Ω' and Ω are the same (except for the fact the one-index formulation only consider deadheading costs).

We show next that Ω respect capacity and odd edge cut-set constraints in $G^{k=0}$. Regarding the *odd edge cut-set constraints*, since Constraints (5) from $RFB(k = 0)$ guarantee routes continuity, then each node has an even number of visiting edges and so these constraints are satisfied. About the *capacity constraints*, they set a lower bound of $2k(S) - |\delta_R(S)|$ deadheading edges, where $k(S) = \lceil \sum_{(i,j)\in E_R^+(S)\cup\delta_R(S)} d_{ij}/D \rceil$, for each cut $\delta(S)$ from $S \subseteq V \setminus \{v_+\}$. In the one-index formulation only the deadheading edges are counted. In contrast, in $RFB(k = 0)$ the required edges are counted by visiting the artificial required arcs (A'_R). We show next that Ω has a lower bound of $k_{RFB}(S')$ deadheading edges, crossing the cut $\delta(S')$ for every subset of nodes $S' \subseteq V \cup V'$ where S' is created based on the set of original nodes $S \subseteq V$ in addition to some artificial nodes, as explained next. Let $S' = S \cup V'(S) \cup V'_{pair}(V'(S))$ where $V'(S) = \{v_{i'} \in V' : v_i \in S \text{ and } \exists (v_i, v_{i'}) \in A'_{in}\}$ and $V'_{pair}(S) = \{v_i \in V' : v_j \in V'(S) \text{ and } \exists (v_i, v_j) \in A'_R\}$. We note that for every feasible solution Ω the number of visited edges in the cut $\delta(S')$ is the same as the number of edges visited in the corresponding constructed solution Ω' in the cut $\delta(S)$, since Constraints (2-4) force each artificial node to be visited once. And Ω' has one visit on each required edge implicit. Therefore, proving a lower bound of deadheading edges in the cut $\delta(S')$ for Ω imply the same lower bound for $\delta(S)$ in the constructed solution Ω'.

We show next the lower bound $k_{RFB}(S')$ is greater than or equal to $k(S)$. We have that $\sum_{i\in S'} d_i = \sum_{(i,j)\in E_R^+(S)\cup\delta_R(S)} d_{ij}$ is the total demand in S'. Summing Constraints (7) to all nodes in S' we have $\sum_{(i,j)\in\delta^-(S')} y_{ij} - \sum_{(i,j)\in\delta^+(S')} y_{ij} = \sum_{i\in S'} d_i$ which means at least $\sum_{i\in S'} d_i$ units of flow are crossing $\delta(S')$ and entering S'. Summing Constraints (8) for each arc $(i, j) \in \delta^-(S')$ we have $\sum_{(i,j)\in\delta^-(S')} y_{ij} \leq \sum_{(i,j)\in\delta^-(S')} (D - d_i)x_{ij}$ which

means $\sum_{(i,j)\in\delta^-(S')} y_{ij} \leq \sum_{(i,j)\in\delta^-(S')} D\ x_{ij}$ is satisfied. Since x variables are integers, this means a feasible solution in $RFB(0)$ has at least $k_{RFB}(S') = \lceil \sum_{(i,j)\in\delta^-(S')} y_{ij}/D \rceil$ edges entering $\delta(S')$. Since

$$k_{RFB}(S') = \lceil \sum_{(i,j)\in\delta^-(S')} y_{ij}/D \rceil \geq \lceil \sum_{i\in S'} d_i/D \rceil = k(S) = \lceil \sum_{(i,j)\in E_R^+(S)\cup\delta_R(S)} d_{ij}/D \rceil$$

we conclude $k_{RFB}(S') \geq k(S)$. Thus, considering routes continuity, at least $2k_{RFB}(S')$ edges cross $\delta(S')$ and so the *capacity constraints* are satisfied. Thus, the constructed solution Ω' is feasible and has the same cost as Ω.

□

Corollary 2. *The optimal solution cost of $RFB(k = 0)$ is greater or equal to the optimal solution cost of the one-index formulation.*

Proof. From Property 4, it follows directly that the optimal solution of $RFB(k = 0)$ is greater or equal to the optimal solution of the *one-index formulation*. □

Corollary 2 is important since it means the potential lower bound generated by $RFB(k = 0)$ is greater than that of the *one-index formulation* (Section 4.4.1). However, it is worth to note Property 4 does not imply that lower bounds obtained by $RFB(k)$ will be in practice better than those obtained by B&C. This happens simply because we cannot guarantee the methodologies will converge to optimum within a limited amount of time.

4.6 Computational Experiments

This section aims to evaluate the performance of the proposed methodologies: B&C (branch-and-cut algorithm, Section 4.4) and $RFB(k)$ (parameterized flow-based formulation, Section 4.5). We compare our methodologies with two methods from literature that are able to compute lower bounds for OCARP: the original MILP formulation from [29] and the B&B (branch-and-bound) algorithm from [28].

Our proposed methodologies used the MILP solver Gurobi version 8 with default settings for solving the mathematical formulations. The implementations were coded in C++ and used the LEMON [12] library for graph algorithms. All experiments were carried on in an Intel Xeon X3430 2.4 GHz processor with 8 GB RAM and running a Linux 64-bit operating system. The $RFB(k)$ was tested using the values of parameter $k \in \{0, 1, 2, 3, \infty\}$. All methods were executed with a time limit of one hour for each instance.

The origin of the benchmark instances employed in the experiment is explained next. Five sets of benchmark instances (gdb, val, egl, A and B) originally developed for the CARP were adapted for the OCARP by considering the depot a common node and keeping the rest of data preserved. With respect to the number of available vehicles (M), three quantities were defined: M^*, $M^* + 1$ and $M^* + 2$, where M^* means the minimum

number of vehicles required for a feasible solution. In the authors website[1] we provide the full data of experiments, including instances and source codes.

- gdb: 23 instances proposed by [16]. Containing from 7 to 27 nodes, 11 to 55 edges (all required) and M^* ranging from 3 to 10.
- val: 34 instances proposed by [7]. Containing from 24 to 50 nodes, 34 to 97 edges (all required) and M^* ranging from 2 to 10.
- egl: 24 instances proposed by [19]. Containing from 77 to 140 nodes, 98 to 190 edges (from 51 to 190 required) and M^* ranging from 5 to 35.
- A: 32 instances proposed by [18]. Containing from 10 to 40 nodes, 15 to 69 edges (from 11 to 63 required) and M^* ranging from 1 to 25.
- B: 24 instances proposed by [18]. Containing from 13 to 40 nodes, 15 to 69 edges (from 11 to 53 required) and M^* ranging from 2 to 21.

Table 4.1 contains the summarizing results for each set of instances. $Gap(\%) = 100 * (UB - LB)/LB$ is the average deviation of upper (UB) and lower (LB) bounds, where the LB values were computed by each lower bounding method and UB is the best known solution cost [3]. B&C+$RFB(2)$ show the best lower bounds obtained combining B&C and $RFB(2)$. Analogously, B&C+RFB^* combine B&C and all $RFB(k), \forall k \in \{0, 1, 2, 3, \infty\}$.

As shown in Table 4.1, the B&C and $RFB(k)$ consistently outperformed the methods from literature, obtaining an overall $Gap(\%)$ at most one third in comparison to the existing methods. Moreover, many instances from *gdb* and *val* sets have their optimal values proved. With respect to the number of vehicles M, the results show that instances with the minimum number of vehicles possible ($M = M^*$) are harder to solve.

The B&C method performed especially well for the egl instances, obtaining $Gap(\%)$ of 4.10, 1.23 and 0.80 in comparison to 11.91, 6.70 and 5.12 from the literature for classes M^*, $M^* + 1$ and $M^* + 2$ respectively. The B&C achieved the best overall results for gdb, val and egl instances for all classes of M.

The $RFB(2)$ was able to obtain the overall best $Gap(\%)$ of 0.69 and for the classes M^*, $M^* + 1$ and $M^* + 2$ the $Gap(\%)$ was 1.46, 0.39 and 0.23 respectively. When compared to B&C, which obtained 1.83, 0.46 and 0.24 we see that the proportional difference is greater for smaller values of number of vehicles M. $RFB(2)$ outperforms B&C for the A and B instances. On the other hand, B&C is significantly better for egl instances, while the gdb and val instances have competitive results by both methodologies. The $RFB(k)$ obtained better results for small and medium sized instances ($|E_R| \leq 100$), while B&C performed better for the larger instances.

In Table 4.1, column B&C+$RFB(2)$ show that these methods exhibit complementary results for the egl instances: some instances were better solved by B&C while others by $RFB(2)$. B&C+$RFB(2)$ obtained an overall $Gap(\%)$ for egl of 1.84 in comparison to 2.04 from B&C and 2.54 from $RFB(2)$. On the other hand, with respect to A and B instances, the best results are from $RFB(k)$ with no complementarity to other methodologies.

[1] `http://www.ic.unicamp.br/~fusberti/problems/ocarp`

In Table 4.1, column B&C+RFB^* show that $RFB(k)$ performance is sensible to parameter $k \in \mathbb{Z}_{\geq 0}$. This difference is most notable for the hardest instances: egl, A and B with M^* class. For these instances B&C+RFB^* obtained an overall $Gap(\%)$ of 3.72, 1.25 and 1.18, respectively, in comparison to 3.95, 1.28 and 1.33 from B&C+$RFB(2)$. This means that $RFB(0)$, $RFB(1)$, $RFB(3)$ and $RFB(\infty)$ can produce better results than B&C and $RFB(2)$ for some instances, even though in general they have inferior results. Thus, the parameter $k \in \mathbb{Z}_{\geq 0}$ should be adjusted in order to obtain the best results.

Table 4.1: Computational experiment overall results.

class	group	$Gap(\%)$									
		Literature		Our methods							
		MILP	B&B	B&C	$RFB(0)$	$RFB(1)$	$RFB(2)$	$RFB(3)$	$RFB(\infty)$	B&C+$RFB(2)$	B&C+RFB^*
M^*	gdb	0.07	0.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	val	1.43	3.16	0.09	0.09	0.09	0.09	0.09	0.11	0.09	0.09
	egl	13.91	11.91	4.10	4.54	4.68	5.16	5.22	6.31	3.95	3.72
	A	3.19	3.58	2.16	1.97	1.53	1.28	1.31	1.35	1.28	1.25
	B	4.28	2.94	3.37	2.48	1.42	1.33	1.41	1.41	1.33	1.18
	overall	4.30	4.23	1.83	1.71	1.45	1.46	1.49	1.70	1.24	1.17
M^*+1	gdb	0.05	0.05	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	val	1.75	2.24	0.00	0.01	0.03	0.02	0.00	0.02	0.15	0.00
	egl	8.66	6.70	1.23	1.50	1.54	1.53	1.54	2.14	1.01	0.98
	A	1.44	1.72	0.47	0.47	0.34	0.30	0.29	0.32	0.30	0.29
	B	1.47	0.73	0.79	0.75	0.29	0.29	0.27	0.26	0.29	0.26
	overall	2.55	2.27	0.46	0.51	0.41	0.39	0.39	0.53	0.30	0.28
M^*+2	gdb	0.02	0.02	0.00	0.00	0.00	0.02	0.00	0.02	0.00	0.00
	val	1.46	1.46	0.00	0.01	0.02	0.07	0.05	0.08	0.00	0.00
	egl	6.58	5.12	0.80	0.94	0.99	0.93	0.97	1.42	0.57	0.55
	A	1.32	1.14	0.25	0.25	0.24	0.16	0.25	0.19	0.16	0.16
	B	0.61	0.23	0.27	0.20	0.04	0.08	0.06	0.08	0.08	0.04
	overall	1.93	1.57	0.24	0.26	0.24	0.23	0.24	0.33	0.15	0.14
overall	gdb	0.05	0.05	0.00	0.00	0.00	0.01	0.00	0.01	0.00	0.00
	val	1.54	2.29	0.03	0.04	0.05	0.06	0.05	0.11	0.03	0.03
	egl	9.72	7.91	2.04	2.32	2.40	2.54	2.58	3.29	1.84	1.75
	A	1.98	2.14	0.96	0.89	0.70	0.58	0.62	0.62	0.58	0.57
	B	2.12	1.30	1.47	1.14	0.58	0.56	0.58	0.58	0.56	0.49
	overall	2.93	2.69	0.85	0.82	0.70	0.69	0.71	0.85	0.56	0.53

In underline: best results regarding $Gap(\%)$, excluding B&C+$RFB(2)$ and B&C+RFB^*.
$Gap(\%)$: average deviation of upper bound (best known) from lower bound (%).
B&C+$RFB(2)$: combination of best lower bounds obtained from B&C and $RFB(2)$.
B&C+RFB^*: combination of best lower bounds obtained from B&C and $RFB(k), \forall k \in \{0, 1, 2, 3, \infty\}$.

4.7 Conclusion

We propose two lower bounding methods for the open capacitated arc routing problem: (i) a branch-and-cut algorithm (B&C) and (ii) a parameterized flow-based formulation method called $RFB(k)$. These methods comprehend relaxed formulations for the OCARP. Lower bounding methods can be used to assess the performance of heuristic methods which represent the state-of-the-art to solve OCARP [3] and many others arc routing problems [4, 9, 11].

The B&C algorithm uses a relaxed *one-index formulation* where all vehicles are aggregated in the same set of variables. Two sets of valid inequalities and their corresponding exact separation algorithms are described.

The $RFB(k)$ method is a relaxed formulation for the OCARP based on network flows. In this formulation, the aggregated vehicles capacities are represented by flow variables. $RFB(k)$ explores the modification of the instance graph in order to achieve flow formulations more or less relaxed. The integer parameter k control the tightness of the formulation, where $RFB(k = 0)$ is the most relaxed version while $RFB(k = \infty)$ is the tightest formulation. As far as we know, the approach adopted by $RFB(k)$ in manipulating an augmented graph for modifying a formulation tightness is quite unique in the arc routing literature.

Some relations between OCARP, $RFB(k)$ and B&C method are shown. We show that $RFB(k), \forall k \geq 0$ is a tighter formulation for the OCARP in comparison to the one-index formulation from B&C. Also, $RFB(k = \infty)$ is shown to be a valid formulation for the OCARP.

Computational experiments revealed that the proposed methods outperformed by a large margin previous methods. The overall optimality gap was reduced from 2.69% to 0.69%. The B&C algorithm obtained the overall best solutions for largest instances (more than 100 required edges), while the $RFB(k)$ method was able to provide the tightest lower bounds for small and medium size instances. The proposed methods B&C and $RFB(k)$ were shown to have complementary results when solving the set of largest instances. The $RFB(k)$ performance was shown to be sensible to parameter k.

Future research topics include extending the ideas presented by $RFB(k)$ to other routing problems and also investigating other ways of generating augmented graphs to yield better results. In relation to the B&C algorithm, studying new inequalities for tightening the formulation even more and also developing heuristic separation algorithms in order to speed up the computation are also interesting research directions.

4.8 Acknowledgments

This work received support from the grants #2016/00315-0 and #2015/11937-9, São Paulo Research Foundation (FAPESP); as well as grants 435520/2018-0 and 314384/2018-9, National Counsel of Technological and Scientific Development (CNPq).

References

[1] Dino Ahr. *Contributions to multiple postmen problems*. PhD thesis, University of Heidelberg, 2004.

[2] Dino Ahr and Gerhard Reinelt. The capacitated arc routing problem: Combinatorial lower bounds. In Ángel Corberán and Gilbert Laporte, editors, *Arc Routing: Problems, Methods, and Applications*, chapter 8, pages 159–181. SIAM, 2014.

[3] Rafael Kendy Arakaki and Fábio Luiz Usberti. Hybrid genetic algorithm for the open capacitated arc routing problem. *Computers & Operations Research*, 90:221–231, 2018.

[4] Rafael Kendy Arakaki and Fábio Luiz Usberti. An efficiency-based path-scanning heuristic for the capacitated arc routing problem. *Computers & Operations Research*, 103:288 – 295, 2019.

[5] José M Belenguer and Enrique Benavent. A cutting plane algorithm for the capacitated arc routing problem. *Computers and Operations Research*, 30(5):705–728, 2003.

[6] José Manuel Belenguer, Enrique Benavent, and Stefan Irnich. The capacitated arc routing problem: Exact algorithms. In Ángel Corberán and Gilbert Laporte, editors, *Arc Routing: Problems, Methods, and Applications*, chapter 9, pages 183–222. SIAM, 2014.

[7] Enrique Benavent, Vicente Campos, Angel Corberán, and Enrique Mota. The capacitated arc routing problem: lower bounds. *Networks*, 22:669–690, 1991.

[8] L. Bodin and L. Levy. The arc partitioning problem. *European Journal of Operational Research*, 53:393–401, 1991.

[9] Yuning Chen, Jin-Kao Hao, and Fred Glover. A hybrid metaheuristic approach for the capacitated arc routing problem. *European Journal of Operational Research*, 253(1):25–39, 2016.

[10] Ángel Corberán and Gilbert Laporte. *Arc Routing: Problems, Methods, and Applications*. SIAM, Philadelphia, PA, USA, 2014.

[11] Jesica de Armas, Peter Keenan, Angel A Juan, and Seán McGarraghy. Solving large-scale time capacitated arc routing problems: from real-time heuristics to metaheuristics. *Annals of Operations Research*, 273(1-2):135–162, 2019.

[12] Balázs Dezs, Alpár Jüttner, and Péter Kovács. Lemon - an open source c++ graph template library. *Electronic Notes in Theoretical Computer Science*, 264(5):23–45, 2011.

[13] Richard YK Fung, Ran Liu, and Zhibin Jiang. A memetic algorithm for the open capacitated arc routing problem. *Transportation Research Part E: Logistics and Transportation Review*, 50:53–67, 2013.

[14] Andrew V Goldberg. An efficient implementation of a scaling minimum-cost flow algorithm. *Journal of algorithms*, 22(1):1–29, 1997.

[15] B. L. Golden and R. T. Wong. Capacitated arc routing problems. *Networks*, 11:305–315, 1981.

[16] L. Bruce Golden, S. James DeArmon, and K. Edward Baker. Computational experiments with algorithms for a class of routing problems. *Computers and Operations Research*, 10:47–59, 1982.

[17] Ralph E Gomory and Tien Chung Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

[18] Alain Hertz. *Recent trends in arc routing. In: Sharda R., Voß S., Golumbic M. C., Hartman IBA, editors. Graph theory, combinatorics and algorithms.* Springer US, 2005.

[19] YO Leon Li and W. Richard Eglese. An interactive algorithm for vehicle routeing for winter-gritting. *Journal of the Operational Research Society*, pages 217–228, 1996.

[20] Rafael Martinelli, Marcus Poggi, and Anand Subramanian. Improved bounds for large scale capacitated arc routing problem. *Computers & Operations Research*, 40(8):2145–2160, 2013.

[21] L. M. Moreira, J. F. Oliveira, A. M. Gomes, and J. S. Ferreira. Heuristics for a dynamic rural postman problem. *Computers and Operations Research*, 34:3281–3294, 2007.

[22] M Cândida Mourão and Leonor S Pinto. An updated annotated bibliography on arc routing problems. *Networks*, 70(3):144–194, 2017.

[23] Manfred W Padberg and M Ram Rao. Odd minimum cut-sets and b-matchings. *Mathematics of Operations Research*, 7(1):67–80, 1982.

[24] Christian Prins. The capacitated arc routing problem: Heuristics. In Ángel Corberán and Gilbert Laporte, editors, *Arc Routing: Problems, Methods, and Applications*, chapter 7, pages 131–157. SIAM, 2014.

[25] Ana Maria Rodrigues and José Soeiro Ferreira. Cutting path as a rural postman problem: solutions by memetic algorithms. *International Journal of Combinatorial Optimization Problems and Informatics*, 3(1):31–46, 2012.

[26] Everton Fernandes Silva, Larissa Tebaldi Oliveira, José Fernando Oliveira, and Franklina Maria Bragion Toledo. Exact approaches for the cutting path determination problem. *Computers & Operations Research*, page 104772, 2019.

[27] H. I. Stern and M. Dror. Routing electric meter readers. *Computers and Operations Research*, 6:209–223, 1979.

[28] Fábio Luiz Usberti. *Heuristic and exact approaches for the open capacitated arc routing problem*. PhD thesis, Universidade Estadual de Campinas, 2012.

[29] Fábio Luiz Usberti, Paulo Morelato França, and André Luiz Morelato França. The open capacitated arc routing problem. *Computers and Operations Research*, 38(11):1543 – 1555, 2011.

Chapter 5

The covering Chinese postman problem

The text presented below is a paper submitted and presented in the International Symposium on Scheduling (ISS 2019) held in Matsue, Japan, on 3-6 July, 2019. The paper is co-authored with Fábio Luiz Usberti. In this manuscript we present a new problem called the covering Chinese postman problem (CCPP). Inspired by applications on the routing of automated meter readers, in this problem each edge covers a subset of edges of the graph. The objective is to find a minimum cost tour such that the edges in the tour collectively cover the whole graph. An exact method and a heuristic method were proposed. Computational experiments show what kind of instances were more difficult to solve.

The covering Chinese postman problem is a combinatorial optimization arc routing problem. Given an undirected graph with cost and covering functions on the edges, this problem aims to find a minimum cost tour that covers all the edges of the network. The problem was modeled by an integer programming formulation. Two solution methods are proposed: an exact branch-and-cut algorithm and a heuristic method. Extensive computational experiments were performed. Branch-and-cut method was able to obtain 325 optimal solutions out of 390 instances, while the heuristic obtained better solutions for some large instances in shorter processing times.

5.1 Introduction

Node routing problems, such as the well known *traveling salesman problem* (TSP), aim to find optimum routes while visiting a set of nodes of a network. On the other hand, arc routing problems aim to determine optimum routes that visit some arcs of the graph, with one or more side constraints [10].

The *covering Chinese postman problem* (CCPP) is an arc routing problem with covering constraints whose solution consist of a minimum cost tour that must visit a set of arcs which covers all the edges of a network. The CCPP is a new problem. To our knowledge the CCPP has not yet been addressed by the literature even though its node routing counterpart, the *covering salesman problem* (CSP) [7], have been objective of research in

the last decades [7, 16].

One practical application of CCPP is the automated meter reading problem [17]. In this problem, utility companies employ automated meter readers with wireless technology to read utility meters from a distance. This way the meter reader does not need to visit every street of the network but rather a subset of them which covers all customers. The aim is to find the minimum cost route that is able to read all customers data using the wireless devices.

Our contribution. This paper proposed an *integer programming* (IP) formulation for the CCPP and a heuristic called *greedy randomized heuristic* (GRH). The problem was solved by both an exact branch-and-cut method and the GRH. Extensive computational experiments were performed to evaluate the proposed methods. Results show that the branch-and-cut was able to obtain 325 optimum solutions from the total of 390 instances. Also, the GRH was able to obtain better solutions than branch-and-cut for 24 large instances while spending less computational time.

Section 5.2 provides the literature background. Section 5.3 formally defines the CCPP. Section 5.4 describe an IP formulation for the problem. In Section 5.5, a heuristic method is proposed. Section 5.6 contains the computational experiments. Finally, Section 5.7 provides the final comments.

5.2 Literature review

The *Chinese postman problem* (CPP) [10] is defined on an undirected graph $G(V, E)$ with a cost function associated to the edges and the objective is to find the minimum cost closed circuit (it ends and starts in the same node) that visit each edge $(u, v) \in E$ at least once. This problem can be solved in polynomial time using an optimal matching algorithm [10]. Many variations of the CPP, however, are NP-hard problems, such as the mixed CPP and windy CPP [14].

Current and Schilling [7] proposed a node routing problem with covering constraints: the *covering salesman problem* (CSP), which is a generalization of the well known traveling salesman problem. In this problem, the objective is to find a minimum cost tour of a subset of cities such that every city not visited by the tour is within some predetermined fixed covering distance of a city that is on the tour. Some papers have discussed on the applications of this problem in the fields of emergency management and disaster planning [15, 16].

Drexl [9] recently proposed the *generalized directed rural postman problem* (GDRPP) which objective is to find the minimum cost tour in a digraph such that the arcs traversed must cover all the predefined subsets of arcs of the digraph. More specifically, at least one arc of each predefined subset must be visited by the tour. Drexl demonstrated how to transform many arc routing problems to the GDRPP, such as: mixed Chinese postman problem, arc routing with turn penalties and others. As solution method, the author proposed a branch-and-cut algorithm that was able to obtain good results including many optimum solutions for the considered instances.

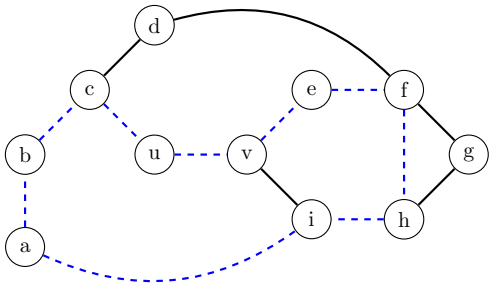

Figure 5.1: Example of a CCPP instance and its solution (in dashed edges).

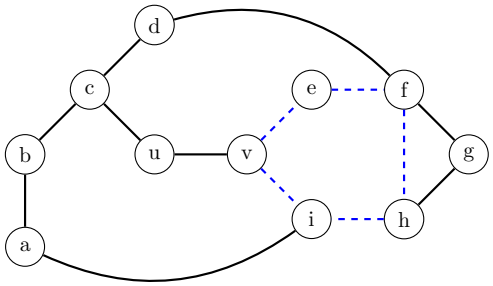

Figure 5.2: Example of a CCPP instance with a wider covering function and its solution (in dashed edges).

5.3 Definition

The covering Chinese postman problem (CCPP) is a new proposed variant of the CPP that considers covering constraints. The CCPP can be formally defined as next. Let $G(V, E)$ be an undirected graph and $c_{uv} \geq 0$ be a non-negative cost associated to each edge $(u, v) \in E$. Consider a covering function $D(u, v)$ associated to each edge $(u, v) \in E$ that represents the set of edges covered by (u, v). Also, it is required that for each $(u, v) \in E$: (i) $(u, v) \in D(u, v)$; and (ii) $(p, q) \in D(u, v) \iff (u, v) \in D(p, q)$. The objective is to find a minimum-cost closed circuit such that all the edges in G are covered by at least one of the edges visited by the circuit.

An example of solution for CCPP is given by Figure 5.1 where all edges have one unit of cost and the covering function for each edge $(u, v) \in E$ is defined as the set of edges that are incident to either the nodes u or v. Figure 5.2 shows a solution for the same graph and costs but defining the covering function of each edge $(u, v) \in E$ as all the edges whose both extreme points are at most from three units of distance from both u and v.

5.4 Mathematical Model

An integer programming mathematical formulation for the CCPP is proposed next. Although the graph G is undirected, the variables in the model are directed. Also, although a CCPP solution can visit an specific edge many times, the variables in the model

are binary. In fact, it can be showed that exist at least one optimal solution for the CCPP that does not visit the same edge $(u, v) \in E$ more than twice and, if it visits (u, v) twice, then it is in opposite directions ((u, v) and (v, u)). Welz [18] proved that property for the *capacitated arc routing problem* but the proof can be also applied to CCPP. Therefore, the model works on $2|E|$ binary variables in the form of directed arcs x_{ij}, x_{ji} representing each edge $(i, j) \in E$ visited in opposite directions.

$$(CCPP)$$

$$MIN \quad \sum_{(i,j)\in E} c_{ij}(x_{ij} + x_{ji}) \tag{1}$$

s.t.

$$\sum_{j\in\delta(i)} x_{ij} = \sum_{j\in\delta(i)} x_{ji} \qquad (i \in V) \tag{2}$$

$$\sum_{(i,j)\in D(u,v)} (x_{ij} + x_{ji}) \geq 1 \qquad ((u, v) \in E) \tag{3}$$

$$\sum_{(i,j)\in(S,\tilde{S})} (x_{ij} + x_{ji}) \geq 2(x_{kl} + x_{mn} - 1)$$

$$(\forall S \subset V, \forall (k, l), (m, n) \in E :$$

$$k, l \in S \text{ and } m, n \in \tilde{S}) \tag{4}$$

$$x_{ij}, x_{ji} \in \{0, 1\}, \forall (i, j) \in E \tag{5}$$

$\delta(i)$: edges incident to node $i \in V$.

In this model, x_{ij} and x_{ji} assume value 1 if and only if the edge $(i, j) \in E$ is visited in each direction. Therefore, the solution cost of the closed circuit is given by objective function (1). Constraints (2) are degree constraints. Constraints (3) force that the solution must visit at least one edge of the covering functions associated to each edge. The connectivity constraints (4) force that every pair of visited edges must be connected in the solution. More specifically, they state that if two edges are selected, respectively belonging to (S, S) and $(\tilde{S}, \tilde{S})$, then at least two cut edges in $(S, \tilde{S})$ must be selected as well, otherwise the solution is not connected. Constraints (5) define the domain of variables x.

In terms of implementation, careful must be taken in relation to constraints (4) since they are of exponential number. A complete enumeration of them is only possible for very small instances. Inspired by the work of Belenguer and Benavent [2], the solution we adopted was to add incrementally the constraints (4) as they are needed in a *branch-and-cut* fashion. The branch-and-cut algorithm iteratively adds a subset of the exponential sized set of constraints (4) that are identified by a *separation algorithm* (Section 5.4.1) especially developed for this formulation. When no violated constraints are found by the separation algorithm and the solution is integer, then that solution is feasible. Details of how a branch-and-cut algorithm operates would be omitted because of complexity, while an interested reader can find it in [13].

5.4.1 Separation algorithm for connectivity constraints

Since connectivity constraints are of exponential number it is impractical to solve a model containing all of them, except for very small instances. Therefore, a polynomial-time separation algorithm was developed to identify violated connectivity constraints (4) on demand and add them to the model. Given a candidate solution x^* for CCPP that obeys constraints (2) and (3), the separation algorithm identifies a subset of connectivity constraints (4) that are violated by x^*. The method works on both integer and real (relaxed integer) variables.

The procedure pseudocode is described in Algorithm 9. Given candidate solution x^*, the separation algorithm start by considering a graph $G^*(V, E^*)$ induced by the edges $E^* = \{(i,j) \in E : x^*_{ij} + x^*_{ji} \geq \delta\}$ for a small $0 < \delta \ll 1$. From that, the Gomory-Hu algorithm [11] is executed on G^* by considering the capacity of each edge $e \in E^*$ as the sum of corresponding arc variables: $x^*_{ij} + x^*_{ji}$. The Gomory-Hu algorithm computes the minimum flow cut edges for each pair of nodes $u, v \in V$. For each found minimum flow cut, the separation algorithm verifies if there is a suitable connectivity constraint (4) violated, in other words, for a given set S it searches for two edges: an edge $(k,l) \in (S,S)$ and an edge $(m,n) \in (\tilde{S},\tilde{S})$ such that the minimum flow cut value is less than $2(x_{kl} + x_{mn} - 1)$. The algorithm output is a set of connectivity constraints, corresponding to minimum flow cuts, that are violated by x^*.

Algorithm 9: Separation algorithm for connectivity constraints.

Data: Given solution x^* for CCPP that obeys constraints (2) and (3).
Result: A set VC of violated connectivity constraints, if there is any.
Consider a graph $G^*(V, E^*)$ induced by set of edges
$E^* = \{(i,j) \in E : x^*_{ij} + x^*_{ji} \geq \delta\}$;
Execute Gomory-Hu algorithm on G^* considering the flow of each $(i,j) \in E^*$ as $x^*_{ij} + x^*_{ji}$;
for each min (flow) cut **S** found by Gomory-Hu algorithm **do**
    $(k,l) \leftarrow$ edge such that $x^*_{kl} \geq x^*_{ij}, \forall (i,j) \in (S,S)$;
    $(m,n) \leftarrow$ edge such that $x^*_{mn} \geq x^*_{ij}, \forall (i,j) \in (\tilde{S},\tilde{S})$;
    if *Constraint (4) is violated by* $S, x^*, (k,l)$ *and* (m,n) **then**
        $v \leftarrow$ violated constraint by $S, (k,l)$ and (m,n);
        $VC \leftarrow VC \cup v$;
return VC ;

5.5 Heuristic method

A heuristic method called *greedy randomized heuristic* (GRH) is proposed for the CCPP. The primary objective of GRH is to generate feasible solutions with reasonable quality for large instances. In addition, GRH intends to consume much less computing time than the exact method (Section 5.4).

The GRH pseudocode is described in Algorithm 10. Given a graph $G(V, E)$, the

covering function D and a parameter k of number of iterations, the algorithm is executed and the output is the best solution found. The GRH first performs a preprocessing procedure and then runs k independent iterations. Each GRH iteration produces one solution and consists of a constructive phase followed by a local search phase.

The preprocessing procedure starts computing a matrix $SP[u][v]$ representing the shortest path from node u to v for each $u, v \in V$. The matrix is computed using the Dijkstra's algorithm [6]. From that, an ordered non-increasing list $CE(u)$ of closest edges, for each node $u \in V$ is computed. More specifically, the distance from node u to edge (i, j) is defined as: $min(SP[u][i], SP[u][j])$, where SP is the matrix of shortest path costs computed earlier. This step is very important for the heuristic performance because the distance between two nodes SP and the list of closest edges CE are computed only once but are intensively used by all following iterations of GRH.

In the constructive phase a feasible solution for CCPP is generated from scratch. The solution is represented by a sequence of directed arcs $(u_1, v_1), (u_2, v_2), ..., (u_n, v_n)$ which gives the tour direction and for each subsequent $v_i \neq u_{i+1}$ a shortest path from v_i to u_{i+1} is implicitly implied. The return from node v_n to u_1 is also implied in this arrangement. The constructive phase starts by selecting one edge (u, v) of E randomly (each edge has equal probability). The edge (u, v) is added to the solution in a directed arc form (u then v) and all edges that are in the covering function $D(u, v)$ are marked. Then, the procedure iteratively selects the closest edge $(i, j) \in E$ to the last visited node v, considering the computed list $CE(v)$, and that is not covered by any edge in the solution. From that the edge (i, j) is added to the solution and all edges in $D(i, j)$ are marked. The process is repeated until all edges are covered (marked) by the edges added to the solution.

Local search phase starts in a feasible solution s and attempts to improve it by considering three sets of moves in order. If, starting from s, any move finds an improving feasible solution s' with lower cost, that move is immediately performed rather than examining all possible moves for a solution. The solution is updated and that set of moves is examined again. After a set of moves is examined to exhaustion without finding any improvement, the next set of moves is examined. Three sets of moves are considered in order:

1. *Remove*: remove from the solution representation any arc (u, v) that can be removed without losing feasibility. In other words, all edges in $D(u, v)$ are covered by at least two edges represented in the solution.

2. *Exchange*: replace in the solution representation an arc (u, v) by an arc (p, q) such that the solution is still feasible and has a lower cost.

3. *2-OPT*: change the order which the edges in the solution are visited [12]. Considering that any pair of subsequent edges in the solution are connected by shortest path links, the 2-OPT move removes two shortest path links and adds two others shortest path links such that it stills represents a closed tour.

The local search ends when the 2-OPT set of moves is examined and no improvement is found.

Algorithm 10: Greedy randomized heuristic (GRH) for CCPP.

Data: Graph $G(V, E)$, covering function D, parameter k of number of iterations.
Result: The best solution produced by GRH in the k iterations.
// preprocessing phase
Compute $SP[u][v]$ for each $u, v \in V$;
Compute $CE(u)$ for each $u \in V$;
$bestSol \leftarrow \emptyset$;
for k iterations **do**
  $sol \leftarrow$ a random edge $(u, v) \in E$;
  $M \leftarrow D(u, v)$;
  $lastNode \leftarrow v$;
  // constructive phase
  while $|M| < |E|$ **do**
    Search in $CE(lastNode)$ the closest edge (i, j) such that $(i, j) \notin M$;
    Add (i, j) to sol at the final position;
    $M \leftarrow M \cup D(i, j)$;
    $lastNode \leftarrow j$;
  // local search phase
  while *sol* has a improving *remove* move **do**
    sol is updated by the first improving *remove* found;
  while *sol* has a improving *exchange* move **do**
    sol is updated by the first improving *exchange* found;
  while *sol* has a improving *2-OPT* move **do**
    sol is updated by the first improving *2-OPT* found;
  if $cost(sol) < cost(bestSol)$ *or* $bestSol = \emptyset$ **then**
    $bestSol \leftarrow sol$;
return *bestSol* ;

5.6 Computational experiments

Computational experiments were performed to compare the proposed exact branch-and-cut (Section 5.4) and heuristic GRH (Section 5.5) methods on benchmark instances from the literature. Experiments were performed in an Intel Core i7-6700K 4.0 GHz, 16 GB RAM and Linux 64-bit operating system. The methods were coded in C++ using the LEMON library for graph algorithms [8] and compiled by g++ v4.8 with "-O3" flag. The branch-and-cut algorithm used the commercial IP solver Gurobi v8.1 with default parameters and time limit of 1800 seconds for each instance. Separation algorithm used $\delta = 0.01$ as tolerance constant. The GRH was executed for $k = 10000$ iterations.

The experiments employed 6 sets of benchmark instances that were originally designed for the *rural postman problem* (RPP): the "Alba" set consisting of two instances modeled from the road-network of the city of Albaida, Spain, from [4]; the 24 computer-generated "P" instances of [3]; the 20 randomly generated "R" instances from [12]; the 36 instances with nodes degree four "D" and the 36 grid instances "G" both from [1]; and 12 "UR500" large instances from [5].

For each RPP instance set, three different covering functions D_α for $\alpha \in \{1.0, 2.0, 3.0\}$

were considered. Therefore, from 6 original instance sets for RPP, a total of 18 instance sets for CCPP were generated.

The covering functions $D_\alpha(u,v)$ are defined for each $(u,v) \in E$ as the set of edges $(i,j) \in E$ such that: $max(SP[i][u], SP[i][v], SP[j][u], SP[j][v]) \leq \alpha \times \mu(c(E))$, where $SP[x][y]$ is the shortest path distance from node x to node y, $\mu(c(E))$ is the average cost of edges in E, and $\alpha \geq 0$ is a real parameter. In other words, one edge cover and is covered by the other when their nodes are close enough. Moreover, $D_\alpha(u,v)$ always include (u,v) independently of α.

Table 5.1 show the data for the instance sets: first column is the set name and in parentheses is the number of instances; $|V|$ and $|E|$ are the number of nodes and edges, respectively; the last three columns are the average size of covering functions D_α for $\alpha = 1$, $\alpha = 2$ and $\alpha = 3$ respectively.

Tables 5.2, 5.3 and 5.4 gives the overall comparison of the branch-and-cut and GRP methods considering the covering functions D_α for $\alpha = 1$, $\alpha = 2$ and $\alpha = 3$ respectively. The first column is the instance set name and in parentheses the number of instances. For each method, OPT is the number of instances with proved optimal solution; $Gap = 100 * (UB - LB)/LB$ is the average gap in percentage between the solution cost found by each method (UB) and the lower bound obtained by branch-and-cut (LB); and Time is the method runtime in seconds.

From Tables 5.2, 5.3 and 5.4 one can observe the following:

- Branch-and-cut method performed very well for all instances with $\alpha = 1$. In fact, 119 instances of 130 were solved to optimality and an average Gap of 0.11 was achieved in reasonable computational time.
- Branch-and-cut also performed well for the instances Alba, P, R, D and G with $\alpha = 2$. The computational time increased in comparison to $\alpha = 1$ for all instance sets.
- Branch-and-cut performed poorly for the UR500 instances with $\alpha = 2$ and $\alpha = 3$.
- Branch-and-cut performed competitively in comparison to GRH for all instances with $\alpha = 3$, except for UR500.
- GRH performed much better than Branch-and-cut for the UR500 instances with $\alpha = 2$ and $\alpha = 3$. In fact, the solutions obtained have Gap at least 5 times smaller and the processing time was at least 4 times faster.
- GRH is much faster than Branch-and-cut for the majority of instance sets, for example the D and G instances with $\alpha = 2$ and $\alpha = 3$ was at least 360 times faster using GRH than Branch-and-cut.
- GRH is faster for greater values of α. For example, the overall processing time for $\alpha = 1$ was 66.9s while for $\alpha = 3$ it was only 17.2s.
- GRH does not degrade its performance when considering instances with larger α values as the branch-and-cut does. For example, GRH obtained 52, 54 and 75 optimum solutions for $\alpha = 1$, $\alpha = 2$ and $\alpha = 3$ respectively.

Table 5.1: Data of instances.

Set	$\|V\|$	$\|E\|$	$\|D_{\alpha=1}\|$	$\|D_{\alpha=2}\|$	$\|D_{\alpha=3}\|$
Alba(2)	90-102	144-160	2.8	9.8	18.4
P(24)	7-50	13-184	4.0	21.4	43.8
R(20)	20-50	37-203	3.8	28.0	53.8
D(36)	16-100	31-200	3.1	10.4	20.9
G(36)	16-100	24-180	1.0	7.6	18.4
UR500(12)	298-499	597-1526	3.6	15.6	35.2

Table 5.2: Experiment results for covering function D_α with $\alpha = 1$.

	Branch-and-cut			GRH		
Set	OPT	*Gap*	Time	OPT	*Gap*	Time
Alba(2)	2	0.00	5.2	0	2.41	3.1
P(24)	24	0.00	18.4	12	1.05	0.6
R(20)	20	0.00	1.2	3	8.43	2.6
D(36)	36	0.00	1.9	9	3.18	1.9
G(36)	36	0.00	0.4	28	0.23	1.3
UR500(12)	1	1.19	1705.3	0	12.64	708.5
overall(130)	119	0.11	161.7	52	3.64	66.9

One can conclude from these experiments that the proposed Branch-and-cut method was especially effective for small and medium sized instances (up to 200 edges) and instances with narrower covering functions (small α). On the other hand, GRH seems to take advantage of wider covering functions in relation to processing time.

Tables 5.5, 5.6 and 5.7 show detailed experiment results for the UR500 instances, which are the largest instances. The first column is instance number; $|V|$ and $|E|$ are number of nodes and edges respectively; LB is the lower bound obtained by branch-and-cut method. For each method, UB is the solution cost and $Gap = 100 * (UB - LB)/LB$ is the average gap in percentage between UB and LB.

In Table 5.5 one can observe that the performance of the branch-and-cut method for the instances with $\alpha = 1$ was much better than the GRH. Moreover, all sets were solved to optimal except for the UR500 set where the average *Gap* was 1.19 for branch-and-cut and 12.64 for GRH.

In Table 5.6 and 5.7 the branch-and-cut display poor results in general for UR500 with $\alpha = 2$ and $\alpha = 3$. Branch-and-cut bad performance can be partly explained because it is an exact exponential-time algorithm in the worst case [13]. Another possibly reason may be because the connectivity constraints (Section 5.4.1) perform worse for instances with wider covering functions.

On the other hand, GRH did not perform so bad as branch-and-cut for UR500 with

Table 5.3: Experiment results for covering function D_α with $\alpha = 2$.

	Branch-and-cut			GRH		
Set	OPT	*Gap*	Time	OPT	*Gap*	Time
Alba(2)	2	0.00	47.2	0	6.49	2.0
P(24)	24	0.00	3.0	21	1.68	0.5
R(20)	20	0.00	16.3	16	1.40	0.7
D(36)	29	0.85	398.2	6	5.83	1.1
G(36)	27	2.71	712.4	11	21.71	0.5
UR500(12)	0	323.30	1804.4	0	62.95	421.2
overall(130)	102	30.83	477.9	54	14.06	39.6

Table 5.4: Experiment results for covering function D_α with $\alpha = 3$.

	Branch-and-cut			GRH		
Set	OPT	*Gap*	Time	OPT	*Gap*	Time
Alba(2)	2	0.00	63.0	0	7.57	1.5
P(24)	24	0.00	1.1	24	0.00	0.1
R(20)	18	9.05	370.1	15	9.62	0.6
D(36)	33	0.24	369.1	12	4.21	0.8
G(36)	27	17.83	578.1	24	25.01	0.4
UR500(12)	0	2010.66	1801.1	0	227.70	181.6
overall(130)	104	191.99	486.7	75	30.71	17.2

$\alpha = 2$ and $\alpha = 3$. Also, GRH has a shorter processing time that decreases for higher values of α. Therefore, it is observable some degree of complementarity in the proposed methods. In the case of wide covering functions, the branch-and-cut can be very sensitive and degrade its performance; on the other hand, GRH does not seem to degrade its performance because of wide covering functions.

Table 5.5: Experiment results detailed for UR500 instances and covering function D_α with $\alpha = 1$.

			Branch-and-cut			GRH	
#	$\|V\|$	$\|E\|$	LB	UB	Gap	UB	Gap
1	298	597	36513	37190	1.85	39298	7.63
2	458	812	30522	30832	1.02	34051	11.56
3	493	868	29823	29823	0.00	33978	13.93
4	343	862	44854	45603	1.67	50052	11.59
5	476	1104	40554	40873	0.79	45255	11.59
6	498	1112	37799	38213	1.10	42502	12.44
7	388	1135	49288	49854	1.15	56257	14.14
8	490	1305	46089	46471	0.83	51602	11.96
9	498	1310	44583	45154	1.28	50296	12.81
10	416	1403	55910	56754	1.51	64703	15.73
11	496	1513	56163	56702	0.96	63056	12.27
12	499	1526	51253	52368	2.18	59443	15.98
avg					1.19		12.64

Table 5.6: Experiment results detailed for UR500 instances and covering function D_α with $\alpha = 2$.

			Branch-and-cut			GRH	
#	$\|V\|$	$\|E\|$	LB	UB	Gap	UB	Gap
1	298	597	23925	25010	4.54	27262	13.95
2	458	812	16926	39161	131.37	20083	18.65
3	493	868	14967	15272	2.04	18091	20.87
4	343	862	21502	82704	284.63	28741	33.67
5	476	1104	14404	47580	230.32	22143	53.73
6	498	1112	12073	58790	386.95	18022	49.28
7	388	1135	18290	48617	165.81	26061	42.49
8	490	1305	8835	68768	678.36	17911	102.73
9	498	1310	10716	59074	451.27	17976	67.75
10	416	1403	12472	97698	683.34	22158	77.66
11	496	1513	6366	40454	535.47	16220	154.79
12	499	1526	8113	34521	325.50	17837	119.86
avg					323.30		62.95

Table 5.7: Experiment results detailed for UR500 instances and covering function D_α with $\alpha = 3$.

			Branch-and-cut			GRH	
#	$\|V\|$	$\|E\|$	LB	UB	Gap	UB	Gap
1	298	597	14933	50045	235.13	18226	22.05
2	458	812	6298	40205	538.38	12949	105.60
3	493	868	5824	38083	553.90	13063	124.30
4	343	862	9492	72590	664.75	15865	67.14
5	476	1104	6312	72296	1045.37	14129	123.84
6	498	1112	3137	64432	1953.94	10869	246.48
7	388	1135	3914	58532	1395.45	10220	161.11
8	490	1305	1721	54973	3094.25	9132	430.62
9	498	1310	2241	78888	3420.21	9716	333.56
10	416	1403	1499	62148	4045.96	6942	363.11
11	496	1513	1526	59138	3775.36	8249	440.56
12	499	1526	2039	71471	3405.20	8442	314.03
avg					2010.66		227.70

5.7 Conclusion

In this paper we have proposed a new combinatorial optimization problem called covering Chinese postman problem (CCPP). The problem was modeled by integer programming. Two solution methods were proposed: an exact branch-and-cut method and a heuristic method called GRP.

Extensive computational experiments with six sets of benchmark instances have been performed. The results show the usefulness of proposed methods. For small and medium instances ($|E| \leq 200$) the branch-and-cut obtained very good results, including many optimal solutions. However for some large instances it was very ineffective. On the other hand, the GRP method was able to obtain reasonable solutions in short computational time for all instances, although for medium instances the branch-and-cut performed better.

In addition to the instance sizes, the covering function was also show to play a major role in instances difficulty. While instances with narrow covering functions tend to be well solved by the branch-and-cut, wider covering functions tend to be more difficult. The GRP was shown to perform better than branch-and-cut method for large instances with wide covering functions.

Future works include improvements on the integer programming formulation by addition of valid inequalities. Another interesting research direction is the development of an efficient metaheuristic for the CCPP, including local search procedures especially tailored for the problem.

References

[1] Julián Aráoz, Elena Fernández, and Oscar Meza. Solving the prize-collecting rural postman problem. *European Journal of Operational Research*, 196(3):886–896, 2009.

[2] M. José Belenguer and Enrique Benavent. A cutting plane algorithm for the capacitated arc routing problem. *Computers & Operations Research*, 30:705–728, 2002.

[3] N Christofides, V Campos, A Corberán, and E Mota. An algorithm for the rural postman problem. Technical report, 1981.

[4] A Corberán and JM Sanchis. A polyhedral approach to the rural postman problem. *European Journal of Operational Research*, 79(1):95–114, 1994.

[5] Angel Corberán, Isaac Plana, and José M Sanchis. A branch & cut algorithm for the windy general routing problem and special cases. *Networks: An International Journal*, 49(4):245–257, 2007.

[6] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2nd edition, 2001.

[7] John R Current and David A Schilling. The covering salesman problem. *Transportation science*, 23(3):208–213, 1989.

[8] Balázs Dezső, Alpár Jüttner, and Péter Kovács. Lemon–an open source c++ graph template library. *Electronic Notes in Theoretical Computer Science*, 264(5):23–45, 2011.

[9] Michael Drexl. On the generalized directed rural postman problem. *Journal of the Operational Research Society*, 65(8):1143–1154, 2014.

[10] Horst A Eiselt, Michel Gendreau, and Gilbert Laporte. Arc routing problems, part i: The chinese postman problem. *Operations Research*, 43(2):231–242, 1995.

[11] Ralph E Gomory and Tien Chung Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

[12] Alain Hertz, Gilbert Laporte, and Pierrette Nanchen Hugo. Improvement procedures for the undirected rural postman problem. *INFORMS Journal on computing*, 11(1):53–62, 1999.

[13] Michael Jünger, Thomas M Liebling, Denis Naddef, George L Nemhauser, William R Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A Wolsey. *50 Years of integer programming 1958-2008: From the early years to the state-of-the-art*. Springer Science & Business Media, 2009.

[14] Wen Lea Pearn and Mao Lin Li. Algorithms for the windy postman problem. *Computers & operations research*, 21(6):641–651, 1994.

[15] DG Reina, SL Toral Marin, Nik Bessis, Federico Barrero, and E Asimakopoulou. An evolutionary computation approach for optimizing connectivity in disaster response scenarios. *Applied Soft Computing*, 13(2):833–845, 2013.

[16] Majid Salari and Zahra Naji-Azimi. An integer programming-based local search for the covering salesman problem. *Computers & Operations Research*, 39(11):2594–2602, 2012.

[17] Robert Shuttleworth, Bruce L Golden, Susan Smith, and Edward Wasil. Advances in meter reading: Heuristic solution of the close enough traveling salesman problem over a street network. In *The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 487–501. Springer, 2008.

[18] Scott Allen Welz. *Optimal solutions for the capacitated arc routing problem using integer programming*. PhD thesis, University of Cincinnati, 1994.

Chapter 6

Discussion

Overview. This thesis presents four articles in Chapters 2, 3, 4 and 5 describing computational methods to solve arc routing problems. Our objective was to explore different algorithmic techniques in order to obtain efficient and reliable methods considering the trade-off between solutions quality and computation processing time. We considered heuristic methods, exact methods and lower bounding methods. The heuristic methods aim to obtain high-quality solutions (i.e. close enough to the optimal solution) in relatively short processing time. Exact methods aim to obtain an optimal solution, which for NP-hard problems can be intractable for large instances given their exponential processing times (unless $\mathcal{P} = \mathcal{NP}$). Moreover, exact methods are interesting since many of them are able to obtain non-trivial feasible solutions and tight lower bounds. Finally, lower bounding methods aim to solely generate good lower bounds in a reasonable time without the commitment of finding solutions. Lower bounds are important since they are used to assess the quality of solutions found by other methods, especially those that do not hold approximation guarantees (e.g. heuristics).

PS-Efficiency. The PS-Efficiency heuristic (Chapter 2) was based on well known previous heuristics for the capacitated arc routing problem (CARP). The PS-Efficiency computational complexity was demonstrated and we executed experiments to verify its performance in practice. Our results show that the heuristic consistently outperformed previous heuristics. This is especially important considering that many good performing metaheuristics were proposed [18, 41, 20] where the previous path-scanning heuristics were employed as subroutines (e.g. to initialize a population of solutions).

Moreover, the PS-Efficiency is a simple and easily implemented heuristic. We believe this heuristic can be applied to other arc routing problems that consider a depot and where more side constraints are required (e.g. distance constraints, time-windows and others). This could be done, for example, by adapting the phase of greedy selection of candidate edges, restricting more the set of candidate edges considering other constraints as well.

Hybrid Genetic Algorithm. The *Hybrid Genetic Algorithm* - HGA (Chapter 3) is a metaheuristic developed for the open capacitated arc routing problem (OCARP). The HGA is a genetic algorithm with feasibilization and local search procedures specially tailored for the OCARP. The feasibilization procedure, as shown in the experiments, was essential to obtain a good number of feasible solutions with good variability in a limited

processing time. Previous approaches [39] have had difficulties regarding feasibility of solutions for instances where the number of available vehicles is low in comparison to the amount of demand. We believe the feasibilization procedure can be applied to other routing problems, specially those where finding feasible solutions is a non-trivial problem. The HGA local search procedure was shown to be effective in finding good solutions for the problem. The proposed inter-route procedure of local search employed constructive heuristics in a destroy-and-repair fashion applied to pair of routes which are close to each other. We believe this approach can be successfully applied to other routing problems as soon as there are known good constructive heuristics available for these problems.

Lower bounding methods. Two lower boundings methods (Chapter 4) were proposed for the open capacitated arc routing problem: the branch-and-cut (B&C) method and the relaxed flow-based formulation ($RFB(k)$). The B&C is a branch-and-cut method applied to a relaxed formulation with two sets of constraints and the corresponding exact separation algorithms. The $RFB(k)$ is a parameterized relaxed flow-based formulation where the integer parameter k can control the tightness of the formulation (e.g. how much its optimal solution is close to the OCARP optimal solution).

Using the proposed lower bounding methods we could prove the optimality of many solutions previously obtained by the HGA. From the computational experiments we observed that the B&C and $RFB(k)$ can be complementary approaches: depending on the instances characteristics and available computing power, one method can obtain better results than the other.

The B&C was inspired by similar methods originally proposed for the CARP [12, 32]. The B&C has shown good performance for large instances, therefore improvements on the efficiency of separation algorithms (e.g. heuristic separations) could provide even more progress in tackling very large instances of OCARP. One downside of this method is that its implementation is not trivial, but was simplified to some extent by adopting a library for graph algorithms called LEMON [22].

The $RFB(k)$ considers flow variables to generate relaxed formulations for the problem. We believe the $RFB(k)$ is an original approach by using a graph generation procedure controlled by an integer parameter k in order to impact the formulation tightness. We believe it has potential to be applied to other routing problems as well by appropriate changes in the phase of graph generation. Regarding the performance, the $RFB(k)$ was able to obtain very tight lower bounds for small and medium instances. Also, choosing the value of parameter k was shown to be very important to obtain the best results.

Covering Chinese postman problem. The covering Chinese postman problem - CCPP (Chapter 5) is a new arc routing problem where covering constraints are considered. We modelled the problem by integer programming. A branch-and-cut exact method and a greedy heuristic were proposed. We show that small and medium instances were completely solved by the branch-and-cut. At the same time, large instances with wide covering functions are much harder to solve by the exact method and were better handled by the greedy heuristic.

Computational experiments have shown that the proposed heuristic and exact methods obtained complementary results. Therefore, we believe the development of a matheuristic method [17] could be promising for this problem [9, 35].

Chapter 7

Conclusion

Arc routing problems have attracted attention from many researchers primarily because of their many real-world applications and difficulty of optimization. In this thesis we have studied and proposed computational methods for three arc routing problems: (i) the capacitated arc routing problem (CARP); (ii) the open capacitated arc routing problem (OCARP); and (iii) the covering Chinese postman problem (CCPP). The study of these problems during the doctorate course culminated with the production of a considerable number of publications which will be summarized next.

Table 7.1 contains a summary of journal articles linked to this thesis. Table 7.1 contains the problem name, references to the papers and chapters, status (i.e. submitted or published), year of publication or submission, journal name and the contributions of the paper. The Chapters 2, 3 and 4 corresponds to three papers submitted to or published in the Computers & Operations Research journal.

Table 7.2 summarizes all conference publications published during the doctorate course. Table 7.2 contains the problem name, references to the papers and chapters, type of publication (full paper, abstract or extended abstract), conference initials and year, city of the venue and the contributions of the paper.

All conference publications in Table 7.2 were peer-reviewed, published and given an oral presentation during the conference. The first three entries of Table 7.2 correspond to papers published and presented in the national conference SBPO (Brazilian Symposium of Operational Research) in the years of 2016, 2018 and 2019 presenting partial research results related to the Chapters 3, 4 and 5 respectively. In 2016, in Vitória, the paper [2] presented a partial research of the *hybrid genetic algorithm* for the OCARP. In 2018, in Rio de Janeiro, the paper [3] presented an earlier version of the *flow-based formulation* for the OCARP. In 2019, Limeira, the paper [4] explained some preliminary methods for the CCPP.

The last three publications in Table 7.2 correspond to abstracts and papers presented in international conferences. Paper [42] was presented in the 30th European Conference on Operational Research - Dublin, Ireland, and described the PS-Efficiency heuristic for the CARP. Paper [5] was presented in the International Symposium on Scheduling 2019 - Matsue, Japan, and reported the newly proposed problem CCPP in addition to exact and heuristic methods for solving it. Paper [8] was presented in the 23rd International Congress on Modelling and Simulation - Canberra, Australia, and described the CCPP,

a practical application and an exact method for solving it. This paper was prized with the *best student presentation* award in the operations research track, given by the ASOR - The Australian Society for Operations Research[1].

In order to computationally treat the proposed arc routing problems many algorithmic techniques were explored: linear integer programming, branch-and-cut, network flows, genetic algorithms, randomized greedy heuristics, and others. Many metaheuristics also required specialized data structures in order to efficiently manipulate all the data involved. We tested our methods in benchmark instances proposed by the literature whenever possible, demonstrating their good performance, considering both quality of solutions and processing time. Moreover, we made the experiments data and our methods source codes public on-line [2].

We believe this thesis presents several advances for the state of the art, advancing the scientific knowledge related to the computational treatment of these combinatorial optimization problems. The advance of science in the treatment of combinatorial optimization problems has practical application in several industries, whether minimizing the cost of operations or increasing their reliability. Thus, the application of these techniques has the potential to increase the competitiveness of companies (e.g. the routing of meter readers employed by utility companies).

Future works. For the OCARP, the study of exact methods based on branch-cut-and-price [26, 16] can be very fruitful since this technique is regarded as one of the best options for exactly solving the VRP. The study of variants such as OCARP with time-windows is also interesting. In the case of the CCPP, we believe that a matheuristic method [17], which plays with the strenghts of both mathematical formulation methods and heuristics, is promising. In addition, the creation of new instances based on real world applications of automated meter reading would be enriching for the literature. Finally, the study of a new problem involving the routing and covering of arc-based customers by drones seems to be a promising research direction [19].

[1] `https://www.asor.org.au/`

[2] `http://www.ic.unicamp.br/~fusberti/problems/carp/` and `http://www.ic.unicamp.br/~fusberti/problems/ocarp/`

Table 7.1: Summary of journal papers concluded during the doctorate.

Problem	References	Status	Year	Journal	Contributions
OCARP	[1] (Chapter 3)	Published	2018	Computers & Operations Research	metaheuristics, experiments
CARP	[6] (Chapter 2)	Published	2019	Computers & Operations Research	heuristics, experiments
OCARP	[7] (Chapter 4)	Submitted	2019	Computers & Operations Research	model, exact and lower bound methods, proof of properties, experiments

Table 7.2: Summary of conference papers published during the doctorate.

Problem	Reference	Type	Conference	City	Contributions
OCARP	[2]	full paper	SBPO 2016	Vitória	metaheuristics, experiments
OCARP	[3]	full paper	SBPO 2018	Rio de Janeiro	lower bounding methods, experiments
CCPP	[4]	full paper	SBPO 2019	Limeira	exact and heuristic methods, experiments
OCARP	[42]	abstract	EURO 2019	Dublin	heuristics, experiments
CCPP	[5] (Chapter 5)	full paper	ISS 2019	Matsue	new problem, model, exact and heuristic methods, experiments
CCPP	[8]	extended abstract	MODSIM 2019	Canberra	new problem, model, exact and heuristic methods, experiments

References

[1] Rafael Kendy Arakaki and Fábio Luiz Usberti. Hybrid genetic algorithm for the open capacitated arc routing problem. *Computers & Operations Research*, 90:221–231, 2018.

[2] Rafael Kendy Arakaki and Fábio Luiz Usberti. *Algoritmo Genético para o Problema de Roteamento em Arcos Capacitado e Aberto. In: XLVIII SBPO Brazilian Symposium of Operational Research.* Annals XLVIII SBPO, Vitória, 2016.

[3] Rafael Kendy Arakaki and Fábio Luiz Usberti. *A relaxed flow-based formulation for the open capacitated arc routing problem. In: L SBPO Brazilian Symposium of Operational Research.* Annals L SBPO, Rio de Janeiro, 2018.

[4] Rafael Kendy Arakaki and Fábio Luiz Usberti. *The covering Chinese postman problem: exact and heuristic methods. In: LI SBPO Brazilian Symposium of Operational Research.* Annals LI SBPO, Limeira, 2019.

[5] Rafael Kendy Arakaki and Fábio Luiz Usberti. *The covering Chinese postman problem. In: International Symposium on Scheduling.* ISS 2019, Matsue, Japan, 2019.

[6] Rafael Kendy Arakaki and Fábio Luiz Usberti. An efficiency-based path-scanning heuristic for the capacitated arc routing problem. *Computers & Operations Research*, 103:288 – 295, 2019.

[7] Rafael Kendy Arakaki and Fábio Luiz Usberti. Lower bounding methods for the open capacitated arc routing problem: a branch-and-cut algorithm and a parameterized flow-based formulation. *Submitted to Computers & Operations Research*, 2019.

[8] Rafael Kendy Arakaki, Mutsunori Yagiura, and Fábio Luiz Usberti. *A branch-and-cut algorithm and valid inequalities for the covering Chinese postman problem. In: 23rd International Congress on Modelling and Simulation.* Book of Abstracts MODSIM 2019, Canberra, 2019.

[9] Claudia Archetti and M Grazia Speranza. A survey on matheuristics for routing problems. *EURO Journal on Computational Optimization*, 2(4):223–246, 2014.

[10] Arjang A Assad and Bruce L Golden. Arc routing methods and applications. *Handbooks in operations research and management science*, 8:375–483, 1995.

[11] J.M. Belenguer and E. Benavent. Polyhedral results on the capacitated arc routing problem. Technical Report Technical Report TR01, University of Valencia, Spain, 1992.

[12] José M Belenguer and Enrique Benavent. A cutting plane algorithm for the capacitated arc routing problem. *Computers and Operations Research*, 30(5):705–728, 2003.

[13] José Manuel Belenguer, Enrique Benavent, and Stefan Irnich. The capacitated arc routing problem: Exact algorithms. In Ángel Corberán and Gilbert Laporte, editors, *Arc Routing: Problems, Methods, and Applications*, chapter 9, pages 183–222. SIAM, 2014.

[14] José-Manuel Belenguer, Enrique Benavent, Philippe Lacomme, and Christian Prins. Lower and upper bounds for the mixed capacitated arc routing problem. *Computers & Operations Research*, 33(12):3363–3383, 2006.

[15] M. José Belenguer and Enrique Benavent. The capacitated arc routing problem: Valid inequalities and facets. *Computational Optimization and Applications*, 10:165–187, 1997.

[16] Claudia Bode and Stefan Irnich. Cut-first branch-and-price-second for the capacitated arc-routing problem. *Operations research*, 60(5):1167–1182, 2012.

[17] Marco A Boschetti, Vittorio Maniezzo, Matteo Roffilli, and Antonio Bolufé Röhler. Matheuristics: Optimization, simulation and control. In *International Workshop on Hybrid Metaheuristics*, pages 171–177. Springer, 2009.

[18] José Brandão and Richard Eglese. A deterministic tabu search algorithm for the capacitated arc routing problem. *Computers and Operations Research*, 35(4):1112–1126, 2008.

[19] James F Campbell, Ángel Corberán, Isaac Plana, and José M Sanchis. Drone arc routing problems. *Networks*, 72(4):543–559, 2018.

[20] Yuning Chen, Jin-Kao Hao, and Fred Glover. A hybrid metaheuristic approach for the capacitated arc routing problem. *European Journal of Operational Research*, 253(1):25–39, 2016.

[21] A Corberan, Isaac Plana, and J Sanchis. The chinese postman problem on directed, mixed, and windy graphs. *Arc Routing: Problems, Methods, and Applications; MOS-SIAM Series on Optimization*, pages 65–84, 2014.

[22] Balázs Dezs, Alpár Jüttner, and Péter Kovács. Lemon - an open source c++ graph template library. *Electronic Notes in Theoretical Computer Science*, 264(5):23–45, 2011.

[23] Jack Edmonds and Ellis L Johnson. Matching, euler tours and the chinese postman. *Mathematical programming*, 5(1):88–124, 1973.

[24] Horst A Eiselt, Michel Gendreau, and Gilbert Laporte. Arc routing problems, part i: The chinese postman problem. *Operations Research*, 43(2):231–242, 1995.

[25] Horst A Eiselt and Gilbert Laporte. A historical perspective on arc routing. In *Arc Routing*, pages 1–16. Springer, 2000.

[26] Ricardo Fukasawa, Humberto Longo, Jens Lysgaard, Marcus Poggi De Aragão, Marcelo Reis, Eduardo Uchoa, and Renato F Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical programming*, 106(3):491–511, 2006.

[27] Gianpaolo Ghiani and Gilbert Laporte. A branch-and-cut algorithm for the undirected rural postman problem. *Mathematical Programming*, 87(3):467–481, 2000.

[28] Bruce L Golden and Richard T Wong. Capacitated arc routing problems. *Networks*, 11(3):305–315, 1981.

[29] L. Bruce Golden, S. James DeArmon, and K. Edward Baker. Computational experiments with algorithms for a class of routing problems. *Computers and Operations Research*, 10:47–59, 1982.

[30] Mei Gu Guan. Graphic programming using odd or even points chinese mathematics. *Chinese Mathematics*, 1:237–277, 1962.

[31] Alain Hertz, Gilbert Laporte, and Michel Mittaz. A tabu search heuristic for the capacitated arc routing problem. *Operations research*, 48:129–135, 1999.

[32] Rafael Martinelli, Marcus Poggi, and Anand Subramanian. Improved bounds for large scale capacitated arc routing problem. *Computers & Operations Research*, 40(8):2145–2160, 2013.

[33] CS Orloff. A fundamental problem in vehicle routing. *Networks*, 4(1):35–64, 1974.

[34] Christos H. Papadimitriou. On the complexity of edge traversing. *Journal of the ACM*, 23:544–554, July 1976.

[35] Victor Pillac, Christelle Gueret, and Andrés L Medaglia. A parallel matheuristic for the technician routing and scheduling problem. *Optimization Letters*, 7(7):1525–1535, 2013.

[36] Michael Polacek, Karl Doerner, Richard Hartl, and Vittorio Maniezzo. A variable neighborhood search for the capacitated arc routing problem with intermediate facilities. *Journal of Heuristics*, 14:405–423, 2008.

[37] Christian Prins. The capacitated arc routing problem: Heuristics. In Ángel Corberán and Gilbert Laporte, editors, *Arc Routing: Problems, Methods, and Applications*, chapter 7, pages 131–157. SIAM, 2014.

[38] Luís Santos, João Coutinho-Rodrigues, and John R. Current. An improved heuristic for the capacitated arc routing problem. *Computers and Operations Research*, 36:2632–2637, September 2009.

[39] Fábio Luiz Usberti. *Heuristic and exact approaches for the open capacitated arc routing problem*. PhD thesis, Universidade Estadual de Campinas, 2012.

[40] Fábio Luiz Usberti, Paulo Morelato França, and André Luiz Morelato França. The open capacitated arc routing problem. *Computers and Operations Research*, 38(11):1543 – 1555, 2011.

[41] Fábio Luiz Usberti, Paulo Morelato França, and André Luiz Morelato França. Grasp with evolutionary path-relinking for the capacitated arc routing problem. *Computers and Operations Research*, 40(12):3206–3217, 2013.

[42] Fábio Luiz Usberti and Rafael Kendy Arakaki. *A cost-efficiency heuristic for arc routing problems. In: 30th European Conference on Operational Research*. Annals of the 30th European Conference on Operational Research, Dublin, 2019.

[43] René van Bevern, Rolf Niedermeier, Manuel Sorge, and Mathias Weller. Chapter 2: The complexity of arc routing problems. In *Arc Routing: Problems, Methods, and Applications*, pages 19–52. SIAM, 2015.

Appendix A

Elsevier copyright policy

According to the Elsevier copyright policy, which can be found in the Internet address `https://www.elsevier.com/about/policies/copyright`, the author is allowed to use the contents of their published papers in their thesis: "... the author is allowed to use their articles, in full or in part, for a range of scholarly, non-commercial purposes, which include: [...] Inclusion in a thesis or dissertation (provided that this is not to be published commercially)". We thank the Elsevier Ltd publishing company for allowing us the inclusion of our articles in this thesis.