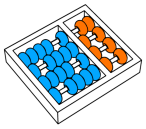
Universidade Estadual de Campinas
Instituto de Computação



Rafael Kendy Arakaki

Estudos Computacionais para o Problema de Roteamento em Arcos Capacitado e Aberto

CAMPINAS
2016

Rafael Kendy Arakaki

Estudos Computacionais para o Problema de Roteamento em Arcos Capacitado e Aberto

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

Orientador: Prof. Dr. Fábio Luiz Usberti

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

CAMPINAS
2016

Agência(s) de fomento e nº(s) de processo(s): CNPq, 162027/2014-1

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

Ar12e

Arakaki, Rafael Kendy, 1993-
Estudos computacionais para o problema de roteamento em arcos capacitado e aberto / Rafael Kendy Arakaki. – Campinas, SP : [s.n.], 2016.

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

1. Problema de roteamento em arcos. 2. Problema de roteamento de leituristas. 3. Algoritmos genéticos. 4. Programação inteira. 5. Otimização combinatória. I. Usberti, Fábio Luiz,1982-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Computational studies for the open capacitated arc routing problem
Palavras-chave em inglês:
Arc routing problem
Meter reading routing problem
Genetic algorithms
Integer programming
Combinatorial optimization
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Fábio Luiz Usberti [Orientador]
Eduardo Candido Xavier
José Federico Vizcaino González
Data de defesa: 08-06-2016
Programa de Pós-Graduação: Ciência da Computação

Universidade Estadual de Campinas
Instituto de Computação

Rafael Kendy Arakaki

Estudos Computacionais para o Problema de Roteamento em Arcos Capacitado e Aberto

Banca Examinadora:

- Prof. Dr. Fábio Luiz Usberti
  Instituto de Computação - UNICAMP
- Prof. Dr. Eduardo Candido Xavier
  Instituto de Computação - UNICAMP
- Prof. Dr. José Federico Vizcaino González
  UNESP - Guaratinguetá

A ata da defesa com as respectivas assinaturas dos membros da banca encontra-se no processo de vida acadêmica do aluno.

Campinas, 08 de junho de 2016

Agradecimentos

Gostaria de agradecer a todas as pessoas que diretamente ou indiretamente contribuíram para que eu pudesse realizar esse trabalho.

Quero agradecer aos meus pais, Paulo e Elaine, à tia Amélia, à tia Alice, à vó Regina e aos demais familiares. Tenho também os meus irmãos, Paulo e Matheus, que agradeço para que não reclamem depois.

Quero agradecer ao meu orientador prof. Fábio Luiz Usberti que me apresentou essa área de pesquisa tão interessante. Além disso, me subsidiou com sua experiência e amizade.

Quero agradecer aos amigos e conhecidos, pelo apoio e momentos de descontração. Em especial, dos colegas da Unicamp, agradeço ao Felipe, Hércules, Leandro e Lucas pelas boas conversas e amizade.

Quero agradecer às diversas pessoas que contribuíram para minha formação acadêmica. Em especial agradeço aos professores da FACOM/UFMS: Débora, Edna, Gonda, Marco Aurélio e Said que de um jeito ou outro me motivaram a continuar na vida acadêmica.

Quero agradecer aos servidores e docentes do IC/UNICAMP, os quais tornam possível esse ambiente acadêmico de excelência que pude usufruir durante o mestrado.

Por fim, agradeço à sociedade brasileira que financia o governo que então financia as agências de fomento que por fim me financiaram. Assim, agradeço ao CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) pelo financiamento desta dissertação.

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. A solução desses problemas remete à diminuição de custos logísticos, melhorando a competitividade das empresas. O Problema de Roteamento em Arcos Capacitado e Aberto (*OCARP - Open Capacitated Arc Routing Problem*) é um problema de otimização combinatória NP-difícil com aplicações práticas, como o problema de roteamento de leituristas e o problema de determinação do caminho de corte.

Esta dissertação de mestrado propõe novos métodos de solução para o OCARP. São propostas uma metaheurística de algoritmos genéticos e uma formulação relaxada de programação linear inteira. Experimentos computacionais comprovam o bom desempenho dos métodos desenvolvidos em comparação aos métodos da literatura. Os resultados demonstram a redução substancial do desvio de otimalidade para todos os grupos de instâncias e parâmetros considerados durante o experimento. Em particular, cinco do total de nove grupos de instâncias foram resolvidos até a otimalidade.

Abstract

Arc routing problems aim at determining the lowest cost routes visiting a subset of edges from a graph, with one or more additional constraints. The solution of these problems leads to lower logistics costs, improving business competitiveness. The Open Capacitated Arc Routing Problem (OCARP) is an NP-hard combinatorial optimization problem with practical applications, such as the meter reading routing problem and the cutting path determination problem.

This master thesis proposes new solution methods for OCARP. It is proposed a genetic algorithm metaheuristic and a relaxed integer linear programming formulation. Computational experiments prove the good performance of the developed methods compared to literature methods. The results show a substantial reduction in optimality deviation for all groups of instances and parameters considered during the experiment. In particular, five from the total of nine instance groups were solved to optimality.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Os problemas de roteamento podem ser subdivididos em duas grandes áreas: os problemas de roteamento em nós e os problemas de roteamento em arcos (*ARPs - Arc Routing Problems*). Os problemas de roteamento em nós, como o clássico Problema do Caixeiro Viajante (*TSP - Travelling Salesman Problem*), objetivam a criação de rota(s) que visita(m) um subconjunto de nós. Por outro lado, os *ARPs* objetivam a criação de rota(s) que visita(m) um subconjunto de arestas. Durante a última década, os problemas de roteamento em arcos têm sido uma área de pesquisa bastante ativa [55].

O problema de roteamento em arcos capacitado e aberto (*OCARP - Open Capacitated Arc Routing Problem*) é um problema de otimização combinatória definido em um grafo conexo não-direcionado com custos e demandas não-negativas nas arestas. É disponibilizada uma frota de veículos com capacidade limitada para atender as demandas das arestas. Para uma aresta ter sua demanda atendida é necessário que um veículo com capacidade suficiente visite-a e decida atender a demanda da aresta, diminuindo por sua vez a capacidade do veículo em questão. O objetivo do OCARP é encontrar um conjunto de rotas que atenda todas as demandas e possua custo mínimo.

O OCARP é um problema NP-difícil, portanto se $P \neq NP$ não existe um algoritmo de tempo polinomial para resolvê-lo [50]. Trata-se de um problema de otimização combinatória que possui diversas aplicações práticas.

Uma das aplicações práticas do OCARP é um problema real de roteamento de leituristas. Empresas distribuidoras de energia elétrica, água e gás possuem colaboradores que registram o consumo de clientes dispersos geograficamente. O roteamento desses funcionários apresenta-se como um problema de difícil solução, o qual consiste em determinar as rotas que os leituristas devem percorrer para realizar as leituras de consumo dos clientes [49]. Nesse problema a demanda configura-se como o tempo de serviço que o leiturista trabalha para realizar a leitura das casas de uma rua e o limite de capacidade é o tempo de serviço de um dia de trabalho. Outra aplicação do OCARP é o problema de determinação do caminho de corte, o qual consiste em guiar o corte de chapas metálicas em peças de formas poligonais utilizadas na indústria metal-mecânica [20, 43].

Apesar de possuir aplicações práticas, a literatura do OCARP é recente e até o momento foram obtidas soluções ótimas somente para instâncias de pequeno porte (menos de 50 arestas) [50]. As metodologias presentes na literatura reportam desvios de otimalidade substanciais (acima de 25%) para as instâncias mais difíceis [50]. Nesse contexto, torna-se

clara a necessidade por novas metodologias de solução.

Considerando a complexidade computacional do OCARP, esta tese propõe uma metodologia de solução heurística baseada em algoritmos genéticos (Capítulo 4). O método desenvolvido melhorou substancialmente as soluções conhecidas para várias instâncias da literatura.

Em relação a métodos exatos, esta tese propõe uma formulação relaxada de *Programação Linear Inteira Mista* (PLIM) para o OCARP (Capítulo 5). Trata-se de uma formulação que consegue produzir limitantes duais mais fortes que o modelo existente na literatura.

Este trabalho está dividido em seis capítulos. O capítulo 2 apresenta conceitos preliminares que são explorados ao longo da tese e uma revisão da literatura. No capítulo 3 são apresentadas a definição e a formulação para o problema de roteamento em arcos capacitado e aberto (OCARP). Adicionalmente, são revisados os principais resultados da literatura relacionada ao OCARP.

O capítulo 4 descreve uma metodologia para solução heurística do OCARP baseada em algoritmos genéticos. Além disso, são apresentados os resultados de experimentos computacionais da heurística. O capítulo 5 define uma formulação matemática relaxada para o OCARP. São apresentados também resultados de experimentos computacionais envolvendo essa formulação. Por fim, o capítulo 6 apresenta as conclusões e comentários finais.

Capítulo 2

Preliminares

2.1 Otimização Combinatória

Otimização combinatória é o ramo da otimização matemática que trabalha com problemas cujo conjunto de soluções factíveis é finito. O conjunto de soluções factíveis, apesar de finito, pode conter um número muito grande de elementos.

A partir do conjunto de soluções factíveis, considera-se normalmente uma função objetivo que atribui para cada solução um valor de custo. Em se tratando de problemas de minimização, o objetivo é encontrar uma solução x^* cujo custo seja o menor possível (nenhuma outra solução possui custo inferior a x^*), e em problemas de maximização procura-se pela solução de maior custo.

Um algoritmo ingênuo para resolver um problema de otimização combinatória é a enumeração de todas as soluções factíveis e a seleção da melhor solução. Entretanto, muitas vezes a utilização desse algoritmo é impraticável, uma vez que o número de soluções factíveis pode ser de ordem exponencial em relação aos dados de entrada (explosão combinatória), gerando assim um algoritmo de complexidade exponencial.

Dentro do ramo da otimização combinatória existem muitas técnicas para tratar esses problemas onde a simples enumeração não é praticável. Este trabalho lida com o estudo de um desses problemas.

2.2 Programação Linear Inteira

Um programa linear pode ser definido como um problema de maximizar ou minimizar uma função linear sujeita a restrições lineares. As restrições são igualdades ou desigualdades que atuam sobre as variáveis de decisão. Em um programa linear as variáveis de decisão são números reais. Considere o seguinte programa linear:

$$MAX \quad \{c^T x : Ax \leq b, x \geq 0\}$$

Onde x representa o vetor de variáveis de decisão (a serem determinadas), c é o vetor de custo que associa para cada variável um peso na função objetivo, b é o vetor de

coeficientes do lado direito das desigualdades (ou igualdades) e A é a matriz de coeficientes das restrições.

A programação linear (PL) é uma ferramenta matemática utilizada em aplicações de diversas áreas, incluindo mas não se restringindo a transportes, energia, telecomunicações, manufaturas, economia e engenharias. Alguns problemas de otimização combinatória podem ser resolvidos através de programação linear. Uma vez que programação linear é um problema polinomial, os problemas NP-difíceis não podem ser resolvidos simplesmente por programação linear (supondo que $P \neq NP$). Sendo assim, esses problemas NP-difíceis precisam de uma modelagem matemática mais genérica e poderosa.

O ramo da Programação Linear Inteira (PLI) trata de programas lineares onde algumas (ou todas) variáveis são inteiras. Isso acrescenta uma flexibilidade muito poderosa aos modelos de PLI. Muitas vezes são utilizadas formulações de programação linear inteira para modelar problemas de otimização combinatória, uma vez que as variáveis discretas podem ser representadas como números inteiros.

Entretanto, diferente da programação linear, a programação linear inteira é um problema *NP-difícil*. Muitas técnicas foram desenvolvidas ao longo dos anos para tratar os problemas de otimização combinatória através de PLI, como os métodos de *branch-and-bound*, *branch-and-cut*, *branch-and-price*, etc.

Para maiores detalhes sobre formulações de programação linear inteira para problemas de otimização, veja [56, 44].

2.3 Limitantes Primais e Duais, Relaxações e Otimalidade

Em problemas de otimização os limitantes primais e duais são valores que limitam o valor (custo) de uma solução ótima. Normalmente o valor da solução ótima é desconhecido, porém através dos limitantes é possível definir um intervalo onde esse valor se encontra. Em se tratando de problemas de minimização, o limitante primal é o limitante superior e o limitante dual é o limitante inferior. Sem perda de generalidade, consideraremos nessa seção o caso dos problemas de minimização.

Seja $c(x)$ o custo de uma solução x. Se existe um limitante primal z_{LP} então há a garantia de que existe uma solução x' tal que $c(x') \leq z_{LP}$. Sabendo que uma solução x^* é ótima segue que $c(x^*) \leq c(x), \forall x$, logo $c(x^*) \leq z_{LP}$, ou seja, o custo da solução ótima é limitado superiormente pelo limitante primal. Portanto uma das maneiras de se obter um limitante primal é encontrar uma solução viável para o problema. Quanto menor o custo dessa solução, menor será o limitante primal e assim mais perto da solução ótima.

Por outro lado, um limitante dual z_{LD} limita inferiormente o custo da solução ótima. Ou seja, garante que não há uma solução de custo menor (melhor) do que z_{LD}. Os limitantes duais podem ser obtidos através de relaxações de um problema. Definimos que um problema é uma relaxação quando o custo de sua solução ótima é sempre inferior ou igual à solução ótima do problema original para a mesma entrada. Isso significa que resolver a relaxação nos dá uma garantia de limitante inferior (limitante dual) z_{LD} para o problema original.

Uma das relaxações mais naturais e diretas para um modelo de Programação Linear Inteira (PLI) é a sua relaxação linear. A relaxação linear de um problema PLI consiste em permitir às variáveis inteiras assumirem valores fracionários, ou seja, a restrição de integralidade das variáveis inteiras é removida. O modelo relaxado possui um conjunto de soluções viáveis que é superconjunto das soluções do modelo PLI. A relaxação linear gera bons limitantes duais quando seu valor ótimo se situa próximo do valor ótimo do modelo PLI. Para muitos problemas, desenvolver um modelo que cumpra esse requisito não é uma tarefa trivial e pode ser considerada uma das principais dificuldades ao se modelar problemas através de programação linear inteira.

Os limitantes primais e duais podem provar a otimalidade de uma solução desde que esses se igualem, isto é, $z_{LD} = c(x) = z_{LP}$. Através desses dois limitantes z_{LP} e z_{LD} podemos definir um desvio de otimalidade $GAP = (z_{LP} - z_{LD})/z_{LD}$ que representa o quão distante z_{LP} se encontra de z_{LD}. Se z_{LP} representa o custo $c(x)$ de uma solução x, então $GAP(x) = (c(x) - z_{LD})/z_{LD}$ representa um limitante superior para o quanto a solução x está distante em relação à solução ótima.

2.4 Problemas de Roteamento em Arcos

Problemas de roteamento em arcos aparecem em diversas áreas de logística e pesquisa operacional. De acordo com Eiselt, Gendreau e Laporte [25] o objetivo dos problemas de roteamento em arcos (*ARPs - Arc Routing Problems*) consiste em determinar a(s) rota(s) de menor custo para visitar alguns arcos do grafo, com uma ou mais restrições complementares.

Muitas aplicações estão relacionadas a problemas de roteamento em arcos, como manutenção de ruas e estradas, medição de leituristas, coleta de lixo, entrega de correios, etc. Detalhes dessas aplicações podem ser encontrados em [26].

Estima-se que bilhões de dólares são despendidos em serviços de roteamento anualmente nos Estados Unidos, principalmente através de recursos públicos, criando-se assim uma demanda por métodos que otimizem os investimentos na área. Nas últimas décadas, o avanço no poder de processamento dos computadores e novas técnicas de otimização contribuíram para a disseminação e adoção de sistemas de *software* para roteamento em arcos [23].

Talvez o primeiro trabalho documentado de *ARPs* seja o problema das sete pontes de Königsberg [27]. Nesse problema a questão é determinar se existe ou não um ciclo que visite exatamente uma vez cada uma das sete pontes do rio Preger na cidade de Königsberg. O matemático Euler descobriu a resposta para esse problema, formalizando as condições para que tal ciclo exista e mostrou que não é possível haver um ciclo assim na cidade de Königsberg.

Ao longo dos anos foram surgindo *ARPs* cada vez mais complexos e visando aplicações mais práticas. Dentre os *ARPs* mais estudados podemos citar o Problema do Carteiro Chinês (Seção 2.4.1), o Problema do Carteiro Rural (Seção 2.4.2) e o Problema de Roteamento em Arcos Capacitado (Seção 2.4.3).

2.4.1 Problema do Carteiro Chinês

O problema do carteiro chinês (*CPP - Chinese Postman Problem*) foi proposto por Meiko [39] que observou o problema ao trabalhar como carteiro durante a revolução cultural chinesa. Nesse problema, o carteiro tem a obrigação de todo dia entregar ou coletar cartas em todas as ruas de uma certa região. Segue a definição formal do problema: dado um grafo $G(V, E)$ com custos não-negativos atribuídos a cada aresta $(i, j) \in E$, o objetivo é encontrar um ciclo de menor custo que visite pelo menos uma vez todas as arestas do grafo.

O CPP é um problema de complexidade polinomial [24]. Eiselt et al. [25] descreve um algoritmo de duas fases para resolvê-lo. A primeira etapa acrescenta um conjunto de arestas de custo mínimo até que o grafo se torne euleriano (todos os nós com grau par). A primeira etapa pode ser reduzida a um problema de emparelhamento de custo mínimo. A segunda etapa constrói um ciclo que visita todas as arestas a partir do grafo euleriano aumentado.

Quando o grafo é misto (contém arestas direcionadas e não-direcionadas) o problema é *NP-difícil* [45] e chamado de problema do carteiro chinês misto (*MCPP - Mixed Chinese Postman Problem*). Frederickson [29] apresentou dois algoritmos 2-aproximados para o MCPP. Além disso, provou que executando-se esses dois algoritmos e escolhendo-se a melhor solução têm-se uma solução 5/3-aproximada. Mais recentemente, Corberán et al. [18] propôs um algoritmo baseado em *branch-and-cut* para a solução exata do problema.

2.4.2 Problema do Carteiro Rural

O Problema do Carteiro Rural (*RPP - Rural Postman Problem*) é similar ao CPP, entretanto no RPP não é requerido que o ciclo visite todas as arestas mas sim um subconjunto de arestas $E_R \subseteq E$, chamado de conjunto de arestas requeridas. O objetivo é encontrar um ciclo de menor custo que visite todas as arestas de E_R.

O RPP foi alvo de muitas pesquisas de metodologias exatas e heurísticas. Em relação a métodos exatos, Corberán, Letchford e Sanchis [17] propuseram um algoritmo de planos de corte que resolveu com otimalidade instâncias com até 196 vértices. Mais tarde, Ghiani e Laporte [31] propuseram um algoritmo *branch-and-cut* capaz de resolver instâncias de até 350 vértices. Mais recentemente, Fernández et al. [28] apresentou um método com uma abordagem diferente que é capaz de utilizar menos variáveis e restrições, alcançando assim resultados melhores que até então.

Em relação a métodos heurísticos, Frederickson [29] desenvolveu um algoritmo 3/2-aproximado. Mais tarde, Fernández de Córdoba et al. [21] propôs uma heurística baseada em métodos de Monte Carlo. Mais recentemente, Hertz et al. [36] descreveram diversos procedimentos de buscas locais para o RPP, essas rotinas podem ser combinadas para compor metaheurísticas para o RPP.

2.4.3 Problema de Roteamento em Arcos Capacitado

O Problema de Roteamento em Arcos Capacitado (CARP) foi proposto por Golden e Wong [33]. Trata-se de um problema de otimização combinatória definido em um grafo

$G(V, E)$ onde existem custos não-negativos c_{ij} e demandas não-negativas d_{ij} associadas a cada aresta. Todas as arestas com demanda positiva precisam ser atendidas por um único veículo de uma frota de M veículos com capacidade limitada D. Ao percorrer o grafo, um veículo pode tanto visitar e atender a demanda de uma aresta ou apenas visitá-la. O CARP considera que há um nó distinto v_0, chamado nó depósito, onde as rotas de cada veículo devem iniciar e terminar. Ao sair do nó depósito o veículo se encontra com capacidade total D que é diminuída após cada atendimento de demanda realizado pelo veículo. O objetivo é encontrar um conjunto de rotas de custo mínimo que atendam todas as demandas das arestas requeridas sem exceder o limite de capacidade dos veículos.

A primeira formulação PLI para o CARP foi descrita por Golden e Wong [33]. Entretanto, essa formulação requer um número exponencial de variáveis e restrições e sua relaxação linear apresenta um limitante dual muito fraco. Posteriormente, Belenguer e Benavent [7] propuseram uma formulação compacta com $O(M|E_R|)$ variáveis inteiras e mais tarde projetaram algoritmos de *branch-and-cut* para essa formulação [8, 9].

Posteriormente Belenguer e Benavent [10] propuseram ainda outra formulação compacta, chamada de formulação superesparsa, com apenas $O(|E|)$ variáveis inteiras. Apesar dessa nova formulação não produzir soluções factíveis para o CARP, ela é capaz de obter bons limitantes duais. Outros limitantes duais foram propostos por Breslin e Keane [15], Amberg e Voss [2], Ahr [1] e Wohlk [54].

Em termos de soluções heurísticas, a literatura do CARP é extensa. Uma das heurísticas mais conhecidas é o Path-Scanning apresentado por Golden et al. [34]. Em relação a metaheurísticas, Hertz et al. [37] apresentou a heurística CARPET baseada em Busca Tabu (*Tabu Search*). Lacomme et al. [40] propôs um algoritmo memético com uma codificação mais simples: uma única rota que atende todas as demandas. Para gerar uma solução CARP, tal rota única é separada em sub-rotas menores através da rotina *Split*.

Mais recentemente, Beullens et al. [13] apresentou uma heurística de Busca Local Guiada (*Guided Local Search*), Brandão e Eglese [14] descreveram uma metaheurística de Busca Tabu determinística e Usberti et al. [52] propuseram uma metaheurística GRASP (*Greedy Randomized Adaptive Search Procedure*).

Mais detalhes de métodos exatos, heurísticos e limitantes inferiores para o CARP podem ser encontrados em [19, 55].

Capítulo 3

Descrição do Problema

3.1 Definição

O Problema de Roteamento em Arcos Capacitado e Aberto (OCARP), proposto por Usberti et al.[53], é definido a seguir. Seja $G(V, E)$ um grafo conexo não-direcionado com custos não-negativos c_{ij} e demandas não-negativas d_{ij} associados a cada aresta. Todas as arestas que possuem demandas positivas ($d_{ij} > 0$) formam o conjunto de arestas requeridas $E_R \subseteq E$. Uma frota de M veículos idênticos com capacidade limitada D é disponibilizada. Ao percorrer o grafo, um veículo pode escolher entre visitar e atender uma aresta ou apenas visitá-la sem atendê-la. No caso de atendimento a sua capacidade é reduzida pela demanda da aresta atendida. Cada aresta requerida deve ser atendida apenas uma vez por um único veículo. Entretanto, as arestas podem ser visitadas múltiplas vezes, por um ou mais veículos.

O OCARP, em contraste com o CARP, considera tanto rotas abertas como fechadas. Uma rota aberta possui nós distintos como nó inicial e nó final, enquanto uma rota fechada possui o mesmo nó inicial e final. Uma solução OCARP consiste em um conjunto de M rotas abertas, uma para cada veículo, para atender todas as arestas requeridas sem exceder a capacidade de nenhum veículo (Figura 3.1). O objetivo do OCARP é minimizar o custo desse conjunto de rotas.

Do ponto de vista teórico, OCARP e CARP são ambos problemas *NP-difíceis*, sendo possível a redução em tempo polinomial de um problema para o outro [50]. Apesar das semelhanças entre CARP e OCARP, existem diferenças importantes que dificultam a adaptação de algoritmos do CARP para se resolver o OCARP. A maioria das metodologias para o CARP considera o número de veículos M como uma variável a ser otimizada, entretanto no OCARP o número de veículos M é um parâmetro fixo. Essa diferença é importante porque torna a geração de uma solução viável para o OCARP mais difícil, uma vez que se o número de veículos e a capacidade dos veículos forem baixos o suficiente, em relação aos valores das demandas, implica-se em resolver um problema *NP-difícil* de empacotamento (*Bin-Packing Problem*) [42].

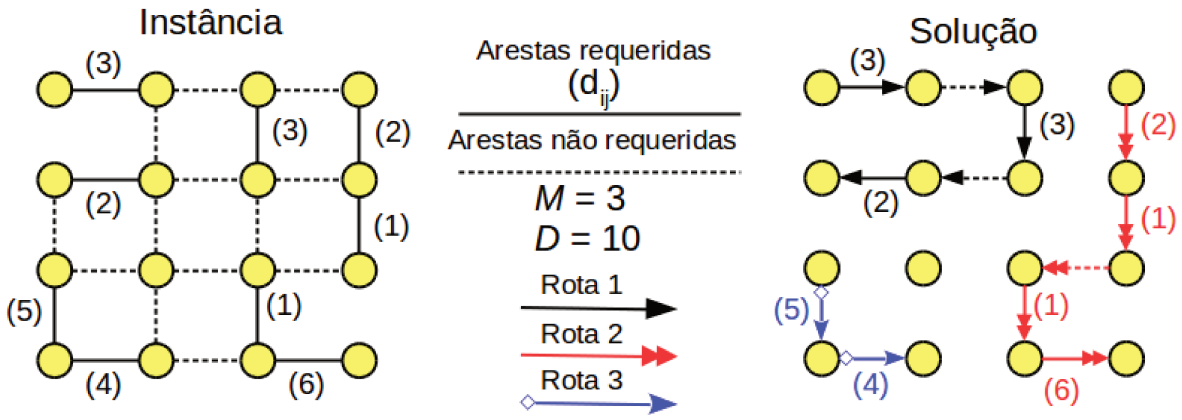


Figura 3.1: OCARP: uma instância e uma solução viável.

3.2 Modelo Matemático

Usberti et al. [53] descreve uma formulação para o OCARP apresentada a seguir.

Apesar do grafo G ser não-direcionado, é conveniente representar soluções OCARP com arcos direcionados para dar a orientação e ordem em que as arestas são visitadas. Desse modo são definidos quatro conjuntos de variáveis:

- $x_{ij}^k = 1$ se a rota k visita o arco (i, j), $x_{ij}^k = 0$ do contrário;
- $l_{ij}^k = 1$ se a rota k atende o arco (i, j), $l_{ij}^k = 0$ do contrário;
- $\alpha_i^k = 1$ se o vértice i é início da rota k, $\alpha_i^k = 0$ do contrário.
- u_S^k, v_S^k e w_S^k $(S \subseteq V)$ são variáveis auxiliares utilizadas no conjunto de restrições (3.7) para eliminação de subciclos ilegais.

A função objetivo (3.1) minimiza o custo total, dado pela soma dos custos das arestas pertencentes a cada rota da solução. Seja $N(i)$ o conjunto de nós vizinhos ao nó i no grafo G, a restrição (3.2) representa a continuidade das rotas e também impõe $\alpha_i^k = 1$ se o vértice i é início da rota k. A restrição (3.3) não permite que haja mais de um vértice como início de rota. A restrição (3.4) estabelece que arcos atendidos precisam ser visitados. A restrição (3.5) força que cada aresta requerida (representada por dois arcos) seja atendida uma única vez por uma única rota. A restrição (3.6) representa o limite de capacidade dos veículos.

O conjunto de restrições (3.7) não permite subciclos ilegais na solução. Para uma certa rota k, a rota é considerada ilegal se existe algum subconjunto de nós S tal que $v_S^k = u_S^k = w_S^k = 1$. A variável u_S^k identifica se há um ciclo em S, a variável v_S^k identifica se não há arestas de corte em $(S, \hat{S})$ e a variável w_S^k identifica se não há um vértice inicial em S. Um exemplo de subciclo ilegal é apresentado na Figura 3.2. O conjunto de restrições (3.7) é de ordem exponencial, portanto na prática essas restrições são adicionadas ao modelo à medida em que são necessárias.

$(OCARP)$

$$MIN \quad \sum_{k=1}^{M} \sum_{(i,j)\in E} c_{ij} x_{ij}^k \tag{3.1}$$

s.a.

$$\sum_{j\in N(i)} (x_{ij}^k - x_{ji}^k) \leq \alpha_i^k \qquad (i \in V, k \in \{1,...,M\}) \tag{3.2}$$

$$\sum_{i\in V} \alpha_i^k \leq 1 \qquad (k \in \{1,...,M\}) \tag{3.3}$$

$$x_{ij}^k \geq l_{ij}^k \qquad ((i,j) \in E_R, k \in \{1,...,M\}) \tag{3.4}$$

$$\sum_{k=1}^{M} (l_{ij}^k + l_{ji}^k) = 1 \qquad ((i,j) \in E_R) \tag{3.5}$$

$$\sum_{(i,j)\in E_R} d_{ij} l_{ij}^k \leq D \qquad k \in \{1,...,M\}) \tag{3.6}$$

$$\left.\begin{aligned} \sum_{(i,j)\in(S,S)} x_{ij}^k - |S|^2 u_S^k \leq |S| - 1 \\ \sum_{(i,j)\in(S,\hat{S})} x_{ij}^k + v_S^k \geq 1 \\ \sum_{i\in S} \alpha_i^k + w_S^k \geq 1 \\ u_S^k + v_S^k + w_S^k \leq 2 \end{aligned}\right\} \qquad (S \subseteq V, \hat{S} = V \setminus S, k \in \{1,...,M\}) \tag{3.7}$$

$$x_{ij}^k \in \{0,1\}, \qquad ((i,j) \in E, k \in \{1,...,M\}) \tag{3.8}$$

$$l_{ij}^k \in \{0,1\}, \qquad ((i,j) \in E_R, k \in \{1,...,M\}) \tag{3.9}$$

$$\alpha_i^k \in \{0,1\}, \qquad (i \in V, k \in \{1,...,M\}) \tag{3.10}$$

$$u_S^k, v_S^k, w_S^k \in \{0,1\}, \qquad (k \in \{1,...,M\}, S \subseteq V) \tag{3.11}$$

Lema 3.2.1. *O conjunto de restrições (3.7) elimina soluções com rotas ilegais (subciclos ilegais) mas não elimina solução viáveis (rotas válidas).*

Demonstração. Primeiro provaremos que o modelo proíbe uma rota ilegal. Suponha uma rota k ilegal e um grafo G' induzido pelas arestas que pertencem à rota k. Se a rota k é ilegal então o grafo G' não é conexo. Pela restrição (3.3) no máximo uma componente de G' terá um vértice de início de rota ($\alpha_i^k = 1$). Excluindo-se a componente que tem o vértice inicial, há pelo menos mais uma. Essa(s) outra(s) componente(s) de G' deverão formar subciclo(s) para manter o equilíbrio da restrição (3.2). Seja S' o conjunto de vértices de uma dessas componentes que não contém o vértice inicial. Então é possível verificar que:

- $u_{S'}^k = 1$, pois S' corresponde a um subciclo.
- $v_{S'}^k = 1$, porque S' é uma componente em G' (não há arestas de corte).

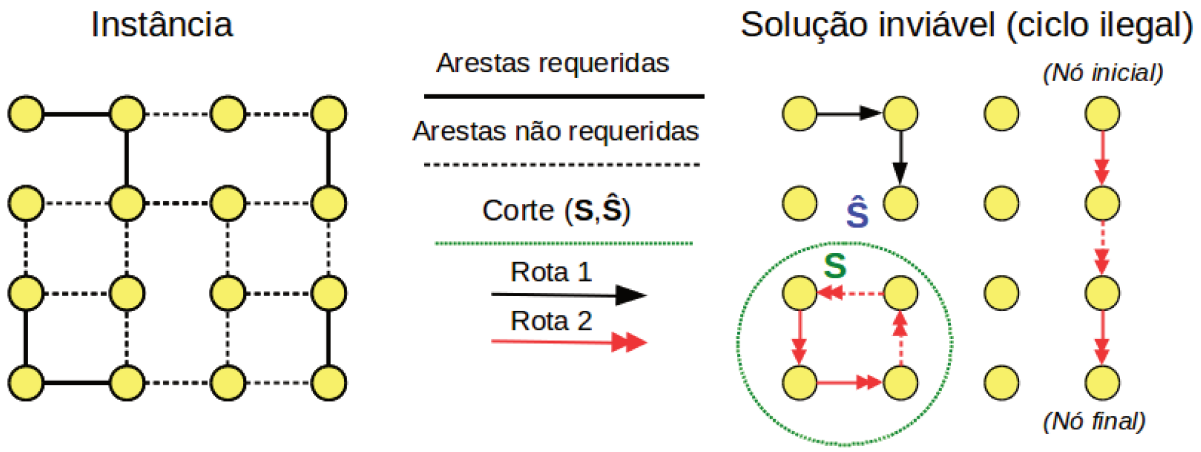


Figura 3.2: Instância OCARP (à esquerda) e uma solução inviável por causa do subciclo ilegal formado na rota 2 (à direita).

- $w^k_{S'} = 1$, não há nó inicial em S'.

Temos portanto que qualquer rota k que gere mais de uma componente em um grafo G' induzido por suas arestas é ilegal no modelo (subciclo ilegal). Para completar a prova, mostraremos que o conjunto de restrições (3.7) não proíbe rotas legais, ou seja, rotas que gerem somente uma componente em um grafo G' induzido pelas suas arestas.

Suponha uma rota válida k e o conjunto R formado pelos vértices que são visitados por esta rota (vértices de G'). Temos então três casos possíveis para um subconjunto de vértices $S \subseteq V$ em relação a R:

1. Se $R \subseteq S$, um dos vértices da rota deve ser o inicial (em caso de rota fechada qualquer vértice pode ser inicial), portanto $w^k_S = 0$.

2. Se $R \not\subseteq S$ e $R \cap S \neq \varnothing$, então há pelo menos uma aresta de corte, que liga um elemento de $R \cap S$ com um elemento de $R \setminus S$, portanto $v^k_S = 0$.

3. Se $R \cap S = \varnothing$, não há arestas $(i, j) \in (S, S)$ na rota, portanto $u^k_S = 0$.

Como conclusão temos que o conjunto de restrições (3.7) proíbe exatamente as rotas que não são conexas (rotas ilegais) e em conjunto com as outras restrições do modelo define um conjunto de rotas válidas para o OCARP. □

3.3 Trabalhos Anteriores

Os experimentos com a formulação original (Seção 3.2) realizados por Usberti et al. [53] reportou bons resultados para instâncias pequenas (menos de 30 nós), porém não alcançou bom desempenho para instâncias maiores (desvios de otimalidade acima de 20%). Conforme reportado pelos autores, a principal deficiência nesse método apresentou ser o limitante dual fraco e a dificuldade em executar na prática um modelo com quantidade exponencial de restrições e variáveis.

Mais tarde outro método de solução exata para o OCARP foi descrito por Usberti et al. [51]. Trata-se de um algoritmo *branch-and-bound* que se vale da relação entre o OCARP e outro problema chamado *Capacity and Degree Constrained Minimum Spanning Forest* (CDCMSF) [50]. Esse método aprimorou os limitantes duais conhecidos, porém para o conjunto de instâncias mais difícil *oegl* ainda foi reportada uma média de desvio de otimalidade substancial (acima de 15%), evidenciando assim a necessidade por métodos mais eficientes.

Em termos de heurísticas, Usberti et al. [53] propuseram uma heurística construtiva de path-scanning que mostrou-se eficaz na construção de soluções viáveis para o OCARP.

3.4 Considerações Finais

O OCARP trata-se de um problema recente da literatura, apresentado em 2011 por Usberti et al. [53]. É um problema de otimização combinatória da classe de roteamento em arcos. Uma vez sendo um problema *NP-difícil*, metodologias heurísticas podem ser consideradas para obtenção de soluções de boa qualidade na prática.

As metodologias propostas para o OCARP na literatura apresentam desvios de otimalidade substanciais (acima de 15%) para muitas instâncias [50], expondo assim a demanda por melhorias, tanto em metodologias exatas quanto heurísticas. Nesse sentido, este trabalho apresenta duas novas metodologias para tratar o problema em duas frentes: uma metaheurística de algoritmos genéticos para gerar soluções primais (Capítulo 4) e uma formulação relaxada de programação linear inteira mista para gerar limitantes duais (Capítulo 5).

Capítulo 4

Algoritmo Genético

Muitos problemas combinatórios NP-difíceis são resolvidos através de métodos heurísticos, uma vez que a solução exata desses problemas requer tempo exponencial [30] (supondo $P \neq NP$). Uma das metaheurísticas mais utilizadas para a solução de problemas de otimização são algoritmos genéticos [32].

Algoritmos genéticos são algoritmos inspirados por elementos da evolução das espécies como seleção, herança, mutação e cruzamento [38]. Resumidamente, os algoritmos genéticos são baseados nas seguintes características:

- Codificação da Solução: Representação de uma solução como uma cadeia de genes, onde se aplicarão as operações de cruzamento e mutação.
- Inicialização da População: Uma população de indivíduos (soluções) é criada para que o algoritmo comece a processar a primeira geração.
- Função de *Fitness*: Avalia uma solução de acordo com a sua aptidão ao problema (em analogia à aptidão do indivíduo ao seu meio ambiente).
- Seleção: Seleciona quais serão os indivíduos que irão reproduzir e quais irão sobreviver até a próxima geração. Os indivíduos com maiores níveis de *fitness* possuem maiores chances de sobreviver e reproduzir.
- Cruzamento (*crossover*): Faz uma recombinação de genes de dois indivíduos progenitores (pais) para formar um novo indivíduo.
- Mutação: Tem como objetivo aumentar a diversidade genética através de mudanças aleatórias nos genes.

Nas últimas décadas diversos trabalhos utilizaram algoritmos genéticos para tratar problemas de roteamento [16, 40, 3]. Historicamente, a maioria desses métodos utilizou a estratégia *cluster-first route-second*, que consiste em primeiramente separar os elementos com demanda (nós ou arestas) em conjuntos que respeitam as restrições específicas do problema e depois são formadas rotas onde cada grupo é atendido por um dos veículos.

Entretanto, têm-se aumentado o número de trabalhos de roteamento que utilizam a estratégia *route-first cluster-second* [6]. Essa estratégia consiste em primeiramente resolver

o problema de roteamento e posteriormente separar esse roteamento em rotas menores que atendam às restrições do problema. O presente trabalho apresenta um algoritmo genético com estratégia *route-first cluster-second* para o OCARP.

4.1 Descrição da Metaheurística

A metaheurística apresentada nesta seção trata-se de um algoritmo genético híbrido. Além das rotinas clássicas de algoritmos genéticos como inicialização de população, seleção e cruzamento, essa heurística também considera as seguintes componentes: (i) rotina de factibilização, (ii) rotina de montagem de codificação, (iii) busca local, (iv) reinício de população. Essa heurística não considera nenhum operador de mutação. O Algoritmo 1 mostra o pseudo-código do algoritmo genético.

A rotina de factibilização é utilizada para se obter uma solução factível a partir da codificação de um indivíduo. Essa rotina é muito importante para o bom desempenho da heurística uma vez que determinar uma solução viável para o OCARP é um problema NP-difícil (discussão na Seção 3.1). A rotina de montagem de codificação possui o objetivo inverso da rotina de factibilização: a partir de uma solução factível procura-se determinar uma codificação.

A busca local é utilizada para aprimorar as soluções geradas ao longo do processo. Por fim, o reinício de população justifica-se para evitar a convergência prematura da população, uma vez que não é considerado nenhum operador de mutação.

Algoritmo 1: Algoritmo Genético

```
início
    Realizar o pré-processamento.
    Gerar população inicial.
    enquanto limite de tempo não for excedido faça
        Selecionar pais para cruzamento.
        para um par de pais selecionados (pai1,pai2) faça
            c = cruzamento(pai1,pai2)
            s = factibilização(c)
            s = busca_local(s)
            c = montagem(s)
            Adicione o indivíduo à população.
        fim
        Selecionar nova população.
        Reinício de população a cada k gerações.
    fim
fim
```

4.1.1 Pré-processamento

A primeira etapa consiste em uma rotina de pré-processamento. Sendo E_R o conjunto de arestas requeridas, cria-se um novo conjunto A_R de arcos requeridos, onde cada aresta requerida $\{i, j\} \in E_R$ gera dois arcos requeridos $(i, j), (j, i) \in A_R$. Em seguida é gerada

uma matriz R de dimensão $|A_R| \times |A_R|$. Cada entrada $R[e, f]$ dessa matriz é definida como o custo do caminho mínimo a partir do vértice fim do arco $e \in A_R$ até o vértice início do arco $f \in A_R$. Desse modo o custo da distância entre arcos requeridos pode ser recuperada em tempo $O(1)$.

4.1.2 Codificação

O algoritmo genético utiliza como codificação C de uma solução OCARP uma sequência de $|E_R|$ arcos requeridos, que corresponde à sequência na qual as arestas requeridas são atendidas. Considera-se que o caminho percorrido entre dois arcos requeridos corresponde a um caminho mínimo obtido na fase de pré-processamento. Desse modo, não há necessidade de codificar arestas não requeridas uma vez que elas estão implícitas nos caminhos mínimos entre pares de arcos requeridos consecutivos de C.

A codificação não possui nenhum delimitador entre as diferentes rotas, de modo que a codificação pode ser interpretada como uma única rota não-capacitada, como mostra a Figura 4.1. A solução OCARP correspondente à codificação somente é obtida após a etapa de factibilização (Seção 4.1.7).

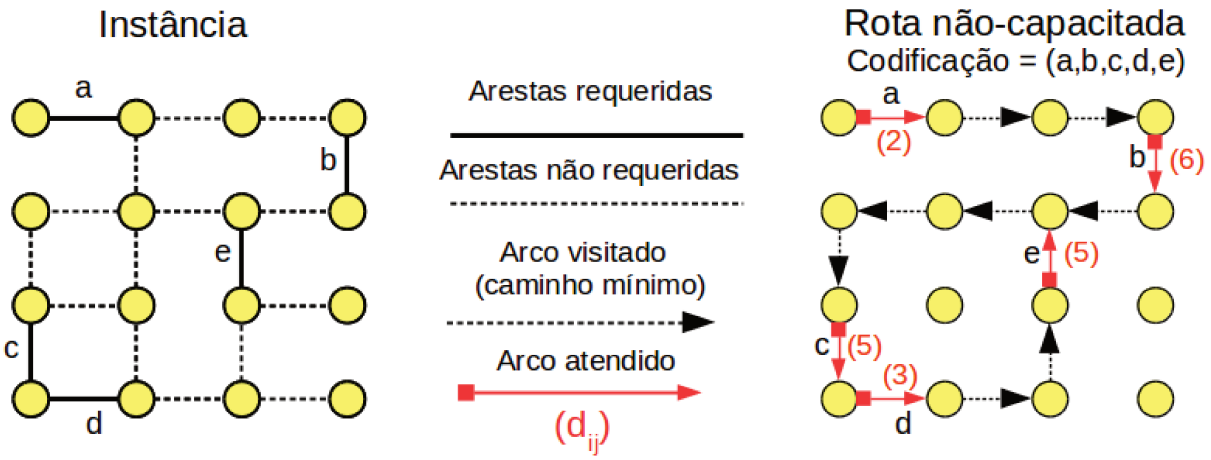


Figura 4.1: Instância com $E_R = \{e, d, c, b, a\}$ e uma codificação com sua rota não-capacitada.

4.1.3 Fitness

A função de *fitness* representa a qualidade de uma solução. A função de *fitness* de uma solução s de custo $c(s)$ é definida como $1/(c(s) - lb_0)$, onde lb_0 é um limitante inferior trivial para o problema: a soma do custo de todas as arestas requeridas. Se for encontrada uma solução s onde $c(s) = lb_0$, então o algoritmo é encerrado pois foi encontrada uma solução ótima. O custo da solução $c(s)$ e seu valor de *fitness* são calculados somente durante a etapa de montagem de codificação (Seção 4.1.9).

4.1.4 Inicialização da População

O objetivo dessa rotina é gerar P indivíduos que correspondam a soluções viáveis OCARP, onde P é o tamanho da população. O processo para gerar cada indivíduo da população é dividido em duas etapas. Na primeira etapa resolve-se o problema de roteamento não-capacitado e na segunda etapa procura-se particionar a rota não-capacitada em sub-rotas capacitadas, gerando assim soluções factíveis para o OCARP.

Na primeira etapa a rota não-capacitada é construída por via de uma adaptação da heurística *Nearest Neighbour* [5]. A rota é inicializada com um arco requerido aleatório e então é aumentada de maneira gulosa e iterativa até conter todas as arestas requeridas. Cada aresta requerida $\{i, j\} \in E_R$ pode ser inserida à rota em uma de duas direções $(i, j), (j, i) \in A_R$ e sua inserção pode ocorrer no início ou ao fim da rota. Se houver empate no critério guloso, a escolha é aleatória.

Em seguida a rota não-capacitada é otimizada pelo método de busca local 2-opt conforme descrito por Groves et al. [35] com a única exceção de que o OCARP trabalha com rotas abertas portanto não há uma ligação do último arco requerido da rota até o primeiro. Para a busca local 2-opt utiliza-se a estratégia *first-improvement*, onde a cada etapa a solução s é substituída por uma solução vizinha s' de menor custo aleatória, não necessariamente a melhor solução vizinha de s (o que caracterizaria uma estratégia *best-improvement*).

A segunda etapa divide a rota não-capacitada em M sub-rotas capacitadas através da rotina de factibilização (Seção 4.1.7). Se não for possível factibilizar a codificação não-capacitada, essa codificação é descartada e outra codificação é gerada. O processo se repete até que toda a população seja preenchida com indivíduos que representam soluções viáveis.

4.1.5 Seleção

São realizados $P - 1$ cruzamentos a cada geração, tal que P é o tamanho da população e cada cruzamento requer a seleção de dois indivíduos.

A seleção dos indivíduos para cruzamentos é realizada através da técnica *Stochastic Universal Selection* de Baker [4]. Essa técnica seleciona os indivíduos em quantidade proporcional à razão de seu *fitness* individual sobre a soma de *fitness* de toda a população e se mostrou mais eficiente do que o método tradicional de roleta.

A nova população é formada pelo melhor indivíduo da população corrente e pelos $P - 1$ indivíduos filhos da geração corrente (seleção elitista). Entretanto, se houver L cruzamentos que geraram indivíduos filhos infactíveis, isto é, para os quais a rotina de factibilização falhou, então são escolhidos aleatoriamente L indivíduos da geração atual para serem inseridos na nova geração. Desse modo a população consiste sempre de indivíduos que representam soluções OCARP viáveis.

4.1.6 Cruzamentos

A etapa de cruzamento tem por objetivo construir uma nova codificação (filho) a partir de duas codificações da população (pais). Foi adotado o operador de cruzamento C1 de

Reeves [47], onde um gene g é escolhido aleatoriamente e copia-se para o filho os genes do início até g do primeiro pai. Em seguida são copiados para o filho os genes que faltam na ordem em que aparecem no segundo pai (Figura 4.2).

Ao final do cruzamento a codificação não-capacitada do filho contém arcos requeridos que correspondem a todo o conjunto de arestas requeridas E_R. Em testes preliminares empíricos o operador de cruzamento C1 apresentou desempenho um pouco melhor do que os operadores OX (*Order Crossover*) e LOX (*Linear Order Crossover*).

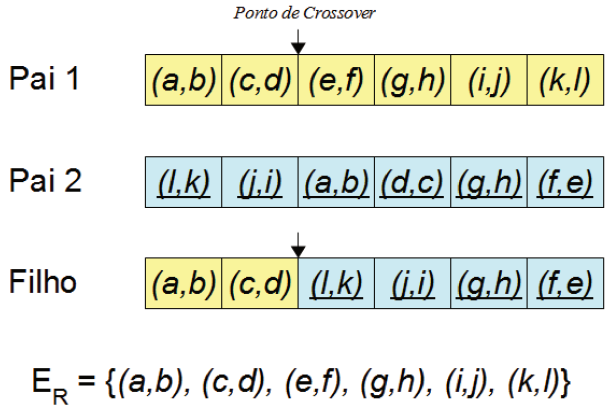


Figura 4.2: Operador de cruzamento $C1$.

4.1.7 Factibilização

O objetivo da rotina de factibilização é dividir a rota não-capacitada (representada pela codificação C) em M sub-rotas capacitadas (solução viável). O algoritmo chave utilizado pela rotina de factibilização é o algoritmo *Split*, descrito primeiramente para o CARP por Ulusoy [48]. O algoritmo *Split* realiza uma decomposição ótima da rota não-capacitada em sub-rotas capacitadas, se houver decomposição viável. A saída do algoritmo *Split* são M sub-rotas que atendem os arcos requeridos na mesma ordem da rota não-capacitada e não excedem o limite de capacidade dos veículos.

O algoritmo *Split* inicia-se com a criação de um grafo auxiliar H, chamado de grafo de Ulusoy. O funcionamento do algoritmo é ilustrado através da Figura 4.3. No retângulo (1) temos a instância e uma codificação não-capacitada que atende todas as arestas requeridas em uma determinada ordem. No retângulo (2) são apresentados os caminhos mínimos entre cada par de arcos requeridos consecutivos. Os caminhos mínimos são obtidos em tempo $O(1)$ por meio da matriz R (Seção 4.1.1). No retângulo (3) se encontra o grafo de Ulusoy H correspondente à codificação. Trata-se de um grafo acíclico e dirigido onde cada vértice representa um arco requerido de C. Além disso, cada aresta de H representa uma sub-rota capacitada de C que é viável em respeito à capacidade. O custo de cada sub-rota é dado pela soma dos caminhos mínimos entre cada par de arcos requeridos consecutivos daquela sub-rota. O pseudo-código do algoritmo *Split* é apresentado no Algoritmo 2.

A solução do algoritmo *Split* é a forma ótima (com relação ao custo) de dividir uma rota não-capacitada em M sub-rotas que respeitam a capacidade dos veículos e cuja ordem de

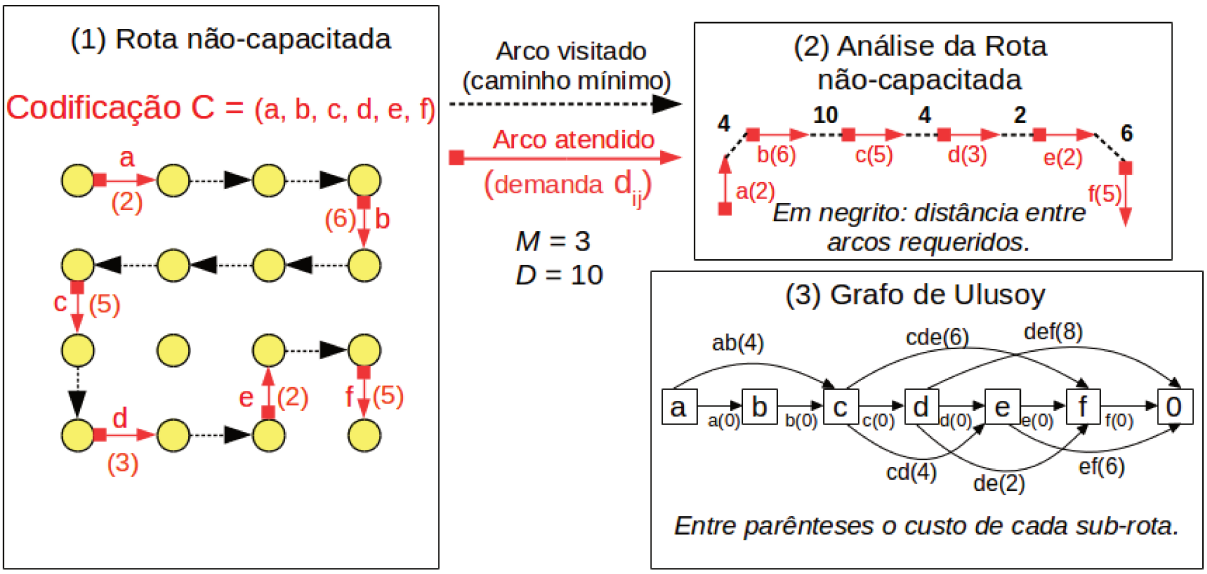


Figura 4.3: Funcionamento do algoritmo *Split*: montagem do grafo H.

atendimento dos arcos requeridos é a mesma da rota não-capacitada. Essa solução é obtida através do caminho mínimo no grafo H do primeiro até o último vértice (ordem topológica) composto por até M arestas. Uma vez que o grafo é acíclico e dirigido o caminho mínimo de até M arcos pode ser obtido através do Algoritmo 3 que trata-se de uma adaptação do algoritmo de Bellman-Ford. A solução OCARP obtida pelo algoritmo *Split* é composta pelas sub-rotas que caracterizam cada arco do caminho mínimo encontrado pelo Algoritmo 3. A Figura 4.4 ilustra a etapa de definição das sub-rotas.

Algoritmo 2: Algoritmo *Split*

Entrada: C: codificação; R: matriz de distâncias entre arcos requeridos; M : número de veículos; D: capacidade dos veículos; d: vetor de demandas dos arcos requeridos;

Saída: sol: solução; $factivel$: viabilidade; $H(V, A)$: grafo de Ulusoy;

Objetivo: Encontrar um particionamento de custo mínimo da rota não-capacitada C em M rotas capacitadas (se houver algum particionamento factível).

início

- $V = \emptyset; A = \emptyset;$
- **para** $i = 1$ *até* $|C| + 1$ **faça**
  - $V = V \cup v_i;$
- **fim**
- **para** $i = 1$ *até* $|C|$ **faça**
  - $cap = d[C_i];$
  - $dist = 0;$
  - $A = A \cup (v_i, v_{i+1})$ com peso 0; // *sub-rota unitária*
  - **para** $j = i + 1$ *até* $|C|$ **faça**
    - $cap = cap + d[C_j];$
    - $dist = dist + R[C_{j-1}][C_j];$
    - **se** $cap \leq D$ **então**
      - $A = A \cup (v_i, v_{j+1})$ com peso $dist$; // *sub-rota viável (capacitada)*
    - **fim**
  - **fim**
- **fim**
- $factivel = false; sol = \emptyset;$
- Obter caminho mínimo em $H(V, A)$ de v_1 até $v_{|C|+1}$ com até M arcos; // Algoritmo 3
- **se** $\exists$ *caminho mínimo de* v_1 *até* $v_{|C|+1}$ **então**
  - $factivel = true;$
  - $sol =$ solução OCARP dada pela codificação C e caminho mínimo em H;
- **fim**
- **return** (sol, $factivel$, H);

fim

Algoritmo 3: Caminho mínimo de até M arcos em DAG.

Entrada: $H(V, A)$: grafo dirigido; $peso$: vetor de pesos para cada arco $\in A$; M: número de veículos; v_1: primeiro vértice da ordenação topológica de $H(V, A)$;

Saída: d: distâncias; π predecessores;

Objetivo: Encontrar o caminho mínimo no grafo H de v_1 até todos os vértices utilizando-se no máximo M arcos.

início

- Inicialize $d[v] = \infty$ e $\pi[v] = NIL$ para cada $v \in V$.
- Faça $d[v_1] = 0$.
- **para** $k = 1$ *até* M **faça**
  - **para** $v_i \in V$ *em ordem topológica inversa* **faça**
    - **para** $(v_i, v_j) \in A$ **faça**
      - **se** $d[v_i] + peso[(v_i, v_j)] < d[v_j]$ **então**
        - $d[v_j] = d[v_i] + peso[(v_i, v_j)];$
        - $\pi[v_j] = v_i;$
      - **fim**
    - **fim**
  - **fim**
- **fim**

fim

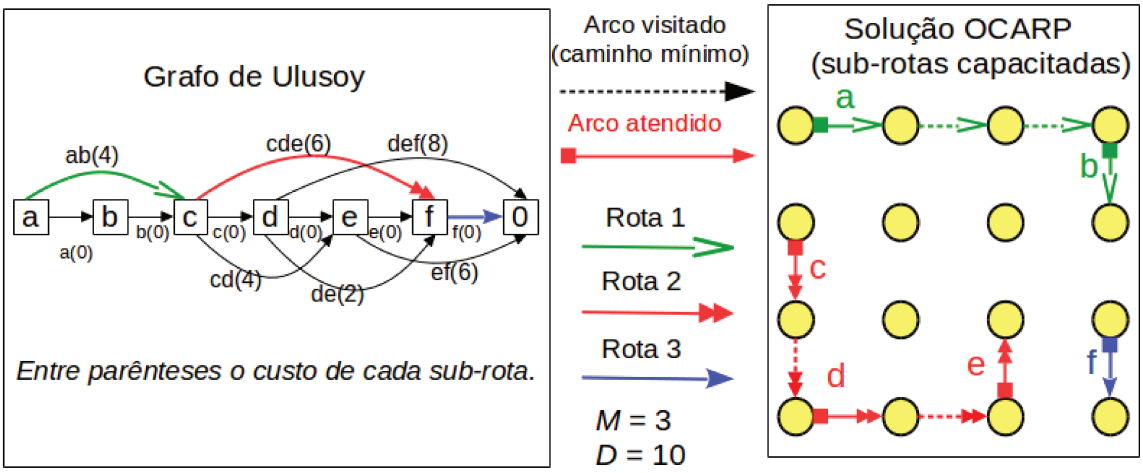


Figura 4.4: Funcionamento do algoritmo *Split*: particionamento da rota.

Devido ao número fixo de veículos e das capacidades limitadas, não é possível garantir que a solução gerada pelo algoritmo *Split* seja factível. Esse caso acontece quando na etapa de obter o caminho mínimo no grafo H descobre-se que não há nenhum caminho de até M arestas do primeiro até o último nó. Nesse caso é adotada uma estratégia para tentar encontrar uma solução factível. Dentre todas as sub-rotas geradas pelo algoritmo *Split* (arcos do grafo H), escolhe-se aquela com a menor capacidade residual e fixa-se ela em uma nova solução parcial (Figura 4.5). Em seguida, são incluídas em uma nova codificação somente as arestas requeridas faltantes utilizando o método guloso descrito na Seção 4.1.4. Finalmente, chama-se a rotina *Split* com essa nova codificação parcial dos arcos restantes, porém dessa vez utilizando um veículo a menos. A cada iteração pelo menos uma sub-rota capacitada é incorporada à solução final. Se após M iterações não for encontrada uma solução factível, o indivíduo é declarado inviável. A estratégia adotada procura fazer um uso econômico das capacidades dos veículos, no sentido de facilitar a busca por uma solução factível nas iterações seguintes.

O Algoritmo 4 descreve o processo da rotina de factibilização em pseudo-código. O objetivo da rotina de factibilização é guiar o algoritmo *Split* para encontrar uma solução factível mesmo se o número de veículos M for muito limitado. Quando o algoritmo *Split* não consegue resolver o problema diretamente, é utilizada a estratégia de fixar a rota de menor capacidade residual. Ao fixar-se a rota que utiliza melhor a capacidade do veículo, aumenta-se as chances de, nas próximas iterações da rotina de factibilização, o algoritmo *Split* encontrar uma solução viável.

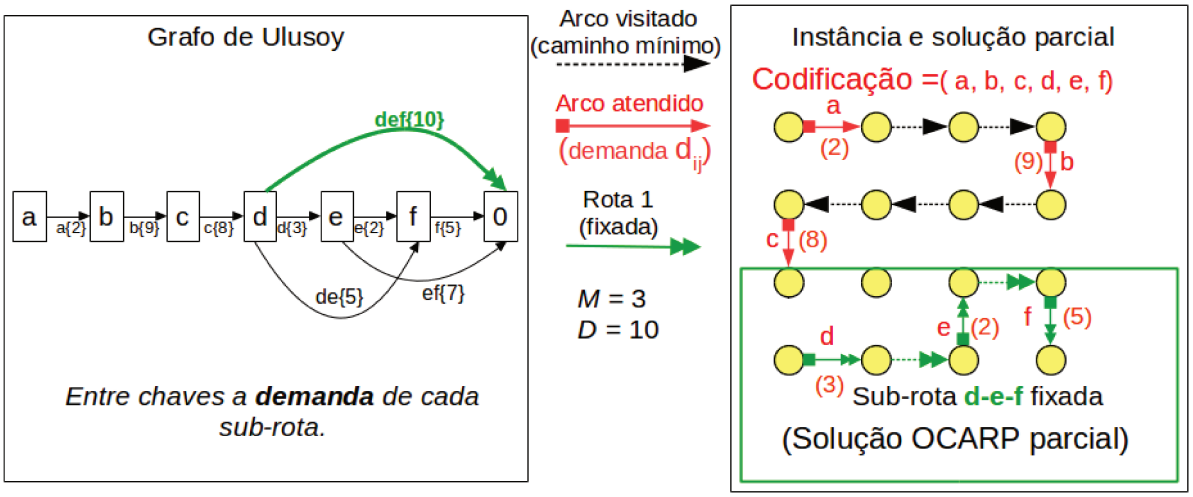


Figura 4.5: Estratégia para lidar com a inviabilidade das soluções.

Algoritmo 4: Rotina de Factibilização

Entrada: C: codificação; R: matriz de distâncias entre arcos requeridos; M: número de veículos; D: capacidade dos veículos; d: demandas;
Saída: sol: solução; $factivel$: viabilidade;
Objetivo: Gerar uma solução OCARP com base na codificação não-capacitada C, utilizando-se do algoritmo *Split* e de uma estratégia para lidar com soluções infactíveis.

```
início
    sol = ∅; factivel = false;
    C' = C;
    para k = 1 até M faça
        M' = M − k + 1;
        sol', H = Split(C', R, M', D, d);
        se sol' é factível então
            factivel = true;
            sol = sol ∪ sol';
            return sol, factivel;
        fim
        senão
            Seja SB_min a sub-rota de menor capacidade residual de H;
            sol = sol ∪ SB_min; // SB_min é uma rota da solução parcial sol.
            C' = C' \ SB_min; // Retire de C' a sub-cadeia que representa SB_min.
            Seja C'' uma codificação não-capacitada criada com o algoritmo de inicialização de
            indivíduo da população, mas roteando apenas os arcos requeridos presentes em C';
            C' = C'';
        fim
    fim
    sol = ∅;
    return sol, factivel;
fim
```

Análise de Complexidade

Para descrever a complexidade da rotina de factibilização (Algoritmo 4) primeiramente serão descritas: (i) a complexidade do algoritmo *Split* (Algoritmo 3), (ii) a complexidade

do algoritmo de inicialização de um indivíduo (Seção 4.1.4).

A complexidade do algoritmo *Split* (Algoritmo 3) gira em torno principalmente do tamanho da codificação C que é $O(|E_R|)$. A etapa de criação das arestas do grafo é $O(|E_R|^2)$. O algoritmo de caminho mínimo com número limitado de arestas é $O(M|V|^2)$. Como $|V|$ é igual a $|C|+1$, temos que a complexidade do algoritmo de caminho mínimo é $O(M|E_R|^2)$. Segue que a complexidade do algoritmo *Split* é $O(|E_R|+|E_R|^2+M|E_R|^2) = O(M|E_R|^2)$.

A complexidade do algoritmo de inicialização de um indivíduo (Seção 4.1.4) é a soma entre as etapas da heurística gulosa *Nearest Neighbour* e de busca local *2-OPT*. O algoritmo *Nearest Neighbour* é um algoritmo guloso e funciona em $|E_R|$ iterações, onde cada iteração verifica-se a distância para os outros $O(|E_R|)$ arcos requeridos, logo possui tempo $O(|E_R|^2)$. A complexidade exata da busca local *2-opt* é difícil de analisar uma vez que o número de iterações pode variar muito até que seja encontrado um mínimo local. Seja c um limitante superior para o número de iterações para a busca local *2-opt* encontrar um mínimo local. A complexidade do *2-opt* é c vezes o custo de cada operação de vizinhança. Cada operação de vizinhança do *2-opt* analisa todos os pares de arcos requeridos em tempo $O(|E_R|^2)$. Conclui-se que o *2-opt* possui complexidade $O(c|E_R|^2)$. Portanto a complexidade do algoritmo de inicialização de um indivíduo é $O(c|E_R|^2)$.

Apresenta-se a seguir a análise de complexidade da rotina de factibilização (Algoritmo 4). O laço principal da rotina é limitado por $O(M)$ iterações. Cada laço principal executa uma rotina *Split* e pode chamar o algoritmo de inicialização de um indivíduo. Portanto a complexidade da rotina de factibilização é $O(M(M|E_R|^2 + c|E_R|^2)) = O(M^2|E_R|^2 + cM|E_R|^2) = O(\max\{c, M\}M|E_R|^2)$.

4.1.8 Busca Local

Algoritmos de busca local realizam pequenas modificações em uma solução visando soluções similares (vizinhas) de melhor custo. As soluções geradas no cruzamento possuem uma probabilidade p de serem submetidas à rotina de busca local.

Boa parte das buscas locais para problemas de roteamento consideram uma vizinhança complexa, muitas vezes sendo realizado um grande esforço computacional para a avaliação dessas vizinhanças [40, 46]. O método proposto neste trabalho é relativamente simples e reutiliza rotinas usadas em outras componentes do algoritmo genético. O Algoritmo 5 descreve a rotina da busca local em pseudo-código.

A busca local proposta utiliza uma estratégia de desconstrução/reconstrução de pares de rotas próximos. A Figura 4.6 ilustra o processo. Para selecionar pares de rotas promissores, primeiramente identifica-se um conjunto S de r rotas suficientemente próximas a um arco requerido (u, v) específico. Em outras palavras, são escolhidas as r rotas que atendem arcos requeridos mais próximos do arco requerido (u, v) para formar o conjunto S. Em seguida, cada par de rotas do conjunto S é submetido à fase de reconstrução.

A fase reconstrução, a partir de duas rotas, procura criar duas novas rotas que atendam às mesmas arestas requeridas. Para isso são utilizados os métodos de inicialização da população (Seção 4.1.4), com o diferencial de que são roteadas apenas as arestas requeridas do par desconstruído e considera-se apenas dois veículos. A substituição das rotas

desconstruídas pelas novas rotas somente é efetivada se diminuir o custo da solução. O Algoritmo 5 descreve a busca local em pseudo-código.

Algoritmo 5: Busca Local

Entrada: sol: solução; r: parâmetro; R: matriz de distâncias entre arcos requeridos; M: número de veículos; D: capacidade dos veículos; d: demandas;
Saída: sol: solução;
Objetivo: Aprimorar uma solução OCARP com base numa estratégia de desconstrução e reconstrução de pares de rotas "próximos".

```
início
    para (u, v) ∈ E_R faça
        Seja S um vetor zerado de M posições.
        // Identifica um conjunto de rotas próximas à aresta requerida (u, v).
        para i = 1 até M faça
            dist = +∞;
            para (a, b) ∈ sol[i] faça
                // Denota-se sol[i] a i-ésima rota da solução sol.
                dist = min{dist, R[(u, v)][(a, b)], R[(u, v)][(b, a)],
                R[(v, u)][(a, b)], R[(v, u)][(b, a)]};
            fim
            S[i] = (dist, i); // S[i] recebe um par ordenado, onde dist é a chave.
        fim
        Ordene o vetor S em relação à chave.
        para i = 1 até r faça
            para j = i + 1 até r faça
                (m1, r1) = S[i];
                (m2, r2) = S[j];
                Seja W o conjunto de arestas requeridas das rotas sol[r1] e sol[r2].
                Seja C_0 o custo de roteamento de sol[r1] e sol[r2] somados.
                Seja N_1 e N_2 as rotas criadas com o algoritmo de inicialização de indivíduo,
                seguido da rotina de factibilização considerando apenas 2 veículos e arestas
                requeridas ∈ W.
                se N_1, N_2 factíveis e seus custos somados inferior a C_0 então
                    sol[r1] = N_1;
                    sol[r2] = N_2;
                fim
            fim
        fim
    fim
    return sol;
fim
```

Análise de Complexidade

A análise de complexidade da busca local (Algoritmo 5) é dada a seguir. O laço principal do algoritmo é executado $O(|E_R|)$ vezes. Dentro do laço existem três grandes seções: a primeira que identifica as distâncias das rotas é $O(M|E_R|)$, a segunda que realiza a

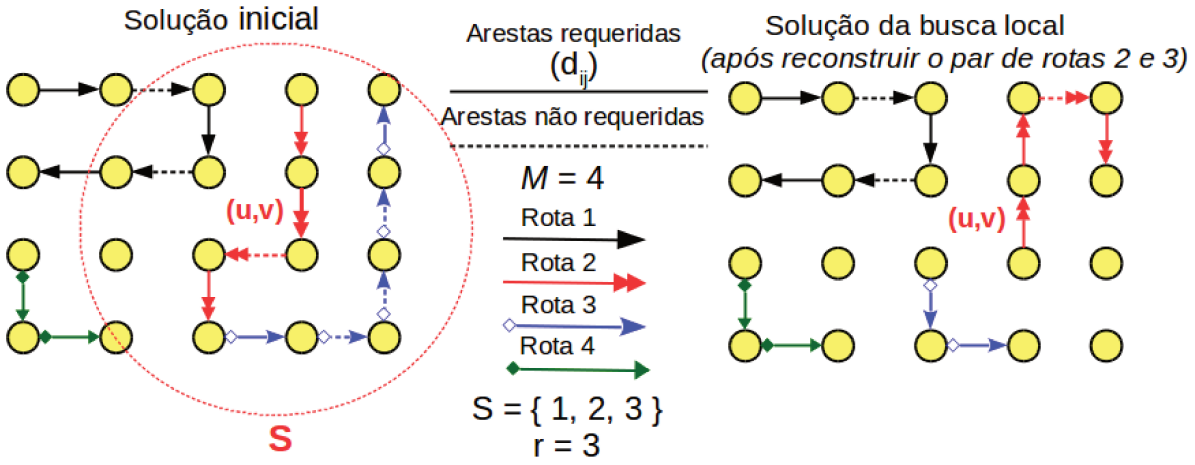


Figura 4.6: Funcionamento da busca local. Em destaque o conjunto S com as r rotas mais próximas à aresta requerida (u, v).

ordenação dos elementos em S é $O(M \log M)$, e a terceira que possui dois laços aninhados que executam os algoritmos de inicialização populacional e de factibilização.

Os dois laços aninhados da terceira etapa executam um número de iterações $O(r^2)$, onde r é um parâmetro constante que regula o tamanho do conjunto de rotas próximas. Em cada iteração o número de elementos de W é $O(|E_R|)$ então o algoritmo de inicialização tem complexidade $O(c|E_R|^2)$, onde c é um limitante superior do número de iterações da busca local *2-opt* (discussão na Seção 4.1.7). A rotina de factibilização para número de rotas $M = 2$ possui complexidade $O(c|E_R|^2)$. Portanto a terceira etapa tem complexidade $O(r^2((c|E_R|^2) + (|E_R|^2))) = O(r^2 c|E_R|^2)$. Na prática r é um parâmetro constante independente da entrada ($r = 4$ é o valor adotado nos experimentos da Seção 4.2), e nessas condições a complexidade pode ser descrita como $O(c|E_R|^2)$.

Somando as complexidades das três etapas e multiplicando pelo número de iterações do laço principal temos $O((M|E_R| + M \log M + c|E_R|^2)|E_R|) = O(c|E_R|^3 + M|E_R|^2 + M \log M|E_R|)$. Um problema OCARP não trivial possui $M < |E_R|$, do contrário temos uma solução ótima onde cada aresta requerida é atendida por um veículo distinto. Logo a complexidade da busca local pode ser descrita por $O(c|E_R|^3)$.

4.1.9 Montagem de Codificação

A entrada do algoritmo de factibilização consiste em uma rota não-capacitada. Entretanto, a saída dos algoritmos de factibilização ou busca local é uma solução OCARP que não necessariamente é representada corretamente pela codificação da rota não-capacitada que originou essa solução. A rotina de montagem de codificação objetiva a montagem de uma codificação compatível com a solução OCARP.

Para que a codificação seja compatível basta que as rotas da solução estejam presentes na codificação em qualquer ordem. A Figura 4.7 mostra uma possível codificação compatível para uma solução OCARP. Essa codificação é dada por *(a,b, c,d, e)*. As seguintes codificações também são compatíveis: *(a,b, e, c,d)* e *(e, a,b, c,d)*. Portanto a ordem das

rotas na codificação é irrelevante para que a codificação seja compatível. A função de *fitness* (Seção 4.1.3) considera o custo da solução capacitada para comparar os indivíduos. Entretanto, é a rota não-capacitada, que representa a solução capacitada, que é utilizada para a reprodução (Seção 4.1.6), portanto é importante que as rotas não-capacitadas tenham baixo custo de roteamento.

Para encontrar uma rota não-capacitada compatível e com baixo custo de roteamento a rotina de montagem de codificação utiliza os métodos de inicialização de indivíduo: constrói-se uma solução com *Nearest Neighbour* e aprimora-se através da busca local *2-opt*. A diferença nesse caso é que as arestas que devem ser roteadas na verdade são as M rotas. Nesse caso, cada rota pode ser representada como uma aresta onde uma das extremidades é o vértice origem do primeiro arco requerido da rota e a outra extremidade é o vértice destino do último arco requerido da mesma rota. As distâncias entre as rotas já foram calculadas (na matriz R). A saída desse algoritmo é uma codificação compatível com a solução OCARP e que possui um custo de roteamento otimizado.

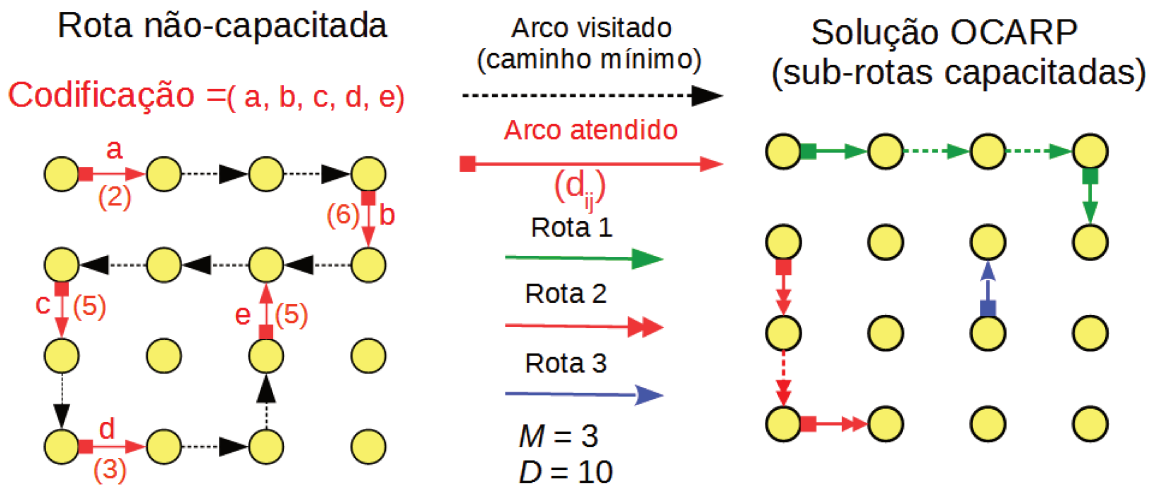


Figura 4.7: Uma solução OCARP (à direita) e uma das possíveis codificações compatíveis (à esquerda).

4.1.10 Reinício de População

A rotina de reinício de população é executada repetidamente ao longo da metaheurística em um intervalo de k gerações. Essa rotina tem o objetivo de evitar a convergência prematura da população.

A cada execução da rotina, metade da população atual é substituída por novas soluções. Os indivíduos substituídos são escolhidos aleatoriamente, entretanto a melhor solução nunca é substituída (seleção elitista). As novas soluções são geradas pelo método de inicialização da população (Seção 4.1.4).

4.2 Experimentos Computacionais

Este experimento visa verificar a qualidade das soluções geradas pelo algoritmo genético. Para isso, é apresentada nesta seção uma comparação entre as soluções obtidas pelo algoritmo genético com as soluções obtidas pela heurística construtiva RPS_ER (*Reactive Path-Scanning with Ellipse Rule*) de Usberti et al. [53].

A máquina utilizada no experimento compõe-se de processador Intel Xeon X3430 2.4 GHz, 8 GB de memória RAM e sistema operacional Linux 64-bit. O algoritmo genético foi implementado em linguagem C++ e utilizou-se a biblioteca LEMON [22] versão 1.3.1 para a computação de algoritmos básicos em grafos. A heurística RPS_ER foi implementada em linguagem C sendo o código cedido pelos autores. Ambas heurísticas foram executadas na mesma máquina e limitou-se o tempo de execução em uma hora para cada instância. Os parâmetros definidos para o algoritmo genético são apresentados na Tabela 4.1.

Os resultados são apresentados nas Tabelas 4.2, 4.3 e 4.4 para os respectivos conjuntos de instâncias: *ogdb* (7-27 nós, 11-55 arestas), *oval* (24-50 nós, 34-97 arestas) e *oegl* (77-140 nós, 98-190 arestas) [53, 34, 12, 41]. Para cada instância, foram considerados três valores distintos para o parâmetro M (número de veículos): M^*, $M^* + 1$ e $M^* + 2$, onde M^* é o número mínimo de veículos necessários para se obter uma solução factível (para obter o valor de M^* foi necessário resolver o *Bin-Packing Problem* [42]).

O valor de LP_1 corresponde à melhor solução obtida pela heurística RPS_ER e LP_2 à melhor solução obtida pelo algoritmo genético. Dados os valores de LP_1 e LP_2 é calculado um valor $\Delta LP(\%)$ que corresponde à diferença proporcional entre os dois limitantes.

Tabela 4.1: Parâmetros do algoritmo genético

Parâmetro	Descrição	Valor
P	Tamanho da população	20
p	Probabilidade de busca local	0.1
r	Tamanho do conjunto de rotas da busca local	4
k	Intervalo de reinício da população (gerações)	250

Tabela 4.2: Resultados da heurística para as instâncias *ogdb*.

		*M**					*M*+1*					*M*+2*				
Instância	M^*	LP_1	LP_2	$\Delta LP(\%)$	CPU_1	CPU_2	LP_1	LP_2	$\Delta LP(\%)$	CPU_1	CPU_2	LP_1	LP_2	$\Delta LP(\%)$	CPU_1	CPU_2
gdb1	5	252	252	0.00	0	0	252	252	0.00	0	0	252	252	0.00	0	0
gdb2	6	291	291	0.00	0	0	291	291	0.00	0	0	291	291	0.00	0	0
gdb3	5	233	233	0.00	0	0	233	233	0.00	0	0	233	233	0.00	0	0
gdb4	4	238	238	0.00	0	0	238	238	0.00	0	0	238	238	0.00	0	0
gdb5	6	316	316	0.00	0	0	316	316	0.00	0	0	316	316	0.00	0	0
gdb6	5	260	260	0.00	0	0	260	260	0.00	0	0	260	260	0.00	0	0
gdb7	5	262	262	0.00	0	0	262	262	0.00	0	0	262	262	0.00	0	0
gdb8	10	210	210	0.00	397	1	210	210	0.00	28	0	210	210	0.00	168	0
gdb9	10	220	219	**0.45**	3571	0	219	219	0.00	77	0	219	219	0.00	8	0
gdb10	4	252	252	0.00	0	0	252	252	0.00	0	0	252	252	0.00	0	0
gdb11	5	362	362	0.00	3	1	360	360	0.00	159	1	360	358	**0.56**	158	1
gdb12	7	336	336	0.00	0	0	336	336	0.00	0	0	336	336	0.00	0	0
gdb13	6	509	509	0.00	0	0	509	509	0.00	0	0	509	509	0.00	0	0
gdb14	5	96	96	0.00	0	0	96	96	0.00	0	0	96	96	0.00	0	0
gdb15	4	56	56	0.00	0	0	56	56	0.00	0	0	56	56	0.00	0	0
gdb16	5	119	119	0.00	0	0	119	119	0.00	0	0	119	119	0.00	0	0
gdb17	5	84	84	0.00	0	0	84	84	0.00	0	0	84	84	0.00	0	0
gdb18	5	158	158	0.00	0	0	158	158	0.00	0	0	158	158	0.00	0	0
gdb19	3	45	45	0.00	0	0	45	45	0.00	0	0	45	45	0.00	0	0
gdb20	4	105	105	0.00	0	0	105	105	0.00	0	0	105	105	0.00	0	0
gdb21	6	149	149	0.00	0	0	149	149	0.00	0	0	149	149	0.00	0	0
gdb22	8	191	191	0.00	0	0	191	191	0.00	0	0	191	191	0.00	0	0
gdb23	10	223	223	0.00	1	0	223	223	0.00	0	0	223	223	0.00	0	0
média				**0.02**	172	< 1			**0.00**	11	< 1			**0.02**	14	< 1

LP_1: limitante primal obtido pela heurística RPS_ER.

LP_2: limitante primal obtido pelo algoritmo genético.

$\Delta LP(\%) = 100(LP_1 - LP_2)/LP_1$: taxa de melhoria de LP_1 para LP_2.

CPU_1 e CPU_2: tempo para heurísticas obterem LP_1 e LP_2 respectivamente (em segundos).

Tabela 4.3: Resultados da heurística para as instâncias *oval*.

		*M**					*M*+1*					*M*+2*				
Instância	M^*	LP_1	LP_2	$\Delta LP(\%)$	CPU_1	CPU_2	LP_1	LP_2	$\Delta LP(\%)$	CPU_1	CPU_2	LP_1	LP_2	$\Delta LP(\%)$	CPU_1	CPU_2
val1A	2	154	154	0.00	0	0	154	151	**1.95**	0	1	154	149	**3.25**	0	1
val1B	3	151	151	0.00	32	1	149	149	0.00	1	1	149	147	**1.34**	1	1
val1C	8	150	148	**1.33**	1478	0	146	146	0.00	3	0	146	146	0.00	0	0
val2A	2	195	195	0.00	0	0	195	192	**1.54**	0	1	195	189	**3.08**	0	1
val2B	3	192	192	0.00	1	0	192	189	**1.56**	1	1	192	186	**3.13**	1	0
val2C	8	185	185	0.00	174	0	185	185	0.00	3	0	185	185	0.00	0	0
val3A	2	71	71	0.00	0	0	71	69	**2.82**	0	1	71	67	**5.63**	0	1
val3B	3	69	69	0.00	0	0	67	67	0.00	4	1	67	66	**1.49**	4	1
val3C	7	66	66	0.00	122	0	65	65	0.00	1	0	65	65	0.00	1	0
val4A	3	358	358	0.00	213	4	358	354	**1.12**	945	4	358	350	**2.23**	950	4
val4B	4	354	354	0.00	453	6	354	350	**1.13**	505	3	354	347	**1.98**	507	2
val4C	5	353	350	**0.85**	720	457	350	347	**0.86**	819	2	350	345	**1.43**	823	2
val4D	9	351	343	**2.28**	1743	326	346	343	**0.87**	1498	1	346	343	**0.87**	1594	1
val5A	3	383	383	0.00	103	3	381	378	**0.79**	713	4	381	374	**1.84**	713	3
val5B	4	378	378	0.00	89	3	376	374	**0.53**	563	2	376	371	**1.33**	563	2
val5C	5	375	374	**0.27**	882	2	372	371	**0.27**	2880	2	372	368	**1.08**	2878	1
val5D	9	370	367	**0.81**	1448	1	370	367	**0.81**	320	1	370	367	**0.81**	320	0
val6A	3	195	195	0.00	12	1	195	193	**1.03**	12	2	195	192	**1.54**	12	1
val6B	4	194	194	0.00	151	1	192	192	0.00	233	1	192	191	**0.52**	233	1
val6C	10	190	190	0.00	314	0	190	190	0.00	80	0	190	190	0.00	67	0
val7A	3	263	263	0.00	700	3	263	259	**1.52**	226	4	263	256	**2.66**	224	3
val7B	4	259	259	0.00	0	2	259	256	**1.16**	1862	3	259	253	**2.32**	1860	2
val7C	9	256	250	**2.34**	124	2	252	249	**1.19**	1004	1	250	249	**0.40**	227	0
val8A	3	364	364	0.00	9	16	359	359	0.00	1435	3	359	354	**1.39**	1436	3
val8B	4	359	359	0.00	448	3	354	354	0.00	499	2	354	351	**0.85**	500	2
val8C	9	352	347	**1.42**	458	11	347	347	0.00	588	0	348	347	**0.29**	373	0
val9A	3	299	298	**0.33**	498	16	299	294	**1.67**	498	11	299	292	**2.34**	497	9
val9B	4	298	294	**1.34**	768	10	298	292	**2.01**	1358	7	298	290	**2.68**	1353	6
val9C	5	296	292	**1.35**	2455	6	294	290	**1.36**	2661	5	294	288	**2.04**	2651	4
val9D	10	292	282	**3.42**	375	152	288	281	**2.43**	84	19	288	280	**2.78**	84	1
val10A	3	403	402	**0.25**	3222	12	403	396	**1.74**	3231	13	403	391	**2.98**	3223	11
val10B	4	399	396	**0.75**	1224	11	399	391	**2.01**	1225	9	399	388	**2.76**	1225	7
val10C	5	397	391	**1.51**	2585	7	394	388	**1.52**	2870	6	394	385	**2.28**	2870	5
val10D	10	391	379	**3.07**	1582	10	387	378	**2.33**	1709	10	387	377	**2.58**	1714	2
média				**0.63**	658	32			**1.01**	818	4			**1.76**	791	3

LP_1: limitante primal obtido pela heurística RPS_ER.

LP_2: limitante primal obtido pelo algoritmo genético.

$\Delta LP(\%) = 100(LP_1 - LP_2)/LP_1$: taxa de melhoria de LP_1 para LP_2.

CPU_1 e CPU_2: tempo para heurísticas obterem LP_1 e LP_2 respectivamente (em segundos).

Tabela 4.4: Resultados da heurística para as instâncias *oegl*.

		*M**					*M*+1*					*M*+2*				
Instância	M^*	LP_1	LP_2	$\Delta LP(\%)$	CPU_1	CPU_2	LP_1	LP_2	$\Delta LP(\%)$	CPU_1	CPU_2	LP_1	LP_2	$\Delta LP(\%)$	CPU_1	CPU_2
egl-e1-A	5	1802	1775	**1.50**	2691	13	1755	1708	**2.68**	2983	11	1755	1659	**5.47**	2974	29
egl-e1-B	7	1823	1749	**4.06**	1134	5	1727	1639	**5.10**	1150	6	1726	1589	**7.94**	2002	568
egl-e1-C	10	1723	1652	**4.12**	2968	65	1651	1576	**4.54**	1238	60	1610	1542	**4.22**	2830	1
egl-e2-A	7	2302	2173	**5.60**	2882	343	2256	2072	**8.16**	2602	56	2256	2043	**9.44**	2594	5
egl-e2-B	10	2249	2079	**7.56**	1466	118	2175	1997	**8.18**	419	1	2166	1971	**9.00**	1635	2
egl-e2-C	14	2424	2084	**14.03**	2936	255	2192	1996	**8.94**	1070	1685	2151	1964	**8.69**	1580	107
egl-e3-A	8	2856	2526	**11.55**	2753	301	2685	2410	**10.24**	1425	364	2676	2366	**11.58**	867	381
egl-e3-B	12	2759	2409	**12.69**	2483	563	2690	2351	**12.60**	2797	14	2596	2321	**10.59**	2207	158
egl-e3-C	17	2804	2357	**15.94**	2550	925	2643	2286	**13.51**	1216	1606	2565	2265	**11.70**	2746	307
egl-e4-A	9	3095	2631	**14.99**	2300	730	2945	2582	**12.33**	1435	39	2825	2556	**9.52**	3234	559
egl-e4-B	14	3198	2696	**15.70**	663	88	2905	2552	**12.15**	3351	2986	2853	2517	**11.78**	2834	328
egl-e4-C	19	3789	2812	**25.79**	714	3004	2923	2515	**13.96**	1364	354	2805	2497	**10.98**	1968	72
egl-s1-A	7	1942	1799	**7.36**	392	29	1787	1683	**5.82**	1957	39	1787	1604	**10.24**	1958	2
egl-s1-B	10	1859	1745	**6.13**	1378	2	1817	1660	**8.64**	1882	1	1729	1579	**8.68**	3391	1173
egl-s1-C	14	2080	1874	**9.90**	1450	1686	1847	1633	**11.59**	1432	2	1757	1512	**13.94**	3101	1
egl-s2-A	14	4303	3689	**14.27**	3306	68	4092	3621	**11.51**	2218	242	4068	3561	**12.46**	3584	463
egl-s2-B	20	4979	3790	**23.88**	913	755	4098	3492	**14.79**	3255	3107	4009	3427	**14.52**	164	276
egl-s2-C	27	4812	3732	**22.44**	2841	1338	4201	3395	**19.19**	2767	1486	3904	3336	**14.55**	1545	1173
egl-s3-A	15	4298	3808	**11.40**	2862	2330	4242	3761	**11.34**	326	2653	4242	3703	**12.71**	327	3229
egl-s3-B	22	4403	3670	**16.65**	751	2554	4204	3582	**14.80**	546	854	4158	3559	**14.41**	276	580
egl-s3-C	29	4782	3742	**21.75**	3269	3233	4253	3564	**16.20**	2433	2651	4102	3492	**14.87**	2322	2613
egl-s4-A	19	5174	4476	**13.49**	1693	3377	5045	4421	**12.37**	2822	193	4965	4399	**11.40**	879	184
egl-s4-B	27	5403	4412	**18.34**	1875	3488	4983	4333	**13.04**	413	1838	4973	4311	**13.31**	3230	942
egl-s4-C	35	8023	5017	**37.47**	3324	3100	5443	4379	**19.55**	3163	1865	5019	4284	**14.64**	1141	2750
média				**14.03**	2066	1182			**11.30**	1844	922			**11.11**	2057	663

LP_1: limitante primal obtido pela heurística RPS_ER.

LP_2: limitante primal obtido pelo algoritmo genético.

$\Delta LP(\%) = 100(LP_1 - LP_2)/LP_1$: taxa de melhoria de LP_1 para LP_2.

CPU_1 e CPU_2: tempo para heurísticas obterem LP_1 e LP_2 respectivamente (em segundos).

A Tabela 4.2 reporta que não houve muita diferença entre os dois métodos para as instâncias *ogdb*, mas isso é principalmente devido a tratar-se de instâncias pequenas que foram bem resolvidas por ambos métodos. Não obstante, é possível verificar a vantagem de desempenho do algoritmo genético referente ao tempo de processamento necessário para se encontrar a melhor solução de algumas instâncias.

A partir das Tabelas 4.3 e 4.4 é possível observar que o algoritmo genético obteve resultados expressivamente melhores do que a metodologia da literatura. Em muitas instâncias a diminuição do custo das soluções superou 15%, com a maior diminuição sendo de 37,47% na instância *egl-s4-C* que é a maior instância considerada nos experimentos. Além disso, em relação ao tempo de processamento, a Tabela 4.3 mostra que para muitas instâncias o algoritmo genético obteve uma solução de melhor custo em um tempo substancialmente inferior à heurística RPS_ER.

Um aspecto que se mostrou de grande importância para a solução de uma instância OCARP é o valor do parâmetro M que limita o número de veículos disponíveis. Nas Tabelas 4.3 e 4.4 é possível verificar que nos casos onde $M = M^* + 1$ ou $M = M^* + 2$ a heurística tomou um tempo de processamento médio consideravelmente menor do que quando $M = M^*$.

É possível observar que o conjunto de instâncias *oegl* foi o que apresentou a maior diferença de limitantes $\Delta LP(\%)$, especialmente quando considerado o número de veículos igual a M^*. Podemos atribuir o bom desempenho do algoritmo genético nessas condições à rotina de factibilização que permite ao algoritmo encontrar soluções factíveis de bom custo mesmo se o número de veículos M é limitado.

4.3 Considerações Finais

Esse capítulo descreveu o algoritmo genético desenvolvido para o OCARP. As principais características dessa heurística são:

- Codificação de uma solução através de uma rota não-capacitada.
- Algoritmo *Split* para decompor uma rota não-capacitada em sub-rotas capacitadas (solução).
- Rotina de Factibilização com estratégias para guiar o algoritmo *Split* e obter soluções viáveis.
- Busca Local de Desconstrução e Reconstrução para aprimorar as soluções.

A ideia de codificação através de rota não-capacitada está consolidada na literatura para diversos problemas de roteamento, assim como o algoritmo *Split* [40, 48]. Duas contribuições deste trabalho são a rotina de factibilização e a busca local de desconstrução e reconstrução. A rotina de factibilização foi projetada visando tratar a dificuldade do OCARP para se encontrar soluções viáveis quando o número de veículos M é muito limitado, configurando assim um problema de empacotamento de difícil resolução. A busca local foi desenvolvida com vistas a melhorar pequenas partes (pares de rotas) de uma solução OCARP.

Os resultados obtidos pelo algoritmo genético são favoráveis e mostram que o método conseguiu superar as metodologias existentes na literatura. Entretanto para se provar a otimalidade dos resultados obtidos pelo algoritmo genético ou pelo menos comprovar seus

desvios de otimalidade são necessários métodos que gerem bons limitantes duais. Para isso serão investigados no Capítulo 5 limitantes duais a partir de uma formulação relaxada do OCARP.

Capítulo 5

Formulação por Fluxos

A formulação por fluxos trata-se de uma formulação de *programação linear inteira mista* (PLIM) relaxada para o OCARP. Essa formulação utiliza conceitos de fluxos em redes sobre um grafo orientado G' gerado a partir do grafo original G.

Tradicionalmente os modelos de fluxos em redes contém demandas em nós e os fluxos transitam através das arestas. Uma vez que as demandas do OCARP se situam nas arestas, estas precisam ter suas demandas transferidas para os nós. Essa atribuição da demanda de uma aresta requerida para um nó precisa ser de modo exclusivo, isto é, nenhum nó pode receber a demanda de duas ou mais arestas requeridas.

O grafo original G não é uma boa representação para tratar o OCARP como um problema de fluxos: suponha um grafo G completo com $n = 4$ vértices ou mais. Esse grafo possui $n(n-1)/2 > n$ arestas. Se houver um número de arestas requeridas m superior ao número de vértices n então não será possível atribuir as demandas de todas as arestas requeridas a nós diferentes. Portanto, para realizar a transferência de demandas o modelo é executado sobre um grafo G' que é supergrafo do grafo original G. Além disso, ao trabalhar com o grafo expandido G' as restrições do modelo se tornam mais simples.

Além de G' conter todos os nós e arestas de G, para cada aresta requerida $(v_i, v_j) \in E_R$ são incluídos em G': (i) dois nós artificiais v'_i e v'_j, (ii) uma aresta artificial requerida (v'_i, v'_j) com custo igual ao da aresta $(v_i, v_j) \in E_R$, (iii) duas arestas artificiais não-requeridas que ligam v_i a v'_i, e v_j a v'_j com custo zero. Por fim, é criado um nó especial v_0 adjacente a todos os nós artificiais com custo zero. Como mostra a Figura 5.1, as arestas requeridas de G não são requeridas em G'.

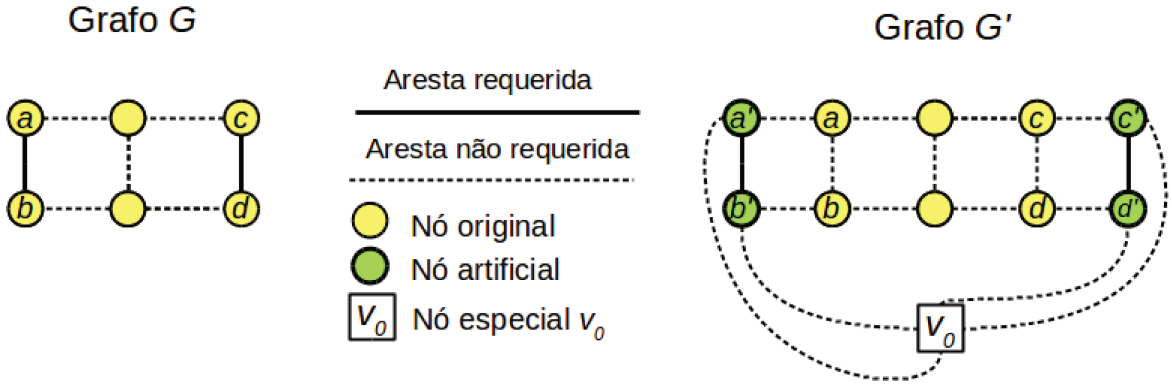


Figura 5.1: Grafos G e G' correspondentes.

É fácil perceber que uma solução OCARP no grafo G' pode ser transformada em uma solução no grafo G, bastando substituir nas rotas dessa solução, as ocorrências de arestas artificiais por suas correspondentes no grafo original G.

Seja V' o conjunto de nós artificiais, seja E' o conjunto de arestas artificiais e seja $E'_R \subset E'$ o conjunto de arestas artificiais requeridas. Para a execução do modelo, para cada aresta artificial requerida $(v'_i, v'_j) \in E'_R$ é escolhido arbitrariamente um nó, digamos v'_i, para ter demanda d'_i igual à demanda da aresta requerida $(v_i, v_j) \in E_R$. A demanda d'_j do outro nó v'_j é zero. Além disso, são criados M fluxos, um para cada rota $k \in [1, M]$ tal que o nó v_0 é a origem dos fluxos. Desse modo, existirão M fluxos percorrendo o grafo G' para atender às demandas dos nós. Cada fluxo pode atender até D unidades de demanda (correspondente à capacidade de um veículo).

O modelo trata-se de uma relaxação do problema OCARP aplicado ao grafo $G'(V \cup V' \cup \{v_0\}, E \cup E')$. O modelo define dois conjuntos de variáveis: (i) x_{ij} representa a quantidade de vezes que o arco (i, j) é visitado considerando-se todas as rotas. (ii) y^k_{ij} representa o fluxo no arco (i, j) pela rota k. Para a execução do modelo, considera-se o grafo G' como um grafo direcionado contendo dois arcos em sentidos opostos para cada aresta não-direcionada.

A função objetivo (5.1) minimiza o custo total, dado pela soma dos custos das arestas. As restrições (5.2) e (5.3) forçam que os vértices artificiais sejam visitados e permite que as rotas possam ser encerradas nesses vértices. A restrição (5.4) assegura que as arestas artificiais requeridas sejam visitadas. A restrição (5.5) representa a continuidade das rotas. A restrição (5.6) fixa o número de rotas (fluxos) saindo do nó de origem (note que $v_0 \notin V'$). A restrição (5.7) limita o valor dos fluxos. As restrições (5.8), (5.9) e (5.10) asseguram o balanceamento dos fluxos em cada nó e o atendimento da demanda dos nós artificiais com demanda. A restrição (5.11) limita inferiormente e superiormente o fluxo total passante em cada arco.

$(Formulacao\ por\ Fluxos)$

$$MIN \quad \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{5.1}$$

s.a.

$$\sum_{j\in N(i)} x_{ji} = 1 \qquad (i \in V') \tag{5.2}$$

$$\sum_{j\in N(i)} x_{ij} \leq 1 \qquad (i \in V') \tag{5.3}$$

$$x_{ij} + x_{ji} = 1 \qquad ((i,j) \in E'_R) \tag{5.4}$$

$$\sum_{j\in N(i)} (x_{ji} - x_{ij}) = 0 \qquad (i \in V) \tag{5.5}$$

$$\sum_{j\in N(0)} x_{0j} = M \tag{5.6}$$

$$\sum_{j\in N(0)} y^k_{0j} = D \qquad (k \in \{1,...,M\})) \tag{5.7}$$

$$\sum_{k=1}^{M} \sum_{j\in N(i)} (y^k_{ji} - y^k_{ij}) = d_i \qquad (i \in V') \tag{5.8}$$

$$\sum_{j\in N(i)} (y^k_{ji} - y^k_{ij}) = 0 \qquad (i \in V, k \in \{1,...,M\})) \tag{5.9}$$

$$\sum_{j\in N(i)} (y^k_{ji} - y^k_{ij}) \geq 0 \qquad (i \in V', k \in \{1,...,M\})) \tag{5.10}$$

$$x_{ij} \leq \sum_{k=1}^{M} y^k_{ij} \leq (D - d_i)x_{ij} \qquad ((i,j) \in E \cup E') \tag{5.11}$$

$$x_{ij} \in \mathbb{Z}_{\geq 0}, \qquad ((i,j) \in E \cup E') \tag{5.12}$$

$$y^k_{ij} \in \mathbb{R}_{\geq 0}, \qquad ((i,j) \in E \cup E', k \in \{1,...,M\}) \tag{5.13}$$

A formulação por fluxos consiste em uma formulação relaxada uma vez que nem sempre as soluções ótimas desse modelo são soluções viáveis para o OCARP. Isso decorre devido à restrição (5.8) não diferenciar o caso onde a demanda de um nó é atendida completamente por um dos fluxos ou atendida parcialmente por dois ou mais fluxos, o que caracteriza um cenário onde a demanda de uma aresta requerida é atendida parcialmente por dois ou mais veículos, o que não é permitido no OCARP. De modo semelhante a restrição (5.11) também não distingue a situação onde dois ou mais fluxos, que somam um valor inferior a D, estão trasitando por um arco (i,j) de outra situação onde apenas um fluxo com valor inferior a D transita pelo arco, desse modo a variável x_{ij} pode assumir um valor inferior ao que seria necessário para uma solução viável.

5.1 Experimentos Computacionais

Esta seção apresenta os resultados de um experimento comparativo entre formulações para o OCARP quanto aos limitantes duais gerados. São comparadas a Formulação Original [53] e a Formulação por Fluxos.

A máquina utilizada no experimento compõe-se de processador Intel Xeon X3430 2.4 GHz, 8 GB de memória RAM e sistema operacional Linux 64-bit. Para a solução da Formulação por Fluxos foi utilizado o *solver* Gurobi versão 6.5 [1] com tempo de execução limitado em uma hora para cada instância. Os resultados para a Formulação Original foram obtidos por Usberti et al. [53] com tempo de execução limitado em uma hora por instância.

Os resultados são apresentados nas Tabelas 5.1, 5.2 e 5.3 para os respectivos conjuntos de instâncias: *ogdb* (7-27 nós, 11-55 arestas), *oval* (24-50 nós, 34-97 arestas) e *oegl* (77-140 nós, 98-190 arestas) [34, 12, 41]. Para cada instância, foram considerados três valores distintos para o parâmetro M (número de veículos): M^*, $M^* + 1$ e $M^* + 2$, onde M^* é o número mínimo de veículos necessários para se obter uma solução factível (para se obter esses valores foi necessário resolver o *Bin-Packing Problem* [42]).

O valor de LD_1 corresponde ao limitante dual obtido pela Formulação Original e LD_2 ao limitantes dual obtido pela Formulação por Fluxos. Dados os valores de LD_1 e LD_2 é calculado um valor $\Delta LD(\%)$ que corresponde à taxa de melhoria obtida por LD_2 em comparação a LD_1.

A Tabela 5.1 permite observar que para as instâncias *ogdb* a taxa de melhoria foi pequena, principalmente porque são instâncias pequenas bem resolvidas em ambas formulações. A Tabela 5.2 mostra que para as instâncias *oval* houve uma taxa de melhoria mais expressiva, aumentando o limitante dual de algumas instâncias até 5%. Além disso, é possível constatar que o modelo teve bom desempenho nessas instâncias em termos de tempo de processamento, tendo obtido uma média de tempo abaixo de dez segundos para as instâncias *ogdb* e abaixo de trinta segundos para as instâncias *oval*.

A Tabela 5.3 permite verificar que para as instâncias *oegl* houve uma taxa de melhoria mais expressiva, alcançando até 18% de melhoria em uma das instâncias. Conforme M se aproxima de M^*, o tempo de processamento é maior, demonstrando assim a influência do parâmetro M na dificuldade de resolução do problema. Não obstante, os limitantes duais da Formulação por Fluxos abriram uma vantagem substancial sobre os limitantes duais obtidos pela Formulação Original.

Também é possível perceber que para muitas instâncias o limite de tempo de processamento foi atingido. Especificamente para a instância *egl-s4-C* quase não houve progresso. Mais especificamente, para $M = M^* + 1$ e $M = M^* + 2$ o limitante dual encontrado foi o valor do limitante dual trivial (soma dos custos de todas arestas requeridas), tanto para LD_1 quanto LD_2. Disso se depreende que para as instâncias mais difíceis o modelo pode acabar por progredir lentamente na poda da árvore de enumeração do algoritmo de *branch-and-bound* e assim não conseguir melhorar os limitantes duais em um tempo de processamento razoável.

[1] Gurobi é um resolvedor comercial de programação linear inteira (www.gurobi.com).

Tabela 5.1: Resultados da formulação por fluxos para as instâncias *ogdb*.

		*M**				*M*+1*				*M*+2*			
Instância	M^*	LD_1	LD_2	$\Delta LD(\%)$	CPU	LD_1	LD_2	$\Delta LD(\%)$	CPU	LD_1	LD_2	$\Delta LD(\%)$	CPU
gdb1	5	252	252	0.00	0	252	252	0.00	1	252	252	0.00	1
gdb2	6	291	291	0.00	2	291	291	0.00	3	291	291	0.00	5
gdb3	5	233	233	0.00	0	233	233	0.00	0	233	233	0.00	3
gdb4	4	238	238	0.00	0	238	238	0.00	1	238	238	0.00	2
gdb5	6	316	316	0.00	2	316	316	0.00	4	316	316	0.00	4
gdb6	5	260	260	0.00	1	260	260	0.00	0	260	260	0.00	3
gdb7	5	262	262	0.00	0	262	262	0.00	2	262	262	0.00	1
gdb8	10	210	210	0.00	9	210	210	0.00	22	210	210	0.00	25
gdb9	10	219	219	0.00	25	219	219	0.00	27	219	219	0.00	33
gdb10	4	252	252	0.00	1	252	252	0.00	2	252	252	0.00	3
gdb11	5	356	362	**1.69**	4	356	360	**1.12**	4	356	358	**0.56**	38
gdb12	7	336	336	0.00	4	336	336	0.00	4	336	336	0.00	5
gdb13	6	509	509	0.00	2	509	509	0.00	5	509	509	0.00	2
gdb14	5	96	96	0.00	1	96	96	0.00	2	96	96	0.00	2
gdb15	4	56	56	0.00	0	56	56	0.00	1	56	56	0.00	1
gdb16	5	119	119	0.00	1	119	119	0.00	1	119	119	0.00	1
gdb17	5	84	84	0.00	2	84	84	0.00	3	84	84	0.00	3
gdb18	5	158	158	0.00	4	158	158	0.00	4	158	158	0.00	5
gdb19	3	45	45	0.00	0	45	45	0.00	0	45	45	0.00	0
gdb20	4	105	105	0.00	0	105	105	0.00	1	105	105	0.00	2
gdb21	6	149	149	0.00	2	149	149	0.00	4	149	149	0.00	6
gdb22	8	191	191	0.00	9	191	191	0.00	5	191	191	0.00	5
gdb23	10	223	223	0.00	12	223	223	0.00	19	223	223	0.00	22
média				**0.07**	4			**0.05**	5			**0.02**	8

LD_1: limitante dual obtido da literatura [53].

LD_2: limitante dual obtido pela Formulação por Fluxos.

$\Delta LD(\%) = 100(LD_2 - LD_1)/LD_1$: taxa de melhoria de LD_1 para LD_2.

CPU: tempo em segundos utilizado pela Formulação por Fluxos.

Tabela 5.2: Resultados da formulação por fluxos para as instâncias *oval*.

		*M**				*M*+1*				*M*+2*			
Instância	M^*	LD_1	LD_2	$\Delta LD(\%)$	CPU	LD_1	LD_2	$\Delta LD(\%)$	CPU	LD_1	LD_2	$\Delta LD(\%)$	CPU
val1A	2	154	154	0.00	1	150	151	**0.67**	2	146	149	**2.05**	3
val1B	3	148	151	**2.03**	2	146	149	**2.05**	2	146	147	**0.68**	7
val1C	8	146	146	0.00	17	146	146	0.00	16	146	146	0.00	13
val2A	2	195	195	0.00	1	191	192	**0.52**	1	185	189	**2.16**	3
val2B	3	191	192	**0.52**	2	185	189	**2.16**	2	185	186	**0.54**	2
val2C	8	185	185	0.00	22	185	185	0.00	20	185	185	0.00	16
val3A	2	71	71	0.00	1	68	69	**1.47**	1	65	67	**3.08**	2
val3B	3	69	69	0.00	1	65	67	**3.08**	1	65	66	**1.54**	2
val3C	7	65	65	0.00	31	65	65	0.00	9	65	65	0.00	10
val4A	3	353	358	**1.42**	6	343	354	**3.21**	12	344	350	**1.74**	14
val4B	4	343	354	**3.21**	7	343	350	**2.04**	12	343	347	**1.17**	13
val4C	5	343	350	**2.04**	12	343	347	**1.17**	15	343	345	**0.58**	33
val4D	9	343	343	0.00	48	343	343	0.00	35	343	343	0.00	68
val5A	3	375	383	**2.13**	23	367	378	**3.00**	9	367	374	**1.91**	11
val5B	4	368	378	**2.72**	11	367	374	**1.91**	11	367	371	**1.09**	13
val5C	5	368	374	**1.63**	14	367	371	**1.09**	15	367	368	**0.27**	17
val5D	9	367	367	0.00	42	367	367	0.00	50	367	367	0.00	43
val6A	3	194	195	**0.52**	2	190	193	**1.58**	3	190	192	**1.05**	6
val6B	4	190	194	**2.11**	4	190	192	**1.05**	5	190	191	**0.53**	15
val6C	10	190	190	0.00	25	190	190	0.00	26	190	190	0.00	31
val7A	3	256	263	**2.73**	4	249	259	**4.02**	5	249	256	**2.81**	9
val7B	4	249	259	**4.02**	5	249	256	**2.81**	10	249	253	**1.61**	11
val7C	9	249	250	**0.40**	25	249	249	0.00	44	249	249	0.00	59
val8A	3	364	364	0.00	7	348	359	**3.16**	11	347	354	**2.02**	9
val8B	4	347	359	**3.46**	7	347	354	**2.02**	41	347	351	**1.15**	13
val8C	9	347	347	0.00	42	347	347	0.00	43	347	347	0.00	24
val9A	3	292	298	**2.05**	9	290	294	**1.38**	18	278	292	**5.04**	20
val9B	4	291	294	**1.03**	11	278	292	**5.04**	27	278	290	**4.32**	22
val9C	5	287	292	**1.74**	41	278	290	**4.32**	23	278	288	**3.60**	42
val9D	10	278	282	**1.44**	103	278	281	**1.08**	112	278	280	**0.72**	122
val10A	3	394	402	**2.03**	51	385	396	**2.86**	38	376	391	**3.99**	39
val10B	4	382	396	**3.66**	40	376	391	**3.99**	45	376	388	**3.19**	35
val10C	5	376	391	**3.99**	23	376	388	**3.19**	61	376	385	**2.39**	72
val10D	10	376	379	**0.80**	124	376	378	**0.53**	160	376	377	**0.27**	69
média				**1.34**	22			**1.74**	26			**1.45**	25

LD_1: limitante dual obtido da literatura [53].

LD_2: limitante dual obtido pela Formulação por Fluxos.

$\Delta LD(\%) = 100(LD_2 - LD_1)/LD_1$: taxa de melhoria de LD_1 para LD_2.

CPU: tempo em segundos utilizado pela Formulação por Fluxos.

Tabela 5.3: Resultados da formulação por fluxos para as instâncias *oegl*.

		M^*				M^*+1				M^*+2			
Instância	M^*	LD_1	LD_2	$\Delta LD(\%)$	CPU	LD_1	LD_2	$\Delta LD(\%)$	CPU	LD_1	LD_2	$\Delta LD(\%)$	CPU
egl-e1-A	5	1595	1775	**11.29**	42	1538	1701	**10.60**	43	1513	1651	**9.12**	51
egl-e1-B	7	1492	1676	**12.33**	56	1478	1626	**10.01**	54	1468	1582	**7.77**	75
egl-e1-C	10	1468	1584	**7.90**	172	1468	1553	**5.79**	157	1468	1527	**4.02**	152
egl-e2-A	7	1955	2151	**10.03**	259	1895	2072	**9.34**	114	1879	2028	**7.93**	112
egl-e2-B	10	1879	2014	**7.18**	228	1879	1982	**5.48**	129	1879	1959	**4.26**	188
egl-e2-C	14	1879	1996	**6.23**	2243	1879	1968	**4.74**	3600	1879	1943	**3.41**	3600
egl-e3-A	8	2198	2477	**12.69**	149	2188	2398	**9.60**	167	2188	2366	**8.14**	154
egl-e3-B	12	2188	2361	**7.91**	302	2188	2332	**6.58**	364	2188	2306	**5.39**	341
egl-e3-C	17	2188	2320	**6.03**	3600	2188	2278	**4.11**	1733	2188	2246	**2.65**	529
egl-e4-A	9	2453	2594	**5.75**	65	2453	2568	**4.69**	29	2453	2554	**4.12**	105
egl-e4-B	14	2453	2620	**6.81**	3600	2453	2519	**2.69**	3158	2453	2505	**2.12**	518
egl-e4-C	19	2453	2572	**4.85**	3600	2453	2492	**1.59**	3600	2453	2481	**1.14**	1867
egl-s1-A	7	1517	1718	**13.25**	3600	1394	1652	**18.51**	781	1394	1582	**13.49**	410
egl-s1-B	10	1394	1637	**17.43**	3600	1394	1551	**11.26**	3600	1394	1476	**5.88**	3600
egl-s1-C	14	1394	1622	**16.36**	3600	1394	1517	**8.82**	3600	1394	1427	**2.37**	3600
egl-s2-A	14	3174	3620	**14.05**	1286	3174	3576	**12.67**	1542	3174	3536	**11.41**	2026
egl-s2-B	20	3174	3502	**10.33**	3600	3174	3420	**7.75**	3600	3174	3381	**6.52**	2453
egl-s2-C	27	3174	3317	**4.51**	3600	3174	3300	**3.97**	3600	3174	3288	**3.59**	3600
egl-s3-A	15	3379	3769	**11.54**	3600	3379	3726	**10.27**	3600	3379	3687	**9.12**	3600
egl-s3-B	22	3379	3592	**6.30**	3600	3379	3567	**5.56**	3448	3379	3542	**4.82**	3600
egl-s3-C	29	3379	3488	**3.23**	3600	3379	3477	**2.90**	3600	3379	3467	**2.60**	3600
egl-s4-A	19	4186	4424	**5.69**	3600	4186	4404	**5.21**	3600	4186	4388	**4.83**	3063
egl-s4-B	27	4186	4321	**3.23**	3600	4186	4309	**2.94**	3600	4186	4299	**2.70**	3600
egl-s4-C	35	4186	4238	**1.24**	3600	4186	4186	0.00	3600	4186	4186	0.00	3600
média				**8.59**	2300			**6.88**	2138			**5.30**	1851

LD_1: limitante dual obtido da literatura [53].

LD_2: limitante dual obtido pela Formulação por Fluxos.

$\Delta LD(\%) = 100(LD_2 - LD_1)/LD_1$: taxa de melhoria de LD_1 para LD_2.

CPU: tempo em segundos utilizado pela Formulação por Fluxos.

5.2 Considerações Finais

A Formulação por Fluxos trata-se de um modelo relaxado para o OCARP, visando principalmente a obtenção de limitantes duais de qualidade. O modelo obteve bons limitantes duais nos experimentos computacionais, demonstrando assim o seu potencial. A Formulação por Fluxos foi inspirada na metodologia de solução que Belenguer e Benavent desenvolveram para o CARP [11], a qual consiste de uma formulação de $O(|E|)$ variáveis inteiras e um algoritmo de plano de cortes para esta formulação.

A vantagem da Formulação por Fluxos, em comparação à Formulação Original, está em seu número reduzido de $O(|E|)$ variáveis inteiras, enquanto a Formulação Original trabalha com um número exponencial de variáveis inteiras e restrições. Assim, a Formulação por Fluxos é um modelo mais simples e com potencial de ser resolvido mais rapidamente pelo *solver* de programação linear inteira. Os resultados comprovam que houve uma melhoria nos limitantes duais em relação à Formulação Original.

A principal desvantagem da Formulação por Fluxos consiste no fato de nem sempre produzir soluções OCARP viáveis, uma vez que trata-se de uma formulação relaxada. No entanto, este ponto pode ser amenizado se existirem métodos heurísticos para obtenção de bons limitantes primais, como por exemplo o algoritmo genético apresentado neste trabalho.

Capítulo 6

Conclusão

O OCARP é um problema *NP-difícil* de otimização combinatória que foi apresentado em 2011 por Usberti et al. [53]. Esse problema se inclui na família de problemas de roteamento em arcos e possui interesse teórico e prático.

O OCARP possui muitas semelhanças com o CARP, entretanto no OCARP as rotas não precisam ser fechadas e não se considera nenhum nó depósito. Além disso, os mais recentes algoritmos e heurísticas para o CARP consideram o número de veículos como uma variável a ser otimizada, enquanto que no OCARP o número de veículos é um parâmetro fixo, assim essas metodologias não são adequadas para se resolver o OCARP.

Uma metaheurística de algoritmos genéticos foi desenvolvida para o OCARP. Essa heurística foi projetada para tratar em especial a dificuldade das instâncias onde o conjunto de demandas e o baixo número de veículos disponíveis torna difícil a geração de uma solução factível, uma vez que há um sub-problema *NP-difícil* de empacotamento que precisa ser resolvido. A heurística obteve bom desempenho superando as metodologias da literatura.

Uma formulação relaxada de programação linear inteira mista foi desenvolvida para o OCARP. Essa formulação foi projetada visando a obtenção de bons limitantes duais. Em particular, essa formulação tem apenas $O(|E|)$ variáveis inteiras, tratando-se de um modelo razoavelmente leve de ser resolvido pelo *solver* de PLI. A formulação desenvolvida apresentou bom desempenho, tendo obtido limitantes duais que se igualaram aos custos ótimos para diversas instâncias.

A Tabela 6.1 condensa os resultados obtidos nos Capítulos 4 e 5 e compara-os com os métodos propostos por Usberti et al. em [53]. Seja GAP_1 o desvio de otimalidade onde o limitante primal é dado pela heurística RPS_ER e o limitante dual é dado pela Formulação Original [53]. Seja GAP_2 o desvio de otimalidade onde o limitante primal é dado pelo algoritmo genético e o limitante dual é dado pela Formulação por Fluxos. Cada desvio i é calculado como $GAP_i(\%) = 100(LP_i - LD_i)/LD_i$, onde LP e LD são os limitantes primais e duais respectivamente. A partir desses valores é definido $\Delta GAP(\%) = 100(GAP_2 - GAP_1)/GAP_1$ como a taxa de redução média dos desvios de otimalidade.

É possível observar pelos resultados da Tabela 6.1 que os métodos propostos reduziram efetivamente os desvios de otimalidade médios de todos os conjuntos de instâncias e parâmetros considerados no experimento. Todas as instâncias *ogdb* foram resolvidas até a otimalidade. Com exceção de duas instâncias o conjunto *oval* também foi resolvido até a otimalidade. As instâncias *oegl* apresentaram uma melhoria muito expressiva nos desvios de otimalidade, não obstante ainda haver espaço para melhorias, principalmente quando se considera o parâmetro $M = M^*$ que possui uma média de desvio de otimalidade quase

Tabela 6.1: Resultados resumidos do experimento comparativo de metodologias.

Instâncias	M	$GAP_1(\%)$	$GAP_2(\%)$	$\Delta GAP(\%)$
ogdb	M^*	0.09	0.00	**-100.00**
	M^*+1	0.05	0.00	**-100.00**
	M^*+2	0.05	0.00	**-100.00**
	todas	0.06	0.00	**-100.00**
oval	M^*	2.07	0.09	**-95.87**
	M^*+1	2.79	0.00	**-100.00**
	M^*+2	3.30	0.00	**-100.00**
	todas	2.72	0.03	**-98.62**
oegl	M^*	33.84	4.93	**-85.44**
	M^*+1	22.74	1.73	**-92.39**
	M^*+2	20.00	1.22	**-93.88**
	todas	25.53	2.63	**-90.57**
todas	M^*	10.92	1.50	**-86.31**
	M^*+1	7.92	0.51	**-93.53**
	M^*+2	7.32	0.36	**-95.04**
	todas	8.72	0.79	**-91.63**

$GAP_1(\%)$: desvio de otimalidade da literatura [53].
$GAP_2(\%)$: desvio de otimalidade dos métodos propostos.
$\Delta GAP(\%)$: taxa de redução no desvio de otimalidade.

três vezes maior em relação ao parâmetro $M = M^* + 1$.

Conclui-se que os métodos propostos nesta dissertação superam os métodos da literatura e contribuem para o avanço do estado da arte de métodos para tratar o OCARP. Um artigo contendo os resultados dos métodos propostos foi submetido para o XLVIII Simpósio Brasileiro de Pesquisa Operacional (atualmente em estágio de revisão). O avanço na pesquisa científica de problemas de otimização combinatória pode contribuir para novas técnicas na indústria, em especial para uma melhor alocação estratégica de recursos escassos. Uma das aplicações práticas do OCARP é o problema de roteamento de leituristas. Portanto, as metodologias propostas neste trabalho tem o potencial de otimizar os recursos investidos pelas empresas de distribuição de água, energia elétrica e gás atualmente em operação no país.

6.1 Trabalhos Futuros

Os métodos apresentados neste trabalho obtiveram bom desempenho em média, porém as instâncias mais difíceis do conjunto *oegl* ainda apresentaram desvios de otimalidade significativos durante os experimentos. Além disso, a formulação relaxada que foi apresentada não garante encontrar uma solução viável para o OCARP, portanto ainda pode ser melhorada, sendo necessário para isso um estudo mais detalhado da formulação.

Um novo campo de investigação que pode ser considerado é a concepção de um algoritmo *branch-and-cut* para a solução exata do OCARP que utilize como base a formulação por fluxos apresentada. Também é possível considerar a integração da heurística ao algoritmo *branch-and-cut* no intuito desse último obter bons limitantes primais durante sua execução a fim de acelerar a exploração da árvore de enumeração. Por fim, a criação de novas instâncias, maiores e mais variadas quanto aos parâmetros de entrada (demandas, número de veículos, esparsidade do grafo, capacidade dos veículos, etc) trata-se de outro trabalho interessante uma vez que os conjuntos de instâncias *ogdb* e *oval* foram praticamente resolvidos até a otimalidade e não deverão servir como instâncias de muito interesse para o futuro.

Pretendemos dar continuidade à pesquisa apresentada neste trabalho, com o apoio de uma bolsa FAPESP, em um curso de doutorado envolvendo principalmente o estudo de métodos exatos para problemas de roteamento em arcos.

Referências Bibliográficas

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

[2] Anita Amberg and Stefan Voss. A hierarchical relaxations lower bound for the capacitated arc routing problem. In *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, pages 1415–1424. IEEE, 2002.

[3] Barrie M Baker and MA Ayechew. A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, 30(5):787–800, 2003.

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

[5] Jørgen Bang-Jensen, Gregory Gutin, and Anders Yeo. When the greedy algorithm fails. *Discrete Optimization*, 1(2):121–127, 2004.

[6] John E Beasley. Route first—cluster second methods for vehicle routing. *Omega*, 11(4):403–408, 1983.

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

[8] J.M. Belenguer and E. Benavent. *A branch and cut algorithm for the Capacitated Arc Routing Problem*. Workshop on Algorithmic Approaches to Large and Complex Combinatorial Optimization Problems, Giens, France, 1994.

[9] José M Belenguer and Enrique Benavent. The capacitated arc routing problem: Valid inequalities and facets. *Computational Optimization and Applications*, 10(2):165–187, 1998.

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

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

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

[13] 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.

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

[15] Peter Breslin. The capacitated arc routing problem: Lower bounds. Master's thesis, University College Dublin, 1997.

[16] Sangit Chatterjee, Cecilia Carrera, and Lucy A Lynch. Genetic algorithms and traveling salesman problems. *European journal of operational research*, 93(3):490–510, 1996.

[17] Angel Corberán, Adam N Letchford, and José María Sanchis. A cutting plane algorithm for the general routing problem. *Mathematical Programming*, 90(2):291–316, 2001.

[18] Angel Corberán, Marcus Oswald, Isaac Plana, Gerhard Reinelt, and José M Sanchis. New results on the windy postman problem. *Mathematical programming*, 132(1-2):309–332, 2012.

[19] Angel Corberán and Christian Prins. Recent results on arc routing problems: An annotated bibliography. *Networks*, 56:50–69, 2009.

[20] Everton Fernandes da Silva. Modelos matemáticos para um problema de caminho de corte. Master's thesis, Universidade de São Paulo, 2016.

[21] P Fernández de Córdoba, LM Garcıa Raffi, and JM Sanchis. A heuristic algorithm based on monte carlo methods for the rural postman problem. *Computers & operations research*, 25(12):1097–1106, 1998.

[22] 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.

[23] Moshe Dror. *Arc routing: theory, solutions and applications*. Springer Science & Business Media, 2012.

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

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

[26] Horst A Eiselt, Michel Gendreau, and Gilbert Laporte. Arc routing problems, part ii: The rural postman problem. *Operations Research*, 43(3):399–414, 1995.

[27] Leonhard Euler. Leonhard euler and the königsberg bridges. *Scientific American*, 189(1):66–70, 1953.

[28] Elena Fernández, Oscar Meza, Robert Garfinkel, and Maruja Ortega. On the undirected rural postman problem: Tight bounds based on a new formulation. *Operations Research*, 51(2):281–291, 2003.

[29] Greg N Frederickson. Approximation algorithms for some postman problems. *Journal of the ACM (JACM)*, 26(3):538–554, 1979.

[30] R. Michael Garey and S. David Johnson. Computers and intractability: a guide to the theory of np-completeness. 1979. *San Francisco, LA: Freeman*, 1978.

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

[32] David Goldberg. Genetic algorithms in optimization, search and machine learning. *Addison Wesley*, 905:205–211, 1988.

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

[34] 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.

[35] GW Groves and JH van Vuuren. Efficient heuristics for the rural postman problem. *ORiON*, 21(1), 2005.

[36] 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.

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

[38] Henry John Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1991.

[39] Mei-Ko Kwan. Graphic programming using odd or even points. *Chinese Math*, 1(273-277):110, 1962.

[40] 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.

[41] 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, 1995.

[42] Silvano Martello and Paolo Toth. *Knapsack problems*. Wiley New York, 1990.

[43] Luís M Moreira, José F Oliveira, A Miguel Gomes, and J Soeiro Ferreira. Heuristics for a dynamic rural postman problem. *Computers & Operations Research*, 34(11):3281–3294, 2007.

[44] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience, New York, NY, USA, 1988.

[45] Christos H Papadimitriou. On the complexity of edge traversing. *Journal of the ACM (JACM)*, 23(3):544–554, 1976.

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

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

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

[49] F. L. Usberti, P. M. França, and A. L. M. França. *Roteamento de Leituristas: Um Problema NP-Difícil (in portuguese). In: XL SBPO Brazilian Symposium of Operational Research*. Annals XL SBPO, João Pessoa, 2008.

[50] Fábio Luiz Usberti. *Métodos heurísticos e exatos para o problema de roteamento em arcos capacitado e aberto*. PhD thesis, Universidade Estadual de Campinas, 2012.

[51] 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.

[52] 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*, 2011. doi: 10.1016/j.cor.2011.10.014.

[53] 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.

[54] Sanne Wøhlk. New lower bound for the capacitated arc routing problem. *Computers & Operations Research*, 33(12):3458–3472, 2006.

[55] Sanne Wøhlk. A decade of capacitated arc routing. In *The vehicle routing problem: latest advances and new challenges*, pages 29–48. Springer, 2008.

[56] Laurence A Wolsey. *Integer programming*, volume 42. Wiley New York, 1998.

Apêndice A

Resultados dos experimentos

Este apêndice apresenta os resultados completos dos experimentos com os métodos propostos nesta tese: o algoritmo genético e a formulação por fluxos, descritos nos Capítulos 4 e 5 respectivamente. Em comparação aos métodos da literatura propostos por Usberti et al. [53].

Sejam LD_1 e LD_2 os limitantes duais obtidos pela Formulação Original [53] e a Formulação por Fluxos, respectivamente. Seja LP_1 o limitante primal obtido pela heurística RPS_ER de Usberti et al. [53] e seja LP_2 o limitante primal obtido pelo Algoritmo Genético. Através desses limitantes, são definidos:

- O desvio de otimalidade(%) formado pelas metodologias da literatura [53]: $GAP_1 = 100 * (LP_1 - LD_1)/LD_1$.
- O desvio de otimalidade(%) formado pelos métodos propostos (Capítulos 4 e 5): $GAP_2 = 100 * (LP_2 - LD_2)/LD_2$.

As Tabelas A.1, A.2 e A.3 apresentam os resultados para três conjuntos de instâncias: ogdb (7-27 nós, 11-55 arestas), oval (24-50 nós, 34-97 arestas) e oegl (77-140 nós, 98-190 arestas) [53, 34, 12, 41]. Para cada instância, foram considerados três valores distintos para o parâmetro M (número de veículos): M^*, $M^* + 1$ e $M^* + 2$, onde M^* é o número mínimo de veículos necessários para se obter uma solução factível (para se obter esses valores foi necessário resolver o *Bin-Packing Problem* [42]).

Os resultados para a Formulação Original foram obtidos por Usberti et al. [53] com tempo de execução limitado em uma hora. Os resultados do algoritmo genético, da heurística RPS_ER e da formulação por fluxos foram obtidos conforme descrito nas Seções 4.2 e 5.1.

Tabela A.1: Resultado dos experimentos computacionais para o conjunto de instâncias *ogdb*.

		M^*						M^*+1						M^*+2					
Instância	M^*	LD_1	LD_2	LP_1	LP_2	GAP_1	GAP_2	LD_1	LD_2	LP_1	LP_2	GAP_1	GAP_2	LD_1	LD_2	LP_1	LP_2	GAP_1	GAP_2
gdb1	5	252	252	252	252	0.00	0.00	252	252	252	252	0.00	0.00	252	252	252	252	0.00	0.00
gdb2	6	291	291	291	291	0.00	0.00	291	291	291	291	0.00	0.00	291	291	291	291	0.00	0.00
gdb3	5	233	233	233	233	0.00	0.00	233	233	233	233	0.00	0.00	233	233	233	233	0.00	0.00
gdb4	4	238	238	238	238	0.00	0.00	238	238	238	238	0.00	0.00	238	238	238	238	0.00	0.00
gdb5	6	316	316	316	316	0.00	0.00	316	316	316	316	0.00	0.00	316	316	316	316	0.00	0.00
gdb6	5	260	260	260	260	0.00	0.00	260	260	260	260	0.00	0.00	260	260	260	260	0.00	0.00
gdb7	5	262	262	262	262	0.00	0.00	262	262	262	262	0.00	0.00	262	262	262	262	0.00	0.00
gdb8	10	210	210	210	210	0.00	0.00	210	210	210	210	0.00	0.00	210	210	210	210	0.00	0.00
gdb9	10	219	219	220	**219**	0.46	**0.00**	219	219	219	219	0.00	0.00	219	219	219	219	0.00	0.00
gdb10	4	252	252	252	252	0.00	0.00	252	252	252	252	0.00	0.00	252	252	252	252	0.00	0.00
gdb11	5	356	**362**	362	362	1.69	**0.00**	356	**360**	360	360	1.12	**0.00**	356	**358**	360	**358**	1.12	**0.00**
gdb12	7	336	336	336	336	0.00	0.00	336	336	336	336	0.00	0.00	336	336	336	336	0.00	0.00
gdb13	6	509	509	509	509	0.00	0.00	509	509	509	509	0.00	0.00	509	509	509	509	0.00	0.00
gdb14	5	96	96	96	96	0.00	0.00	96	96	96	96	0.00	0.00	96	96	96	96	0.00	0.00
gdb15	4	56	56	56	56	0.00	0.00	56	56	56	56	0.00	0.00	56	56	56	56	0.00	0.00
gdb16	5	119	119	119	119	0.00	0.00	119	119	119	119	0.00	0.00	119	119	119	119	0.00	0.00
gdb17	5	84	84	84	84	0.00	0.00	84	84	84	84	0.00	0.00	84	84	84	84	0.00	0.00
gdb18	5	158	158	158	158	0.00	0.00	158	158	158	158	0.00	0.00	158	158	158	158	0.00	0.00
gdb19	3	45	45	45	45	0.00	0.00	45	45	45	45	0.00	0.00	45	45	45	45	0.00	0.00
gdb20	4	105	105	105	105	0.00	0.00	105	105	105	105	0.00	0.00	105	105	105	105	0.00	0.00
gdb21	6	149	149	149	149	0.00	0.00	149	149	149	149	0.00	0.00	149	149	149	149	0.00	0.00
gdb22	8	191	191	191	191	0.00	0.00	191	191	191	191	0.00	0.00	191	191	191	191	0.00	0.00
gdb23	10	223	223	223	223	0.00	0.00	223	223	223	223	0.00	0.00	223	223	223	223	0.00	0.00
média						0.09	**0.00**					0.05	**0.00**					0.05	**0.00**

Tabela A.2: Resultado dos experimentos computacionais para o conjunto de instâncias *oval*.

		M^*						M^*+1						M^*+2					
Instância	M^*	LD_1	LD_2	LP_1	LP_2	GAP_1	GAP_2	LD_1	LD_2	LP_1	LP_2	GAP_1	GAP_2	LD_1	LD_2	LP_1	LP_2	GAP_1	GAP_2
val1A	2	154	154	154	154	0.00	0.00	150	**151**	154	**151**	2.67	**0.00**	146	**149**	154	**149**	5.48	**0.00**
val1B	3	148	**151**	151	151	2.03	**0.00**	146	**149**	149	149	2.05	**0.00**	146	**147**	149	**147**	2.05	**0.00**
val1C	8	146	146	150	**148**	2.74	**1.37**	146	146	146	146	0.00	0.00	146	146	146	146	0.00	0.00
val2A	2	195	195	195	195	0.00	0.00	191	**192**	195	**192**	2.09	**0.00**	185	**189**	195	**189**	5.41	**0.00**
val2B	3	191	**192**	192	192	0.52	**0.00**	185	**189**	192	**189**	3.78	**0.00**	185	**186**	192	**186**	3.78	**0.00**
val2C	8	185	185	185	185	0.00	0.00	185	185	185	185	0.00	0.00	185	185	185	185	0.00	0.00
val3A	2	71	71	71	71	0.00	0.00	68	**69**	71	**69**	4.41	**0.00**	65	**67**	71	**67**	9.23	**0.00**
val3B	3	69	69	69	69	0.00	0.00	65	**67**	67	67	3.08	**0.00**	65	**66**	67	**66**	3.08	**0.00**
val3C	7	65	65	66	66	1.54	1.54	65	65	65	65	0.00	0.00	65	65	65	65	0.00	0.00
val4A	3	353	**358**	358	358	1.42	**0.00**	343	**354**	358	**354**	4.37	**0.00**	344	**350**	358	**350**	4.07	**0.00**
val4B	4	343	**354**	354	354	3.21	**0.00**	343	**350**	354	**350**	3.21	**0.00**	343	**347**	354	**347**	3.21	**0.00**
val4C	5	343	**350**	353	**350**	2.92	**0.00**	343	**347**	350	**347**	2.04	**0.00**	343	**345**	350	**345**	2.04	**0.00**
val4D	9	343	343	351	**343**	2.33	**0.00**	343	343	346	**343**	0.87	**0.00**	343	343	346	**343**	0.87	**0.00**
val5A	3	375	**383**	383	383	2.13	**0.00**	367	**378**	381	**378**	3.81	**0.00**	367	**374**	381	**374**	3.81	**0.00**
val5B	4	368	**378**	378	378	2.72	**0.00**	367	**374**	376	**374**	2.45	**0.00**	367	**371**	376	**371**	2.45	**0.00**
val5C	5	368	**374**	375	**374**	1.90	**0.00**	367	**371**	372	**371**	1.36	**0.00**	367	**368**	372	**368**	1.36	**0.00**
val5D	9	367	367	370	**367**	0.82	**0.00**	367	367	370	**367**	0.82	**0.00**	367	367	370	**367**	0.82	**0.00**
val6A	3	194	**195**	195	195	0.52	**0.00**	190	**193**	195	**193**	2.63	**0.00**	190	**192**	195	**192**	2.63	**0.00**
val6B	4	190	**194**	194	194	2.11	**0.00**	190	**192**	192	192	1.05	**0.00**	190	**191**	192	**191**	1.05	**0.00**
val6C	10	190	190	190	190	0.00	0.00	190	190	190	190	0.00	0.00	190	190	190	190	0.00	0.00
val7A	3	256	**263**	263	263	2.73	**0.00**	249	**259**	263	**259**	5.62	**0.00**	249	**256**	263	**256**	5.62	**0.00**
val7B	4	249	**259**	259	259	4.02	**0.00**	249	**256**	259	**256**	4.02	**0.00**	249	**253**	259	**253**	4.02	**0.00**
val7C	9	249	**250**	256	**250**	2.81	**0.00**	249	249	252	**249**	1.20	**0.00**	249	249	250	**249**	0.40	**0.00**
val8A	3	364	364	364	364	0.00	0.00	348	**359**	359	**359**	3.16	**0.00**	347	**354**	359	**354**	3.46	**0.00**
val8B	4	347	**359**	359	359	3.46	**0.00**	347	**354**	354	354	2.02	**0.00**	347	**351**	354	**351**	2.02	**0.00**
val8C	9	347	347	352	**347**	1.44	**0.00**	347	347	347	347	0.00	0.00	347	347	348	**347**	0.29	**0.00**
val9A	3	292	**298**	299	**298**	2.40	**0.00**	290	**294**	299	**294**	3.10	**0.00**	278	**292**	299	**292**	7.55	**0.00**
val9B	4	291	**294**	298	**294**	2.41	**0.00**	278	**292**	298	**292**	7.19	**0.00**	278	**290**	298	**290**	7.19	**0.00**
val9C	5	287	**292**	296	**292**	3.14	**0.00**	278	**290**	294	**290**	5.76	**0.00**	278	**288**	294	**288**	5.76	**0.00**
val9D	10	278	**282**	292	**282**	5.04	**0.00**	278	**281**	288	**281**	3.60	**0.00**	278	**280**	288	**280**	3.60	**0.00**
val10A	3	394	**402**	403	**402**	2.28	**0.00**	385	**396**	403	**396**	4.68	**0.00**	376	**391**	403	**391**	7.18	**0.00**
val10B	4	382	**396**	399	**396**	4.45	**0.00**	376	**391**	399	**391**	6.12	**0.00**	376	**388**	399	**388**	6.12	**0.00**
val10C	5	376	**391**	397	**391**	5.59	**0.00**	376	**388**	394	**388**	4.79	**0.00**	376	**385**	394	**385**	4.79	**0.00**
val10D	10	376	**379**	391	**379**	3.99	**0.00**	376	**378**	387	**378**	2.93	**0.00**	376	**377**	387	**377**	2.93	**0.00**
média						2.07	**0.09**					2.79	**0.00**					3.30	**0.00**

Tabela A.3: Resultado dos experimentos computacionais para o conjunto de instâncias *oegl*.

		M^*						M^*+1						M^*+2					
Instância	M^*	LD_1	LD_2	LP_1	LP_2	GAP_1	GAP_2	LD_1	LD_2	LP_1	LP_2	GAP_1	GAP_2	LD_1	LD_2	LP_1	LP_2	GAP_1	GAP_2
egl-e1-A	5	1595	**1775**	1802	**1775**	12.98	**0.00**	1538	**1701**	1755	**1708**	14.11	**0.41**	1513	**1651**	1755	**1659**	15.99	**0.48**
egl-e1-B	7	1492	**1676**	1823	**1749**	22.18	**4.36**	1478	**1626**	1727	**1639**	16.85	**0.80**	1468	**1582**	1726	**1589**	17.57	**0.44**
egl-e1-C	10	1468	**1584**	1723	**1652**	17.37	**4.29**	1468	**1553**	1651	**1576**	12.47	**1.48**	1468	**1527**	1610	**1542**	9.67	**0.98**
egl-e2-A	7	1955	**2151**	2302	**2173**	17.75	**1.02**	1895	**2072**	2256	**2072**	19.05	**0.00**	1879	**2028**	2256	**2043**	20.06	**0.74**
egl-e2-B	10	1879	**2014**	2249	**2079**	19.69	**3.23**	1879	**1982**	2175	**1997**	15.75	**0.76**	1879	**1959**	2166	**1971**	15.27	**0.61**
egl-e2-C	14	1879	**1996**	2424	**2084**	29.00	**4.41**	1879	**1968**	2192	**1996**	16.66	**1.42**	1879	**1943**	2151	**1964**	14.48	**1.08**
egl-e3-A	8	2198	**2477**	2856	**2526**	29.94	**1.98**	2188	**2398**	2685	**2410**	22.71	**0.50**	2188	**2366**	2676	**2366**	22.30	**0.00**
egl-e3-B	12	2188	**2361**	2759	**2409**	26.10	**2.03**	2188	**2332**	2690	**2351**	22.94	**0.81**	2188	**2306**	2596	**2321**	18.65	**0.65**
egl-e3-C	17	2188	**2320**	2804	**2357**	28.15	**1.59**	2188	**2278**	2643	**2286**	20.80	**0.35**	2188	**2246**	2565	**2265**	17.23	**0.85**
egl-e4-A	9	2453	**2594**	3095	**2631**	26.17	**1.43**	2453	**2568**	2945	**2582**	20.06	**0.55**	2453	**2554**	2825	**2556**	15.17	**0.08**
egl-e4-B	14	2453	**2620**	3198	**2696**	30.37	**2.90**	2453	**2519**	2905	**2552**	18.43	**1.31**	2453	**2505**	2853	**2517**	16.31	**0.48**
egl-e4-C	19	2453	**2572**	3789	**2812**	54.46	**9.33**	2453	**2492**	2923	**2515**	19.16	**0.92**	2453	**2481**	2805	**2497**	14.35	**0.64**
egl-s1-A	7	1517	**1718**	1942	**1799**	28.02	**4.71**	1394	**1652**	1787	**1683**	28.19	**1.88**	1394	**1582**	1787	**1604**	28.19	**1.39**
egl-s1-B	10	1394	**1637**	1859	**1745**	33.36	**6.60**	1394	**1551**	1817	**1660**	30.34	**7.03**	1394	**1476**	1729	**1579**	24.03	**6.98**
egl-s1-C	14	1394	**1622**	2080	**1874**	49.21	**15.54**	1394	**1517**	1847	**1633**	32.50	**7.65**	1394	**1427**	1757	**1512**	26.04	**5.96**
egl-s2-A	14	3174	**3620**	4303	**3689**	35.57	**1.91**	3174	**3576**	4092	**3621**	28.92	**1.26**	3174	**3536**	4068	**3561**	28.17	**0.71**
egl-s2-B	20	3174	**3502**	4979	**3790**	56.87	**8.22**	3174	**3420**	4098	**3492**	29.11	**2.11**	3174	**3381**	4009	**3427**	26.31	**1.36**
egl-s2-C	27	3174	**3317**	4812	**3732**	51.61	**12.51**	3174	**3300**	4201	**3395**	32.36	**2.88**	3174	**3288**	3904	**3336**	23.00	**1.46**
egl-s3-A	15	3379	**3769**	4298	**3808**	27.20	**1.03**	3379	**3726**	4242	**3761**	25.54	**0.94**	3379	**3687**	4242	**3703**	25.54	**0.43**
egl-s3-B	22	3379	**3592**	4403	**3670**	30.30	**2.17**	3379	**3567**	4204	**3582**	24.42	**0.42**	3379	**3542**	4158	**3559**	23.05	**0.48**
egl-s3-C	29	3379	**3488**	4782	**3742**	41.52	**7.28**	3379	**3477**	4253	**3564**	25.87	**2.50**	3379	**3467**	4102	**3492**	21.40	**0.72**
egl-s4-A	19	4186	**4424**	5174	**4476**	23.60	**1.18**	4186	**4404**	5045	**4421**	20.52	**0.39**	4186	**4388**	4965	**4399**	18.61	**0.25**
egl-s4-B	27	4186	**4321**	5403	**4412**	29.07	**2.11**	4186	**4309**	4983	**4333**	19.04	**0.56**	4186	**4299**	4973	**4311**	18.80	**0.28**
egl-s4-C	35	4186	**4238**	8023	**5017**	91.66	**18.38**	4186	4186	5443	**4379**	30.03	**4.61**	4186	4186	5019	**4284**	19.90	**2.34**
média						33.84	**4.93**					22.74	**1.73**					20.00	**1.22**