Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação

FÁBIO LUIZ USBERTI

"Métodos Heurísticos e Exatos para o Problema de Roteamento em Arcos Capacitado e Aberto"

"Heuristic and Exact Approaches for the Open Capacitated Arc Routing Problem"

Tese de doutorado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e de Computação da Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas para obtenção do título de Doutor em Engenharia Elétrica, na área de concentração Automação.

Doctorate thesis presented to the Electrical and Computer Engineering Postgraduation Programm of the School of Electrical and Computer Engineering of the University of Campinas to obtain the Ph.D. grade in Electrical Engineering, with specialization in Automation.

Orientador: Prof. Dr. André Luiz Morelato França.
Co-Orientador: Prof. Dr. Paulo Morelato França.

ESTE EXEMPLAR CORRESPONDE À VERSÃO FINAL DA TESE.

CAMPINAS
2012

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

Us17m

Usberti, Fábio Luiz
Métodos heurísticos e exatos para o problema de roteamento em arcos capacitado e aberto / Fábio Luiz Usberti. --Campinas, SP: [s.n.], 2012.

Orientador: André Luiz Morelato França.
Co-Orientador: Paulo Morelato França.
Tese de Doutorado - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Roteamento (Administração de redes de computadores). 2. Método de varredura. 3. Heurística. 4. Programação heurística. 5. GRASP (Sistema operacional de computador). I. França, André Luiz Morelato. II. França, Paulo Morelato. III. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. IV. Título.

Título em Inglês: Heuristic and exact approaches for the open capacitated arc routing problem
Palavras-chave em Inglês: Routing administration (computer networks), Method of scanning, Heuristics, Heuristic programming, GRASP (computer operating system)
Área de concentração: Automação
Titulação: Doutor em Engenharia Elétrica
Banca examinadora: Marcos Jose Negreiros Gomes, Horácio Hideki Yanasse, Cid Carvalho de Souza, Fernando José Von Zuben
Data da defesa: 13-02-2012
Programa de Pós Graduação: Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE DOUTORADO

Candidato: Fábio Luiz Usberti

Data da Defesa: 13 de fevereiro de 2012

Título da Tese: "Métodos Heurísticos e Exatos para o Problema de Roteamento em Arcos Capacitado e Aberto"

Prof. Dr. André Luiz Morelato França (Presidente): _____

Prof. Dr. Horácio Hideki Yanasse: _____

Prof. Dr. Marcos Jose Negreiros Gomes: _____

Prof. Dr. Cid Carvalho de Souza: _____

Prof. Dr. Fernando José Von Zuben: _____

This work was supported by CNPq (Brazilian Research Agency)

Essa obra é dedicada aos meus pais,
Roberto e Maria Helena.

Perfection is achieved not when there
is nothing else to be included, but when
there is nothing left to be removed.
Antoine de Saint-Exupéry (1900 – 1944)

Agradecimentos

A Deus	*por um universo sujeito à otimização.*
Papai e mamãe	*minha eterna gratidão.*
Fran	*todo meu amor.*
André, Christiano, Celso e Paulo	*minha admiração.*
Meus irmãos, Rô e Bea	*total confiança.*
Daniel, Amanda, Betinho e Gú	*muita alegria.*
José	*grande amizade.*
Baca, Cris, Hugo, Laura, Lú e Spock	*muitos cafés com "menossuns".*
LABORE	*trabalho e bem-estar em harmonia.*
FEEC	*segundo lar.*
FEAGRI e IC	*boas lembranças.*
UNICAMP	*uma vida.*
CNPq	*um suporte.*

Abstract

The Open Capacitated Arc Routing Problem (OCARP) is an NP-hard combinatorial optimization problem where, given an undirected graph, the objective is to find a minimum cost set of tours that services a subset of edges with positive demand under capacity constraints. This problem is related to the Capacitated Arc Routing Problem (CARP) but differs from it since OCARP does not consider a depot, and tours are not constrained to form cycles. Applications to OCARP from literature are discussed. An integer linear programming formulation is given, followed by some properties of the problem. A Greedy Randomized Adaptive Search Procedure (GRASP) with path-relinking (PR) solution method is proposed and compared with other successful metaheuristics. Some features of this GRASP with PR are (i) reactive parameter tuning, where the metaheuristic parameters values are stochastically selected biased in favor of those values which produced the best solutions in average; (ii) a statistical filter, which discards initial solutions if they are unlikely to improve the incumbent best solution; (iii) infeasible local search, where high-quality solutions, though infeasible, are used to explore the feasible/infeasible boundaries of the solution space; (iv) evolutionary PR, a recent trend in which a pool of elite solutions is progressively improved by relinking pairs of elite solutions. Computational tests were conducted for both CARP and OCARP instances, and results reveal that the GRASP with PR is very competitive, achieving the best overall deviation from lower bounds. This work also proposes an exact algorithm for OCARP, based on the branch-and-bound paradigm. Three lower bounds are proposed, one of them uses a subgradient method to solve a Lagrangian relaxation. The computational tests compared the proposed branch-and-bound with a commercial state-of-the-art ILP solver. Results reveal that the branch-and-bound outperformed CPLEX in the overall average deviation from lower bounds.

Keywords: arc routing, path-scanning heuristic, greedy randomized adaptive search procedure, path-relinking, branch-and-bound, Lagrangian relaxation, subgradient method.

Resumo

O problema de roteamento em arcos capacitado e aberto (*open capacitated arc routing problem, OCARP*) é um problema de otimização combinatorial NP-difícil em que, dado um grafo não-direcionado, o objetivo consiste em encontrar um conjunto de rotas de custo mínimo para veículos com capacidade restrita que atendam a demanda de um subconjunto de arestas. O OCARP está relacionado com o problema de roteamento em arcos capacitado (*capacitated arc routing problem*, CARP), mas difere deste pois o OCARP não possui um nó depósito e as rotas não estão restritas a ciclos. Aplicações da literatura para o OCARP são discutidas. Uma formulação de programação linear inteira é fornecida junto com propriedades do problema. Uma metaheurística GRASP (*greedy randomized adaptive search procedure*) com reconexão por caminhos (*path-relinking*) é proposta e comparada com outras metaheurísticas bem-sucedidas da literatura. Algumas características do GRASP são: (i) ajuste reativo de parâmetros, cujos valores são estocasticamente selecionados com viés àqueles valores que produziram, em média, as melhores soluções; (ii) um filtro estatístico que descarta soluções iniciais caso estas tenham baixa probabilidade de superar a melhor solução incumbente; (iii) uma busca local infactível que gera soluções de baixo custo utilizadas para explorar fronteiras factíveis/infactíveis do espaço de soluções; (iv) a reconexão por caminhos evolutiva aprimora progressivamente um conjunto de soluções de elevada qualidade (soluções elites). Testes computacionais foram conduzidos com instâncias CARP e OCARP e os resultados mostram que o GRASP é bastante competitivo, atingindo os melhores desvios entre os custos das soluções e limitantes inferiores conhecidos. Este trabalho também propõe um algoritmo exato para o OCARP que se baseia no paradigma branch-and-bound. Três limitantes inferiores são propostos e um deles utiliza o método dos subgradientes para resolver uma relaxação lagrangeana. Testes computacionais comparam o algoritmo branch-and-bound com o CPLEX resolvendo um modelo reduzido OCARP de programação linear inteira. Os resultados revelam que o algoritmo branch-and-bound apresentou resultados melhores que o CPLEX no que diz respeito aos desvios entre limitantes e ao número de melhores soluções.

Palavras-chave: roteamento em arcos, heurística de varredura de caminhos, GRASP, reconexão por caminhos, branch-and-bound, relaxação lagrangeana, métodos dos subgradientes.

Contents

Agradecimentos **xi**

Abstract **xiii**

Resumo **xv**

List of Publications **xxi**

List of Tables **xxiii**

List of Figures **xxv**

List of Abbreviations **xxvii**

List of Symbols **xxix**

Introduction **1**

Introdução **3**

1 Arc Routing Problems **5**

1.1 Chinese Postman Problem 5

1.2 Rural Postman Problem 6

1.3 Capacitated Arc Routing Problem 6

1.3.1 Heuristics and Metaheuristics 7

1.3.2 Approximation Algorithms 7

1.3.3 Lower Bounds 8

1.3.4 Exact Algorithms 8

2 Open Capacitated Arc Routing Problem **9**

2.1 Problem Statement 9

2.2 Applications 10

2.2.1 Meter Reader Routing Problem . . . 10
2.2.2 Cutting Path Determination Problem . . . 11
2.3 ILP Model . . . 12
2.3.1 OCARP properties . . . 16
2.4 Complexity Study . . . 18
2.5 Solution Strategy . . . 20
2.6 Final Remarks . . . 21

3 Heuristics Approaches 23
3.1 Introduction . . . 23
3.2 Greedy Randomized Adaptive Search Procedure . . . 23
3.2.1 Constructive Phase . . . 24
3.2.2 Local Search Phase . . . 30
3.2.3 Statistical Filter . . . 33
3.3 Path-Relinking . . . 35
3.3.1 Solution distance metric . . . 35
3.3.2 Relinking of Solutions . . . 36
3.3.3 Elite solutions pool . . . 37
3.3.4 GRASP and Path-relinking coupling . . . 38
3.3.5 GRASP with Path-relinking pseudo-code . . . 41
3.4 Computational Experiments . . . 41
3.4.1 OCARP case study . . . 43
3.4.2 CARP case study . . . 46
3.5 Final Remarks . . . 52

4 Exact Approaches 57
4.1 Introduction . . . 57
4.2 Motivation . . . 57
4.3 Branch-and-Bound General Concepts . . . 59
4.4 Branch-and-Bound for the CDCMSFP . . . 60
4.4.1 Implementation Decisions . . . 60
4.4.2 Lower Bounds . . . 62
4.4.3 Lower Bounds Tightness . . . 67
4.4.4 Lower Bounds Composition . . . 67
4.4.5 Upper Bounds . . . 67
4.4.6 Problem Reduction . . . 68
4.4.7 Branch-and-Bound Pseudo-Code . . . 68
4.4.8 Complexity Study . . . 68
4.5 Alternative Lower Bounds . . . 70

4.6 Computational Experiments 71
4.7 Final Remarks 73

Conclusions **77**

Conclusões **80**

References **85**

List of Publications

Journals

Usberti, F. L., França, P. M. and França, A. L. M.: 2011a, Grasp with evolutionary path-relinking for the capacitated arc routing problem, *Computers and Operations Research* . doi: 10.1016/j.cor.2011.10.014.

Usberti, F. L., França, P. M. and França, A. L. M.: 2011b, The open capacitated arc routing problem, *Computers and Operations Research* **38**(11), 1543 – 1555.

Conferences

Assis, L. S., França, P. M. and Usberti, F. L.: 2009, *Reagrupamento Capacitado: Problema de Redistritamento de Lotes de Faturamento (in portuguese). In: XLI SBPO Brazilian Symposium of Operational Research*, Annals XLI SBPO, Porto Seguro.

Assis, L. S., França, P. M. and Usberti, F. L.: 2011, *Multicriteria Capacitated Districting Problem: Study Case on Power Distribution Companies In: IFORS 2011*, IFORS 2011, Melbourne.

Bacalhau, E. T., Usberti, F. L. and Filho, C. L.: 2011, *Dynamic Programming for Optimal Allocation of Maintenance Resources on Power Distribution Networks In: IFORS 2011*, IFORS 2011, Melbourne.

França, P. M., Assis, L. S. and Usberti, F. L.: 2009, *Metaheurística GRASP para o Problema de Agrupamento (in portuguese). In: XXXII CNMAC Congresso Nacional de Matemática Aplicada e Computacional*, Annals XXXII CNMAC, Cuiabá.

França, P. M., Assis, L. S., Usberti, F. L. and Garcia, V. J.: 2009, *Multicriteria Capacitated Redistricting Problem. In: 23rd EURO European Conference on Operational Research 2009*, Annals 23rd EURO, Bonn.

França, P. M., Garcia, V. J., França, A. L. M. and Usberti, F. L.: 2007a, *Algoritmos para Roteamento de Leituristas (in portuguese). In: XXX CNMAC Congresso Nacional de Matemática Aplicada e Computacional*, Annals XXX CNMAC, Florianópolis.

França, P. M., Garcia, V. J., França, A. L. M. and Usberti, F. L.: 2007b, *Enfoque Multicritério*

para o Problema de Redistritamento Capacitado (in portuguese). In: XXXIX SBPO Brazilian Symposium of Operational Research, Annals XXXIX SBPO, Fortaleza.

Reis, P. A., Lyra, C., Cavellucci, C., Zuben, F. J. V., Usberti, F. L., Gonzalez, J. F. V., Coelho, G. P. and Ferreira, H. M.: 2008, *Problema de Alocação Ótima de Recursos de Manutenção: Formulação e Estudos de Caso (in portuguese). In: XL SBPO Brazilian Symposium of Operational Research*, Annals XL SBPO, João Pessoa.

Usberti, F. L.: 2007, *SIMANFIS: Simplificação da Arquitetura Neuro-Fuzzy ANFIS (in portuguese). In: XXXIX SBPO Brazilian Symposium of Operational Research*, Annals XXXIX SBPO, Fortaleza.

Usberti, F. L., França, P. M. and França, A. L. M.: 2008a, *Heuristics for the Capacitated Rural Postman Problem. In: 18th IFORS Triennial Conference of the International Federation of Operational Research Societies*, Annals 18th IFORS, Sandton.

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

Usberti, F. L., França, P. M. and França, A. L. M.: 2009, *The Open Capacitated Arc Routing Problem. In: 8th CTW Cologne-Twente Workshop on Graphs and Combinatorial Optimization*, Annals 8th CTW, Paris.

Usberti, F. L., França, P. M. and França, A. L. M.: 2010, *GRASP-Tabu Search for the Open Capacitated Arc Routing Problem. In: ALIO-INFORMS Joint International Meeting*, Annals of ALIO-INFORMS, Buenos Aires.

Usberti, F. L., França, P. M. and França, A. L. M.: 2011a, *GRASP with Path-Relinking for two Arc Routing Problems In: IFORS 2011*, Annals IFORS 2011, Melbourne.

Usberti, F. L., França, P. M. and França, A. L. M.: 2011b, *New Lower and Upper Bounds for the Open Capacitated Arc Routing Problem In: Optimization 2011*, Annals Optimization 2011, Lisboa.

Usberti, F. L., França, P. M. and França, A. L. M.: 2011c, *On the complexity and heuristic methods for a new arc routing problem In: ORP3 2011*, Annals ORP3 2011, Cádiz.

Usberti, F. L., Gonzalez, J. F. V., Lyra, C. and Cavellucci, C.: 2009, *Maintenance resources allocation on power distribution networks with a multi-objective framework. In: 8th Cologne-Twente Workshop on Graphs and Combinatorial Optimization*, Annals 8th CTW, Paris.

Usberti, F. L., Lyra, C., Cavellucci, C. and Gonzalez, J. F. V.: 2010, *Two-level multicriteria optimization of maintenance activities on power distribution networks. In: XLII SBPO Brazilian Symposium of Operational Research*, Annals XLII SBPO, Bento Gonçalves.

List of Tables

3.1 Tuning scheme for the parameters γ and g. 29
3.2 GRASP parameters. 43
3.3 Comparison results between path-scanning heuristics. 45
3.4 GRASP with PR results for *ogdb* instances. 46
3.5 GRASP with PR results for *oval* instances. 47
3.6 GRASP with PR results for *oegl* instances. 48
3.7 Comparison of constructive heuristics. 49
3.8 GRASP results for *gdb* instances 52
3.9 GRASP results for *val* instances 53
3.10 GRASP results for *egl* instances 54
3.11 Summary results for CARP metaheuristics. 55

4.1 Branch-and-bound parameters. 71
4.2 Branch-and-bound overall results. 73
4.3 Branch-and-bound results for *ogdb* instances. 74
4.4 Branch-and-bound results for *oval* instances. 75
4.5 Branch-and-bound results for *oegl* instances. 76

List of Figures

2.1 An OCARP instance and a corresponding feasible solution. 10
2.2 Transforming a CPDP instance into an OCARP instance. 12
2.3 OCARP illegal subcycle. 14
2.4 CARP graph G transformed into OCARP graph G_1. 18
2.5 OCARP graph G transformed into CARP graph G_1. 20

3.1 Evolution of the parameter γ for instance *oegl-s4-C*. 27
3.2 Neighborhood moves with possible reversals (in red). 31
3.3 Distribution of the ratio c_{ini}/c_{ls} for instance *oegl-e1-A*. 34
3.4 Relinking two solutions. 37
3.5 Evolution of the parameter γ for instance *oegl-e4-C*. 44
3.6 Run time distributions for GRASP with (Gf) and without (Gnf) filtering on instance *egl-s4-C*. 50
3.7 Run time distributions for GRASP with (EvPR) and without (PR) evolutionary path-relinking on instance *egl-s4-C*. 51

4.1 Transforming OCARP graph $G(V, E)$ into the augmented graph $\tilde{G}(\tilde{V}, \tilde{E})$. . 58
4.2 Example of the branch-and-bound search tree expansion. 61
4.3 Branch-and-bound search tree for OCARP. 62

List of Abbreviations

ACO	ant colony optimization.
CARP	capacitated arc routing problem.
CARP-TW	CARP with time windows.
CCPP	capacitated CPP.
CDCMSFP	capacity and degree constrained minimum spanning forest problem.
CPDP	cutting path determination problem.
CPLEX	solver for linear programming problems.
CPP	chinese postman problem.
CVRP	capacitated vehicle routing problem.
DCARP	directed CARP.
DCMSTP	degree constrained minimum spanning tree problem.
egl	set of CARP instances (Li and Eglese 1996).
EvPR	evolutionary path-relinking.
FFD	first-fit-decreasing.
GCH	greedy constructive heuristic.
gdb	set of CARP instances (Golden, DeArmon and Baker 1983).
Gf	GRASP with filtering.
Gnf	GRASP without filtering.

GRASP	greedy randomized adaptive search procedure.
ILP	integer linear programming.
LB0	trivial CARP and OCARP lower bound.
LB1	minimum cost edges lower bound.
LB2	minimum cost spanning forest lower bound.
LB3	degree constrained minimum cost spanning forest lower bound.
MCARP	mixed CARP.
MRRP	meter reader routing problem.
OCARP	open CARP.
oegl	set of OCARP instances based the egl instances.
ogdb	set of OCARP instances based the gdb instances.
oval	set of OCARP instances based the val instances.
PR	path-relinking.
PS_ER	path-scanning with ellipse rule.
RCL	restricted candidate list.
RPP	rural postman problem.
TS	tabu-search.
TTT	time-to-target plots.
val	set of CARP instance (Benavent, Campos, Corberán and Mota 1992).
VNS	variable neighborhood search.

List of Symbols

α_i^k	binary decision variable; $\alpha_i^k = 1$ if node i is the source of tour k, $\alpha_i^k = 0$ otherwise.
$\alpha(.)$	inverse Ackermann function.
α	restricted candidate list parameter.
β	ellipse rule parameter.
δ	arbitrary positive demand ($\delta > 0$).
δ_{ij}	broken pairs distance between two solutions S_i and S_j.
δ_{min}	minimum distance between solutions belonging to the pool.
ΔLB	deviation from lower bound $\Delta LB = (UB - LB)/LB$.
$\Delta LB^{\max}$	maximum average deviation from lower bound.
γ	cost-demand edge-selection rule parameter.
λ	Lagrangian multiplier.
$\leqslant_p$	polynomial reduction, e.g., $P_A \leqslant_p P_B$ denotes that problem P_A can be polynomially reduced in problem P_B.
μ	average value of the ratio c_{ini}/c_{ls}.
Π	set of values for a generic parameter π.
ψ	path-scanning heuristic edge-selection rule.
$\tilde{\psi}$	path-scanning heuristic cost-demand edge-selection rule.
$\tilde{\psi}_{\max}$	maximum evaluation of the cost-demand edge-selection rule.

$\tilde{\psi}_{\min}$ minimum evaluation of the cost-demand edge-selection rule.

σ standard deviation of the ratio c_{ini}/c_{ls}.

τ parameter that prevents the step size becoming to small in the subgradient method.

θ parameter that adjusts the step size in the subgradient method.

A possible values for α.

B possible values for β.

c_{best} incumbent best solution cost.

c_{ini} initial solution cost.

c_{ls} local search solution cost.

$c(e) = c_{ij}$ cost of edge $e = (i, j)$.

$\overline{c}_i$ average cost of the solutions obtained by using $\pi = \pi_i$.

$\overline{c}_{\max}$ maximum average cost ($\overline{c}_{\max} = \max_i \overline{c}_i$).

$\overline{c}_{\min}$ minimum average cost ($\overline{c}_{\min} = \min_i \overline{c}_i$).

C arbitrary high cost ($C \gg SP^{\max}$).

CPU average execution time.

$d^{\max}$ maximum edge demand ($d^{\max} = \max_{e \in E_R} d(e)$).

$d(e) = d_{ij}$ demand of edge $e = (i, j)$.

D vehicle capacity.

$e = [v_i, v_j] = (i, j)$ representations for edges linking nodes $v_i = i$ and $v_j = j$ of graph $G(V, E)$.

E_{R0} set of dummy required edges connecting a dummy node $v \in V_0$ with the depot v_0.

E_0 set of dummy non-required edges linking v_0 to every node in G.

E_1 set of edges resulting from the addition of the dummy edges.

E_R set of required edges ($(i, j) \in E_R | (i, j) \in E$ and $d_{ij} > 0$).

$\tilde{E}_{NR}$ set of non-required edges from $\tilde{E}_S$.

$\tilde{E}_R$ set of required edges from $\tilde{E}_S$.

$\tilde{E}_S$ set of edges from $\tilde{F}$.

$\tilde{F}(\tilde{V}, \tilde{E}_S)$ spanning forest in $\tilde{G}$, representing an OCARP solution.

$Feas$ ratio of feasible solutions obtained by a heuristic compared to the whole number of solutions generated.

$G_1(V_1, E_1)$ graph resulting from the addition of the dummy nodes and edges.

$G(V, E)$ undirected graph with a set of vertices V and a set of edges E.

$\tilde{G}(\tilde{V}, \tilde{E})$ augmented complete graph with a set of vertices $\tilde{V}$ and a set of edges $\tilde{E}$.

g cost-demand edge-selection rule metaparameter.

I_L list of the indices for each edge in L ($I_L[e_i] = i$).

k^{end} maximum number of GRASP iterations.

k^{eps} number of GRASP iterations between evolutionary path-relinking executions.

k^{filter} number of iterations to calibrate the filter threshold.

l_{ij}^k binary decision variable; $l_{ij}^k = 1$ if tour k services arc (i, j), $l_{ij}^k = 0$ otherwise.

L sorted list of all non-required edges $\tilde{E} \setminus \tilde{E}_R$ in increasing order of cost.

LB_{bb} lower bound obtained with the proposed branch-and-bound algorithm.

LB_{best} best lower bound between the branch-and-bound and CPLEX ($LB_{best} = \max\{LB_{cplex}, LB_{bb}\}$).

LB_{cplex} lower bound obtained with CPLEX.

LB_0 trivial CARP and OCARP lower bounds ($LB_0 = \sum_{e \in E_R} c(e)$).

LB lower bound.

m number of possible values for a generic parameter π ($m = |\Pi|$).

M number of vehicles available.

M^* minimum number of vehicles necessary for a feasible solution.

n_{best} number of best solutions obtained by a heuristic.

n_{inf} maximum number of infeasible moves to execute with the infeasible local search.

n_{me} number of missing edges to complete an OCARP solution.

ned number of required edges ($ned = |E_R|$).

$N(i)$ set of adjacent nodes of node i ($j \in N(i) | j \in V$ and $(i,j) \in E$).

N_{tree} total amount of nodes in the branch-and-bound search tree.

p_i probability of choosing the value π_i for a generic parameter π.

$p_{\max}$ probability associated to the generic parameter value π_i with the lowest average solution cost ($\overline{c}_i = \overline{c}_{\min}$).

$p_{\min}$ probability associated to the generic parameter value π_i with the highest average solution cost ($\overline{c}_i = \overline{c}_{\max}$).

$P_{carp}(G)$ induced graph by a CARP solution in G.

$P_{ocarp}(G)$ induced graph by an OCARP solution in G.

$P^*_{carp}(G)$ induced graph by a CARP optimal solution in G.

$P^*_{ocarp}(G)$ induced graph by an OCARP optimal solution in G.

q_i variable inversely proportional to $\overline{c}_i$, and which is used to determine p_i.

S CARP or OCARP solution.

$SP^{\max}$ cost of the greatest minimum shortest path between any two nodes ($SP^{\max} = \max_{i,j \in V} SP(i,j)$).

$SP(i,j)$ shortest path cost between nodes i and j.

tc total cost from required edges ($tc = \sum_{e \in E_R} c(e)$).

td total demand to be serviced ($td = \sum_{e \in E_R} d(e)$).

$\tilde{T}$ tree in $\tilde{G}$, representing an OCARP tour.

u^k_S binary auxiliary variable; if $u^k_S = 0$ then tour k does not have a sufficient number of arcs to form a cycle in S ($S \subseteq V$).

UB upper bound.

v_S^k binary auxiliary variable; if $v_S^k = 0$ then tour k traverses the cut-set $(S, \tilde{S})$ $(S \subseteq V, \tilde{S} = V \setminus S)$.

v_0 depot node.

$v_i = i$ representations for nodes of graph $G(V, E)$.

V_0 set of dummy nodes.

V_1 set of nodes resulted from the union with the dummy nodes set.

v_l last node visited by a tour.

w_S^k binary auxiliary variable; if $w_S^k = 0$ then tour k contains a source node in S $(S \subseteq V)$.

x_{ij}^k binary decision variable; $x_{ij}^k = 1$ if tour k traverses arc (i, j), $x_{ij}^k = 0$ otherwise.

Introduction

The Capacitated Arc Routing Problem (CARP), proposed by Golden and Wong (1981), is a combinatorial optimization problem defined in a connected undirected graph $G(V, E)$ with non-negative costs and demands on its edges. There is a fleet of identical vehicles with limited capacity that must service all edges with positive demand (required edges). The objective is to search for a set of minimum cost tours that start and finish in a distinguished node, called depot.

Many real world applications have been related to arc routing problems, such as street sweeping, garbage collection, mail delivery, school bus routing, meter reading etc, and estimates on the expenditure involved in these services reach billions of dollars annually in the US, thus revealing a substantial savings potential (Dror 2001).

This work objective is to study the Open Capacitated Arc Routing Problem (OCARP) (Usberti, França and França 2011b). The OCARP is similar to CARP but in the first problem tours are not constrained to form cycles. Therefore both open and closed tours are permitted. In both problems there are required edges, which must be serviced, and non-required edges, used within shortcut paths from one required edge to another. Considering that a tour start at node v_s and finish at node v_t, in CARP $v_s = v_t = v_0$ for all tours, being a particular case of OCARP. Consequently, the OCARP can be seen as a CARP generalization, and to the best of our knowledge this problem has never been formally reported in the literature, in spite of important practical problems be easily modeled as an OCARP.

This thesis is organized in four chapters. Chapter 1 gives a background in arc routing problem, describing three well-known examples such as the Chinese Postman Problem (CPP), the Rural Postman Problem (RPP), and the CARP. A more detailed presentation is given for the later, including a formulation, heuristics, lower bounding procedures and exact algorithms.

Chapter 2 formally introduces the OCARP. For this problem real-life applications are presented, such as the Meter Reader Routing Problem and the Cutting Path Determination Problem. The OCARP is formulated as an integer linear programming problem, and properties are given. A complexity study reveals that OCARP is NP-hard. A polynomial reduction is proposed allowing, under certain conditions, CARP methods to be applied to OCARP instances.

A heuristic approach to solve OCARP is presented in Chapter 3. This heuristic is based on

the greedy randomized adaptive search procedure (GRASP) with evolutionary path-relinking (PR), and it was developed to solve both CARP and OCARP. Some features of this GRASP with PR are (i) reactive parameter tuning, where the parameters values are tuned based on those values which historically produced the best solutions in average; (ii) a statistical filter, which discards initial solutions if they are unlikely to improve the incumbent best solution; (iii) infeasible local search, where high-quality solutions, though infeasible, are used to explore the feasible/infeasible boundaries of the solution space; (iv) evolutionary PR, a recent trend in which a pool of elite solutions is progressively improved by relinking pairs of elite solutions. Computational tests were conducted for both CARP and OCARP instances, and results reveal that the GRASP is very competitive compared to other metaheuristics from the literature, achieving the best overall deviation from lower bounds.

Chapter 4 shows an exact branch-and-bound algorithm proposed to solve OCARP. The algorithm is motivated by the similarity between OCARP and a spanning tree problem. Three lower bounds are proposed with distinct trade-offs between computational effort and bound tightness. One of the lower bounds is based on Lagrangian relaxation solved by a subgradient method. Computational experiments are conducted in a set of literature instances, where the branch-and-bound results are compared with CPLEX.

Finally, the section Conclusions highlights the contributions of this work and points out future lines of investigation.

Introdução

O problema de roteamento em arcos capacitado (*capacitated arc routing problem*, CARP), proposto por Golden and Wong (1981), é um problema de otimização combinatória definido em um grafo conexo não-direcionado $G(V, E)$ com custos e demandas não-negativas nas arestas. Uma frota de veículos idênticos com capacidade limitada precisa atender todas as arestas com demanda positiva (arestas requeridas). O objetivo do CARP consiste em encontrar um conjunto de rotas de custo mínimo que se iniciem e terminem em um nó distinto, denominado depósito.

Muitas aplicações reais estão relacionadas com problemas de roteamento em arcos, como varrição de ruas, coleta de resíduos, entrega de correios, roteamento de ônibus escolares, leitura de medidores de energia, água e gás. Estimativas sobre os custos envolvidos nesses serviços atingem bilhões de dólares anualmente nos Estados Unidos, revelando um enorme potencial para economia de recursos via otimização (Dror 2001).

Este trabalho tem por objetivo o estudo do problema de roteamento em arcos capacitado e aberto (*open capacitated arc routing problem*, OCARP) (Usberti et al. 2011b). O OCARP é similar ao CARP, porém, no primeiro problema, as rotas não estão restritas a ciclos. Logo tanto rotas abertas quanto fechadas são permitidas. Nos dois problemas existem as arestas requeridas, que devem ser atendidas, e as arestas não-requeridas, que são utilizadas como atalhos entre uma aresta requerida e outra. Considerando que uma rota se inicia em um nó v_s e termina em um nó v_t, no CARP tem-se que $v_s = v_t = v_0$ para todas as rotas, sendo assim um caso particular do OCARP. Consequentemente, o OCARP pode ser enxergado como uma generalização do CARP e, após uma exaustiva busca, constatou-se que esse problema nunca foi formalmente reportado na literatura, apesar de relevantes problemas práticos poderem ser modelados como um OCARP.

Este trabalho está organizado em quatro capítulos. O Capítulo 1 fornece um histórico de problemas de roteamento em arcos, descrevendo três exemplos conhecidos da literatura como o problema do carteiro chinês (*chinese postman problem*, CPP), o problema do carteiro rural (*rural postman problem*, RPP), e o CARP. Uma descrição mais detalhada deste último é fornecida, incluindo formulação, heurísticas, algoritmos para geração de limitantes inferiores e algoritmos exatos.

O Capítulo 2 introduz formalmente o OCARP. Para esse problema, são apresentadas aplicações reais, como o problema de roteamento de leituristas e o problema de determinar o caminho de

corte. O OCARP é formulado como um problema de programação linear inteira e algumas propriedades interessantes são fornecidas. Um estudo de complexidade revela que o OCARP é NP-difícil. Uma redução polinomial é proposta permitindo, sob certas condições, aplicar métodos do CARP para instâncias do OCARP.

Um método heurístico para resolver o OCARP é apresentado no Capítulo 3. Essa heurística é baseada na metaheurística GRASP (*greedy randomized adaptive search procedure*) com reconexão por caminhos evolutiva (*evolutionary path-relinking*), concebida para resolver ambos CARP e OCARP. Algumas características desse GRASP são: (i) ajuste reativo de parâmetros, onde os valores dos parâmetros são selecionados com base naqueles valores que, em média, produzem as melhores soluções; (ii) um filtro estatístico que descarta soluções iniciais se estas provavelmente não conseguiriam aprimorar a melhor solução incumbente; (iii) uma busca local infactível, capaz de produzir soluções de baixo custo, contudo infactíveis, que são utilizadas na exploração das fronteiras factível/infactível do espaço de soluções; (iv) reconexão por caminhos evolutiva, considerada uma nova tendência em otimização, aperfeiçoa continuamente um conjunto de soluções de elite a partir de sucessivas reconexões de pares de soluções de elevada qualidade. Testes computacionais foram conduzidos com instâncias CARP e OCARP, e os resultados mostram que o GRASP é muito competitivo se comparado a outras metaheurísticas da literatura, atingindo os menores desvios entre os custos das soluções e limitantes inferiores conhecidos.

O Capítulo 4 mostra um algoritmo branch-and-bound proposto para resolver de forma exata o OCARP. O algoritmo é motivado pela similaridade entre o OCARP e um problema de árvore geradora. Três limitantes inferiores são propostos, e cada um possui uma relação distinta entre o custo computacional de obtê-lo e a qualidade do limitante produzido. Um dos limitantes inferiores é baseado em relaxação lagrangeana e calculado a partir do método de subgradientes. Experimentos computacionais são conduzidos com instâncias da literatura, e os resultados obtidos com o algoritmo branch-and-bound proposto são comparados com um *solver* comercial estado-da-arte para problemas de programação linear inteira.

O capítulo de conclusões destaca as contribuições deste trabalho e aponta para futuras linhas de investigação.

Chapter 1

Arc Routing Problems

The objective of arc routing problems, according to Eiselt, Gendreau and Laporte (1995a), consists in determining the least cost traversal of a given subset of edges in a graph, with one or more collateral constraints. Three important arc routing problems described in this chapter are the Chinese Postman Problem (CPP), the Rural Postman Problem (RPP), and the Capacitated Arc Routing Problem (CARP).

1.1 Chinese Postman Problem

The Chinese Postman Problem (CPP) was introduced by Mei-ko (1962) and its objective is to find a minimum cost tour for a connected graph $G(V, E)$ with non-negative costs on the edges, and the tour must visit all the graph edges at least once.

A polynomial exact algorithm to solve the CPP is described by Eiselt et al. (1995a), and it can be divided in two phases. The first is to determine a minimum cost graph augmentation duplicating a sufficient number of edges so that the graph becomes Eulerian, i.e., all nodes having an even degree.

Edmonds and Johnson (1973) revealed an efficient polynomial algorithm to accomplish the first phase for directed and undirected graphs by solving a minimum cost matching problem.

The second phase constructs a traversal of this augmented graph, which is an easy task for Eulerian graphs. When the graph is mixed (containing both directed and undirected edges), Papadimitriou (1976) shows that solving the first phase becomes an NP-hard problem. Negreiros, Coelho Júnior, Palhano, Coutinho, de Castro, Gomes, Barcellos, Rezende and Pereira (2009) developed efficient implementations for the CPP on symmetric, directed and mixed graphs, which solve real-life instances (up to $|V| = 1383$ and $|E| = 2486$).

1.2 Rural Postman Problem

In many real-life problems, only a subset of edges in the graph $G(V, E)$ requires some kind of service. That is the case in the Rural Postman Problem (RPP), whose objective is to find a minimum cost tour that traverses a subset of edges (required edges, $E_R \subseteq E$). This problem is NP-hard for undirected, directed and mixed graphs (Eiselt, Gendreau and Laporte 1995b). An approximation algorithm proposed by Frederickson (1979) is based on the CPP exact algorithm. This approximation algorithm tries to find a low cost graph augmentation to make the graph Eulerian, and then it traces a tour that covers all required edges. To perform the graph augmentation, the minimum cost matching and the minimum cost spanning tree algorithms are used.

1.3 Capacitated Arc Routing Problem

The Undirected Capacitated Arc Routing Problem (CARP or UCARP), proposed by Golden and Wong (Golden and Wong 1981), is a combinatorial optimization problem defined in a connected undirected graph $G(V, E)$ where non-negative costs c_{ij} and demands d_{ij} are assigned to each edge $e = [v_i, v_j]$. All edges with positive demand (required edges, $E_R \subseteq E$) must be serviced by a fleet of identical vehicles with limited capacity D. While traversing the graph, a vehicle might (i) service an edge, which deducts the demand from the vehicle capacity and increases the solution cost, or (ii) deadhead an edge (traverse an edge without servicing it), which only increases the solution cost. A tour is defined feasible when it starts and ends at a distinguished node v_0, called depot, and the sum of the demands serviced by that vehicle is less than or equal to D. A feasible solution is formed by a family of feasible tours, which services all required edges. The CARP objective is to search for a minimum cost feasible solution.

Other problems related to CARP are the Directed CARP or DCARP (directed graph), the Mixed CARP or MCARP (mixed graph), the Capacitated Chinese Postman Problem or CCPP ($E_R = E$), the CARP with Time Windows or CARP-TW (required edges must be serviced within a given time interval).

Many real world applications have been related to CARP, such as street sweeping, garbage collection, mail delivery, school bus routing, meter reading etc. Estimates on the expenditure involved in these services reaches billions of dollars annually only in the United States, thus revealing a substantial savings potential. Details on these applications are provided in Eiselt et al. (1995b), Assad and Golden (1995) and Dror (2001).

As Dror (2001) observed, there are two CARP versions with respect to the number of vehicles. In the first one, which corresponds to the original CARP conception, the number of vehicles is a fixed parameter. The second version considers the number of vehicles as a decision variable which means that, in this version, CARP algorithms account for an unlimited fleet of vehicles.

Welz (1994) observed that determining whether a feasible solution exists for a given number of tours is already NP-hard, since it requires solving a bin-packing problem. This may be the reason why most state-of-the-art heuristics deal with the second CARP version.

An integer linear programming (ILP) model for CARP, proposed by Golden and Wong (1981), is described in Chapter 2 since it will be used to derive a valid ILP model for the OCARP.

A more detailed overview on the CARP complexity, polyhedral results, exact, approximate and heuristic algorithms can be found in the references Eiselt et al. (1995b), Dror (2001), Hertz (2005), Wøhlk (2008b), and Corberán and Prins (2010).

1.3.1 Heuristics and Metaheuristics

Due to the CARP complexity, many real world instances are intractable for exact algorithms, hence opening research for heuristics which, despite being unable to guarantee optimality, perform well in most cases, providing high-quality solutions on average. Examples of constructive heuristics for the CARP are path-scanning (Golden et al. 1983; Santos, Coutinho-Rodrigues and Current 2009), augment-merge (Golden and Wong 1981), and augment-insert (Pearn 1991). Better CARP solutions were obtained through metaheuristics such as tabu search (Eglese and Li 1996; Hertz, Laporte and Mittaz 2000; Brandão and Eglese 2008), genetic algorithm (Lacomme, Prins and Ramdane-Chérif 2004), hybrid tabu-scatter search (Greistorfer 2003), guided local search (Beullens, Muyldermans, Cattrysse and Oudheusden 2003), variable neighborhood search (Hertz and Mittaz 2001; Polacek, Doerner, Hartl and Maniezzo 2008; Maniezzo and Roffilli 2008), and ant colony optimization (Santos, Coutinho-Rodrigues and Current 2010). Two GRASPs with path-relinking (Prins and Calvo 2005; Labadi, Prins and Reghioui 2008) were developed for the CARP (and CARP-TW in the case of Labadi et al. (2008)). The solution quality of these two GRASPs, however, were outperformed by the three most recent metaheuristics (Brandão and Eglese 2008; Polacek et al. 2008; Santos et al. 2010).

1.3.2 Approximation Algorithms

In terms of approximation algorithms, it has been shown that even the $\frac{3}{2}$-approximation for the CARP is already NP-hard (Golden and Wong 1981). The current best approximation factor is $\frac{7}{2} - \frac{3}{D}$ (Wøhlk 2008a), where D is the vehicle capacity. This algorithm assumes that the matrix of edges costs satisfies the triangle inequality, otherwise, finding an α-approximation for the CARP would be NP-hard for any $\alpha > 0$. The idea of this approximation algorithm is to construct a single tour servicing all required edges, and then using dynamic programming to optimally partition the tour into smaller tours that respect the vehicle capacity.

1.3.3 Lower Bounds

Lower bounding schemes for CARP, such as *Capacity constraints*, *Odd Edge Cutset constraints*, *Disjoint Paths* (Belenguer and Benavent 2003), and *Multiple Cuts Node Duplication Lower Bound* (Wøhlk 2006) affirm that, for a subset $S \subseteq V \setminus \{v_0\}$, some edges must be deadheaded depending on the total demand to be serviced in S, the number of required edges in the cutset $(S, S \setminus V)$, and possibly some additional demand on the path from v_0 to S. There are some additional references concerning lower bounding procedures for the CCPP (Assad, Pearn and Golden 1987), DCARP (Mourão and Almeida 2000), CARP (Pearn 1988; Benavent et al. 1992), and MCARP (Belenguer, Benavent, Lacomme and Prins 2006; Gouveia, Mourão and Pinto 2010).

1.3.4 Exact Algorithms

Despite CARP being NP-hard, attempts were made towards solving CARP to optimality, including a branch-and-bound algorithm (Hirabayashi, Saruwatari and Nishida 1992), and a CARP reduction into the capacitated vehicle routing problem (CVRP), which is then solved by a branch-and-cut-and-price algorithm (Longo, de Aragão and Uchoa 2006). This reduction to the CVRP augments the original graph into a complete graph with $2|E_R| + 1$ vertices, where E_R is the number of required edges. Augmenting the graph reflects negatively on the size of the instances the branch-and-cut-and-price algorithm can manage. The largest instance solved to optimality by this method contains 77 nodes and 87 required edges.

Chapter 2

Open Capacitated Arc Routing Problem

This chapter gives a formal description to OCARP in Section 2.1, followed by some applications in Section 2.2. An integer linear programming (ILP) is formulated in Section 2.3, deriving also some interesting properties. The OCARP complexity is analyzed in Section 2.4 and a solving strategy is given in Section 2.5. Section 2.6 presents the final remarks which closes this chapter.

2.1 Problem Statement

Let $G(V, E)$ be an undirected connected graph where non-negative costs c_{ij} and demands d_{ij} are assigned to each edge $e = [v_i, v_j]$. All edges with positive demands, called required edges ($E_R \subseteq E$), must be serviced once by a single vehicle. Nonetheless, all edges may be traversed multiple times, by one or more vehicles. A fleet of M identical vehicles with limited capacity D is available. While traversing the graph, a vehicle might (i) service an edge, which deducts the demand from the vehicle capacity and increases the solution cost, or (ii) deadhead an edge, same as traverse an edge without servicing it, which only increases the solution cost.

Despite graph G being undirected, it is convenient to represent OCARP solutions with directed arcs. These arcs gives orientation and order in which the edges of graph G are traversed. Throughout the text, undirected edges are referring to the OCARP instance, and directed arcs are referring to the OCARP solution (see example in Figure 2.1).

The OCARP considers both open and closed tours. An open tour uses distinct nodes to start (source node) and end (sink node) the tour, while in a closed tour the source and sink nodes are the same. A feasible OCARP solution is formed by a family of feasible tours, which services all required edges and does not violate any vehicle capacity. The objective of OCARP is to find the minimum cost family of tours (Usberti et al. 2011b).

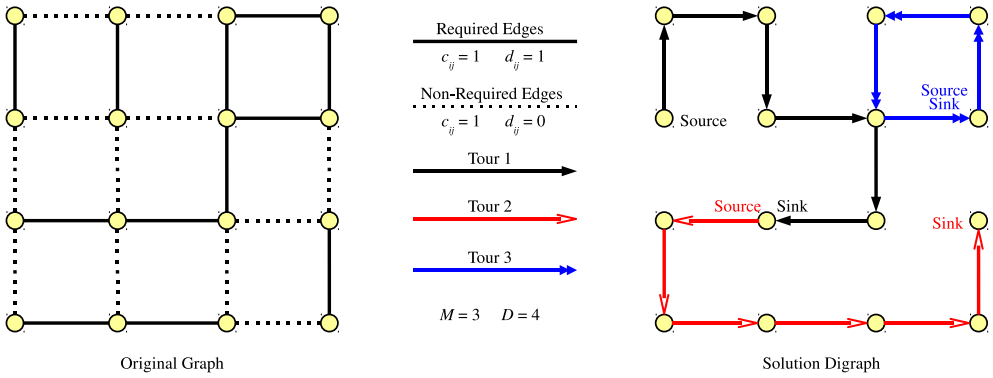


Figure 2.1: An OCARP instance and a corresponding feasible solution.

2.2 Applications

Many combinatorial optimization problems can be represented as an OCARP instance. For the sake of simplicity this section selects only two problems from literature which revealed themselves as interesting applications for the OCARP.

2.2.1 Meter Reader Routing Problem

The Meter Reader Routing Problem (MRRP) interests major electric, water and gas distribution companies which periodically needs to meter read their clients. Like the OCARP, the MRRP does not consider a depot since the employees responsible for metering, called meter readers, are taken by auto from the office to the address of their first card, and after completing their routes, they take public transportation to return home. The objective is to find a set of tours for meter readers, with limited amount of working time, that visit every street segment containing clients in a minimum traversal time. Service time is incurred whenever an employee meter reads, while a shorter deadheading time is computed when the employee is not reading. All street segments have positive deadhead time, however some may have zero service time, which means there are no clients on that segment.

Stern and Dror (1979) routed meter readers for the state power company from Beersheva, Israel, and developed a *route first, cluster second* heuristic, where initially the problem is treated as non-capacitated, and a single route covers all required edges. This single route is partitioned into segments, each designated to a meter reader.

Wunderlich, Collette, Levy and Bodin (1992) routed meter readers for the Southern California Gas Company (SOCAL) from Los Angeles, USA, using an adapted arc partitioning algorithm developed and further improved by Bodin and Levy (1989) and Bodin and Levy (1991). In their algorithm, the graph is partitioned into meter readers territories, followed by tour construction

inside each territory. This algorithm represents a reverse strategy compared to Stern and Dror proposition, i.e., a *cluster first, route second* heuristic.

The transformation from MRRP to OCARP comes naturally:

1. Vehicles correspond to meter readers, with capacity equal to their working time.
2. Vertices correspond to street intersections.
3. Edges correspond to street segments.
4. Edge cost represents deadheading time.
5. Edge demand represents servicing time.

A singularity of the MRRP is that even when the reader is not servicing an edge, he is using part of his working time by deadheading. Therefore the corresponding OCARP vehicle should have its remaining capacity decreased not only when servicing, but also when deadheading.

2.2.2 Cutting Path Determination Problem

In the Cutting Path Determination Problem (CPDP) the trajectories of a set of blowtorches must be determined for a cut pattern on a quadrilateral steel plate in order to produce a predefined set of polygonal pieces in minimum time. A piece is produced when its shape is fully traversed by one or more blowtorches. These blowtorches have a limited amount of energy to spend and must not traverse the interior of any shape, but they may dislocate above the plate level, reflecting additional elevating and lowering maneuvers times. Moreira, Oliveira, Gomes and Ferreira (2007) investigated a version of CPDP using the concept of a dynamic rural postman problem. In their dynamic version, the related graph changes during the cutting process because when a piece is produced it falls off into a special container, therefore giving new possible paths for the blowtorches to take.

The CPDP may also be modeled as an OCARP through the following transformation:

1. Graph vertices correspond to polygons vertices. A vertex can be removed if only two non-required edges incides on it, and a single non-required edge replaces both inciding edges. The cost of the new edge is made equal to the sum of the costs of the previous ones.
2. Graph edges correspond to polygons edges.
3. Non-required edges are formed by the set of rectilinear trajectories between all pairs of vertices. This set of non-required edges is partitioned into *upper* and *lower* non-required edges. The lower edges are those which do not overrun the interior of any polygon. The upper edges represent a blowtorch dislocation above the plate level.

4. Edge cost corresponds to traversal time, reminding that non-required upper edges imply additional times due to elevating and lowering maneuvers. Moreover, cutting an edge is a slower process than to only traversing it, thus required edges have higher costs compared to non-required edges.

5. Demand represents the energy spent to cut through the plate (while traversing required and lower non-required edges). It is assumed that energy is spent only when the blowtorchs are cutting.

6. Vehicles correspond to blowtorches, with capacity equivalent to the amount of energy they are allowed to spend.

7. The objective function seeks the minimum makespan.

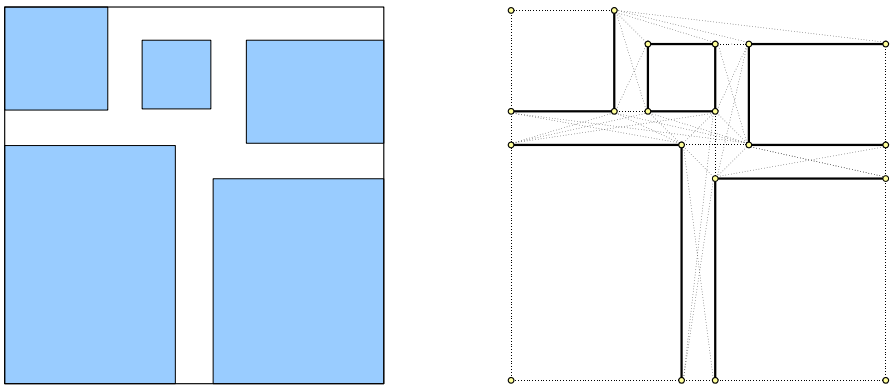


Figure 2.2: Transforming a CPDP instance into an OCARP instance.

This transformation has polynomial complexity bounded by $O(n^2)$, where n is the number of vertices. Figure 2.2 shows an example of a cutting pattern (plate with the shapes to be produced) and the equivalent OCARP graph. Required edges are represented by continuous lines, and non-required by dotted lines. Given the large amount of upper non-required edges, these were omitted.

2.3 ILP Model

In this section an integer linear programming model for OCARP is proposed, starting with the following CARP model from Golden and Wong (1981) and then discussing the differences. This model considers only directed edges, hence each edge $(i, j) \in E$ from the undirected CARP graph $G(V, E)$ is treated as two arcs (i, j) and (j, i). It is assumed that node v_0 is the depot,

and that the number of vehicles M is a fixed parameter. $N(i)$ denotes the nodes adjacent to node i in G.

Two sets of decision variables are defined: $x_{ij}^k = 1$ if tour k traverses arc (i,j), $x_{ij}^k = 0$ otherwise; $l_{ij}^k = 1$ if tour k services arc (i,j), $l_{ij}^k = 0$ otherwise. There are the auxiliary variables u_S^k and v_S^k ($S \subseteq V, \tilde{S} = V \setminus S$): if $u_S^k = 0$ then tour k does not have a sufficient number of arcs to form an illegal subcycle in S; if $v_S^k = 0$ then tour k traverses the cut-set $(S, \tilde{S})$. It is worth mentioning that the converse for these two conditional statements does not hold true, meaning that it is not possible to predict the values of u_S^k and v_S^k given the arcs of tour k S and $(S, \tilde{S})$.

(CARP)

$$Minimize \quad \sum_{k=1}^{M} \sum_{(i,j)\in E} c_{ij} x_{ij}^k \tag{2.1}$$

s.t.

$$\sum_{j\in N(i)} (x_{ji}^k - x_{ij}^k) = 0 \qquad (i \in V, k \in \{1,\dots,M\}) \tag{2.2}$$

$$x_{ij}^k \geqslant l_{ij}^k \qquad ((i,j) \in E_R, k \in \{1,\dots,M\}) \tag{2.3}$$

$$\sum_{k=1}^{M} (l_{ij}^k + l_{ji}^k) = 1 \qquad ((i,j) \in E_R) \tag{2.4}$$

$$\sum_{(i,j)\in E_R} d_{ij} l_{ij}^k \leqslant D \qquad (k \in \{1,\dots,M\}) \tag{2.5}$$

$$\left.\begin{array}{l} \displaystyle\sum_{(i,j)\in(S,S)} x_{ij}^k - |S|^2 u_S^k \leqslant |S| - 1 \\ \displaystyle\sum_{(i,j)\in(S,\tilde{S})} x_{ij}^k + v_S^k \geqslant 1 \\ u_S^k + v_S^k \leqslant 1 \end{array}\right\} \quad (S \subseteq V \setminus \{v_0\}, \tilde{S} = V \setminus S, k \in \{1,\dots,M\}) \tag{2.6}$$

$$x_{ij}^k \in \{0,1\} \qquad ((i,j) \in E, k \in \{1,\dots,M\}) \tag{2.7}$$

$$l_{ij}^k \in \{0,1\} \qquad ((i,j) \in E_R, k \in \{1,\dots,M\}) \tag{2.8}$$

$$u_S^k, v_S^k \in \{0,1\} \qquad (k \in \{1,\dots,M\}, S \subseteq V \setminus \{v_0\}) \tag{2.9}$$

The objective function (2.1) minimizes the solution total cost. Constraints (2.2) maintain routes continuity (every node must have equal indegree and outdegree). Constraints (2.3) state that serviced arcs must also be traversed; (2.4) force each required edge (represented by two arcs) to be serviced in a unique direction and by a single vehicle; (2.5) are the capacity constraints; and (2.6) eliminate illegal subcycles.

For a CARP illegal subcycle to occur, referring to some tour k, two necessary conditions must be satisfied under subset S, strictly containing the nodes of the subcycle (except the depot):

1. The number of tour arcs in S is at least $|S|$, otherwise there would not be enough arcs to form the subcycle.
$$\sum_{(i,j)\in(S,S)} x_{ij}^k \geqslant |S| \tag{2.10}$$

2. There are no arcs in the cut $(S, \tilde{S})$, meaning that the subcycle is disconnected from the tour.
$$\sum_{(i,j)\in(S,\tilde{S})} x_{ij}^k = 0 \tag{2.11}$$

Constraints (2.6), by using the auxiliary variables u_S^k and v_S^k, state that at most one of the two conditions (2.10,2.11) can be satisfied for any $S \subseteq V \setminus \{v_0\}$.

Constraints (2.2) are not valid for OCARP since open tours are feasible. Each directed open tour contains a source and a sink node, which can be detected by the difference between their indegree and outdegree (if the tour is closed (cycle), then any node can be the source and the sink). That said, valid continuity constraints for the OCARP are represented by (2.12,2.13,2.14).

$$\sum_{j\in N(i)} (x_{ij}^k - x_{ji}^k) \leqslant \alpha_i^k \qquad (i \in V, k \in \{1,\ldots,M\}) \tag{2.12}$$

$$\sum_{i\in V} \alpha_i^k \leqslant 1 \qquad (k \in \{1,\ldots,M\}) \tag{2.13}$$

$$\alpha_i^k \in \{0,1\} \qquad (i \in V, k \in \{1,\ldots,M\}) \tag{2.14}$$

The auxiliary variable $\alpha_i^k = 1$ if node i is the source of tour k, $\alpha_i^k = 0$ otherwise. While constraints (2.12) detect which nodes are sources, constraints (2.13) declare that each tour may contain at most one source. It should be noticed that there is no need to restrain the number of sink nodes, since this number is implicitly restricted by the graph degree balance.

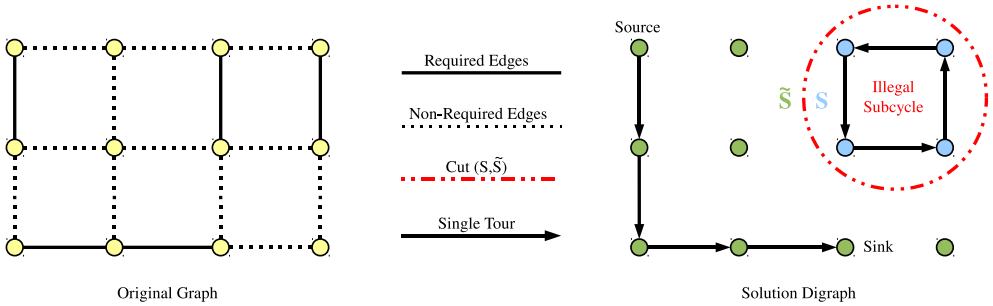


Figure 2.3: OCARP illegal subcycle.

Constraints (2.6) are based on the fact that a cycle is illegal for CARP if it is disconnected from the tour origin, the depot. This is not valid for OCARP, since any node can be the

origin. The OCARP subcycle in Figure 2.3, for example, is illegal because it is disconnected from the tour source. Surely, a disconnected cycle in any subset $S \subseteq V$ is valid for OCARP if S contains the tour source. Therefore, it is possible to extend CARP illegal subcycle necessary conditions (2.10,2.11) to OCARP, under subset S strictly containing the nodes of the subcycle, by comprising a third condition (2.15).

3. There is no source node in S.

$$\sum_{i \in S} \alpha_i^k = 0 \tag{2.15}$$

With this third condition, an OCARP adaptation of the subcycle elimination constraints is given in (2.16,2.17). The auxiliary variable $w_S^k = 0$ if tour k contains a source node in S ($S \subseteq V$). These constraints work by allowing at most two of three necessary conditions (2.10,2.11,2.15) to form an illegal OCARP subcycle on any subset $S \subseteq V$.

$$\left.\begin{array}{l} \displaystyle\sum_{(i,j) \in (S,S)} x_{ij}^k - |S|^2 u_S^k \leqslant |S| - 1 \\ \displaystyle\sum_{(i,j) \in (S,\tilde{S})} x_{ij}^k + v_S^k \geqslant 1 \\ \displaystyle\sum_{i \in S} \alpha_i^k + w_S^k \geqslant 1 \\ u_S^k + v_S^k + w_S^k \leqslant 2 \end{array}\right\} \quad (S \subseteq V, \tilde{S} = V \setminus S, k \in \{1, \ldots, M\}) \tag{2.16}$$

$$u_S^k, v_S^k, w_S^k \in \{0,1\} \qquad (k \in \{1, \ldots, M\}, S \subseteq V) \tag{2.17}$$

Replacing constraints (2.2,2.6,2.9) by (2.12,2.13,2.14,2.16,2.17) in the CARP model, leads to the following valid OCARP integer linear programming model.

(OCARP)

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

s.t.

$$\sum_{j\in N(i)} (x_{ij}^k - x_{ji}^k) \leqslant \alpha_i^k \qquad (i \in V, k \in \{1,\dots,M\})$$

$$\sum_{i\in V} \alpha_i^k \leqslant 1 \qquad (k \in \{1,\dots,M\})$$

$$x_{ij}^k \geqslant l_{ij}^k \qquad ((i,j) \in E_R, k \in \{1,\dots,M\})$$

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

$$\sum_{(i,j)\in E_R} d_{ij} l_{ij}^k \leqslant D \qquad (k \in \{1,\dots,M\})$$

$$\left.\begin{array}{l} \displaystyle\sum_{(i,j)\in(S,S)} x_{ij}^k - |S|^2 u_S^k \leqslant |S| - 1 \\ \displaystyle\sum_{(i,j)\in(S,\tilde{S})} x_{ij}^k + v_S^k \geqslant 1 \\ \displaystyle\sum_{i\in S} \alpha_i^k + w_S^k \geqslant 1 \\ u_S^k + v_S^k + w_S^k \leqslant 2 \end{array}\right\} \qquad (S \subseteq V, \tilde{S} = V \setminus S, k \in \{1,\dots,M\})$$

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

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

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

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

The binary constraints (2.7) under the decision variables x_{ij}^k may induce that an OCARP tour could traverse an arc only once, which is not true. However, in the next section, Property 4 shows that there always exists an optimal OCARP solution in which all arcs are traversed at most once.

2.3.1 OCARP properties

Some interesting OCARP properties are:

Property 1 *Given any OCARP instance with M vehicles and at least M required edges, there exists an optimal solution which uses all vehicles.*

Proof: Consider an optimal solution which uses less than M vehicles. Since there are more than $(M-1)$ required edges, at least one vehicle is traversing two or more required edges. In this case, we can split this vehicle tour into two, leaving at least one required edge in each tour, and the solution cost would remain the same. This procedure can be repeated until all vehicles attend a required edge, giving the property correctness. ∎

A consequence of this property is that instances where $M \geqslant |E_R|$ are trivially solvable by assigning one vehicle per required edge. Therefore, unlike CARP which admits M being a decision variable, OCARP has only meaning if M is a fixed parameter.

Property 2 *Given any OCARP instance, there exists an optimal solution in which all tours start and finish with required edges.*

Proof: Consider an optimal tour with a terminal non-required edge. If this edge is simply removed from the tour, feasibility is maintained. This process can be iterated until there are no longer any terminal non-required edges, giving the property correctness. ∎

A consequence of this property is that if there is a closed tour belonging to an optimal solution, then all of its edges are required (excluding non-required edges with zero cost). Otherwise, it would be possible to reduce the solution cost by removing one of the non-required edges from the closed tour, turning it into an open tour, still feasible.

Property 3 *Given any OCARP instance, there exists an optimal solution in which all tours are formed by alternating required edges with shortest paths linking one required edge to another.*

Proof: In the case where a solution is formed by two adjacent required edges, it can be considered that there is an *empty path* between these edges, which configures their shortest path. If there is a tour with two required edges connected by a path of length greater than the shortest path, then it would be possible to simply replace this suboptimal path by the shortest one, without loss of feasibility. ∎

Property 4 *There is an optimal OCARP solution which does not traverse an edge twice in the same direction by the same vehicle.*

Proof: Welz (1994) demonstrated this same property for CARP within a two-step proof. First, he has shown that the inequality $x_{ij}^k + x_{ji}^k \leqslant 2$ $(k \in \{1, \ldots, M\}, (i,j) \in E)$ is valid for an optimal CARP solution, otherwise there would exist a deadheading cycle that could be removed without loss of feasibility. For the same reasons, this inequality remains valid for OCARP. The second step was to show that the CARP solution induced graph is Eulerian, since all nodes have even degree. This imply the existence of an Eulerian tour, which traverses each edge exactly once. As for OCARP, only the source and sink nodes may have odd degrees, meaning that the OCARP solution induced graph is either Eulerian or semi-Eulerian. This last case implies the existence of an Eulerian path which also traverses each edge exactly once. ∎

2.4 Complexity Study

Through a polynomial reduction CARP $\leqslant_p$ OCARP, this section intends to prove that the latter is at least as hard as the former. Furthermore, knowing if a problem, such as OCARP, is NP-hard, justifies the employment of heuristics to find good quality solutions, since exact methods are likely inadequate to solve realistic sized instances.

Theorem 1 *CARP can be polynomially reduced into OCARP.*

Starting from any CARP instance $G(V, E)$, with M vehicles and a depot node v_0, add $2M$ dummy nodes (V_0) and $2M$ dummy required edges (E_{R0}), with relatively high costs for any $e \in E_{R0}$, $c(e) = C \gg SP^{\max}$ (where $SP^{\max}$ is the cost of the maximum shortest path between any two nodes of G), and demands for any $e \in E_{R0}$, $d(e) = \delta > 0$ (any positive value suffices), linking the dummy nodes to v_0. High costs are attributed to dummy required edges so that an optimal solution traverses each one of them a single time. Finally, the vehicles capacities should be increased to $D + 2\delta$.

A new graph $G_1(V_1, E_1)$ is then formed, where $V_1 = V \cup V_0$ and $E_1 = E \cup E_{R0}$. This transformation has complexity $O(M)$, and assuming $M < |E_R|$ (Property 1), the reduction at hand is linear with respect to the size of G. Figure 2.4 shows an example of the reduction just described.

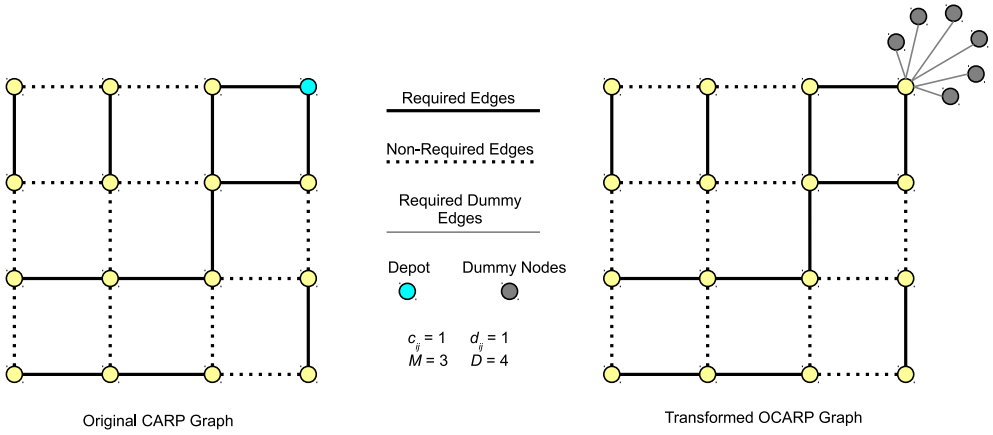


Figure 2.4: CARP graph G transformed into OCARP graph G_1.

Consider $P^*_{carp}(G)$ and $P^*_{ocarp}(G_1)$ the induced graphs by CARP and OCARP optimal solutions in G and G_1, respectively. The relationship between them are described in the following:

1. $P^*_{carp}(G) = P^*_{ocarp}(G_1 \setminus V_0)$, i.e., for every OCARP optimal solution in G_1, there is a corresponding CARP solution in G that can be obtained by removing the set of dummy nodes and edges.

2. $c(P^*_{carp}(G)) = c(P^*_{ocarp}(G_1)) - 2MC$, i.e., the CARP optimal solution cost is equal to the corresponding OCARP optimal solution cost subtracting the costs of the dummy edges.

To prove Theorem 1 it will be demonstrated that in an OCARP optimal solution all tours traverse exactly two dummy edges by starting and ending at distinct dummy nodes (Lemma 1.1), and after extracting all dummy edges from this solution, a feasible CARP solution emerge (Corollary 1.1), i.e., a solution which traverses all required edges, attends the vehicles capacities (Lemma 1.2), and is formed by closed tours which visit the depot (Lemma 1.3). The proof is complete when, beyond feasibility, optimality is also verified (Lemma 1.4). For the following, consider $P_{ocarp}(G)$ as the induced graph from an OCARP optimal solution after removing dummy nodes and edges, i.e., $P_{ocarp}(G) = P^*_{ocarp}(G_1 \setminus V_0)$.

Lemma 1.1 *All tours from $P^*_{ocarp}(G_1)$ traverse exactly two distinct dummy edges.*

Proof: Since the transformed OCARP instance has M vehicles and $2M$ dummy edges, then two possibilities arise: (1) all tours traverse exactly two dummy edges or (2) at least one tour traverses more than two dummy edges. In the first case, two dummy edges can be traversed by a single tour without the requirement of revisiting any dummy edge. This can be accomplished if and only if the tour starts at a dummy node and ends at another dummy node. In the second case, if a tour visits more than two dummy edges, at least one of them will necessarily be revisited, which would result in a needless cost increase, given the high cost of traversing a dummy edge. Therefore the second possibility could not occur in an optimal solution. ∎

Lemma 1.2 *All required edges from G are serviced in $P_{ocarp}(G)$ without overloading any vehicle capacity.*

Proof: Since $P_{ocarp}(G)$ is formed by extracting two dummy edges with demand δ from each $P^*_{ocarp}(G_1)$ tour, then naturally $P_{ocarp}(G)$ traverses all required edges from G. It should be noticed that the vehicles remaining capacities are exactly the same for both solutions, since the original vehicles have 2δ less capacity than the upgraded vehicles. ∎

Lemma 1.3 *$P_{ocarp}(G)$ is formed by a set of closed tours that visit the depot v_0.*

Proof: It was claimed by the proof of Lemma 1.1 that all $P^*_{ocarp}(G_1)$ tours have two distinct dummy edges as terminals. Since all dummy edges are linked to G by the depot v_0, removing them transforms the OCARP tours into cycles that visit v_0. ∎

Corollary 1.1 *$P_{ocarp}(G)$ is feasible for the CARP.*

Note: As an immediate consequence of the previous lemmas, $P_{ocarp}(G)$ does not overload any vehicle capacity, traverses all the required edges, and it is formed by closed tours that visit the depot v_0.

Lemma 1.4 *$P_{ocarp}(G)$ represents an optimal CARP solution.*

Proof: The OCARP optimal solution cost for G_1 can be decomposed as $c(P^*_{ocarp}(G_1)) = c(P^*_{ocarp}(G_1 \setminus V_0)) + 2MC$, where $2MC$ is a lower bound from Lemma 1.1. It follows that $c(P^*_{ocarp}(G_1 \setminus V_0))$ must be minimum, and so is $c(P_{ocarp}(G))$, according to the hypothesis $P_{ocarp}(G) = P^*_{ocarp}(G_1 \setminus V_0)$. ∎

Corollary 1.2 *OCARP is NP-hard.*

Note: CARP is an NP-hard problem (Golden and Wong 1981) which is polynomially reducible to OCARP.

2.5 Solution Strategy

This section reveals an inverse polynomial reduction of the one proposed in Section 2.4, i.e., OCARP $\leqslant_p$ CARP. This is relevant, once it admits solving OCARP through CARP algorithms.

Theorem 2 *The OCARP can be polynomially reduced into CARP.*

Consider an OCARP instance $G(V, E)$ with M vehicles. Add a dummy depot v_0 and a set of dummy non-required edges (E_0), with relatively high costs for any $e \in E_0$, $c(e) = C \gg SP^{\max}$, and demands for any $e \in E_0$, $d(e) = 0$, linking v_0 to every node in G. The costs of the dummy non-required edges are made high so that an optimal solution traverses these edges strictly when the vehicle is leaving and returning to the depot. A new graph $G_1(V_1, E_1)$ is then formed, where $V_1 = V \cup \{v_0\}$ and $E_1 = E \cup E_0$. This reduction has linear complexity $O(|V|)$. Figure 2.5 gives an example of the proposed reduction.

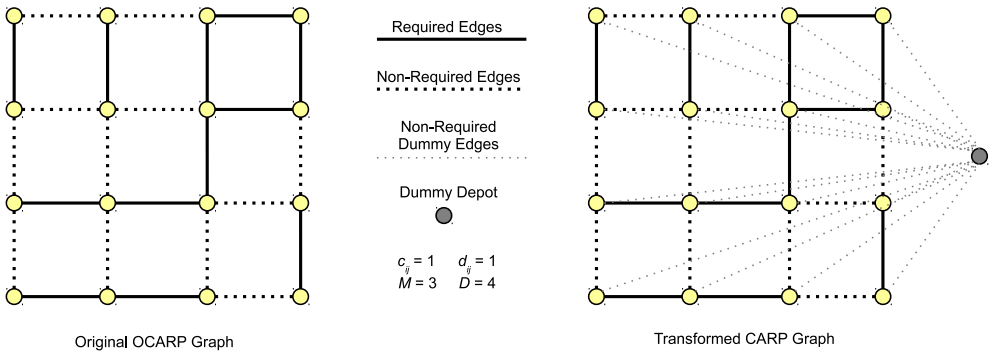


Figure 2.5: OCARP graph G transformed into CARP graph G_1.

Consider $P^*_{ocarp}(G)$ and $P^*_{carp}(G_1)$ the induced graphs by OCARP and CARP optimal solutions in G and G_1, respectively. The relationship between them are described in the following:

1. $P^*_{ocarp}(G) = P^*_{carp}(G_1) \setminus \{v_0\}$, i.e., for every CARP optimal solution in G_1, there is a corresponding OCARP solution in G that can be obtained by removing the dummy depot.

2. $c(P^*_{ocarp}(G)) = c(P^*_{carp}(G_1)) - 2MC$, i.e., the OCARP optimal solution cost is equal to the corresponding CARP optimal solution cost subtracting twice the number of vehicles times the cost of a dummy edge.

Proof: Consider $P_{carp}(G)$ as the induced graph from a CARP optimal solution after removing the dummy depot, i.e., $P_{carp}(G) = P^*_{carp}(G_1 \setminus \{v_0\})$. Given the simplicity of this transformation, it will be demonstrated that $P_{carp}(G)$ represents a feasible OCARP solution and concluded that it is also optimal.

An OCARP solution is considered feasible if it is formed by tours (open or not) that traverse all required edges without overloading the capacity of any vehicle. The induced graph $P_{carp}(G)$ is formed by extracting only dummy edges from $P^*_{carp}(G_1)$, therefore $P_{carp}(G)$ tours still traverse all the original required edges while respecting the capacity constraints. In addition, since the depot is linked to V only through dummy edges, then every $P^*_{carp}(G_1)$ tour contains exactly two dummy edges, and not more given their high costs. If two adjacent edges are extracted from a cycle, as in every $P^*_{carp}(G_1)$ tour to generate $P_{carp}(G)$, the result will be an OCARP tour. This concludes $P_{carp}(G)$ feasibility. Now about $P_{carp}(G)$ being optimal, suppose there exists $P^*_{ocarp}(G)$, such that $c(P^*_{ocarp}(G)) < c(P_{carp}(G))$. Then by simply adding two dummy edges, with cost C, at both the beginning and the end of each tour, it would be possible to transform $P^*_{ocarp}(G)$ into an induced CARP solution graph, $P_{carp}(G_1)$, with cost $c(P_{carp}(G_1)) = c(P^*_{ocarp}(G)) + 2MC < c(P_{carp}(G)) + 2MC = c(P^*_{carp}(G_1))$, which would be a contradiction. ∎

From Property 2, Theorem 2 remains valid even if the set E_0 is the set of dummy edges linking v_0 with only the nodes incident to required edges. This gives a simplification for the proposed reduction.

Corollary 2.1 *OCARP and CARP are polynomially equivalent.*

Note: Since both polynomial reductions OCARP $\leqslant_p$ CARP and CARP $\leqslant_p$ OCARP exist, then the complexity of solving these problems differs from a polynomial function.

2.6 Final Remarks

This chapter has introduced the open capacitated arc routing problem (OCARP), a combinatorial optimization problem of theoretical and practical interest belonging to the family of arc routing problems. At least two applications from literature can be modeled as an OCARP, the Meter Reader Routing Problem and the Cutting Path Determination Problem. An integer linear programming model was proposed, followed by some interesting properties of the problem. The

OCARP complexity has been proven NP-hard through a polynomial reduction of the CARP. A reverse reduction was also proposed, showing that algorithms for the CARP could also be suitable for the OCARP, as long as they assume a fixed number of vehicles.

Chapter 3

Heuristics Approaches

3.1 Introduction

This chapter describes a GRASP with path-relinking (PR) developed to solve both the capacitated arc routing problem (CARP) and the Open CARP (OCARP). Section 3.2 reviews the general structure of a Greedy Randomized Adaptive Search Procedure (GRASP) and gives a thorough description of the proposed GRASP to solve the CARP, including the constructive phase, cost-demand edge-selection rule, reactive adjustment, local search, and the statistical solution filtering. To strengthen the search for high-quality solutions, a path-relinking was coupled to the GRASP, mirroring several successful experiences in literature, which are referred to in Section 3.3. Still on this section, the detailed *modus operandi* of the proposed path-relinking is provided, with special attention to the metric used to measure the distance between a pair of solutions, the operator used to progressively transform an initial solution towards a guiding solution, the admission policy for the elite solutions pool, and the way how GRASP and path-relinking were jointed. Computational experiments were conducted for both OCARP and CARP instances and the results are presented in Section 3.4. The final remarks close this chapter in Section 3.5.

3.2 Greedy Randomized Adaptive Search Procedure

A Greedy Randomized Adaptive Search Procedure (GRASP) (Feo and Resende 1995) is a memoryless multi-start metaheuristic, where each iteration consists of two phases:

- *construction phase*: initial solutions are built, one element at a time, with a greedy randomized heuristic. At each construction iteration, the next element to be added is determined by ordering all elements in a candidate list with respect to a greedy function that estimates the benefit of selecting each element. The probabilistic component of a GRASP

is characterized by randomly choosing one of the best candidates in the list, not always the top best.

- *local search*: the neighborhood of the initial solutions is explored. The solutions generated by a GRASP construction are not guaranteed to be locally optimal. Hence, it is almost always beneficial to apply a local search to attempt to improve each constructed solution. A local search algorithm works in an iterative fashion by successively replacing the current solution by a better one from its neighborhood. It terminates when there is no better solution in the neighborhood.

The best solution over all GRASP iterations is returned as the result. Success for a local search algorithm depends on an efficient neighborhood search technique and a good starting solution provided by the construction phase. A GRASP can be seen as a metaheuristic which captures good features of pure greedy algorithms (intensification) and also of random construction procedures (diversification).

Competitive results have been reported in literature using GRASP-based metaheuristics in different routing problems such as the vehicle routing problem (Prins 2009), the truck and trailer routing problem (Villegas, Prins, Prodhon, Medaglia and Velasco 2011), and the CARP-TW (Labadi et al. 2008). According to Resende and Ribeiro (2005), the performance of GRASP can be enhanced by using reactive parameter tuning mechanisms, multiple neighborhoods, and path-relinking. These features were incorporated in the proposed GRASP, whose components and the general structure follows.

3.2.1 Constructive Phase

The GRASP constructive phase was developed based on Santos et al. (2009) path-scanning heuristic with ellipse rule. This heuristic was adapted to include a restricted candidate list, responsible for holding a set of good and diversified elements to embody the solution under construction. The parameters which control the balance of goodness and diversity have their values reactively adjusted according to the average solution cost these values provide.

3.2.1.1 Path-Scanning Heuristics

The path-scanning heuristics developed for CARP construct each solution by adding to a path starting at the depot, one required edge at a time. To determine the next edge to add, an *edge-selection rule* $\psi(v_l, e)$ is used (3.1), where $e = [v_i, v_j]$ is a candidate for the next required edge to be visited starting from v_i to v_j, v_l is the last node visited by the tour, and SP represents the shortest path cost between two nodes. Every unserviced required edge e whose demand d_{ij} is less than the vehicle remaining capacity is a possible candidate, and the heuristic will select the one which minimizes $\psi(v_l, e)$.

$$\psi(v_l, e) = \min(SP(v_l, v_i), SP(v_l, v_j)) \tag{3.1}$$

There are cases where more than one candidate edge minimizes $\psi(v_l, e)$, specially when they are incident to v_l. In these situations, a tie breaking rule is considered, and this rule represents the major difference between CARP path-scanning heuristics. Golden and Wong (1981) have used five criteria to break ties:

1. minimize $\frac{c_{ij}}{d_{ij}}$.
2. maximize $\frac{c_{ij}}{d_{ij}}$.
3. minimize the cost back to depot.
4. maximize the cost back to depot.
5. criterion 3 if the vehicle has used more than half of its capacity; criterion 4, otherwise.

A problem instance is solved five times, using a different criterion each time, and the best of the five solutions is taken. Pearn (1989) modified this approach by selecting one of the five criteria at random, with equal probability, whenever a tie occurs. Belenguer et al. (2006) simplified the tie breaking rule by randomly selecting one tied edge. This was adopted by Santos et al. (2009), in their path-scanning heuristic with ellipse rule, explained in the following paragraphs.

Recently, Santos et al. (2009) developed a path-scanning heuristic which makes use of an ellipse rule. When a vehicle is near its full capacity, this rule enforces the vehicle to service only edges inside an ellipse containing the shortest path between the last serviced edge and the depot, following the rationale that a heavily loaded vehicle should stay closer to the depot in order to reduce its returning cost. These authors define $ned = |E_R|$, td the total demand to be serviced, tc the total cost from edges with positive demand, v_0 the depot node, $[v_h, v_l]$ the last serviced edge on the tour, and β a real parameter. If the remaining vehicle capacity is less than or equal to $\beta\,(td/ned)$, then the next edge to be serviced $[v_i, v_j]$ must be the nearest edge to $[v_h, v_l]$ ($v_l = v_i$, if the edges are adjacent) satisfying the condition:

$$SP(v_l, v_i) + c_{ij} + SP(v_j, v_0) \leqslant \frac{tc}{ned} + SP(v_l, v_0) \tag{3.2}$$

If no candidate edge satisfies (3.2) then the vehicle returns to the depot. Through the ellipse rule, the authors obtained 44% reduction in overall average deviation from lower bounds with little or no increase in solution time, compared to previous path-scanning heuristics.

Solving OCARP using the path-scanning heuristic with ellipse rule was attempted. This however was not successful given that this heuristic was developed for the CARP version where

the number of vehicles is a decision variable (Section 1.3), and in OCARP the number of vehicles is a fixed parameter (Property 1). In fact, the OCARP computational experiments (Section 3.4.1) reveal that this heuristic has failed to find even a single feasible solution for some instances. This can be explained by the fact that underneath the tour cost optimization problem relies a bin-packing subproblem of assigning the required edges to the vehicles, given their limited amount and capacity. By relaxing the number of vehicles available, common procedure among the latest CARP algorithms, this subproblem cease to exist. As for the OCARP, the problem of finding a feasible solution is already NP-Hard. Therefore, efficient OCARP heuristics should take both of these optimization problems into consideration, and for this reason a reactive path-scanning heuristic with ellipse rule is proposed.

3.2.1.2 Cost-demand edge-selection rule

One well-known bin-packing algorithm (Martello and Toth 1990) is the *first fit decreasing* (FFD) that operates by sorting the elements in a decreasing order of their demands, and then inserting each element into the first bin with sufficient remaining capacity. This concept of prioritizing the elements of higher demands inspired a new edge-selection rule $\tilde{\psi}(e)$ (3.3), redesigned in order to consider not only the shortest path cost, but also the demands to be collected from the candidate edges. These two objectives are weighted through parameter $\gamma \in [0,1]$, $e = [v_i, v_j]$ is a candidate edge, v_l is the last node visited by the tour, $SP^{\max}$ is the length of the maximum shortest path between any two nodes of the graph and $d^{\max}$ is the maximum edge demand.

$$\tilde{\psi}(e) = \gamma \frac{\min(SP(v_l, v_i), SP(v_l, v_j))}{SP^{\max}} + (1-\gamma)(1 - \frac{d_{ij}}{d^{\max}}) \tag{3.3}$$

It should be noticed that if $\gamma = 1$, then this edge-selection rule becomes equivalent to (3.1) (except by the normalization factor $SP^{\max}$). When $\gamma = 0$, the edge-selection rule works as the FFD heuristic, in an attempt to use the vehicle capacity more efficiently. The parameter γ is self-tuned through a metaparameter $g \in [0.5, 1)$, according to the feasibility of the previous solutions obtained by the heuristic.

Let γ_k and g_k be the values of these parameters at iteration k. Table 3.1 gives the tuning mechanism for these parameters. The values of γ and g are set so that initially $\tilde{\psi}(e)$ works such as in the original heuristic. When infeasible solutions are obtained in sequel, an exponentially fast adjustment of γ is performed by successive multiplications with g. During this adjustment, when a feasible solution appears, γ is readjusted in favor of solution cost and g is taken closer to 1 making future adjustments increasingly smoother (Figure 3.1).

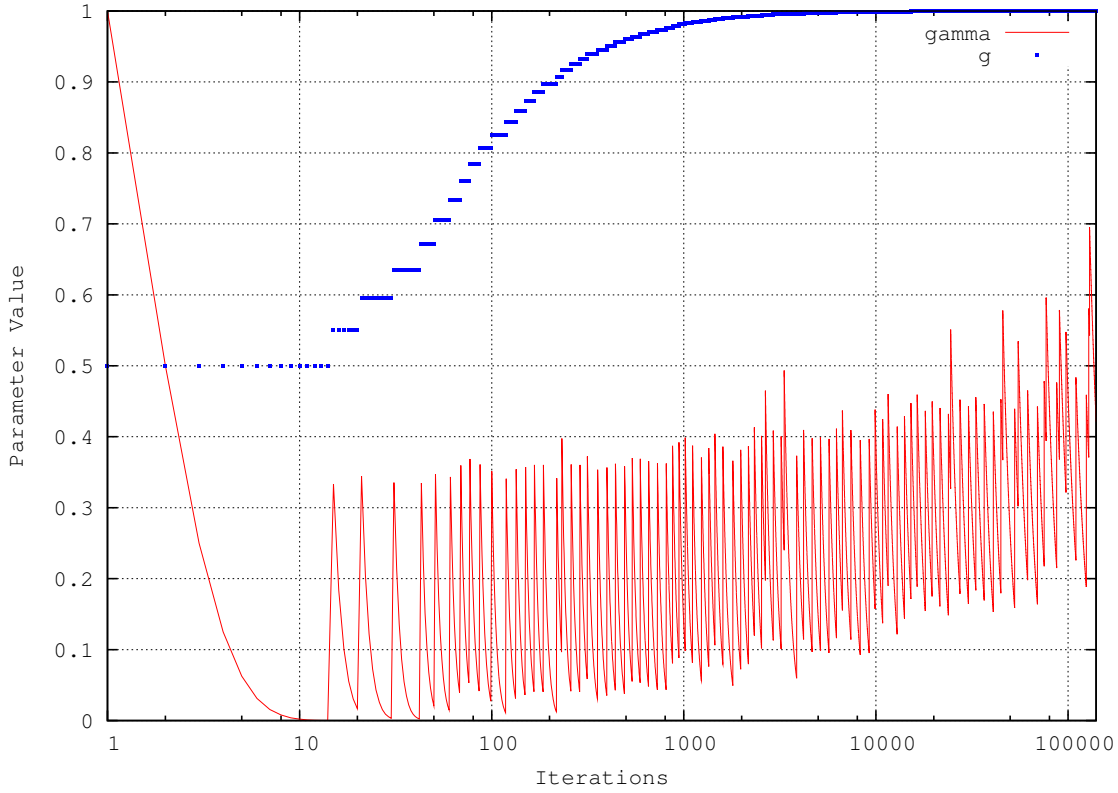


Figure 3.1: Evolution of the parameter γ for instance *oegl-s4-C*.

3.2.1.3 Constructive Heuristic

The path-scanning heuristic with ellipse rule was adapted into a GRASP constructive heuristic by replacing the edge-selection rule with the restricted candidate list (RCL) (3.4), which is filled with the best candidate edges according to the cost-demand edge-selection rule (3.3), limited by a threshold parameter α.

$$e \in RCL \Rightarrow \tilde{\psi}(v_l, e) \leq \alpha(\tilde{\psi}_{\max} - \tilde{\psi}_{\min}) + \tilde{\psi}_{\min} \tag{3.4}$$

where $\tilde{\psi}_{\min} = \min_e \tilde{\psi}(v_l, e)$ and $\tilde{\psi}_{\max} = \max_e \tilde{\psi}(v_l, e)$.

The constructive heuristic pseudo-code is presented in Algorithm 1. The computational complexity of this heuristic in a worst-case scenario is bounded by $O(|E_R|^2)$, since the main loop (line 7) iterates $|E_R|$ times and at each iteration the heuristic tries to find the best among $|E_R|$ positions for a required edge. Also, the evaluation of the edge-selection rule $\tilde{\psi}$ is pre-computed for all required edges ($O(|E_R|^2)$).

Algorithm 1 constructivePhase$(G, D, \alpha, \beta, \gamma)$

Input: G – instance graph, D – vehicle capacity, α – RCL parameter, β – ellipse rule parameter, γ – cost-demand edge-selection rule parameter
Output: S – feasible CARP solution

$td = \sum_{e \in E_R} d(e)$, $ned = |E_R|$
$S \leftarrow \varnothing$
$t \leftarrow 1$ // tour index
$tour_t \leftarrow \varnothing$ // ordered set of edges representing a tour
$rvc \leftarrow D$ // remaining vehicle capacity
$v_l = 0$ // starting tour on depot (node 0)
for $(i = 1$ **to** $ned)$ **do**
  $RCL \leftarrow \varnothing$
  $\tilde{\psi}_{\min} \leftarrow \min_{e \in E_R \setminus S} \tilde{\psi}(v_l, e)$, $\tilde{\psi}_{\max} \leftarrow \max_{e \in E_R \setminus S} \tilde{\psi}(v_l, e)$
  for $(\forall e \in E_R \setminus S)$ **do**
    if $\tilde{\psi}(v_l, e) \leq \alpha(\tilde{\psi}_{\max} - \tilde{\psi}_{\min}) + \tilde{\psi}_{\min}$ **then**
      $RCL \leftarrow RCL \cup \{e\}$
    end if
  end for
  if $rvc \leqslant \beta \dfrac{td}{ned}$ **then**
    $RCL \leftarrow RCL \setminus \{$edges violating ellipse rule (3.2)$\}$
  end if
  if $RCL = \varnothing$ **then**
    // starting new tour
    $S \leftarrow S \cup \{tour_t\}$
    $t \leftarrow t + 1$
    $tour_t \leftarrow \varnothing$
    $rvc \leftarrow D$
    $v_l = 0$
  else
    $e = [v_i, v_j] \leftarrow$ **randomEdge**(RCL) // randomly selects an edge from set RCL
    $tour_t \leftarrow tour_t \cup \{e\}$
    $rvc \leftarrow rvc - d(e)$
    $v_l \leftarrow v_j$
  end if
end for
return S

Table 3.1: Tuning scheme for the parameters γ and g.

Initial values	Feasible solution at iteration k	Infeasible solution at iteration k
$\gamma_1 = 1$	$\gamma_{(k+1)} = \frac{2\gamma_k+1}{3}$	$\gamma_{(k+1)} = g_k\gamma_k$
$g_1 = 0.5$	$g_{(k+1)} = \frac{9g_k+1}{10}$	$g_{(k+1)} = g_k$

3.2.1.4 Reactive Parameters

The proposed constructive heuristic has two parameters, α and β, that directly affect the heuristic performance and ergo must be properly adjusted. The RCL parameter α controls the greediness of the candidate edge selection ($\alpha = 0$ pure greedy; $\alpha = 1$ pure random). The ellipse rule parameter β is responsible for controlling the ellipse shape, or in other words, how active is this rule ($\beta = 0$, inactive; $0 < \beta < D(ned/td)$, it depends on the vehicle remaining capacity; $\beta \geqslant D(ned/td)$, always active).

A reactive parameter adjustment, based on the work of Prais and Ribeiro (2000), was implemented to select the values for α and β at each iteration of the constructive heuristic from a discrete set of possible values. Let $\Pi = \{\pi_1, \ldots, \pi_m\}$ be the set of possible values for a given parameter π. The probabilities associated with the choice of each value are all initially made equal to $p_i = 1/m$, $(i = 1, \ldots, m)$. Furthermore, let c_{best} be the cost of the incumbent best solution and $\overline{c}_i$ the average cost of all solutions obtained by using $\pi = \pi_i$. In Prais and Ribeiro (2000), the selection probabilities are periodically reevaluated through (3.5).

$$p_i = \frac{q_i}{\sum_{j=1}^{n} q_j} \qquad q_i = \frac{c_{best}}{\overline{c}_i} \qquad (i = 1, \ldots, m) \tag{3.5}$$

It is intended that the values of π_i producing good solutions on average will generate larger q_i, which in turn increases the probabilities p_i associated to them. However, it turned out that through equation (3.5), the probabilities are not expressing well the relative differences between their associated average costs. For most CARP instances, these probabilities would hardly differ in more than one percent. An alternative reactive scheme is proposed (3.6), which preserves the main idea of the previous one, but amplifies the effect of the average costs in their associated probabilities.

$$q_i = 1 - \left(\frac{m-1}{m}\right)\left(\frac{\overline{c}_i - \overline{c}_{\min}}{\overline{c}_{\max} - \overline{c}_{\min}}\right) \qquad (i = 1, \ldots, m) \tag{3.6}$$

where $\overline{c}_{\min}$ and $\overline{c}_{\max}$ are the minimum and maximum average costs, and p_i is calculated the same way as before (3.5).

Let $p_{\max}$ and $p_{\min}$ be the probabilities associated to the best ($\overline{c}_i = \overline{c}_{\min}$) and worst ($\overline{c}_i = \overline{c}_{\max}$) parameters, respectively. Then, by means of equation (3.6), $p_{\max} = mp_{\min}$, giving a much better

probability distribution, in the sense that the best parameter will have m times better chance to be chosen than the worst parameter.

Algorithm 2 describes the pseudo-code for the reactive parameter adjustment. The main loop (2) iterates m times, where m is a constant referring to the number of possible values each parameter has. This leads to a computational complexity bounded by a constant $O(1)$.

Algorithm 2 reactiveChoice(C, N)

Input: $C = \{\overline{c}_1, \ldots, \overline{c}_m\}$ - average solution costs for each parameter value
$N = \{n_1, \ldots, n_m\}$ - number of solutions obtained for each parameter value
Output: $i \in \{1, \ldots, m\}$ - index of the parameter value

$q_{sum} \leftarrow 0, \quad c_{\min} \leftarrow \min_i \overline{c}_i, \quad c_{\max} \leftarrow \max_i \overline{c}_i$
for $(i = 1$ **to** $m)$ **do**
  $q_i \leftarrow 1$
  if $n_i > 0$ **and** $c_{\min} \neq c_{\max}$ **then**
    $q_i \leftarrow q_i - \left(\frac{m-1}{m}\right)\left(\frac{\overline{c}_i - \overline{c}_{\min}}{\overline{c}_{\max} - \overline{c}_{\min}}\right)$
  end if
  $q_{sum} \leftarrow q_{sum} + q_i$
end for
$n_{rand} \leftarrow$ **randomNumber**$(0, 1)$ // real random number between $[0, 1]$
$p_{sum} \leftarrow 0$
for $(i = 1$ **to** $m)$ **do**
  $p_{sum} \leftarrow p_{sum} + \frac{q_i}{q_{sum}}$
  if $n_{rand} \leqslant p_{sum}$ **then**
    break for
  end if
end for
return i

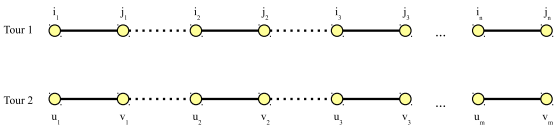
3.2.2 Local Search Phase

After an initial solution is generated by the constructive phase, the local search tries to improve it by exploring neighbor solutions defined by a set of moves which operate on the required edges order and orientation. The solutions are encoded as a list of required edges with implicit shortest paths between them, following the ideas in Lacomme et al. (2004) and Beullens et al. (2003).

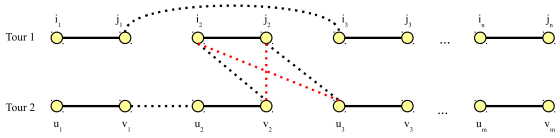
3.2.2.1 Neighborhood Moves

Four types of moves, used by the most successful CARP metaheuristics, were considered. These neighborhood moves have a good trade-off between solution improvement and computational effort. Also, they are applicable for inter-routes and intra-routes, as shown by the examples of Figure 3.2.

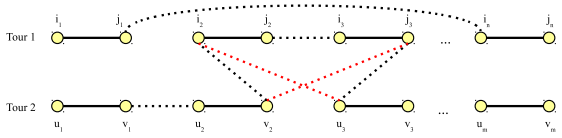
- *single-insertion + reversal*: a required edge is removed from its current position and placed in another one, reversed or not (Figure 3.2(b)). A reversal means that the edge will be traversed in the opposite direction as from before.
- *double-insertion + reversal*: two adjacent required edges are removed from their current positions and placed in another ones, possibly reversing both of them (Figure 3.2(c)).
- *swap + reversal*: two required edges switch their current positions, possibly reversing one or both required edges (Figure 3.2(d)).
- *block-insertion*: a block of adjacent required edges are removed from its current position and placed in another one (Figure 3.2(e)).



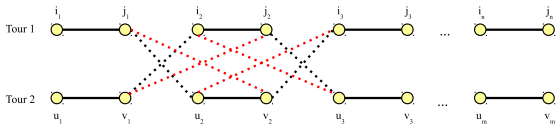
(a) Original tours.



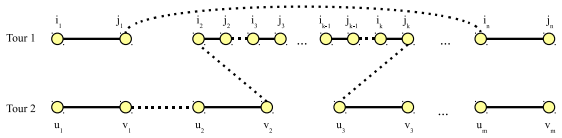
(b) Single-insertion: edge (i_2, j_2) is placed between edges (u_2, v_2) and (u_3, v_3).



(c) Double-insertion: edges (i_2, j_2) and (i_3, j_3) are placed between edges (u_2, v_2) and (u_3, v_3).



(d) Swap: edges (i_2, j_2) and (u_2, v_2) switch positions.



(e) Block-insertion: edges (i_2, j_2), (i_3, j_3), ..., (i_k, j_k) are placed between edges (u_2, v_2) and (u_3, v_3).

Figure 3.2: Neighborhood moves with possible reversals (in red).

The local search phase uses the first three moves, while block-insertion is used as the path-relinking operator (Section 3.3.2). To achieve a local optimal solution, the *best improvement* scheme was adopted, where the selected move in each local search iteration is the one which achieves the greatest reduction in solution cost, preserving feasibility, i.e., the vehicle capacity constraints (Algorithm 3).

The local search complexity is defined by the size of the neighborhood and the complexity of evaluating each possible move. The size of all four neighborhoods is $O(|E_R|^2)$, because they can be seen as selecting the best possible 2-combination of required edges. To evaluate a move from

any of the four proposed neighborhoods requires a constant computational effort $O(1)$, because all shortest paths between required edges are previously computed in a preprocessing phase. Therefore, the best-improvement move can be determined for any solution with a computational complexity of $O(|E_R|^2)$.

Algorithm 3 localSearch(S)

Input: S - CARP solution
Output: S_{ls} - locally optimal CARP solution
1: $c_{last} \leftarrow \infty$, $S_{ls} \leftarrow S$
2: **while** (**cost**$(S_{ls}) < c_{last}$) **do**
3: $c_{last} \leftarrow$ **cost**(S_{ls})
4: $S_{si} \leftarrow$ **applyBestSingleInsert**(S_{ls})
5: $S_{di} \leftarrow$ **applyBestDoubleInsert**(S_{ls})
6: $S_{sw} \leftarrow$ **applyBestSwap**(S_{ls})
7: **if cost**$(S_{si}) <$ **cost**(S_{ls}) **then**
8: $S_{ls} \leftarrow S_{si}$
9: **end if**
10: **if cost**$(S_{di}) <$ **cost**(S_{ls}) **then**
11: $S_{ls} \leftarrow S_{di}$
12: **end if**
13: **if cost**$(S_{sw}) <$ **cost**(S_{ls}) **then**
14: $S_{ls} \leftarrow S_{sw}$
15: **end if**
16: **end while**
17: **return** S_{ls}

3.2.2.2 Infeasible Local Search

A diversification strategy was incorporated into local search by allowing capacity infeasible moves. Since an integer linear programming problem optimal solution must reside on the boundary of the feasible convex hull, then this optimal solution is adjacent to the infeasible space, making search techniques which explore the infeasible solution space an interesting field of investigation.

Glover (2007) draws some light on the importance of exploring feasible/infeasible boundaries in the solution space of combinatorial optimization problems. This work proposes an *infeasible local search*, which receives as input a feasible solution, and probably returns a better cost solution, infeasible with respect to the vehicles capacities (Algorithm 4). It works mostly like the normal local search, except for two differences: (i) at each iteration, the search ignores if a move violates a vehicle capacity; (ii) the search is interrupted after a given number of infeasible moves, preventing the solution going too deep in the infeasible space, and possibly harming its way back.

The *infeasible local search* provides these infeasible solutions as initial solutions to the path-relinking, with means to explore paths traversing the infeasible/feasible boundaries of the so-

lution space. The computational complexity of the infeasible local search uses the same neighborhoods from the regular local search, therefore the computational complexity of these two heuristics are the same ($O(|E_R|^2)$).

Algorithm 4 infeasibleLocalSearch(S, n_{inf})

Input: S - CARP solution
Output: S_{ils} - CARP solution, likely infeasible
1: $c_{last} \leftarrow \infty$, $S_{ls} \leftarrow$**localSearch**(S), $S_{ils} \leftarrow S_{ls}$
2: **for** $(i = 1$ **to** $n_{inf})$ **do**
3: $c_{last} \leftarrow$ **cost**(S_{ils})
4: $S_{isi} \leftarrow$ **applyBestInfeasibleSingleInsert**(S_{ils})
5: $S_{idi} \leftarrow$ **applyBestInfeasibleDoubleInsert**(S_{ils})
6: $S_{isw} \leftarrow$ **applyBestInfeasibleSwap**(S_{ils})
7: **if cost**$(S_{isi}) <$ **cost**(S_{ils}) **then**
8: $S_{ils} \leftarrow S_{isi}$
9: **end if**
10: **if cost**$(S_{idi}) <$ **cost**(S_{ils}) **then**
11: $S_{ils} \leftarrow S_{idi}$
12: **end if**
13: **if cost**$(S_{isw}) <$ **cost**(S_{ils}) **then**
14: $S_{ils} \leftarrow S_{isw}$
15: **end if**
16: **if** (**cost**$(S_{ils}) \geqslant c_{last}$) **then**
17: **break**
18: **end if**
19: **end for**
20: **return** S_{ils}

3.2.3 Statistical Filter

In general, good solutions uncovered by local search comes from good initial solutions found in the constructive phase. In addition, local search is often the most demanding phase of a GRASP in terms of computational effort. Therefore, it seems unwise and computationally expensive to explore the neighborhood of all initial solutions, including low-quality ones. Instead, poor quality initial solutions should be rather discarded, and with the computational time saved, other more promising solution space regions can be explored. This strategy is called GRASP filtering (Feo, Resende and Smith 1994). Prais and Ribeiro (2000) propose a filtering by storing the average value (μ) of the ratio between initial (c_{ini}) and local search (c_{ls}) solutions costs. After the first 100 iterations, they use this information to decide whether each constructed solution will be submitted to local search or not. Their idea is based on the rationale that if some reasonable threshold applied to the cost of the constructed solution leads to a value much higher than the cost of the best solution already found, it is unlikely that local search could produce a better solution than the current best. Their threshold is determined by (3.7), where

an initial solution passes through the filter only if 90% of the ratio c_{ini}/c_{best} is less than or equal to the average ratio (μ).

$$0.9c_{ini} \leqslant \mu c_{best} \tag{3.7}$$

This work addresses GRASP filtering with a different approach, where a statistically meaningful filter is proposed. This filter is able to classify bad solutions within a certain confidence interval. For this, an additional variable is needed to determine the threshold, the standard deviation (σ) of the ratio between initial and local search solutions costs. A solution is considered good, and passes through the filter, when it satisfies the following condition:

$$c_{ini} \leqslant (\mu + 2\sigma)c_{best} \tag{3.8}$$

The filter accepts an initial solution to undergo local search when the ratio c_{ini}/c_{best} is less than the average ratio plus two times its standard deviation, which gives a confidence interval of slightly more than 95% probability that a rejected solution could not be improved by local search further than c_{best}, assuming of course that c_{ini}/c_{ls} is an independent random variable with normal distribution. Figure 3.3 clearly illustrates the normal distribution of 10000 samples of the ratio c_{ini}/c_{ls} randomly generated by solving instance *oegl-e1-A* (see Section 3.4) with the constructive heuristic (Algorithm 1), to obtain c_{ini}, and with the local search (Algorithm 3), to obtain c_{ls}.

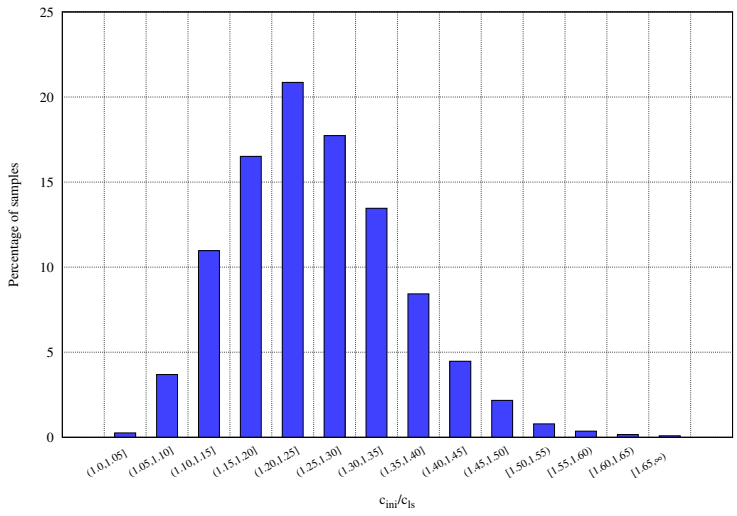


Figure 3.3: Distribution of the ratio c_{ini}/c_{ls} for instance *oegl-e1-A*.

3.3 Path-Relinking

Path-relinking (PR) was introduced by Glover (1996), in the context of tabu and scatter searches, as a mechanism to combine intensification and diversification by exploring trajectories connecting high-quality (elite) solutions previously produced during the search. These elite solutions often share a significant portion of their attributes, for example the nodes and edges of a graph. Paths between a pair of solutions (S_1, S_2) in the search space traverse other solutions that share these attributes contained in S_1 and S_2. Such paths may be generated by applying neighborhood moves to the initial solution S_1, which progressively introduces attributes from the guiding solution S_2. This generates a sequence of intermediate solutions, often not locally optimal, however improvable by local search and possibly better than S_1 and S_2.

Labadi et al. (2008) observed that, despite GRASP simplicity and speed, it is often less effective than its counterpart metaheuristics, like tabu search, and they explain this may be due to the independent (memoryless) GRASP iterations, using no information to sample good regions of the solution space. This may be remedied hybridizing PR with GRASP, as Resende and Ribeiro (2005) suggest, in order to improve the performance of the latter by tackling the memoryless criticism faced by the basic GRASP scheme.

The use of path-relinking within a GRASP procedure can be done as an *intensification strategy* to each local optimum obtained after the local search phase, and/or as a *post-optimization strategy* to all pairs of elite solutions. Labadi et al. (2008) uses both strategies separately to solve the CARP-TW, and conclude that the intensification strategy provided a better average deviation from lower bound in exchange for a higher computational time.

A relatively recent trend in literature is the *evolutionary PR*, where pairs of elite set solutions are continuously relinked while improvements in quality are observed on the elite set (Villegas et al. 2011; Resende and Werneck 2004), with a close relation to the evolutionary behaviour of a genetic algorithm population. This work uses the evolutionary PR as an intensification strategy, following some ideas of Resende, Martí, Gallego and Duarte (2010).

Details on the path-relinking distance metric, neighborhood operator, management of the elite solutions pool, and the PR implementation are in the remainder of this section.

3.3.1 Solution distance metric

The distance between two solutions is a measure that reflects how these solutions differ from one another. For example, the *Hamming distance* is a straightforward metric for a problem whose solutions are represented by binary strings. This distance corresponds to the number of positions in the string in which both solutions have different values.

For other representations of solutions, a suitable distance metric could be the minimum number of moves of a given neighborhood operator to transform an initial solution S_i into a guiding solution S_j. However, a minimum number of moves also implies shorter trajectories to

explore. Thus, the minimum number of moves may not be the best option of choice if a more explorative path-relinking is intended.

This work have used the *broken pairs distance* δ_{ij} as the metric adopted in the proposed path-relinking. This distance, following the ideas of Labadi et al. (2008), is measured by the relative position of each required edge in a given solution S_i compared to a second solution S_j. The relative position of a required edge is defined by its predecessor required edge. It is worth reminding that an OCARP solution is formed by family of tours, represented as a sequence of required edges with shortest paths of non-required edges linking one required edge to another (see Figure 3.2).

The distance δ_{ij} refers to the number of required edges whose predecessors are different in two solutions S_i and S_j (see example in Figure 3.4). It should be easy to realize that this metric always satisfies the inequality $0 \leqslant \delta_{ij} \leqslant |E_R|$.

3.3.2 Relinking of Solutions

Block-insertion was the neighborhood move used to generate the path between the initial and guiding solutions (see Section 3.2.2.1). Through block-insertion it is easy to generate a series of moves which monotonically decreases the distance from the initial to the guiding solution. To achieve this, instead of moving only one misplaced required edge, a whole block of adjacent edges is moved altogether. This block is formed by a starting misplaced edge and a following sequence of edges, already on their correct relative position. This guarantees that on each move, the resulting solution will be closer to the guiding solution by one or two units of distance.

An example of the relinking process is given in Figure 3.4, where each capital letter represents a required edge, written in italics when its relative position is incorrect, and in bold when it has been moved to the correct position. A block of required edges moved by a block insertion is represented by a rectangle.

It is not an easy task to assure feasibility of intermediate solutions obtained through block insertion, given that it would lead to a bin packing problem of arranging these blocks among the vehicles, regarding capacity constraints. However, in an attempt to preserve feasibility, the decision on the next moving block is made by taking the lightest displaced block from the fullest vehicle.

To explore these intermediate solutions, the local search phase is repeatedly applied once the current solution is four units closer to the guiding solution. The four units of distance is not arbitrary, but recommended by Ribeiro and Resende (2011) as the minimum number of differing components between a pair of solutions to find a better local minimum.

Algorithm 5 gives the pseudo-code for relinking a pair of solutions. This algorithm has a main loop 4 that repeatedly applies block insertions to an initial solution until this solution is sufficiently close to the guiding solution. Theoretically, the maximum distance between two

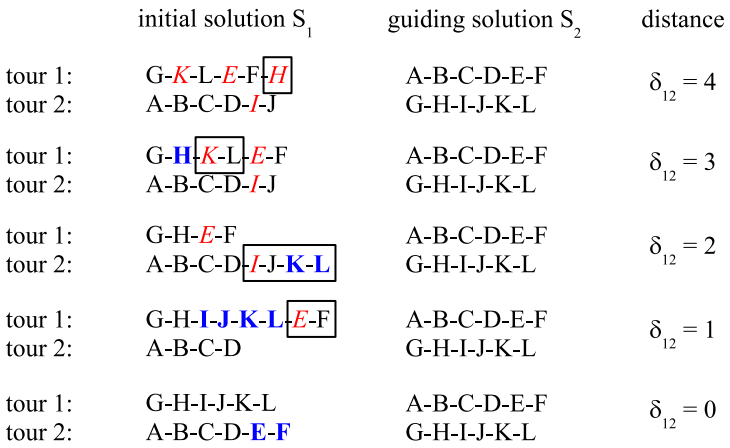


Figure 3.4: Relinking two solutions.

solutions can be at most E_R, and each block insertion reduces this distance in one or two units. Every time the initial solution is four units closer to the guiding solution, local search is applied to the initial solution in search of a better one. Since the local search computational complexity is bounded by $O(|E_R|^2)$, then the relinking algorithm has a worst-case complexity of $O(|E_R|^3)$.

3.3.3 Elite solutions pool

The elite solutions pool represents a set of the best solutions found by the metaheuristic that still preserve some diversity among them. An invariant of this pool $P = \{S_1, S_2, \ldots, S_n\}$ is that for all pairs (i, j) with $i \neq j$, then $\delta_{ij} \geqslant \delta_{min}$, where δ_{min} is a diversity parameter which sets the minimum distance between solutions belonging to the pool.

In order for a candidate solution (S_k) to be considered entering the pool, it must satisfy one of the following conditions:

- pool is not full, and there are no elite solutions S_i such that $cost(S_i) \leqslant cost(S_k)$ and $\delta_{ik} < \delta_{min}$.

- pool is full, there are no elite solutions S_i such that $cost(S_i) \leqslant cost(S_k)$ and $\delta_{ik} < \delta_{min}$, and there are at least one elite solution S_j such that $cost(S_j) > cost(S_k)$.

Once the candidate solution S_k is admitted in the pool, every elite solution S_i $(i \neq k)$ with $\delta_{ik} < \delta_{min}$, if any, are excluded from the pool. If still the pool size remains above its capacity, then the worst elite solution is excluded from the pool (Algorithm 6).

Algorithm 5 solutionRelinking(S_i, S_j, c_{filter})

Input: S_i, S_j - pair of initial-guiding solutions, c_{filter} - local search filter threshold
Output: S_{best} is the lowest cost solution obtained on the path between S_i and S_j

$S_{best} \leftarrow S_i$
$\delta_{ij} \leftarrow$ **distance**(S_i, S_j) // distance between solutions, see Section 3.3.1
$\delta_{next} \leftarrow \delta_{ij} - 4$
while $(\delta_{ij} \geqslant 1)$ **do**
  $tour \leftarrow$ most loaded tour in S_i containing a required edge on an incorrect relative position
  $[e_{ini}, e_{end}] \leftarrow$ less demanding incorrectly positioned block of required edges from $tour$
  $e_{pred} \leftarrow$ predecessor edge of e_{ini} in S_j
  $S_i \leftarrow$ **blockInsert**$(S_i, e_{pred}, [e_{ini}, e_{end}])$ // move block to its correct relative position
  $\delta_{ij} \leftarrow$ **distance**(S_i, S_j)
  if $\delta_{ij} \leqslant \delta_{next}$ **then**
    if (**isFeasible**(S_i)) **and** (**cost**$(S_i) < c_{filter}$) **then**
      $S_{ls} \leftarrow$ **localSearch**(S_i)
      if (**cost**$(S_{ls}) <$ **cost**(S_{best})) **then**
        $S_{best} \leftarrow S_{ls}$
      end if
    end if
    $\delta_{next} \leftarrow \delta_{ij} - 4$
  end if
end while
return S_{best}

3.3.4 GRASP and Path-relinking coupling

The path-relinking proposed in this work was implemented as an intensification strategy for the GRASP, combined with the concepts of evolutionary path-relinking. At every GRASP iteration, the solution generated after the local search phase is tested for membership of the elite pool, and relinked with the five best elite solutions (iterative PR). The best solution obtained from each path is tested for membership of the elite pool. At every 100 iterations, an evolutionary PR is executed, where each solution from the pool is relinked with the five best solutions from the same pool. The rationale of this strategy is to initially fill the elite pool with high-quality and diverse solutions generated by the iterative PR. The quality of the pool is then improved with the evolutionary PR, and in order to maintain diversity, another 100 iterations of the GRASP with iterative PR are executed. This is repeated for 10000 iterations or while the average cost of the elite solutions is improved.

The path between two solutions is always explored in both directions, i.e., each solution acts as initial and guiding. To sum up some diversity in the path-relinking, the solution space exploration is not restrained to the feasible space between two solutions, but also to promising unfeasible regions. Given a pair of solutions, one of them acts as initial PR solution, after going through the infeasible local search (Section 3.2.2), while the other acts as the guiding solution, unchanged. This strategy leads to an alternative path traversing the feasible-infeasible boundary between the initial and guiding solutions.

Algorithm 6 insertPool(P, S, δ_{min})

Input: P - pool of elite solutions, S - solution, δ_{min} - minimum distance between solutions in the pool.
Output: **true** if solution was inserted in pool, **false** otherwise.

```
 1: insert ← false
 2: n ← lastIndex(P) // index of the last solution in the pool
 3: S_worst ← P_n // worst solution in the pool
 4: for (i = 1 to n) do
 5:    if ((distance(S, P_i) < δ_min) and (cost(S) > cost(P_i))) then
 6:       return insert
 7:    end if
 8: end for
 9: if (cost(S) < cost(S_worst)) then
10:    insert ← true
11:    for (i = 1 to n) do
12:       if (distance(S, P_i) < δ_min) then
13:          P_i ← ∅ // remove solution of index i from the pool
14:       end if
15:    end for
16:    resortPool(P) // resort solutions of the pool in increasing order of cost
17:    n ← lastIndex(P) // index of the last solution in the pool
       // if pool is not full
18:    if (n < poolSize(P)) then
19:       P_{n+1} ← S
20:    else
21:       P_n ← S
22:    end if
23: end if
24: return insert
```

Algorithms 7 and 8 give the pseudo-code for the iterative PR and evolutionary PR. For simplicity, it is considered that the elite solutions in the pool are sorted by costs in increasing order. Both path-relinking strategies use the solution relinking operator (Algorithm 5) a constant number of times, thus following the worst-case computational complexity of $O(|E_R|^3)$. The management of the elite pool involves sorting the solutions by their cost and keeping the list sorted when a new solution is inserted. Considering the constant size of the pool, its management is computationally bounded by $O(1)$, thus not adding complexity to both path-relinking strategies.

Algorithm 7 iterativePR(P, S, c_{filter})

Input: P - pool of elite solutions, S - solution, c_{filter} - local search filter threshold

$S_{best} \leftarrow S$
$n_{inf} \leftarrow 4$ // number of moves to execute with the infeasible local search
insertPool(P, S)
$S_{inf} \leftarrow$ **infeasibleLocalSearch**(S, n_{inf}) // relink the five best elite solutions with S and S_{inf}
for $(i = 1$ **to** $5)$ **do**
    $S_{pool} \leftarrow P_i$
    $S_{pr1} \leftarrow$ **solutionRelinking**$(S_{pool}, S, c_{filter})$
    $S_{pr2} \leftarrow$ **solutionRelinking**$(S, S_{pool}, c_{filter})$
    $S_{pr3} \leftarrow$ **solutionRelinking**(S_{inf}, S, c_{filter})
    insertPool(P, S_{pr1}), **insertPool**(P, S_{pr2}), **insertPool**(P, S_{pr3})
end for

Algorithm 8 evolutionaryPR(P, c_{filter})

Input: P - pool of elite solutions, c_{filter} - local search filter threshold

$poolSize \leftarrow 100, \quad \delta_{\min} \leftarrow 0.4|E_R|$
$P_{new} \leftarrow$ **newPool**$(poolSize, \delta_{\min})$
$n_{inf} \leftarrow 4$ // number of moves to execute with the infeasible local search
for $(i = 1$ **to** $poolSize)$ **do**
    $S_1 \leftarrow P_i$
    insertPool(P_{new}, S_1)
    $S_{ils} \leftarrow$ **infeasibleLocalSearch**(S_1, n_{inf})
    // relink the five best solutions with all solutions from the pool
    for $(j = 1$ **to** $5)$ **do**
        $S_2 \leftarrow P_j$
        $S_{pr1} \leftarrow$ **solutionRelinking**(S_1, S_2, c_{filter})
        $S_{pr2} \leftarrow$ **solutionRelinking**(S_2, S_1, c_{filter})
        $S_{pr3} \leftarrow$ **solutionRelinking**$(S_{ils}, S_2, c_{filter})$
        insertPool(P_{new}, S_{pr1}), **insertPool**(P_{new}, S_{pr2}), **insertPool**(P_{new}, S_{pr3})
    end for
end for
$P \leftarrow P_{new}$

3.3.5 GRASP with Path-relinking pseudo-code

Algorithm 9 shows the pseudo-code for the proposed GRASP with path-relinking. This metaheuristic has a main loop starting in line 13. Inside the loop, the values of parameters α and β are reactively selected, followed by the construction of an initial solution. This solution may be infeasible if the heuristic is solving an instance with a limited number of vehicles. Thus the feasibility is verified, and the cost-demand edge-selection rule parameter γ adjusted accordingly. If the solution is feasible and has a sufficiently low cost to pass the filter threshold, local search is performed, followed by an iterative path-relinking. Also, at every k_{pr} iterations the evolutive path-relinking is executed. The heuristic halts when a given number of iterations k_{end} is performed, or when the average cost of the elite solutions pool does not improve after an evolutive path-relinking.

This metaheuristic was developed to solve CARP instances (Usberti, França and França 2011a), but it can solve the OCARP as well. If the number of vehicles is finite, as in OCARP instances, the contructive heuristic may not deliver a feasible solution. However, the metaheuristic addresses the vehicles limitation by adjusting the cost-demand edge-selection into a more conservative use of the vehicles capacities, therefore reducing the number of vehicles used by a solution.

The computational complexity of the metaheuristic is mainly attributed to both path-relinking strategies (lines 35 and 38), which are executed a constant number of times inside the main loop. As a consequence, the worst-case complexity of the metaheuristic is bounded by $O(|E_R|^3)$.

3.4 Computational Experiments

This section reveals the computational experiments using the GRASP with PR to solve both OCARP (Section 3.4.1) and CARP (Section 3.4.2) instances. The standard set of CARP instances[1] was referred to, which includes 23 *gdb* (7-27 nodes, 11-55 edges) (Golden et al. 1983), 34 *val* (24-50 nodes, 34-97 edges) (Benavent et al. 1992), and 24 *egl* (77-140 nodes, 98-190 edges) (Li and Eglese 1996), totaling 81 instances.

Algorithms were implemented in C language, and compiled using the GNU compiler collection (gcc). Tests were executed in a Intel Core 2 Quad 3.0 GHz with 4 GB of RAM, using Linux 64 bits as the operating system.

Table 3.2 lists the GRASP parameters and their values used in the computational experiments.

[1] http://www.uv.es/~belengue/carp.html

Algorithm 9 GRASP with evolutionary PR

Input: $G(V, E)$ - instance graph, D - vehicle capacity
Output: S_{best} is the lowest cost feasible solution obtained

$c_{filter} \leftarrow c_{lsBest} \leftarrow 2|E_R| \sum_{e \in E} c(e)$ // trivial upper bound
$c_{worst} \leftarrow \sum_{e \in E_R} c(e)$ // trivial lower bound
$A \leftarrow \{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5\} \leftarrow \{0.0, 0.5, 1.0, 1.5, 2.0\}$ // possible values for α
$B \leftarrow \{\beta_1, \beta_2, \beta_3, \beta_4, \beta_5\} \leftarrow \{1.0, 1.25, 1.5, 1.75, 2.0\}$ // possible values for β
$N_\alpha \leftarrow \{n_{\alpha_1}, n_{\alpha_2}, n_{\alpha_3}, n_{\alpha_4}, n_{\alpha_5}\} \leftarrow \{0, 0, 0, 0, 0\}$ // number of solutions for each α
$N_\beta \leftarrow \{n_{\beta_1}, n_{\beta_2}, n_{\beta_3}, n_{\beta_4}, n_{\beta_5}\} \leftarrow \{0, 0, 0, 0, 0\}$ // number of solutions for each β
$C_\alpha \leftarrow \{\overline{c}_{\alpha_1}, \overline{c}_{\alpha_2}, \overline{c}_{\alpha_3}, \overline{c}_{\alpha_4}, \overline{c}_{\alpha_5}\} \leftarrow \{0.0, 0.0, 0.0, 0.0, 0.0\}$ // average solution cost for each α
$C_\beta \leftarrow \{\overline{c}_{\beta_1}, \overline{c}_{\beta_2}, \overline{c}_{\beta_3}, \overline{c}_{\beta_4}, \overline{c}_{\beta_5}\} \leftarrow \{0.0, 0.0, 0.0, 0.0, 0.0\}$ // average solution cost for each β
$\gamma \leftarrow 1.0$, $g \leftarrow 0.5$
$poolSize \leftarrow 100$, $\delta_{\min} \leftarrow 0.4|E_R|$ // defining pool size and minimum solution distance
$P \leftarrow$ **newPool**$(poolSize, \delta_{\min})$
$k_{end} \leftarrow 10000$, $k_{filter} \leftarrow k_{pr} \leftarrow 100$, $c_{aver} \leftarrow \infty$
for $k = 1$ **to** k_{end} **do**
    $i \leftarrow$ **reactiveChoice**(C_α, N_α), $j \leftarrow$ **reactiveChoice**(C_β, N_β)
    $\alpha \leftarrow \alpha_i$, $\beta \leftarrow \beta_j$
    $S \leftarrow$ **constructionPhase**$(G, D, \alpha, \beta, \gamma)$ // see Algorithm 2
    if (**feasible**(S) = **true**) **then**
        $\gamma \leftarrow \frac{2\gamma + 1}{3}$, $g \leftarrow \frac{9g + 1}{10}$
        $c_{ini} \leftarrow$ **cost**(S)
        if $(c_{ini} > c_{worst})$ **then**
            $c_{worst} \leftarrow c_{ini}$
        end if
        if $(c_{ini} \leqslant c_{filter})$ **then**
            $S_{ls} \leftarrow$ **localSearch**(S)
            $c_{ls} \leftarrow$ **cost**(S_{ls})
            if $(c_{ls} < c_{lsBest})$ **then**
                $c_{lsBest} \leftarrow c_{ls}$
            end if
            if $(k \leqslant k_{filter})$ **then**
                $(\mu, \sigma) \leftarrow$ update average and standard deviation of ratio $\frac{c_{ini}}{c_{ls}}$
            else
                // update filter threshold
                $c_{filter} \leftarrow (\mu + 2\sigma) c_{lsBest}$
            end if
            iterativePR(P, S_{ls}, c_{filter})
            if $(k \equiv 0 \pmod{k_{filter}})$ **then**
                // apply evolutionary PR once every k_{filter} iterations
                evolutionaryPR(P, c_{filter})
                if (**averageCost**$(P) \geqslant c_{aver}$) **then**
                    break for // no pool improvement, stop
                end if
                $c_{aver} \leftarrow$ **averageCost**(P)
            end if
        end if
    else
        $\gamma \leftarrow g\gamma$
        $c_{ini} \leftarrow c_{worst}$ // penalizing cost of infeasible solutions
    end if
    $n_{\alpha_i} \leftarrow n_{\alpha_i} + 1$, $n_{\beta_j} \leftarrow n_{\beta_j} + 1$
    $\overline{c}_{\alpha_i} \leftarrow \overline{c}_{\alpha_i} + \frac{c_{ini} - \overline{c}_{\alpha_i}}{n_{\alpha_i}}$, $\overline{c}_{\beta_i} \leftarrow \overline{c}_{\beta_i} + \frac{c_{ini} - \overline{c}_{\beta_i}}{n_{\beta_i}}$ // updating the average solution cost
end for
return $S_{best} \leftarrow P_1$

Table 3.2: GRASP parameters.

$k^{end} = 10000$	maximum number of GRASP iterations.
$k^{filter} = 100$	number of iterations to calibrate the filter threshold.
$k^{eps} = 100$	number of GRASP iterations between evolutionary path-relinking executions.
$A = \{0.0, 0.5, 1.0, 1.5, 2.0\}$	possible values for the RCL parameter α.
$B = \{1.0, 1.25, 1.5, 1.75, 2.0\}$	possible values for ellipse rule parameter β.
$\gamma = 1$	initial value for the cost-demand edge selection rule paramenter.
$g = 0.5$	metaparameter value for adjusment of parameter γ.
$n_{inf} = 4$	maximum number of infeasible moves to execute with the infeasible local search.
$poolSize = 100$	size of the elite solutions pool.
$\delta_{\min} = 0.4\|E_R\|$	minimum distance between solutions in the elite solutions pool.

3.4.1 OCARP case study

Having in mind that OCARP is a new NP-hard combinatorial optimization problem, the objective of these experiments consisted in forming a benchmark set of instances and conferring the first upper bounds to the optimal costs. The depot in the standard CARP set of instances was considered a common node while the rest of the data left intact, leading to three groups of instances referred as *ogdb*, *oval*, and *oegl*. The solutions costs for these instances were compared with trivial lower bounds ($LB_0 = \sum_{e \in E_R} c(e)$), in lack of a better alternative.

Algorithm 9 was engineered to build tours starting and ending at a depot node. Knowing that OCARP does not concern a depot, this issue was addressed by transforming the OCARP graphs using the reduction described in Section 2.5, where a dummy depot connecting all required edges is included. From this virtual depot all vehicles start and end their tours.

Property 1, in Section 2.3, has shown that OCARP is only an interesting problem when a fixed number of vehicles M is considered. For this reason, the OCARP computational tests has considered three classes of parameterization: $M = M^*$, $M = M^* + 1$ and $M = M^* + 2$, where M^* represents the minimum number of vehicles necessary to find a feasible solution, which is known for all instances. Consequently, from each of the 81 instances, three different numbers of vehicles are considered, thus deriving 243 OCARP instances, divided into three groups (*ogdb*, *oval*, and *oegl*), and three classes (M^*, $M^* + 1$, $M^* + 2$).

3.4.1.1 Parameter γ behavior

Figure 3.5 reveals the behavior of the cost-demand edge-selection rule parameter γ (Section 3.2.1.2) for instance *oegl-e4-C*. The intricacy of seeking for a feasible solution which uses at most M^* vehicles led to the prompt adjustment of γ towards edge demand. As more vehicles are allowed in the solution ($M^* + 1$ and $M^* + 2$), less active is the fitting of γ. It can be noticed that all three curves have a long-term ascendant tendency, which is a consequence of metaparameter

g, which is asymptotically driven towards 1 for every feasible solution found (represented by a peak). The rationale of these adjustments is that, as soon as a feasible solution emerges, the heuristic should invest more iterations biasing the edge-selection rule in the direction of the shortest path cost, thus giving the opportunity for better solutions to arise.

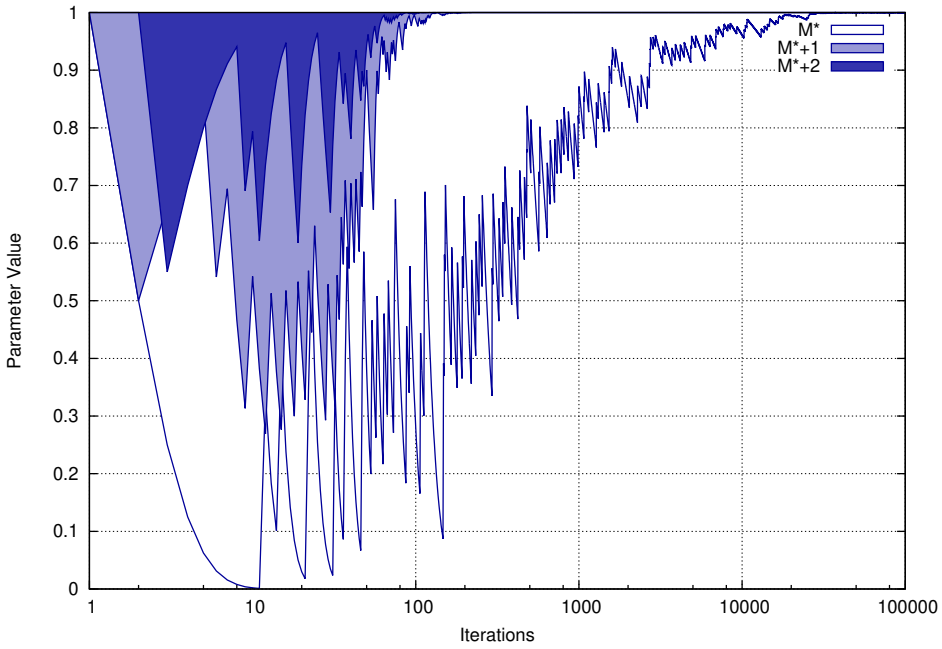


Figure 3.5: Evolution of the parameter γ for instance *oegl-e4-C*.

3.4.1.2 Constructive heuristic results to OCARP

The GRASP constructive heuristic (GCH, Algorithm 1) was compared with the path-scanning heuristics from Belenguer et al. (2006) (PS) and Santos et al. (2009) (PS_ER) (described in Section 3.2.1), the last one considering three configurations for the ellipse rule parameter β (Table 3.3). The methods execution times were very alike, so only the GCH times were reported. When considering each group-class of instances, the GCH heuristic achieved the smallest $\Delta LB = (UB - LB)/LB$ for all but one group-class, namely *ogdb*-($M^* + 2$). In addition, GCH has also achieved the lowest $\Delta LB^{\max}$ for six group-classes, which is at least two times better than the other heuristics. In the overall comparison, GCH outperformed its siblings in both average and maximum deviations with $\Delta LB = 12.24\%$ and $\Delta LB^{\max} = 115.77\%$.

As mentioned in Section 3.2.1, heuristic PS_ER was unable to find feasible solutions for some instances of several group-classes (represented by a dash '-' in Table 3.3). This effect is

amplified for higher values of β, since it shortens the tours more prematurely. Nevertheless, the looser instances for which PS_ER did find feasible solutions had their ΔLB decreased as β increased, meaning that the ellipse rule behaves more effectively for these group-classes. A glimpse on the hardness to attain a feasible solution for each instance group-class is given by $Feas$, which represents the percentage of feasible solutions obtained compared to the total number of iterations.

Table 3.3: Comparison results between path-scanning heuristics.

		PS			PS_ER									GCH			
					$\beta = 0.5$			$\beta = 1.0$			$\beta = 1.5$						
group	class	ΔLB	$\Delta LB^{\max}$	$Feas$	ΔLB	$\Delta LB^{\max}$	$Feas$	ΔLB	$\Delta LB^{\max}$	$Feas$	ΔLB	$\Delta LB^{\max}$	$Feas$	ΔLB	$\Delta LB^{\max}$	$Feas$	CPU
ogdb	M^*	0.50	5.48	94.96	0.38	3.33	90.34	**0.29**	2.74	77.82	1.90	17.29	62.01	**0.29**	**2.38**	77.43	0.17
	M^*+1	0.48	5.02	100.00	0.36	3.81	100.00	0.22	**1.9**	100.00	**0.2**	**1.9**	99.00	**0.2**	2.25	98.44	0.16
	M^*+2	0.48	5.02	100.00	0.36	3.81	100.00	0.22	1.90	100.00	**0.13**	**1.69**	100.00	0.20	2.25	99.95	0.17
	overall	0.49	5.48	98.32	0.37	3.81	96.78	0.24	2.74	92.61	0.74	17.29	87.00	**0.23**	**2.38**	91.94	0.17
oval	M^*	5.12	**13.7**	94.36	4.91	29.45	91.42	-	-	84.48	-	-	77.40	**4.22**	18.49	83.65	1.59
	M^*+1	5.93	9.71	100.00	5.32	9.71	100.00	4.50	9.35	99.91	4.43	9.35	97.39	**4.21**	**8.27**	97.96	1.64
	M^*+2	6.46	10.07	100.00	5.85	10.07	100.00	5.01	9.35	100.00	4.91	9.35	100.00	**4.67**	**9.23**	99.70	1.57
	overall	5.84	**13.7**	98.12	5.36	29.45	97.14	-	-	94.80	-	-	91.60	**4.36**	18.49	93.77	1.60
oegl	M^*	53.24	132.85	82.87	-	-	69.58	-	-	34.99	-	-	9.66	**42.76**	**115.77**	49.07	17.03
	M^*+1	48.18	71.95	98.69	34.77	58.32	95.05	30.99	83.80	81.88	-	-	61.80	**28.41**	**40.82**	80.65	6.98
	M^*+2	47.08	71.95	100.00	34.33	58.32	99.88	27.73	48.64	95.32	25.38	**36.48**	83.45	**25.21**	39.60	90.73	7.34
	overall	49.50	132.85	93.85	-	-	88.17	-	-	70.73	-	-	51.64	**32.12**	**115.77**	73.48	10.45
overall	M^*	19.62	132.85	90.73	-	-	83.78	-	-	65.76	-	-	49.69	**15.75**	**115.77**	70.05	5.76
	M^*+1	18.20	71.95	99.56	13.48	58.32	98.35	11.90	83.80	93.93	-	-	86.06	**10.94**	**40.82**	92.35	2.80
	M^*+2	18.01	71.95	100.00	13.51	58.32	99.96	10.99	48.64	98.44	10.14	**36.48**	94.48	**10.02**	39.60	96.79	2.88
	overall	18.61	132.85	96.76	-	-	94.03	-	-	86.04	-	-	76.75	**12.24**	**115.77**	86.40	3.82

PS - Path-Scanning Heuristic with Random Selection of Tied Edges (Belenguer et al. 2006). PS_ER - Path-Scanning Heuristic with Ellipse Rule (Santos et al. 2009).
GCH - GRASP Constructive Heuristic. β - ellipse rule parameter. M^* - maximum number of vehicles. CPU - running time (s)
ΔLB: average deviation from lower bound (%). $\Delta LB^{\max}$: maximum average deviation from lower bound (%). $Feas$: feasible solutions against all solutions (%).

3.4.1.3 Metaheuristic results to OCARP

Tables 3.4-3.6 show the individual results for all instances after executing a single time the GRASP with PR metaheuristic detailed in Algorithm 9. From the set of 243 instances, 87 solutions $(35,80\%)$ were proven to be optimal ($LB = UB$). Average deviations from lower bound (ΔLB) were 7.53%, 3.74%, and 2.66%, for groups M^*, M^*+1, and M^*+2, respectively. The overall average deviation was 4.64%, against the GCH overall average deviation of 12.24%, representing more than 62% reduction.

The CPU time was 8.47 minutes on average per instance, with less than one-fourth of instances above this average. The field $Feas$ on Tables 3.4-3.6 gives the percentage of feasible solutions compared to the number of executed iterations for each instance, thus showing those instances with a difficult bin-packing subproblem. For example, instance *oegl-s4-C*, class M^*, is certainly the hardest instance to find a feasible solution, and this is confirmed by the ratio 0.59% of feasible solutions against iterations.

Table 3.4: GRASP with PR results for *ogdb* instances.

			$M = M^*$				$M = M^* + 1$				$M = M^* + 2$			
	M^*	LB_0	UB	ΔLB_0	CPU	$Feas$	UB	ΔLB_0	CPU	$Feas$	UB	ΔLB_0	CPU	$Feas$
ogdb1	5	252	**252**	0.00	0.00	100.00	**252**	0.00	0.00	100.00	**252**	0.00	0.00	100.00
ogdb2	6	291	**291**	0.00	0.00	100.00	**291**	0.00	0.00	100.00	**291**	0.00	0.00	100.00
ogdb3	5	233	**233**	0.00	0.00	100.00	**233**	0.00	0.00	100.00	**233**	0.00	0.00	100.00
ogdb4	4	238	**238**	0.00	0.01	100.00	**238**	0.00	0.00	100.00	**238**	0.00	0.00	100.00
ogdb5	6	316	**316**	0.00	0.01	100.00	**316**	0.00	0.00	100.00	**316**	0.00	0.00	100.00
ogdb6	5	260	**260**	0.00	0.00	100.00	**260**	0.00	0.00	100.00	**260**	0.00	0.00	100.00
ogdb7	5	262	**262**	0.00	0.00	100.00	**262**	0.00	0.00	100.00	**262**	0.00	0.00	100.00
ogdb8	10	210	**210**	0.00	0.30	100.00	**210**	0.00	0.03	100.00	**210**	0.00	0.00	100.00
ogdb9	10	219	**219**	0.00	1.24	96.67	**219**	0.00	0.09	100.00	**219**	0.00	0.02	100.00
ogdb10	4	252	**252**	0.00	0.02	100.00	**252**	0.00	0.01	100.00	**252**	0.00	0.01	100.00
ogdb11	5	356	362	1.69	40.93	100.00	360	1.12	62.44	100.00	358	0.56	65.37	100.00
ogdb12	7	336	**336**	0.00	0.00	100.00	**336**	0.00	0.00	100.00	**336**	0.00	0.00	100.00
ogdb13	6	509	**509**	0.00	0.14	17.82	**509**	0.00	0.00	100.00	**509**	0.00	0.00	100.00
ogdb14	5	96	**96**	0.00	0.00	100.00	**96**	0.00	0.00	100.00	**96**	0.00	0.00	100.00
ogdb15	4	56	**56**	0.00	0.00	100.00	**56**	0.00	0.00	100.00	**56**	0.00	0.00	100.00
ogdb16	5	119	**119**	0.00	0.01	100.00	**119**	0.00	0.00	100.00	**119**	0.00	0.00	100.00
ogdb17	5	84	**84**	0.00	0.01	100.00	**84**	0.00	0.00	100.00	**84**	0.00	0.00	100.00
ogdb18	5	158	**158**	0.00	0.00	100.00	**158**	0.00	0.01	100.00	**158**	0.00	0.00	100.00
ogdb19	3	45	**45**	0.00	0.00	100.00	**45**	0.00	0.00	100.00	**45**	0.00	0.00	100.00
ogdb20	4	105	**105**	0.00	0.02	77.78	**105**	0.00	0.00	100.00	**105**	0.00	0.00	100.00
ogdb21	6	149	**149**	0.00	0.02	100.00	**149**	0.00	0.00	100.00	**149**	0.00	0.00	100.00
ogdb22	8	191	**191**	0.00	0.05	100.00	**191**	0.00	0.00	100.00	**191**	0.00	0.00	100.00
ogdb23	10	223	**223**	0.00	0.01	75.00	**223**	0.00	0.00	100.00	**223**	0.00	0.01	100.00

M^*: minimum number of vehicles to attain a feasible solution. LB_0: trivial lower bound.
UB: solution obtained by the GRASP with PR. ΔLB_0: average deviation from lower bound (%).
CPU: running time (s) to attain UB. $Feas$: feasible solutions against all solutions (%).

3.4.2 CARP case study

Considering that the main ideas of the GRASP with PR are perfectly suitable for CARP instances as well as OCARP instances, this work has verified the efficiency of the proposed metaheuristic against the latest metaheuristics developed for CARP. As usual for this problem, the number of vehicles available is infinite, without any costs related to using them. In addition, the lower bounds identified by Longo et al. (2006) are used to identify the gaps for the costs of the solutions obtained.

3.4.2.1 Constructive heuristic results to CARP

In order to show the effectiveness of the reactive parameter tuning scheme, described in Section 3.2.1, the path-scanning heuristic with ellipse rule (PS_ER) (Santos et al. 2009) was implemented with a fixed $\beta = 1.5$, and compared with the proposed GRASP constructive heuristic (GCH). Fifteen runs of the PS_ER and GCH with 10000 iterations were executed for each of the 81 instances.

Table 3.7 shows that practically with the same computational effort, GCH was able to reduce the average deviation from lower bound from every instance set, and all but one (*val* set) maximum average deviation from lower bound. In addition, due to the *restricted candidate list*,

Table 3.5: GRASP with PR results for *oval* instances.

			$M = M^*$				$M = M^* + 1$				$M = M^* + 2$			
	M^*	LB_0	UB	ΔLB_0	CPU	$Feas$	UB	ΔLB_0	CPU	$Feas$	UB	ΔLB_0	CPU	$Feas$
oval1A	2	146	154	5.48	49.48	100.00	151	3.42	43.11	100.00	149	2.05	41.04	100.00
oval1B	3	146	151	3.42	16.21	69.97	149	2.05	35.61	100.00	147	0.68	35.13	100.00
oval1C	8	146	159	8.90	20.19	3.55	**146**	0.00	0.01	100.00	**146**	0.00	0.01	100.00
oval2A	2	185	195	5.41	27.20	100.00	192	3.78	26.65	100.00	189	2.16	22.34	100.00
oval2B	3	185	192	3.78	17.31	100.00	189	2.16	19.02	100.00	186	0.54	31.42	100.00
oval2C	8	185	**185**	0.00	0.49	52.94	**185**	0.00	0.01	100.00	**185**	0.00	0.01	100.00
oval3A	2	65	71	9.23	23.07	100.00	69	6.15	24.84	100.00	67	3.08	33.79	100.00
oval3B	3	65	69	6.15	16.14	100.00	67	3.08	27.47	100.00	66	1.54	23.92	100.00
oval3C	7	65	66	1.54	9.85	73.17	**65**	0.00	0.01	100.00	**65**	0.00	0.02	100.00
oval4A	3	343	358	4.37	363.86	100.00	354	3.21	435.68	100.00	350	2.04	680.68	100.00
oval4B	4	343	354	3.21	344.86	100.00	350	2.04	489.60	100.00	347	1.17	734.55	100.00
oval4C	5	343	350	2.04	165.61	100.00	347	1.17	517.90	100.00	345	0.58	641.61	100.00
oval4D	9	343	**343**	0.00	83.54	100.00	**343**	0.00	8.56	100.00	**343**	0.00	2.00	100.00
oval5A	3	367	383	4.36	284.83	100.00	378	3.00	386.42	100.00	374	1.91	447.48	100.00
oval5B	4	367	378	3.00	223.11	100.00	374	1.91	382.62	100.00	371	1.09	479.20	100.00
oval5C	5	367	374	1.91	142.02	100.00	371	1.09	335.60	100.00	368	0.27	528.59	100.00
oval5D	9	367	**367**	0.00	2.35	100.00	**367**	0.00	2.77	100.00	**367**	0.00	0.66	100.00
oval6A	3	190	195	2.63	105.99	100.00	193	1.58	217.72	100.00	192	1.05	193.11	100.00
oval6B	4	190	194	2.11	65.63	100.00	192	1.05	147.72	100.00	191	0.53	188.99	100.00
oval6C	10	190	**190**	0.00	0.39	100.00	**190**	0.00	0.12	100.00	**190**	0.00	0.02	100.00
oval7A	3	249	263	5.62	284.33	100.00	259	4.02	459.07	100.00	256	2.81	553.60	100.00
oval7B	4	249	259	4.02	258.32	100.00	256	2.81	398.66	100.00	253	1.61	479.44	100.00
oval7C	9	249	250	0.40	49.31	99.92	**249**	0.00	8.43	100.00	**249**	0.00	1.11	100.00
oval8A	3	347	364	4.90	223.38	100.00	359	3.46	305.47	100.00	354	2.02	330.93	100.00
oval8B	4	347	359	3.46	163.73	100.00	354	2.02	226.39	100.00	351	1.15	377.57	100.00
oval8C	9	347	**347**	0.00	26.29	97.29	**347**	0.00	0.26	100.00	**347**	0.00	0.14	100.00
oval9A	3	278	298	7.19	1080.65	100.00	294	5.76	1194.49	100.00	292	5.04	1227.03	100.00
oval9B	4	278	294	5.76	802.44	100.00	292	5.04	1184.15	100.00	290	4.32	1339.11	100.00
oval9C	5	278	292	5.04	644.09	100.00	290	4.32	1155.24	100.00	288	3.60	1360.46	100.00
oval9D	10	278	283	1.80	409.27	100.00	281	1.08	1126.40	100.00	280	0.72	1308.58	100.00
oval10A	3	376	402	6.91	1052.22	100.00	396	5.32	1267.92	100.00	391	3.99	1586.57	100.00
oval10B	4	376	396	5.32	1176.71	100.00	391	3.99	1331.65	100.00	388	3.19	1798.88	100.00
oval10C	5	376	391	3.99	888.24	100.00	388	3.19	1421.52	100.00	385	2.39	1749.59	100.00
oval10D	10	376	379	0.80	400.95	100.00	378	0.53	1059.55	100.00	377	0.27	1430.68	100.00

M^*: minimum number of vehicles to attain a feasible solution. LB_0: trivial lower bound.
UB: solution obtained by the GRASP with PR. ΔLB_0: average deviation from lower bound (%).
CPU: running time (s) to attain UB. $Feas$: feasible solutions against all solutions (%).

GCH provides a much more diverse set of initial solutions, which is an important diversification ingredient for the local search and path-relinking.

3.4.2.2 Time-to-target plots

Run time distributions or time-to-target (TTT) plots display the probability that an algorithm will find a solution at least as good as a given target value within a given running time. Time-to-target plots were first used by Feo et al. (1994), and give subsidy to characterize the running times of stochastic algorithms for combinatorial optimization problems. Such plots are very useful in the comparison of different algorithms for solving a given problem.

To plot the empirical runtime distribution of a given stochastic algorithm, a solution target value is fixed and each algorithm is executed N times, recording the instant t_i when a solution

Table 3.6: GRASP with PR results for *oegl* instances.

			$M = M^*$				$M = M^* + 1$				$M = M^* + 2$			
	M^*	LB_0	UB	ΔLB_0	CPU	$Feas$	UB	ΔLB_0	CPU	$Feas$	UB	ΔLB_0	CPU	$Feas$
oegl-e1-A	5	1468	1775	20.91	76.12	99.95	1708	16.35	217.04	100.00	1659	13.01	287.82	100.00
oegl-e1-B	7	1468	1749	19.14	44.31	99.80	1639	11.65	149.34	100.00	1589	8.24	272.82	100.00
oegl-e1-C	10	1468	1652	12.53	44.41	99.55	1576	7.36	57.39	100.00	1542	5.04	91.36	100.00
oegl-e2-A	7	1879	2177	15.86	262.75	100.00	2072	10.27	441.48	100.00	2043	8.73	760.54	100.00
oegl-e2-B	10	1879	2080	10.70	97.65	99.93	1997	6.28	233.64	100.00	1971	4.90	505.74	100.00
oegl-e2-C	14	1879	2084	10.91	68.20	76.61	1997	6.28	94.88	99.75	1964	4.52	148.59	100.00
oegl-e3-A	8	2188	2526	15.45	185.72	98.16	2410	10.15	430.75	100.00	2372	8.41	1256.29	100.00
oegl-e3-B	12	2188	2411	10.19	157.45	98.83	2359	7.82	314.28	100.00	2321	6.08	647.77	100.00
oegl-e3-C	17	2188	2364	8.04	135.03	78.64	2308	5.48	142.48	99.67	2270	3.75	298.89	100.00
oegl-e4-A	9	2453	2693	9.78	234.87	99.70	2582	5.26	960.25	100.00	2556	4.20	2358.99	100.00
oegl-e4-B	14	2453	2786	13.58	296.66	87.58	2567	4.65	276.93	100.00	2517	2.61	595.38	100.00
oegl-e4-C	19	2453	3383	37.91	135.64	1.03	2515	2.53	155.93	80.21	2497	1.79	194.91	98.58
oegl-s1-A	7	1394	1799	29.05	132.72	100.00	1683	20.73	475.12	100.00	1604	15.06	710.46	100.00
oegl-s1-B	10	1394	1745	25.18	97.67	100.00	1659	19.01	323.03	100.00	1579	13.27	466.02	100.00
oegl-s1-C	14	1394	1876	34.58	102.41	85.99	1633	17.14	94.15	100.00	1512	8.46	147.65	100.00
oegl-s2-A	14	3174	3697	16.48	504.32	100.00	3621	14.08	1998.68	100.00	3561	12.19	4928.34	100.00
oegl-s2-B	20	3174	4309	35.76	806.30	7.62	3498	10.21	737.66	99.90	3427	7.97	1080.76	100.00
oegl-s2-C	27	3174	3928	23.76	1051.34	18.29	3414	7.56	537.12	98.75	3342	5.29	526.80	100.00
oegl-s3-A	15	3379	3828	13.29	1785.78	100.00	3764	11.39	5247.81	100.00	3734	10.51	6809.36	100.00
oegl-s3-B	22	3379	3680	8.91	1039.18	97.90	3588	6.19	853.82	100.00	3564	5.47	2444.62	100.00
oegl-s3-C	29	3379	3814	12.87	1606.15	55.96	3625	7.28	560.43	99.80	3495	3.43	807.41	100.00
oegl-s4-A	19	4186	4500	7.50	3899.02	100.00	4421	5.61	4382.59	100.00	4405	5.23	12448.65	100.00
oegl-s4-B	27	4186	4469	6.76	2040.83	92.27	4360	4.16	2834.19	100.00	4328	3.39	2662.37	100.00
oegl-s4-C	35	4186	7789	86.07	249.51	0.59	4502	7.55	2243.15	66.66	4320	3.20	2707.30	98.80

M^*: minimum number of vehicles to attain a feasible solution. LB_0: trivial lower bound.
UB: solution obtained by the GRASP with PR. ΔLB_0: average deviation from lower bound (%).
CPU: running time (s) to attain UB. $Feas$: feasible solutions against all solutions (%).

with cost at least as good as the target value is found. For each algorithm, the i-th sorted running time t_i is associated with probability $p_i = (i - 1/2)/N$. Determining these probabilities are explained by Aiex, Resende, Ribeiro, Celso and Ribeiro (2000). The TTT plot represents the points (t_i, p_i), for $i = 1, \ldots, N$. In this work, a sample of $N = 200$ runs were collected for each evaluated algorithm.

3.4.2.3 GRASP filtering effect on runtime

To establish the effect of filtering in the GRASP runtime, TTTs were drawn (Figure 3.6) for two basic GRASP heuristics, with (Gf) and without (Gnf) filtering, applied to the hardest instance $(egl - s4 - C)$. Three targets were selected, in order that the heuristics would not take too long to hit. Still, when the heuristics reached a limit of 2000 seconds, they were interrupted.

Figure 3.6 reveals that for all three targets Gf has improved runtime, which is minor for the highest target, but increases substantially for harder ones. For instance, with 50% probability, Gf hits the 21500 target in less than 500 seconds, while Gnf takes almost 900 seconds. Lower targets reflect better the filter effect on the TTTs, once high-quality solutions require many GRASP iterations to appear, and the filter safely eliminates unpromising initial solutions, which in turn saves plenty of the heuristic computational time.

Table 3.7: Comparison of constructive heuristics.

		gdb	val	egl	overall
PS_ER	ΔLB	1.99	5.01	9.23	5.40
	$\Delta LB_{\max}$	9.81	12.24	13.35	13.35
	CPU	0.15	0.61	1.70	0.80
GCH	ΔLB	1.82	4.98	8.32	5.07
	$\Delta LB_{\max}$	9.37	12.62	11.60	12.62
	CPU	0.15	0.62	1.71	0.81

Reported results were obtained after fifteen runs of 10000 iterations.
PS_ER - Path-Scanning with Ellipse Rule (Santos et al. 2009).
GCH - GRASP Constructive Heuristic.
ΔLB - average deviation from lower bound (%).
$\Delta LB_{\max}$ - maximum average deviation from lower bound (%).
CPU - average execution time per run in seconds.

3.4.2.4 Evolutionary PR effect on runtime

To quantify the evolutionary path-relinking contribution on the solution space search, two GRASP heuristics were compared using TTTs (Figure 3.7). The first one (EvPR) is the complete proposed GRASP, as described in Algorithm 9, while the second (PR) is the same heuristic except by the *evolutionary PR* (step 29), which was removed, and only the *iterative PR* is executed. Hence, this comparison tries to verify if the additional computational effort of EvPR is only an extra weight for the metaheuristic, or it effectively helps finding better solutions in reduced execution times.

Figure 3.7 clearly shows that EvPR is very effective in finding high-quality solutions in less time. For example, it has 50% chance to hit the 20900 target, for instance egl-s4-c, in less than 500 seconds. With the same probability, it requires 30% additional time in average (650 seconds or less) to hit the same target without EvPR. Nonetheless, EvPR loses its relative efficiency for targets that are not difficult to reach only with PR (e.g., 21100 target).

3.4.2.5 Metaheuristic results to CARP

The GRASP with EvPR was compared to three high-performance metaheuristics (Usberti et al. 2011b), based on their average deviation from lower bounds reported in literature, which are:

- TS - Tabu Search proposed by Brandão and Eglese (2008), which executed a single run for each instance on a Pentium Mobile at 1.4 GHz.
- VNS - Variable Neighborhood Search proposed by Polacek et al. (2008), which executed ten runs for each instance (except for the *gdb* set) on a Pentium IV at 3.6 GHz.

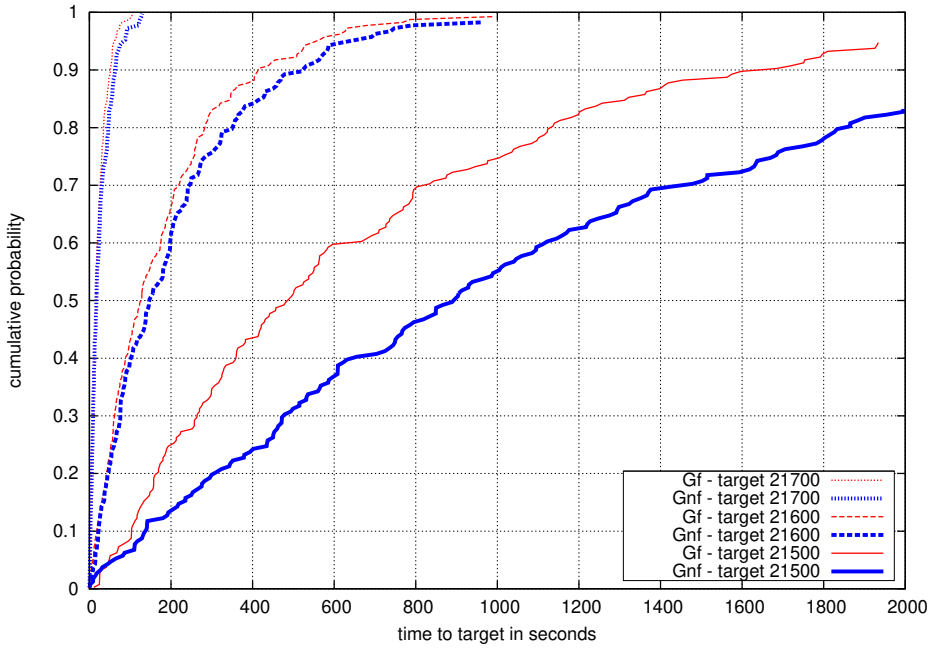


Figure 3.6: Run time distributions for GRASP with (Gf) and without (Gnf) filtering on instance *egl-s4-C*.

- ACO - Ant Colony Optimization proposed by Santos et al. (2010), which executed fifteen runs for each instance on a Pentium III at 1.0 GHz.

Tables 3.8, 3.9, and 3.10 report the results for sets *gdb*, *val*, and *egl* after fifteen runs of the GRASP with EvPR (Algorithm 9), and compare the best solutions obtained by each metaheuristic. The best lower bounds (LB) (Longo et al. 2006) and upper bounds (UB) (Santos et al. 2010) for each instance were also reproduced. It should be noticed that UB can be lower than the best solution reported for some instances (e.g., $val10D$). The values of UB were generated after additional experiments with CARP metaheuristics, for example, by particular parameter tunning for each instance. Thus UB values are used only as an information of the current best known solution cost for each instance, and are not comparable with the metaheuristics best results. A similar study was made with the GRASP, and through it five new best upper bounds (in italics) were discovered for instances *egl-e4-C* ($UB = 11559$), *egl-s2-B* ($UB = 13088$), *egl-s3-C* ($UB = 17189$), *egl-s4-B* ($UB = 16267$), and *egl-s4-C* ($UB = 20484$).

Table 3.11 summarizes CARP metaheuristics results on computational effort and solution quality. The average execution times reported are multiples of the metaheuristics original times, where the factor was determined by the processor frequency ratio between the original machine and the machine used in this work. The intention was to make a reasonably fair execution time

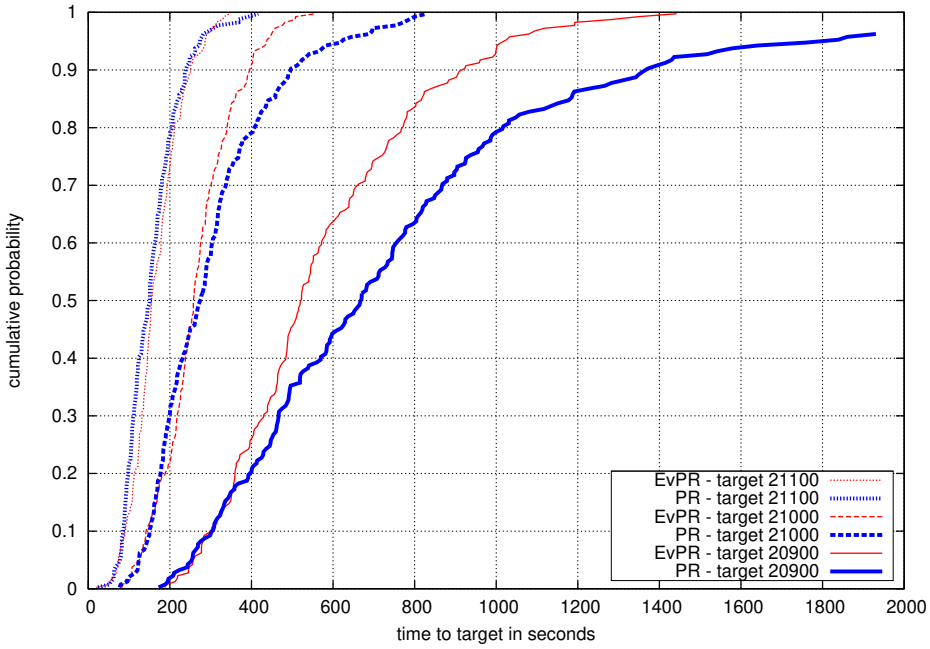


Figure 3.7: Run time distributions for GRASP with (EvPR) and without (PR) evolutionary path-relinking on instance *egl-s4-C*.

comparison, despite distinct programming languages, operating systems, and other particular configurations of each machine.

On the one hand, the GRASP is the most computer demanding metaheuristic, reaching over four minutes of CPU time per instance, on average. On the other hand, this additional time is highly compensated by presenting the best overall results in both average deviation from lower bounds ($\Delta LB = 0.33\%$) and number of best solutions ($n_{best} = 72$). In addition, GRASP was the only metaheuristic to achieve the optimal solution for every *gdb* instance. It is also worth noticing the GRASP excellent performance in the hardest set *egl* where, compared to the ACO, it reduced ΔLB from 0.9% to 0.79% and found five additional best solutions.

Table 3.8: GRASP results for *gdb* instances

			GRASP								
			mean			median			best cost		
instance	*LB*	*UB*	cost	*CPU*	*iter*	cost	*CPU*	*iter*	TS	ACO	GRASP
gdb1	316	316	316.0	0.01	2.40	316	0.01	2	**316**	**316**	**316**
gdb2	339	339	339.0	0.25	16.13	339	0.05	13	**339**	**339**	**339**
gdb3	275	275	275.0	0.02	3.67	275	0.03	3	**275**	**275**	**275**
gdb4	287	287	287.0	0.00	2.47	287	0.00	2	**287**	**287**	**287**
gdb5	377	377	377.0	0.13	10.27	377	0.14	10	**377**	**377**	**377**
gdb6	298	298	298.0	0.03	5.13	298	0.01	5	**298**	**298**	**298**
gdb7	325	325	325.0	0.00	1.80	325	0.01	1	**325**	**325**	**325**
gdb8	348	348	349.5	31.96	648.80	350	42.16	800	**348**	**348**	**348**
gdb9	303	303	303.6	38.43	752.20	303	6.49	755	**303**	**303**	**303**
gdb10	275	275	275.0	0.02	3.53	275	0.00	4	**275**	**275**	**275**
gdb11	395	395	395.0	0.87	9.53	395	0.58	9	**395**	**395**	**395**
gdb12	458	458	458.0	0.19	24.40	458	0.20	13	**458**	**458**	**458**
gdb13	536	536	542.6	6.86	490.20	544	6.46	500	540	**536**	**536**
gdb14	100	100	100.0	0.01	2.87	100	0.02	2	**100**	**100**	**100**
gdb15	58	58	58.0	0.00	1.67	58	0.00	2	**58**	**58**	**58**
gdb16	127	127	127.0	0.10	7.67	127	0.07	7	**127**	**127**	**127**
gdb17	91	91	91.0	0.00	1.40	91	0.00	1	**91**	**91**	**91**
gdb18	164	164	164.0	0.08	2.80	164	0.14	3	**164**	**164**	**164**
gdb19	55	55	55.0	0.00	3.20	55	0.00	3	**55**	**55**	**55**
gdb20	121	121	121.0	0.71	54.33	121	0.10	46	**121**	**121**	**121**
gdb21	156	156	156.0	0.42	15.93	156	0.15	13	**156**	**156**	**156**
gdb22	200	200	200.0	0.69	19.27	200	0.03	15	**200**	**200**	**200**
gdb23	233	233	234.7	36.78	1119.93	235	35.40	1100	235	235	**233**

Reported GRASP results were obtained with fifteen runs. *LB* - best known lower bound. *UB* - best known feasible solution cost.
CPU - execution time in seconds. Median *CPU* refers to the execution time for the median cost solution.
iter - number of GRASP iterations executed. Best solutions between metaheuristics in bold.

3.5 Final Remarks

This chapter has shown a high-end metaheuristic to solve the capacitated arc routing problem (CARP) and the Open CARP (OCARP), based on a greedy randomized adaptive search procedure (GRASP) with evolutionary path-relinking.

The GRASP constructive heuristic (GCH) was based on Santos et al. (2009) path-scanning heuristic with ellipse rule with many original additions which improved overall performance, such as: (i) a cost-demand edge-selection rule, self-tuned according to the instance hardness to achieve feasible solutions; (ii) a restricted candidate list containing the best candidate edges to be included in the solution; (iii) an ellipse rule, which shortens the tours in favor of solution cost; (iv) reactive parameters α and β, where the restricted candidate list parameter α and the ellipse rule parameter β had their values selected from a set of possible values based on the average solution cost induced by each of them.

In the GRASP local search phase, not all initial solutions had their neighborhood explored. A filter prevents low-quality solutions going through local search by defining a statistical threshold, which gives more than 95% probability of not throwing away an initial solution that would otherwise outperform the incumbent best. The proposed filter was demonstrated by time-to-target plots (TTT) to improve GRASP average runtime.

Table 3.9: GRASP results for *val* instances

			GRASP									
			mean			median			best cost			
instance	*LB*	*UB*	cost	*CPU*	*iter*	cost	*CPU*	*iter*	TS	VNS	ACO	GRASP
val1A	173	173	173.0	0.12	3.07	173	0.29	3	**173**	**173**	**173**	**173**
val1B	173	173	173.0	19.80	227.07	173	40.89	100	**173**	**173**	**173**	**173**
val1C	245	245	245.0	0.49	21.87	245	0.75	21	**245**	**245**	**245**	**245**
val2A	227	227	227.0	0.08	2.87	227	0.14	3	**227**	**227**	**227**	**227**
val2B	259	259	259.0	0.43	9.53	259	0.22	7	**259**	**259**	**259**	**259**
val2C	457	457	457.3	3.96	205.87	457	4.30	100	**457**	**457**	**457**	**457**
val3A	81	81	81.0	0.12	3.27	81	0.10	3	**81**	**81**	**81**	**81**
val3B	87	87	87.0	0.36	8.07	87	0.83	7	**87**	**87**	**87**	**87**
val3C	138	138	138.0	0.50	27.53	138	0.89	27	**138**	**138**	**138**	**138**
val4A	400	400	400.0	4.98	8.40	400	3.73	7	**400**	**400**	**400**	**400**
val4B	412	412	412.0	11.97	21.67	412	7.70	16	**412**	**412**	**412**	**412**
val4C	428	428	430.3	98.02	774.53	428	72.76	1000	**428**	**428**	**428**	**428**
val4D	526	530	531.0	136.58	1580.00	530	159.32	1500	**530**	**530**	**530**	**530**
val5A	423	423	423.0	4.72	8.80	423	8.21	8	**423**	**423**	**423**	**423**
val5B	446	446	446.0	3.03	8.13	446	4.39	7	**446**	**446**	**446**	**446**
val5C	470	474	474.0	98.33	1353.33	474	99.92	1400	**474**	**474**	**474**	**474**
val5D	573	575	584.5	120.90	2246.67	583	115.55	2200	583	**575**	577	581
val6A	223	223	223.0	0.53	4.33	223	0.27	4	**223**	**223**	**223**	**223**
val6B	231	233	233.0	50.72	846.67	233	45.88	800	**233**	**233**	**233**	**233**
val6C	313	317	317.0	52.86	906.67	317	50.26	900	**317**	**317**	**317**	**317**
val7A	279	279	279.0	3.56	8.67	279	4.11	7	**279**	**279**	**279**	**279**
val7B	283	283	283.0	1.31	6.00	283	0.60	6	**283**	**283**	**283**	**283**
val7C	334	334	334.0	34.37	219.47	334	19.89	200	**334**	**334**	**334**	**334**
val8A	386	386	386.0	2.04	5.67	386	1.23	5	**386**	**386**	**386**	**386**
val8B	395	395	395.0	10.69	30.07	395	5.50	15	**395**	**395**	**395**	**395**
val8C	518	521	526.5	115.43	1906.67	527	101.91	1900	529	**521**	**521**	522
val9A	323	323	323.0	65.91	30.53	323	161.37	23	**323**	**323**	**323**	**323**
val9B	326	326	326.0	52.50	31.60	326	30.00	25	**326**	**326**	**326**	**326**
val9C	332	332	332.0	139.90	91.40	332	89.11	100	**332**	**332**	**332**	**332**
val9D	385	389	392.1	246.30	1833.33	391	255.74	1800	391	**389**	391	391
val10A	428	428	428.0	73.04	26.00	428	30.48	14	**428**	**428**	**428**	**428**
val10B	436	436	436.0	211.84	86.87	436	24.54	67	**436**	**436**	**436**	**436**
val10C	446	446	446.2	211.37	333.33	446	327.91	69	**446**	**446**	**446**	**446**
val10D	525	525	530.6	314.67	2093.33	531	293.83	1800	530	**526**	**526**	527

Reported GRASP results were obtained with fifteen runs. *LB* - best known lower bound. *UB* - best known feasible solution cost.
CPU - execution time in seconds. Median *CPU* refers to the execution time for the median cost solution.
iter - number of GRASP iterations executed. Best solutions between metaheuristics in bold.

A path-relinking, whose elite solution pool progressively improves itself (*evolutionary*), was proposed based on the work of Resende et al. (2010). The proposed metaheuristic alternates GRASP iterations with the evolutionary path-relinking, in an attempt to intensify the search, while preserving some diversity. As recommended by Glover (1996), this work does not constrain the search in the feasible solution space, but also explores paths traversing the feasible/infeasible boundaries. This is accomplished by an *infeasible local search*, which reduces the cost of a locally optimum feasible solution through capacity infeasible moves. The resulting solutions are then used as initial solutions for the path-relinking. The effectiveness of evolutionary path-relinking in the metaheuristic runtime was demonstrated by TTT plots.

The computational experiments were divided into two case studies, one for the CARP and the other for OCARP. Three groups of CARP instances were used (*gdb* (Golden et al. 1983),

Table 3.10: GRASP results for *egl* instances

			GRASP									
			mean			median			best cost			
instance	*LB*	*UB*	cost	*CPU*	*iter*	cost	*CPU*	*iter*	TS	VNS	ACO	GRASP
egl-e1-A	3548	3548	3548.0	0.91	22.27	3548	0.70	21	**3548**	**3548**	**3548**	**3548**
egl-e1-B	4468	4498	4508.6	72.16	1286.67	4498	68.31	1300	4533	**4498**	**4498**	**4498**
egl-e1-C	5542	5595	5615.3	91.19	14.53	5615	80.80	1400	**5595**	**5595**	**5595**	**5595**
egl-e2-A	5011	5018	5018.0	137.02	920.00	5018	129.09	900	**5018**	**5018**	**5018**	**5018**
egl-e2-B	6280	6317	6330.7	184.26	1646.67	6334	221.54	1600	6343	6321	**6317**	**6317**
egl-e2-C	8234	8335	8335.8	181.53	1626.67	8335	169.50	1600	8347	**8335**	**8335**	**8335**
egl-e3-A	5898	5898	5898.0	63.37	378.00	5898	30.56	370	5902	**5898**	**5898**	**5898**
egl-e3-B	7697	7775	7787.3	261.09	1686.67	7787	266.53	1600	7816	**7775**	7777	7777
egl-e3-C	10163	10292	10296.5	273.30	1733.33	10292	266.80	1700	10309	**10292**	**10292**	**10292**
egl-e4-A	6395	6444	6461.1	321.18	1700.00	6464	272.68	1700	6473	6446	6456	**6444**
egl-e4-B	8884	8983	9037.1	348.97	2180.00	9038	355.22	2100	9063	9004	**8990**	9002
egl-e4-C	11427	*11559*	11670.0	439.83	1886.67	11629	431.63	2000	11627	11652	**11624**	11626
egl-s1-A	5014	5018	5038.9	65.46	706.67	5019	25.73	400	5072	**5018**	**5018**	**5018**
egl-s1-B	6379	6388	6388.4	218.12	2060.00	6388	252.94	1900	**6388**	**6388**	**6388**	**6388**
egl-s1-C	8480	8518	8521.5	187.70	1753.33	8518	171.86	1700	8535	**8518**	**8518**	**8518**
egl-s2-A	9824	9884	9980.5	1193.05	2493.33	9983	1038.93	2500	10038	9944	**9895**	9903
egl-s2-B	12968	*13088*	13240.6	1250.71	2680.00	13232	1071.18	2700	13178	**13167**	13194	13169
egl-s2-C	16353	16425	16539.9	1633.59	2793.33	16525	1220.26	2700	16505	16491	16461	**16442**
egl-s3-A	10143	10220	10276.1	1583.73	2166.67	10280	1928.24	2200	10451	10259	10249	**10221**
egl-s3-B	13616	13682	13860.7	1317.04	2420.00	13852	1377.34	2300	13981	13751	13786	**13694**
egl-s3-C	17100	*17189*	17277.7	2081.06	2733.33	17265	1759.64	2600	17346	17299	17269	**17221**
egl-s4-A	12143	12268	12406.5	1741.85	2566.67	12416	1637.19	2400	12462	12375	12324	**12297**
egl-s4-B	16093	*16267*	16432.0	2207.21	3400.00	16441	2023.98	3300	16490	16353	16428	**16333**
egl-s4-C	20375	*20484*	20660.5	3311.59	3433.33	20646	2819.00	3300	20733	20640	20595	**20563**

Reported GRASP results were obtained with fifteen runs. *LB* - best known lower bound. *UB* - best known feasible solution cost.
CPU - execution time in seconds. Median *CPU* refers to the execution time for the median cost solution.
iter - number of GRASP iterations executed. Best solutions between metaheuristics in bold.

val (Benavent et al. 1992), and *egl* (Li and Eglese 1996)), 81 in total. For the OCARP case study, the original depot was considered a common node, and the group of instances renamed as *ogdb*, *oval*, and *oegl*. In addition, each group was divided into three classes, according to the number of vehicles available, $M = M^*$, $M = M^* + 1$ and $M = M^* + 2$, where M^* represents the minimum number of vehicles for a feasible solution. The number of OCARP instances is therefore three times greater than those of CARP, totalling 243 instances.

In the OCARP case study, the proposed GRASP constructive heuristic outperformed two other path-scanning heuristics from literature (Belenguer et al. 2006; Santos et al. 2009) with respect to the overall average and maximum deviations from lower bound. Next, the complete metaheuristic solved the OCARP instances, providing optimal solutions for more than 35% of them, and reducing the overall average deviation from lower bound in more than 62%, compared to the proposed constructive heuristic.

In the CARP case study, the GCH has been shown successful to reduce the initial solutions average and maximum deviations from lower bounds with almost none additional computational effort, compared with the path-scanning with ellipse rule heuristic (Santos et al. 2009). The GRASP with PR was compared with a tabu search (Brandão and Eglese 2008), variable neighborhood search (Polacek et al. 2008), and ant colony optimization (Santos et al. 2010) metaheuristics. Results show that the GRASP outperformed all other metaheuristics with re-

Table 3.11: Summary results for CARP metaheuristics.

		TS	VNS	ACO	GRASP
CPU factor		1.4/3.0	3.6/3.0	1.0/3.0	1.0
CPU	gdb	1.2	-	1.1	5.1
	val	9.4	52.7	8.4	61.5
	egl	136.0	603.8	167.7	798.6
	overall	44.6	-	53.5	264.0
ΔLB	gdb	0.07	-	0.04	0.00
	val	0.30	0.17	0.20	0.23
	egl	2.18	0.94	0.90	0.79
	overall	0.56	-	0.40	0.33
n_{best}	gdb	22	-	22	23
	val	30	34	32	30
	egl	4	12	14	19
	overall	56	-	68	72

CPU factor - used to scale execution times.

CPU - average execution time per run in seconds.

ΔLB - average deviation from lower bound (%).

n_{best} - number of best solutions.

spect to the overall average deviation from lower bounds and number of best solutions found, in spite of additional execution time.

Chapter 4

Exact Approaches

4.1 Introduction

This chapter describes an exact algorithm to solve the open capacitated arc routing problem (OCARP). The proposed algorithm is motivated by the similarity between OCARP and a tree problem, namely the *capacity and degree constrained minimum spanning forest problem* (CDCMSFP) (Section 4.2). The branch-and-bound paradigm is used as background to guarantee optimality (Section 4.3). Three lower bounding schemes are proposed with distinct trade-offs between computational effort and bound tightness (Section 4.4). Lower bounds for CARP are discussed and also the possibility of extending them to OCARP (Section 4.5). Computational experiments are conducted on a set of literature instances, where the branch-and-bound results are compared with the ones provided by a state-of-the-art integer linear programming (ILP) solver (Section 4.6). Section 4.7 ends this chapter with the final remarks.

4.2 Motivation

In Chapter 2 the OCARP has been shown to be NP-hard. This is a negative result since it means that the complexity of an exact algorithm will not be polynomial, unless $P = NP$. Nonetheless, the extent by which OCARP instances can be optimally solved is an interesting investigation. This chapter describes an algorithm that finds an optimal solution to any OCARP instance. It will be shown that the complexity of this exact algorithm is exponential, however it performs better than solving the OCARP model using CPLEX as an ILP solver.

The branch-and-bound algorithm is based on a transformation performed on the original graph $G(V, E)$. This graph can be transformed into an equivalent full graph $\tilde{G}(\tilde{V}, \tilde{E})$ by replacing all non-required edges of G with minimum shortest paths between required edges. Figure 4.1 shows an example of this transformation with an OCARP instance containing 24 edges, six of

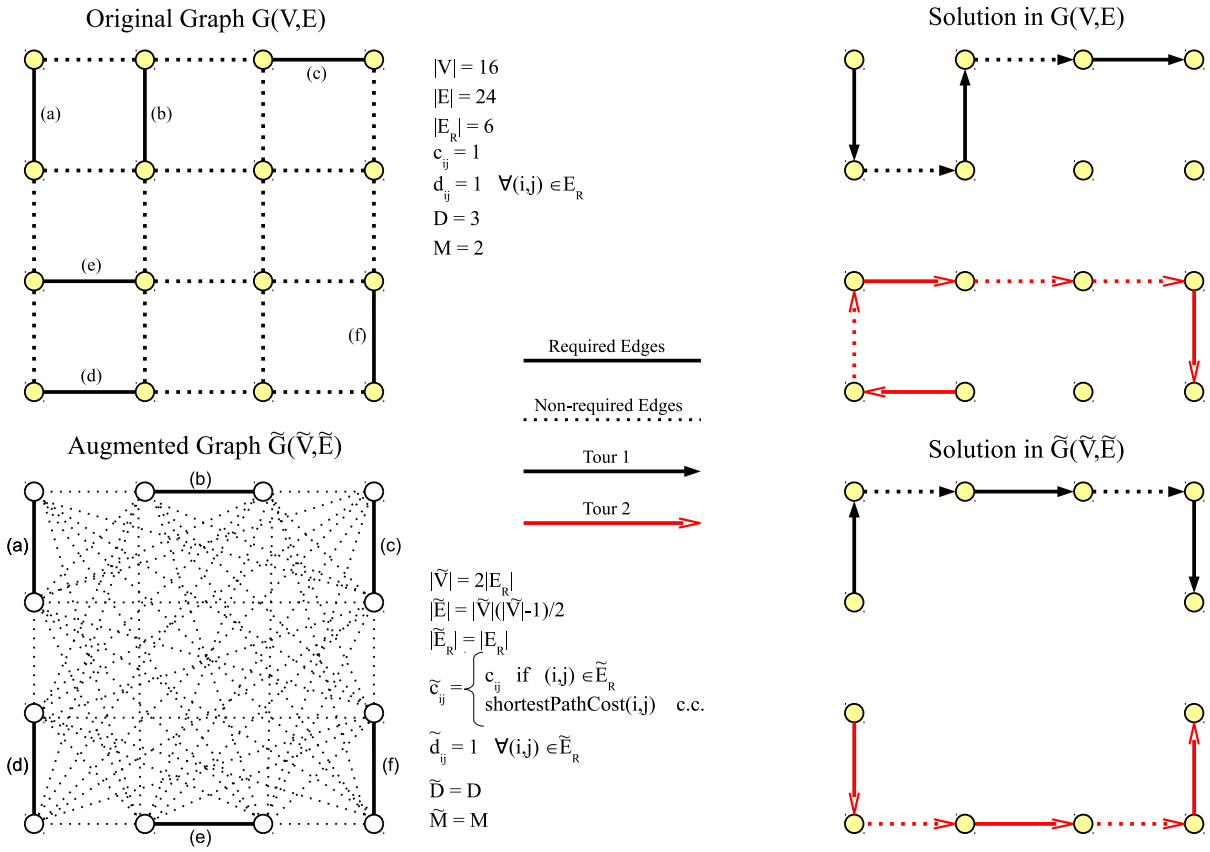


Figure 4.1: Transforming OCARP graph $G(V, E)$ into the augmented graph $\tilde{G}(\tilde{V}, \tilde{E})$.

them required. In the example, the augmented graph has 66 edges, six required (unchanged from the original graph), and 60 non-required edges connecting every pair of required edges. The costs of required edges remain the same in the augmented graph. As for the non-required edges, their costs are shortest path costs to traverse the original graph from one node to another, both nodes belonging to required edges. In case of two adjacent required edges, there will be a zero cost non-required edge connecting them in the augmented graph.

The augmented graph $\tilde{G}$ is frequently used in literature to solve the CARP (Lacomme et al. 2004; Beullens et al. 2003) since two required edges are always connected by a shortest path in an optimal solution (Property 3).

An optimal OCARP solution in G, when migrated to the augmented graph $\tilde{G}$, will represent a spanning forest $\tilde{F}(\tilde{V}, \tilde{E}_S)$. The set of edges $\tilde{E}_S$ represents the OCARP tours and it is formed by the union of required edges ($\tilde{E}_R$) and a subset of non-required edges ($\tilde{E}_{NR}$) from the augmented graph ($\tilde{E}_S = \tilde{E}_R \cup \tilde{E}_{NR}$). A spanning forest $\tilde{F}$ is considered a feasible OCARP solution when it has the following properties:

- $\tilde{F} = \left\{\tilde{T}_1, \tilde{T}_2, \ldots, \tilde{T}_M\right\}$ – one tree per OCARP vehicle.
- $|\tilde{E}_S| = |\tilde{V}| - M \Rightarrow |\tilde{E}_{NR}| = |\tilde{E}_R| - M$ – the number of non-required edges is equal to $|\tilde{E}_R| - M$.
- $1 \leqslant \delta(v) \leqslant 2 \qquad (v \in \tilde{V}, \delta(v) = \text{degree of } v)$ – all nodes have degree of 1 or 2.
- $\sum_{(i,j)\in\tilde{T}_k} d_{ij} \leqslant D \qquad (k \in \{1, 2, \ldots, M\})$ – tree demand will satisfy capacity D.

One advantage in solving the OCARP using the augmented graph $\tilde{G}$ is that the problem becomes equivalent to that of finding a minimum cost spanning forest with M trees, constrained by tree capacity and node degree. In other words, the combinatorics of the problem resides in finding the best $|R|-M$ non-required edges of minimum cost, attending these constraints. In the following, this problem is addressed as the *capacity and degree constrained minimum spanning forest problem* (CDCMSFP).

A second advantage in solving the OCARP as a minimum spanning forest is that there are easily computable lower bounds. These lower bounds will be useful within the branch-and-bound framework to solve OCARP, topic of the following section.

4.3 Branch-and-Bound General Concepts

According to Papadimitriou and Steiglitz (1982), the branch-and-bound method is based on the idea of efficiently enumerating all feasible points of the solution space of a combinatorial optimization problem. The enumeration however is implicit, since it is impracticable to evaluate all feasible solutions. One way of describing the branch-and-bound is to say that this algorithm tries to construct a proof that a solution is optimal, based on successive partitioning of the solution space. The branch in branch-and-bound refers to this partitioning process; the bound refers to lower bounds (for minimization problems) that are used to construct a proof of optimality without exhaustive search.

The search tree exploration in the branch and bound algorithm involves at least two phases: (i) *Branching* – a set of solutions, represented by a node, is partitioned into mutually exclusive subsets. Each partition subset is represented by a child node. (ii) *Lower bounding* – an algorithm is available for calculating a lower bound on the cost of any solution in a given subset.

It is possible to formulate the branch-and-bound method for any optimization problem in which (i) and (ii) are available, whether or not the objective function or constraints are linear. Papadimitriou and Steiglitz (1982) mention there are at least three implementation aspects that should be addressed beforehand:

1. *Branching scheme* – or how to partition the solution space.

2. *Node selection* – at each main iteration of the algorithm, a node must be chosen to branch from. The usual alternatives are *least-lower-bound*, *depth-first search*, or *breadth-first search*.

3. *Lower bound calculation* – in many occasions there are several lower bounding procedures with different trade-offs between bounds that are relatively tight but require relatively large computation time, and bounds that are not so tight but can be computed fast.

If an initial upper bound UB is known to the problem, it can be used to prune many nodes from the search tree, and save computational effort. A good initial upper bound can also be used to reduce the search space by removing, for example, candidate edges from the graph that could not belong to an optimal solution. An upper bound can be determined by any heuristic capable of generating a feasible solution. Again, the trade-off between the time required for the heuristic against the upper bound quality should be taken into consideration.

Depending on the problem, the branch-and-bound algorithm might have an exponential worst-case complexity, thus it is often terminated before optimality is reached, either by design or necessity. In such a case, taking the best feasible solution cost UB, and the lowest lower bound LB of the set of active node provides a ratio of $(UB - LB)/LB$ from the optimal solution cost.

Figure 4.2 shows an example of the branch-and-bound search tree expansion for a general problem starting with given $LB = 5$ and $UB = 15$. This example uses the *least-lower-bound* as the node selection criteria, and the order by which the nodes are visited is *a-b-d-h*. The search tree in this example has been developed until optimality was attained by branching the node with $LB = 10$ and $UB = 12$. After this branch, the active nodes are all checked and the global bounds updated with the smallest lower and upper bounds. In this case they are both equal ($LB = UB = 11$). This means optimality has been attained and the search stops.

4.4 Branch-and-Bound for the CDCMSFP

4.4.1 Implementation Decisions

The branch-and-bound implementation decisions regarding the branching scheme, node selection and lower bound calculations are described.

First of all, a solution of the CDCMSFP, partial or complete, is represented by the set of required and non-required edges of $\tilde{G}$ the solution contains.

With respect to the branching scheme, Figure 4.3 assists the comprehension of how the solution space is partitioned. Consider that all non-required edges $\tilde{E} \setminus \tilde{E}_R$ are sorted in a list $L = \{e_1, e_2, \ldots, e_n\}$ by increasing order of cost. In each level of the search tree, a non-required edge is included into the partial solution from the level above. Thus each search tree node containing a subset of non-required edges is branched to include one additional non-required

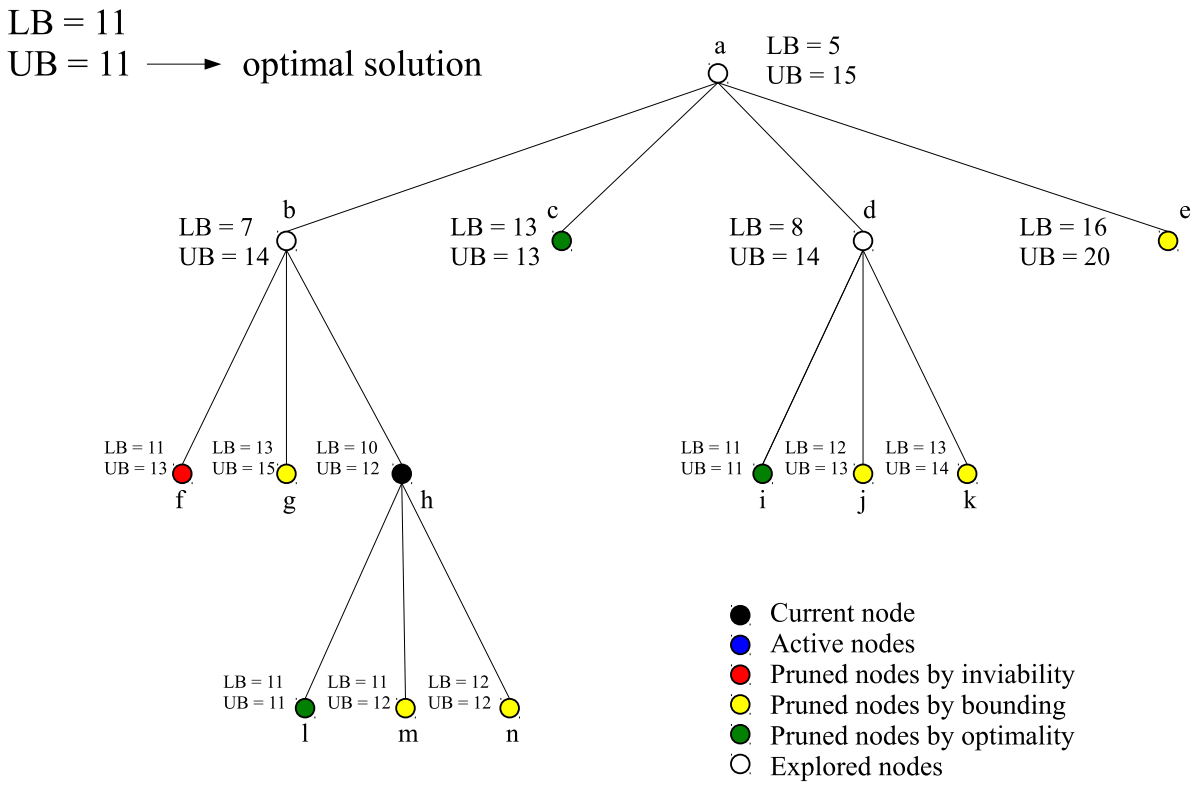


Figure 4.2: Example of the branch-and-bound search tree expansion.

edge. The branching scheme takes the highest index i from the most costly edge e_i in the current search tree node, as long as $i < n$, and divides the solution space into $n - i$ mutually exclusive partitions. In each partition, an edge e_j ($\forall j \in [i+1, n]$) is included in the corresponding partial solution.

The node selection scheme adopted in the proposed branch-and-bound algorithm is the least-lower-bound. One advantage of this scheme is that the current node lower bound is also a global lower bound, since it is, by definition, the smallest lower bound among all active nodes. In addition, if the current node represents a complete feasible solution, then it must be optimal. Another advantage relies on the fact that this scheme favors the improvement of the global lower bound against the global upper bound. This may be viewed as negative as well, because complete solutions are discovered less often than using, for example, the depth-first search.

As for the lower bound calculation, the proposed algorithm uses three procedures detailed in the following section.

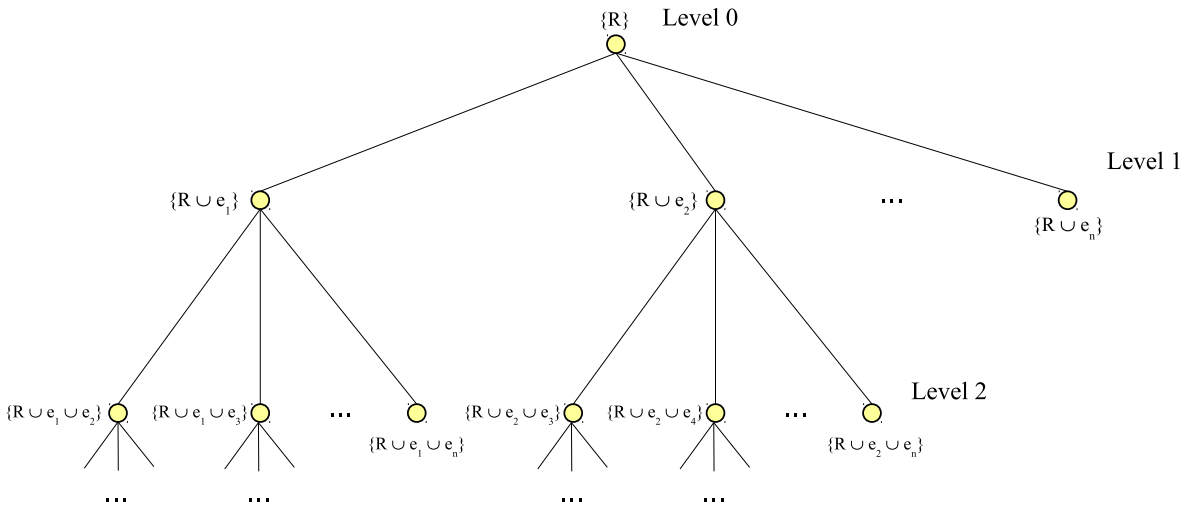


Figure 4.3: Branch-and-bound search tree for OCARP.

4.4.2 Lower Bounds

In the developed branch-and-bound algorithm, three lower bounds were tested, with different trade-offs of complexity and computational effort, and this section shows how these lower bounds were combined with the algorithm. Assume that $\tilde{E}_{PS}$ is the subset of edges representing a given partial CDCMSFP solution, $L = \{e_1, e_2, \ldots, e_n\}$ is a sorted list of all non-required edges $\tilde{E} \setminus \tilde{E}_R$ in increasing order of cost, and I_L is the list of the indices for each edge in L ($I_L[e_i] = i$). The following describes three proposed lower bounds.

LB_1 – minimum cost edges

Lower bound LB_1 is computed by relaxing the illegal subcycle, degree and capacity constraints of the CDCMSFP. Supposing $e_{\max}$ is the maximum cost edge in $\tilde{E}_{PS}$, then LB_1 can be calculated by adding to the cost of the partial solution, the cost of the cheapest n_{me} edges in L starting from index $I_L[e_{\max}] + 1$, where $n_{me} = |\tilde{E}_R| - M - |\tilde{E}_{PS}|$ is the number of missing edges to complete the solution.

Lower bound LB_1 can be computed in $O(1)$, by simply fetching the missing edges cost from a matrix $C_{m \times n}$ ($m = |\tilde{E}|$, $n = |\tilde{E}_R| - M$). The pre-processing phase of constructing matrix C requires $O(|\tilde{E}||\tilde{E}_R|)$ of computational effort, however the pre-processing is performed only once, while LB_1 is calculated numerous times during execution. The row index of matrix C represents the edge index in list L from which to start the sum of the missing edges costs, and this index should be the highest cost edge index from the partial solution plus one ($I_L[e_{\max}] + 1$). The column index of matrix C represents the number of missing edges (n_{me}). Equation 4.1 shows how to fill matrix C for any OCARP instance.

$$C[i,j] = \begin{cases} \sum_{k=i}^{i+j-1} cost(L[k]) & \text{if } i+j-1 \leqslant |\tilde{E}| \\ \infty & \text{otherwise} \end{cases} \tag{4.1}$$

Algorithm 10 shows how to determine LB_1 for a given partial solution $\tilde{E}_{PS}$.

Algorithm 10 LB1 – minimum cost edges lower bound

Input: $\tilde{E}_{PS}$ – set of non-required edges from the partial solution
Output: LB_1 – minimum cost edges lower bound
1: $e_{\max} \leftarrow$ getMaxCostEdge($\tilde{E}_{PS}$) – returning the highest cost edge in $\tilde{E}_{PS}$
2: $n_{me} \leftarrow |\tilde{V}| - M - |\tilde{E}_{PS}|$ – returning the number of missing edges in $\tilde{E}_{PS}$
3: $i \leftarrow I_L[e^{\max}] + 1$ – determining the row of matrix C
4: $j \leftarrow n_{me}$ – determining the column of matrix C
5: $LB_1 \leftarrow cost(\tilde{E}_{ps}) + C[i,j]$
6: **return** LB_1

LB_2 – minimum cost spanning forest

Lower bound LB_2 is computed by relaxing the degree and capacity constraints of the CDCMSFP. In other words, lower bound LB_2 computes the minimum cost spanning forest that completes the partial solution, and this is done by adding the cheapest $n_{me} = |E_R| - M - |\tilde{E}_{ps}|$ edges which do not form a cycle in the solution (Algorithm 11). This is done within almost-linear time $O(|\tilde{E}|\alpha(\tilde{V}))$ with Kruskal algorithm implemented using disjoint-set-forest and union-by-rank (Cormen, Stein, Rivest and Leiserson 2001). For all practical purposes, the inverse Ackermann function regards the inequality $\alpha(.) \leqslant 5$. The algorithm halts once the solution complies $|E_R| - M$ non-required edges (or equivalently M trees).

Algorithm 11 LB2 – minimum cost spanning forest lower bound

Input: $\tilde{E}_{PS}$ – set of non-required edges from the partial solution
Output: S – optimal spanning forest
1: $e_{\max} \leftarrow$ getMaxCostEdge($\tilde{E}_{PS}$) – returning the highest cost edge in $\tilde{E}_{PS}$
2: $n_{me} \leftarrow |\tilde{V}| - M - |\tilde{E}_{PS}|$ – returning the number of missing edges in $\tilde{E}_{PS}$
3: $S \leftarrow \tilde{E}_{PS}$
4: $i \leftarrow I_L[e^{\max}] + 1$
5: **while** $(n_{me} > 0)$ **do**
6: $\quad e \leftarrow L[i]$
7: $\quad$ **if** (containsCycle($S \cup \{e\}$) = **false**) **then**
8: $\quad\quad S \leftarrow S \cup \{e\}$
9: $\quad\quad n_{me} \leftarrow n_{me} - 1$
10: $\quad$ **end if**
11: $\quad i \leftarrow i + 1$
12: **end while**
13: **return** S

LB_3 – degree constrained minimum cost spanning forest

Lower bound LB_3 is computed by relaxing only the capacity constraints of the CDCMSFP. This lower bound is based on a heuristic for the *degree constrained minimum spanning tree problem* (DCMSTP), by Andrade, Lucena and Maculan (2006), which uses Lagrangian dual information to derive the lower bounds. Let $E(S) = \{(i,j) \in E | i \in S, j \in S\}$ be the set of edges contained in the node set S, and $N(i) = \{j \in V | (i,j) \in E\}$ the set of nodes adjacent to node i. Consider the following ILP model for the DCMSTP.

(DCMSTP)

$$MIN \quad Z = \sum_{(i,j)\in E} c_{ij} x_{ij} \tag{4.2}$$

st

$$\sum_{(i,j)\in E} x_{ij} = |V| - 1 \tag{4.3}$$

$$\sum_{(i,j)\in E(S)} x_{ij} \leqslant |S| - 1 \qquad \forall S \subset V \tag{4.4}$$

$$\sum_{j\in N(i)} x_{ij} \leqslant d_i \qquad \forall i \in V \tag{4.5}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in E \tag{4.6}$$

The objective function (4.2) minimizes the solution total cost; constraint (4.3) forces $|V|-1$ edges in the solution; constraints (4.4) are for illegal subcycle elimination; constraints (4.5) limit the degrees of the nodes. If the problem is to generate a forest (more than one tree), then the right-hand side of constraint (4.3) should be replaced by $|V| - t$, where t is the number of desired trees ($t = M$ in this work). The Lagrangian relaxation used by Andrade et al. (2006) and equally in this work, is obtained by relaxing contraints (4.5), which turns into the following relaxed λ-DCMSTP formulation.

(λ-DCMSTP)

$$MIN \quad Z_\lambda = \sum_{(i,j)\in E} c_{ij}x_{ij} - \sum_{i\in V} \lambda_i \left(d_i - \sum_{j\in N(i)} x_{ij} \right) \tag{4.7}$$

st

$$\sum_{(i,j)\in E} x_{ij} = |V| - 1 \tag{4.8}$$

$$\sum_{(i,j)\in E(S)} x_{ij} \leqslant |S| - 1 \qquad \forall S \subset V \tag{4.9}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in E \tag{4.10}$$

Equation 4.11 is obtained by algebrically manipulating the objective function 4.7.

$$\begin{aligned} Z_\lambda &= \sum_{(i,j)\in E} c_{ij}x_{ij} - \sum_{i\in V} \lambda_i \left(d_i - \sum_{j\in N(i)} x_{ij} \right) \\ &= \sum_{(i,j)\in E} c_{ij}x_{ij} + \sum_{i\in V} \lambda_i \left(\sum_{j\in N(i)} x_{ij} \right) - \sum_{i\in V} \lambda_i d_i \\ &= \sum_{(i,j)\in E} c_{ij}x_{ij} + \sum_{(i,j)\in E} (\lambda_i + \lambda_j) x_{ij} - \sum_{i\in V} \lambda_i d_i \\ &= \sum_{(i,j)\in E} (c_{ij} + \lambda_i + \lambda_j) x_{ij} - \sum_{i\in V} \lambda_i d_i \end{aligned} \tag{4.11}$$

This way, the relaxed problem is to find the minimum spanning tree with Lagrangian costs $l_{ij} = c_{ij} + \lambda_i + \lambda_j$. From the Lagrangian duality theory, it is straightforward to establish that the optimal cost Z^*_λ is a lower bound for the optimal cost Z^* for any given set of multipliers λ. There is also the Lagrangian dual problem, which is to find the set of multipliers λ^* which maximizes Z^*_λ, in other words, finds the maximum lower bound for Z^*. To solve the Lagrangian dual the subgradient method (Beasley 1993), which generates a sequence of multipliers which converge to λ^*, is employed.

Algorithm 12 computes lower bound LB_3 by solving several times a minimum cost spanning forest with Lagrangian costs on the edges using Algorithm 11. Both of the parameters θ and τ regarding the step size have been set with the values suggested by Beasley (1993) (lines 1 and 2). The Lagrangian multipliers are initially set with zeros (line 5) so that in the first iteration the Lagrangian costs are equal to the original costs. In line 9, the costs of the edges are updated by the Lagrangian costs ($l_{ij} = c_{ij} + \lambda_i + \lambda_j$). The extents of node degree infeasibilities of the spanning forest S_λ generated in line 10 is measured by the subgradients (line 15), which are used to determine the step size (line 18) for the Lagrangian multipliers adjustment (line 23).

Algorithm 12
LB3 – degree constrained minimum cost spanning forest lower bound

Input: $\tilde{E}_{PS}$ – set of non-required edges from the partial solution, UB – an upper bound for the DCMSTP, obtained heuristically, $maxIter$ – maximum number of iterations, used as stopping criterion
Output: LB_3 – the best lower bound obtained by solving the dual Lagrangian problem

$\theta \leftarrow 2.0$ – adjusts the step size depending on how tight is the GAP between LB and UB bounds
$\tau \leftarrow 1.05$ – prevents the step size from becoming to small when the same happens with the GAP
$iter \leftarrow 0$
for $((\forall i \in V))$ **do**
    $\lambda_i \leftarrow 0$
end for
$LB_3 \leftarrow -\infty$
while $((iter \leqslant maxIter)$ and $(UB > LB_3))$ **do**
    updateEdgeCosts$(\tilde{E}_{PS}, \lambda)$ – apply Lagrangian costs to the edges
    $S_\lambda \leftarrow$ LB2$(\tilde{E}_{PS})$ – solve the (λ-DCMSTP) with Kruskal
    if $(cost(S_\lambda) > LB_3)$ **then**
        $LB_3 \leftarrow cost(S_\lambda)$
    end if
    for $(\forall i \in V)$ **do**
        $G_i \leftarrow d_i -$ nodeDegree(S_λ, i)
    end for
    if $(\sum_{i \in V} G_i > 0)$ **then**
        $t \leftarrow \dfrac{\theta(\tau UB - LB_3)}{\sum_{i \in V} G_i^2}$
    else
        return LB_3 – optimal solution cost
    end if
    for $(\forall i \in V)$ **do**
        $\lambda_i \leftarrow \max(0, \lambda_i + tG_i)$
    end for
    $iter \leftarrow iter + 1$
end while
return LB_3

4.4.3 Lower Bounds Tightness

A trivial lower bound LB_0 for an optimal OCARP solution cost may be computed by summing all required edges costs ($LB_0 = \sum_{e \in E_R} c_e$). This however can be very loose when optimal solutions require deadheading. The following paragraphs will show that $LB_3 \geqslant LB_2 \geqslant LB_1 \geqslant LB_0$.

1. $LB_1 \geqslant LB_0$: Lower bound LB_1 considers the cost of the partial solution ($\tilde{E}_{PS}$), which in turn contains all required edges ($E_R \subseteq \tilde{E}_{PS}$).

2. $LB_2 \geqslant LB_1$: Both lower bounds LB_1 and LB_2 are formed by adding the same number of edges (n_{me}) to the partial solution. However, LB_1 always adds the least cost edges.

3. $LB_3 \geqslant LB_2$: The Lagrangian multipliers in Algorithm 12 are initially set with zeros, meaning that the first minimum cost spanning forest obtained in line 10 will have cost equal to LB_2.

4.4.4 Lower Bounds Composition

Empirical studies have revealed that the branch-and-bound algorithm performs better when the lower bounds are combined, rather than used individually. This is because there are moments during the search tree exploration when a fast computable lower bound is preferred instead of a costly one, despite this last being a tighter bound.

As explained in Section 4.4.1, for every child node generated in the branching phase a non-required edge is added to the corresponding partial solution. In the proposed branch-and-bound, these child nodes are formed in an orderly manner of decreasing costs, starting with a child node that receives the highest cost non-required edge. This denotes that the first child nodes should likely contain a relatively high cost partial solutions. Thus lower bounds for these solutions do not need to be tight, and fast lower bounds would save substantial amounts of execution time.

Let LB and UB be the incumbent best lower and upper bounds at a given branch-and-bound iteration. At the start of a branching phase, the lower bounding procedure of choice is $LB1$, the fastest one. If a child node results in a lower bound $LB1$ such that $LB1 < LB + (UB - LB)/10$, then the lower bounding procedure is replaced by $LB2$. This test is repeated for every generated child node, and if $LB2$ also fails to pass the test, it is replaced by $LB3$, which assumes the job of deriving lower bounds until the last child node is created in the course of that branching phase. The choice of using the relative value $(UB - LB)/10$ was made after several empirical tests.

4.4.5 Upper Bounds

Considering the quality of the GRASP with path-relinking described in Chapter 3, this same metaheuristic was used to provide the branch-and-bound algorithm with a high-quality starting

upper bound.

It was considered during the implementation of the branch-and-bound algorithm to develop a constructive heuristic which takes a partial solution, complete a feasible solution, and then improve it with some local search procedure. This idea was rejected afterwards, once it has been verified that the quality of the proposed GRASP with path-relinking was a good enough metaheuristic, and that it would be a waste of effort to dedicate the branch-and-bound time to pursue better upper bounds.

4.4.6 Problem Reduction

In order to save computational effort, a problem reduction is performed so that non-required edges, with sufficiently high costs, can be eliminated from the instance graph without loss of optimality. For each non-required edge $e \in \tilde{E}$, a lower bound LB_e is computed by forcing edge e into the solution. If $LB_e > UB$ then edge e cannot take part in any optimal solution, thus can be removed from the instance graph.

The problem reduction method uses all three lower bounds, and start with the weakest lower bound LB_1, which is computed for each edge in decreasing order of cost. It continues using LB_1 while it keeps removing the edges from the graph instance ($LB_1 > UB$). When $LB_1 \leqslant UB$, lower bound LB_2 is activated and it is used as the lower bounding procedure until $LB_2 \leqslant UB$, when it is replaced by LB_3 which is used until all edges are considered.

4.4.7 Branch-and-Bound Pseudo-Code

Algorithm 13 shows the proposed exact algorithm to solve the CDCMSFP, and begins by creating a node representing the partial solution containing all and only the required edges. This first node is then added to the heap (line 3). Before starting the search, a reduction of the problem is attempted by eliminating some candidate non-required edges (line 4). The main loop of the algorithm is located in line 5, and it halts when the algorithm finds the optimal solution or a sufficient amount of time has ellapsed or the heap size exceeds a given threshold. The search commences by extracting a heap node (line 6). The lower bound is updated (line 7), and tests are performed to verify if optimality has been attained (lines 8 and 12). The branching phase begins in the loop of line 16. If the partial solution remains feasible after the addition of a non-required edge (line 17), it then goes through one or more lower bounding procedures (lines 19, 21, and 23), according to the strategy described in Section 4.4.4.

4.4.8 Complexity Study

The computational complexity of Algorithm 13 can be bounded by the number of nodes in the tree. This can be quantified with support of Figure 4.3. Each i-th level of this tree contains

Algorithm 13 Branch-and-Bound for the CDCMSFP

Input: $G(V, E)$ – undirected complete graph, UB – an upper bound for the DCMSTP, obtained heuristically, $timeLimit$ – execution time limit, used as stopping criterion, $heapSize$ – memory usage limit, used as stopping criterion.

1: $\tilde{E}_{PS} \leftarrow E_R$ – setting an initial solution containing only the required edges
2: $z_0 \leftarrow cost(\tilde{E}_{PS})$ – trivial lower bound for the empty solution
3: insertHeap($activeNodes, \tilde{E}_{PS}, z_0$) – inserting partial solution $\tilde{E}_{PS}$ with key z_0 in the heap $activeNodes$
4: problemReduction($G(V, E), UB$) – reducing the number of candidate edges
5: **while** (($activeSet \neq \varnothing$) and ($time \leqslant timeLimit$) and ($activeSet.size \leqslant heapSize$)) **do**
6: $(\tilde{E}_{PS}, z) \leftarrow$ extractMinHeap($activeNodes$)
7: $LB \leftarrow z$
8: **if** ($UB - LB < 1$) **then**
9: $LB \leftarrow UB$
10: **exit while**
11: **end if**
12: **if** (completeSolution($\tilde{E}_{PS}$) = **true**) **then**
13: $UB \leftarrow LB$
14: **exit while**
15: **end if**
16: **for** ($\forall e \in \tilde{E}_{PS} \setminus E_R$) **do**
17: $S_{child} \leftarrow \tilde{E}_{PS} \cup \{e\}$
18: **if** (feasible(S_{child}) = **true**) **then**
19: $z_{child} \leftarrow$ LB1(S_{child})
20: **if** ($z_{child} \leqslant LB + \frac{UB - LB}{10}$) **then**
21: $z_{child} \leftarrow$ LB2(S_{child})
22: **if** ($z_{child} \leqslant LB + \frac{UB - LB}{10}$) **then**
23: $z_{child} \leftarrow$ LB3(S_{child})
24: **end if**
25: **end if**
26: **if** ($z_{child} \leqslant UB$) **then**
27: insertHeap($activeNodes, S_{child}, z_{child}$)
28: **end if**
29: **end if**
30: **end for**
31: **end while**
32: **return** (LB, UB)

all possible solutions (not necessarily feasible) with i non-required edges. Therefore, the total amount of nodes in the search tree (N_{tree}) corresponds to the sum of nodes in all its $|E_R| - M$ levels (Equation (4.12)).

$$N_{tree} = \sum_{i=0}^{|E_R|-M} \binom{|\tilde{E}|}{i} < \sum_{i=0}^{|\tilde{E}|} \binom{|\tilde{E}|}{i} = 2^{|\tilde{E}|} \tag{4.12}$$

Each node in the search tree goes through a heap insertion, a heap deletion, and possibly the calculation of all three lower bounds LB_1, LB_2, and LB_3. Thus the computational effort performed on each node is bounded by $O(|\tilde{E}|\alpha(\tilde{V}))$. These calculations are performed N_{tree} times, making the computational complexity of the proposed method to be bounded by $O(|\tilde{E}|\alpha(\tilde{V})2^{|\tilde{E}|})$.

4.5 Alternative Lower Bounds

Welz (1994) has observed that the linear relaxation of the CARP model which excludes the subtour elimination constraints (2.6), always provides a lower bound equal to LB_0. This is extendable to OCARP, i.e., the linear relaxation of the OCARP model, excluding the subtour elimination constraints (2.16), generates a lower bound equal to LB_0 (the proof is equivalent to the one given for CARP (Welz 1994)).

Lower bounding schemes for CARP, such as *Capacity constraints*, *Odd Edge Cutset constraints*, *Disjoint Paths* (Belenguer and Benavent 2003), and *Multiple Cuts Node Duplication Lower Bound* (Wøhlk 2006) affirm that, for a subset $S \subseteq V \setminus \{v_0\}$, some edges must be deadheaded depending on the total demand to be serviced in S, the number of required edges in the cutset $(S, S \setminus V)$, and possibly some additional demand on the path from v_0 to S. However, these constraints are valid based on the fact that every CARP vehicle which enters subset S must exit from it to return to the depot. Since OCARP tours may start and end at any node, these inequalities are not valid for OCARP.

Two attempts were made to improve LB_0 using the optimization software CPLEX 12. The first one was by solving OCARP linear relaxation model including only part ($S \subseteq V, |S| \leqslant 3$) of the subtour elimination constraints (2.16). This strategy however did not improve LB_0. The second approach consisted in solving the integer reduced OCARP model, neglecting all subtour elimination constraints. The stopping criteria were defined by execution time (one hour), memory usage (2.5 GB) or optimality. When one of these criteria was reached, the best lower bound obtained in the process (LB_{cplex}) was stored.

4.6 Computational Experiments

The branch-and-bound algorithm was tested with the same instances used in the GRASP with PR computational tests (Section 3.4). The standard set of CARP instances[1] was referred to, which includes 23 *gdb* (7-27 nodes, 11-55 edges) (Golden et al. 1983), 34 *val* (24-50 nodes, 34-97 edges) (Benavent et al. 1992), 24 *egl* (77-140 nodes, 98-190 edges) (Li and Eglese 1996), totaling 81 instances. The depot was considered a common node while the rest of the data left intact, leading to three groups of instances referred as *ogdb*, *oval*, and *oegl*.

Since OCARP is only meaningful with a fixed M (Property 1), the computational tests have considered three classes of parameterization: $M = M^*$, $M = M^* + 1$ and $M = M^* + 2$, where M^* represents the minimum number of vehicles necessary for a feasible solution. Consequently, from each of the 81 CARP instances, three different numbers of vehicles are considered, thus deriving 243 OCARP instances.

Table 4.1 lists the branch-and-bound parameters and their values used in the computational experiments. The same time limit was adopted to CPLEX. The branch-and-bound algorithm was implemented in C programming language, and all tests were executed in a Intel Core 2 Quad 3.0 GHz with 4 Gb of RAM.

The computational experiments use as the starting upper bound (UB) the best solution obtained by the GRASP with PR, presented in Tables 3.4-3.6.

Table 4.1: Branch-and-bound parameters.

$timeLimit = 3600$	execution time limit in seconds.
$heapSize = 2 \times 10^7$	heap size limit in number of nodes.
$\lambda_0 = [0, 0, \ldots, 0]$	initial Lagrangian multipliers for the subgradient method.
$\theta = 2.0$	adjusts the step size depending on the GAP in the subgradient method.
$\tau = 1.05$	prevents the step size from becoming too small in the subgradient method.

Table 4.2 reveals the overall results of the branch-and-bound algorithm for each group-class of instances. These results inform on the quality of the branch-and-bound lower bounds (LB_{bb}), which are compared with the integer reduced model lower bounds obtained by CPLEX (LB_{cplex}). The trivial lower bounds (LB_0) and the best lower bounds ($LB_{best} = \max\{LB_{cplex}, LB_{bb}\}$) are also discriminated for a thorough comparison analysis.

Considering the average deviation from lower bound ($\Delta LB = \frac{UB-LB}{LB}$) in Table 4.2, both LB_{bb} and LB_{cplex} performed equally to group of instances *ogdb*, not improving the trivial bounds. However, this group has only three instances yet to attain optimality, leaving a small margin for improvements. Considering group *oval*, the branch-and-bound yielded minor improvements to the trivial lower bounds, with a gap reduction from 2.45% to 2.37%. On the one hand, CPLEX performed much better for this group, significantly reducing the deviation to 1.62%,

[1] http://www.uv.es/~belengue/carp.html

and matching the best lower bounds deviation ($LB_{cplex} = LB_{best}$) for all instances in this group. On the other hand, the branch-and-bound reduced the trivial lower bound deviation from 12.15% to 9.88%, outperforming CPLEX that handed a 11.63% gap. Moreover, the branch-and-bound had the best performance in the overall comparison, achieving an average deviation of 3.94%, while CPLEX yielded an average deviation of 4.14%, and the trivial deviation was 4.64%.

Table 4.2 also shows the number of best lower bounds (n_{best}) for each group-class of instances. These numbers draw practically the same conclusions as the average deviations from lower bounds. That is, for group *ogdb*, methods branch-and-bound and CPLEX performed equally; for group *oval*, CPLEX yielded 102 best lower bounds against the branch-and-bound 77 best lower bounds; for group *oegl*, the branch-and-bound outperformed CPLEX with 72 to 33 best lower bounds; lastly, for the overall comparison, the branch-and-bound delivered the highest number of best lower bounds 218, while CPLEX provided only 204.

The performance complementarity between both algorithms has been registered by the overall best lower bounds deviations (3.62%) and number of best lower bounds (243), which are significantly better than the results presented by each algorithm alone. One cause of this complementarity could be the absense of capacity lower bounding schemes in the branch-and-bound algorithm. It may happen that the *oval* instances are more susceptible to capacity constraints than node degree constraints. If this is the case, the OCARP model capacity constraints should have benefited CPLEX. Still, the reduced integer linear model was too onerous to solve the *oegl* instances, which explains CPLEX poor performance to this group.

Tables 4.3-4.5 shows the branch-and-bound results for all instances individually. The memory stopping criterion was not triggered for any instance. From the set of 243 instances, 92 solutions (37.86%) were proven optimal ($UB = LB_{best}$).

The only way the branch-and-bound algorithm could improve the initial upper bounds was by fetching a complete solution from the heap, and this solution would already be optimal for the reasons given in Section 4.3. This event only happened during the experiments when the initial upper bound was already optimal. Therefore, the branch-and-bound has not improved any of the initial upper bounds, and this is likely due to the already high-quality bounds provided by the GRASP with PR metaheuristic. Besides, the branch-and-bound was engineered with a least-lower-bound branching scheme that favors the search for higher lower bounds over lower cost feasible solutions, which in turn is supported by other approaches, for example, a depth-first search.

Table 4.2: Branch-and-bound overall results.

		ΔLB				n_{best}			
group	class	ΔLB_0	ΔLB_{cplex}	ΔLB_{bb}	ΔLB_{best}	LB_0	LB_{cplex}	LB_{bb}	LB_{best}
ogdb	M^*	0.07	**0.07**	**0.07**	0.07	23	**23**	**23**	23
	M^*+1	0.05	**0.05**	**0.05**	0.05	23	**23**	**23**	23
	M^*+2	0.02	**0.02**	**0.02**	0.02	23	**23**	**23**	23
	overall	0.05	**0.05**	**0.05**	0.05	69	**69**	**69**	69
oval	M^*	3.61	**1.66**	3.41	1.66	16	**34**	16	34
	M^*+1	2.27	**1.75**	2.24	1.75	28	**34**	28	34
	M^*+2	1.46	**1.46**	**1.46**	1.46	33	**34**	33	34
	overall	2.45	**1.62**	2.37	1.62	77	**102**	77	102
oegl	M^*	20.22	19.09	**17.13**	17.13	10	10	**24**	24
	M^*+1	9.37	9.08	**7.17**	7.17	11	11	**24**	24
	M^*+2	6.87	6.73	**5.34**	5.34	12	12	**24**	24
	overall	12.15	11.63	**9.88**	9.88	33	33	**72**	72
overall	M^*	7.53	**6.37**	6.53	5.79	49	**67**	63	81
	M^*+1	3.74	3.44	**3.08**	2.87	62	68	**75**	81
	M^*+2	2.66	2.61	**2.21**	2.20	68	69	**80**	81
	overall	4.64	4.14	**3.94**	3.62	179	204	**218**	243

$LB_0 = \sum_{e \in R} c_e$: trivial lower bound. LB_{cplex}: cplex lower bounds.
LB_{bb}: branch-and-bound lower bounds. $\Delta LB_{best} = \min\{\Delta LB_{cplex}, \Delta LB_{bb}\}$: best lower bound.
ΔLB: average deviation from lower bound (%). n_{best}: number of best lower bounds.
M^*: minimum number of vehicles to attain a feasible solution.

4.7 Final Remarks

An algorithm based on the branch-and-bound paradigm was developed to solve OCARP optimally. Three lower bounding schemes were derived, regarding the number of edges (LB_1), the absense of cycles (LB_2) and the node degree constraints (LB_3). The computational complexity of the algorithm is exponential, meaning that the execution time may be too high for realistic size instances. Nonetheless, computational experiments were performed halting the algorithm after a given execution time. The best bounds encountered in the process are returned as result. These bounds were compared with the bounds generated by solving OCARP model with a high-end ILP solver. The results show that the branch-and-bound algorithm outperformed CPLEX in the overall comparison.

Table 4.3: Branch-and-bound results for *ogdb* instances.

			$M = M^*$				$M = M^* + 1$				$M = M^* + 2$			
	M^*	LB_0	UB	LB	ΔLB	CPU	UB	LB	ΔLB	CPU	UB	LB	ΔLB	CPU
ogdb1	5	252	**252**	252	0.00	0.00	**252**	252	0.00	0.00	**252**	252	0.00	0.01
ogdb2	6	291	**291**	291	0.00	0.00	**291**	291	0.00	0.00	**291**	291	0.00	0.00
ogdb3	5	233	**233**	233	0.00	0.00	**233**	233	0.00	0.00	**233**	233	0.00	0.00
ogdb4	4	238	**238**	238	0.00	0.00	**238**	238	0.00	0.00	**238**	238	0.00	0.00
ogdb5	6	316	**316**	316	0.00	0.00	**316**	316	0.00	0.00	**316**	316	0.00	0.00
ogdb6	5	260	**260**	260	0.00	0.00	**260**	260	0.00	0.00	**260**	260	0.00	0.00
ogdb7	5	262	**262**	262	0.00	0.00	**262**	262	0.00	0.00	**262**	262	0.00	0.00
ogdb8	10	210	**210**	210	0.00	0.00	**210**	210	0.00	0.00	**210**	210	0.00	0.00
ogdb9	10	219	**219**	219	0.00	0.00	**219**	219	0.00	0.00	**219**	219	0.00	0.00
ogdb10	4	252	**252**	252	0.00	0.00	**252**	252	0.00	0.00	**252**	252	0.00	0.00
ogdb11	5	356	362	356	1.69	3600.50	360	356	1.12	3600.53	358	356	0.56	3600.43
ogdb12	7	336	**336**	336	0.00	0.01	**336**	336	0.00	0.01	**336**	336	0.00	0.01
ogdb13	6	509	**509**	509	0.00	0.00	**509**	509	0.00	0.00	**509**	509	0.00	0.00
ogdb14	5	96	**96**	96	0.00	0.00	**96**	96	0.00	0.00	**96**	96	0.00	0.00
ogdb15	4	56	**56**	56	0.00	0.00	**56**	56	0.00	0.00	**56**	56	0.00	0.00
ogdb16	5	119	**119**	119	0.00	0.00	**119**	119	0.00	0.00	**119**	119	0.00	0.00
ogdb17	5	84	**84**	84	0.00	0.00	**84**	84	0.00	0.00	**84**	84	0.00	0.00
ogdb18	5	158	**158**	158	0.00	0.00	**158**	158	0.00	0.00	**158**	158	0.00	0.00
ogdb19	3	45	**45**	45	0.00	0.00	**45**	45	0.00	0.00	**45**	45	0.00	0.00
ogdb20	4	105	**105**	105	0.00	0.00	**105**	105	0.00	0.00	**105**	105	0.00	0.00
ogdb21	6	149	**149**	149	0.00	0.00	**149**	149	0.00	0.00	**149**	149	0.00	0.00
ogdb22	8	191	**191**	191	0.00	0.00	**191**	191	0.00	0.00	**191**	191	0.00	0.00
ogdb23	10	223	**223**	223	0.00	0.00	**223**	223	0.00	0.00	**223**	223	0.00	0.00

M^*: minimum number of vehicles to attain a feasible solution. LB_0: trivial lower bound.
UB: solution obtained by the GRASP with PR. LB: best lower bound provided by the branch-and-bound algorithm.
ΔLB: average deviation from lower bound (%). CPU: running time (s) of the branch-and-bound algorithm.

Table 4.4: Branch-and-bound results for *oval* instances.

			$M = M^*$				$M = M^* + 1$				$M = M^* + 2$			
	M^*	LB_0	UB	LB	ΔLB	CPU	UB	LB	ΔLB	CPU	UB	LB	ΔLB	CPU
oval1A	2	146	154	*147*	4.76	3600.66	151	146	3.42	3600.72	149	146	2.05	3600.77
oval1B	3	146	151	146	3.42	3600.77	149	146	2.05	3600.84	147	146	0.68	3600.99
oval1C	8	146	159	146	8.90	3600.63	**146**	146	0.00	3600.48	**146**	146	0.00	3600.63
oval2A	2	185	195	*191*	2.09	3602.13	192	*187*	2.67	3601.19	189	185	2.16	3601.01
oval2B	3	185	192	*187*	2.67	3601.62	189	185	2.16	3601.21	186	185	0.54	3601.85
oval2C	8	185	**185**	185	0.00	0.01	**185**	185	0.00	0.00	**185**	185	0.00	0.01
oval3A	2	65	71	*66*	7.58	3600.67	69	65	6.15	3600.62	67	65	3.08	3600.67
oval3B	3	65	69	65	6.15	3600.62	67	65	3.08	3600.64	66	65	1.54	3600.72
oval3C	7	65	66	65	1.54	3601.32	**65**	65	0.00	562.71	**65**	65	0.00	546.19
oval4A	3	343	358	343	4.37	3600.71	354	343	3.21	3600.36	350	343	2.04	3600.29
oval4B	4	343	354	343	3.21	3600.31	350	343	2.04	3600.31	347	343	1.17	3600.31
oval4C	5	343	350	343	2.04	3600.46	347	343	1.17	3600.45	345	343	0.58	3600.46
oval4D	9	343	**343**	343	0.00	3600.59	**343**	343	0.00	3600.75	**343**	343	0.00	3600.64
oval5A	3	367	383	367	4.36	3600.34	378	367	3.00	3600.32	374	367	1.91	3600.29
oval5B	4	367	378	367	3.00	3600.32	374	367	1.91	3600.28	371	367	1.09	3600.32
oval5C	5	367	374	367	1.91	3600.28	371	367	1.09	3600.25	368	367	0.27	3600.26
oval5D	9	367	**367**	367	0.00	0.00	**367**	367	0.00	0.01	**367**	367	0.00	0.00
oval6A	3	190	195	190	2.63	3600.57	193	190	1.58	3600.62	192	190	1.05	3600.53
oval6B	4	190	194	190	2.11	3600.53	192	190	1.05	3600.65	191	190	0.53	3600.73
oval6C	10	190	**190**	190	0.00	0.00	**190**	190	0.00	0.00	**190**	190	0.00	0.00
oval7A	3	249	263	249	5.62	3600.24	259	249	4.02	3600.15	256	249	2.81	3600.20
oval7B	4	249	259	249	4.02	3600.20	256	249	2.81	3600.19	253	249	1.61	3600.24
oval7C	9	249	250	249	0.40	3601.51	**249**	249	0.00	3601.41	**249**	249	0.00	3601.34
oval8A	3	347	364	347	4.90	3600.34	359	347	3.46	3600.22	354	347	2.02	3600.24
oval8B	4	347	359	347	3.46	3600.22	354	347	2.02	3600.25	351	347	1.15	3600.21
oval8C	9	347	**347**	347	0.00	0.01	**347**	347	0.00	0.00	**347**	347	0.00	0.00
oval9A	3	278	298	278	7.19	3600.15	294	278	5.76	3600.17	292	278	5.04	3600.24
oval9B	4	278	294	278	5.76	3600.21	292	278	5.04	3600.30	290	278	4.32	3600.24
oval9C	5	278	292	278	5.04	3600.21	290	278	4.32	3600.24	288	278	3.60	3600.26
oval9D	10	278	283	278	1.80	3600.26	281	278	1.08	3600.14	280	278	0.72	3600.16
oval10A	3	376	402	376	6.91	3600.15	396	376	5.32	3600.18	391	376	3.99	3600.14
oval10B	4	376	396	376	5.32	3600.17	391	376	3.99	3600.19	388	376	3.19	3600.22
oval10C	5	376	391	376	3.99	3600.24	388	376	3.19	3600.23	385	376	2.39	3600.20
oval10D	10	376	379	376	0.80	3600.18	378	376	0.53	3600.14	377	376	0.27	3600.15

M^*: minimum number of vehicles to attain a feasible solution. LB_0: trivial lower bound.
UB: solution obtained by the GRASP with PR. LB: best lower bound provided by the branch-and-bound algorithm.
ΔLB: average deviation from lower bound (%). CPU: running time (s) of the branch-and-bound algorithm.

Table 4.5: Branch-and-bound results for *oegl* instances.

			$M = M^*$				$M = M^* + 1$				$M = M^* + 2$			
	M^*	LB_0	UB	LB	ΔLB	CPU	UB	LB	ΔLB	CPU	UB	LB	ΔLB	CPU
oegl-e1-A	5	1468	1775	*1673*	6.10	3600.28	1708	*1629*	4.85	3600.22	1659	*1590*	4.34	3600.21
oegl-e1-B	7	1468	1749	*1591*	9.93	3600.22	1639	*1556*	5.33	3600.25	1589	*1524*	4.27	3600.34
oegl-e1-C	10	1468	1652	*1523*	8.47	3600.27	1576	*1504*	4.79	3600.32	1542	*1490*	3.49	3600.34
oegl-e2-A	7	1879	2177	*2019*	7.83	3600.17	2072	*1990*	4.12	3600.12	2043	*1965*	3.97	3600.12
oegl-e2-B	10	1879	2080	*1944*	7.00	3600.17	1997	*1927*	3.63	3600.23	1971	*1912*	3.09	3600.30
oegl-e2-C	14	1879	2084	*1900*	9.68	3600.20	1997	*1889*	5.72	3600.27	1964	1879	4.52	3600.29
oegl-e3-A	8	2188	2526	*2277*	10.94	3600.14	2410	*2260*	6.64	3600.15	2372	*2245*	5.66	3600.32
oegl-e3-B	12	2188	2411	*2221*	8.55	3600.29	2359	*2212*	6.65	3600.18	2321	*2203*	5.36	3600.23
oegl-e3-C	17	2188	2364	2188	8.04	3600.17	2308	2188	5.48	3600.25	2270	2188	3.75	3600.25
oegl-e4-A	9	2453	2693	2453	9.78	3600.39	2582	2453	5.26	3600.29	2556	2453	4.20	3600.33
oegl-e4-B	14	2453	2786	2453	13.58	3600.37	2567	2453	4.65	3600.18	2517	2453	2.61	3600.34
oegl-e4-C	19	2453	3383	2453	37.91	3600.25	2515	2453	2.53	3600.21	2497	2453	1.79	3600.25
oegl-s1-A	7	1394	1799	*1584*	13.57	3600.20	1683	*1541*	9.21	3600.28	1604	*1503*	6.72	3600.34
oegl-s1-B	10	1394	1745	*1475*	18.31	3600.20	1659	*1447*	14.65	3600.23	1579	*1426*	10.73	3600.20
oegl-s1-C	14	1394	1876	*1415*	32.58	3600.30	1633	*1403*	16.39	3600.28	1512	*1397*	8.23	3600.30
oegl-s2-A	14	3174	3697	*3228*	14.53	3600.11	3621	*3217*	12.56	3600.40	3561	*3205*	11.11	3600.17
oegl-s2-B	20	3174	4309	*3176*	35.67	3600.32	3498	3174	10.21	3600.26	3427	3174	7.97	3600.21
oegl-s2-C	27	3174	3928	3174	23.76	3600.16	3414	3174	7.56	3600.21	3342	3174	5.29	3600.20
oegl-s3-A	15	3379	3828	*3393*	12.82	3600.30	3764	*3386*	11.16	3600.32	3734	*3381*	10.44	3600.30
oegl-s3-B	22	3379	3680	3379	8.91	3600.32	3588	3379	6.19	3600.14	3564	3379	5.47	3600.10
oegl-s3-C	29	3379	3814	3379	12.87	3600.15	3625	3379	7.28	3600.37	3495	3379	3.43	3600.20
oegl-s4-A	19	4186	4500	4186	7.50	3600.25	4421	4186	5.61	3600.15	4405	4186	5.23	3600.24
oegl-s4-B	27	4186	4469	4186	6.76	3600.13	4360	4186	4.16	3600.42	4328	4186	3.39	3600.22
oegl-s4-C	35	4186	7789	4186	86.07	3600.29	4502	4186	7.55	3600.32	4320	4186	3.20	3600.15

M^*: minimum number of vehicles to attain a feasible solution. LB_0: trivial lower bound.
UB: solution obtained by the GRASP with PR. LB: best lower bound provided by the branch-and-bound algorithm.
ΔLB: average deviation from lower bound (%). CPU: running time (s) of the branch-and-bound algorithm.

Conclusions

On the problem.

This work introduced the open capacitated arc routing problem (OCARP), a NP-hard combinatorial optimization problem of theoretical and practical interest belonging to the family of arc routing problems. At least two real-life applications can be modeled as an OCARP, the Meter Reader Routing Problem (Stern and Dror 1979) and the Cutting Path Determination Problem (Moreira et al. 2007).

The OCARP has been formulated as an integer linear programming problem, with an exponential number of variables and constraints. It is unknown if there is a compact formulation to OCARP, and if there is any polynomial algorithm to separate the illegal subcycle elimination constraints. Some interesting properties has been derived to OCARP, regarding the number of vehicles a solution would use, characterization of the beginning and end of each vehicle tour, and attributes of a cyclic tour.

Through a polynomial reduction of the CARP to the OCARP, it was possible to prove that the later is NP-hard. Also, it was possible to demonstrate the backward reduction of the OCARP to the CARP. This reduction can be used by CARP algorithms to also solve OCARP. However, the latest CARP exact algorithms and heuristics assume the number of vehicles as a decision variable, making them unsuitable for OCARP, where the number of vehicles is fixed.

On the heuristic approach.

Based on a greedy randomized adaptive search procedure (GRASP) with evolutionary path-relinking (PR), a high-end metaheuristic was developed to solve both CARP and OCARP.

The GRASP constructive phase (GCH) was based on an efficient CARP path-scanning heuristic (Santos et al. 2009). This heuristic uses an ellipse rule that is triggered when the vehicle capacity is below a given threshold relying on a parameter β. Once active, the ellipse rule directs the tour to the depot, potentially reducing the cost of the way back. Despite OCARP not having a depot, it has empirically been verified that the ellipse rule, by shortening the tours, reduces the solutions cost on average. Other features of the GCH are (i) a cost-demand edge-selection rule, self-tuned according with the instance hardness to achieve feasible solutions; (ii) a restricted candidate list (RCL) containing candidate edges to enter the solution that are at

most α (%) distant from the best candidate; (iii) reactive parameters α and β, meaning that their values are selected from a set of possible values once every iteration, based on the average solution cost induced by each value.

After the GCH has created an initial feasible solution, the local search tries to explore the solution neighborhood in search for a better solution. Four types of neighborhoods moves are defined *single-insert*, *double-insert*, *swap*, and *block-insert*. The *best-improve* scheme is used to select the next move to execute. In addition, a filter was created to prevent low-quality solutions going through local search. This filter uses a statistical threshold that gives more than 95% probability of not discarding an initial solution that would otherwise outperform the incumbent best. The proposed filter was demonstrated by time-to-target plots (TTT) to improve GRASP runtime on average.

Another trend included in the metaheuristic is evolutionary path-relinking, which uses a pool of high-quality solutions (elite solutions) stored along execution. The main idea of path-relinking is to explore the solution space surrounding the path connecting a pair of initial and guiding solutions, one of them from the elite pool. These paths does not need to be constrained in the feasible solution space. This work also explores paths traversing the feasible/infeasible boundaries. This is accomplished by an *infeasible local search*, which produces low cost infeasible solutions, using them as initial solutions to the path-relinking. The effectiveness of evolutionary path-relinking in the metaheuristic runtime was also shown by TTT plots.

The OCARP computational experiments used 81 CARP instances (23 *gdb* (Golden et al. 1983), 34 *val* (Benavent et al. 1992), and 24 *egl* (Li and Eglese 1996)). The original depot was considered a common node, and each group renamed as *ogdb*, *oval*, and *oegl*. These groups of instances were also divided into three classes, according to the number of vehicles available, $M = M^*$, $M = M^* + 1$ and $M = M^* + 2$, where M^* represents the minimum number of vehicles for a feasible solution. Results showed that GCH performed much better than two other path-scanning heuristics, mostly because these last two heuristics were developed for CARP, without concern of the number of vehicles limitation. The complete metaheuristic was also tested and its results reduced the average deviation from lower bounds to 4.64%, starting from the average deviation of 12.24% from the GCH initial solutions.

Additional experiments revealed the metaheuristic performance for the CARP. First, the GCH was tested against two other path-scanning heuristics from literature (Belenguer et al. 2006; Santos et al. 2009), outperforming them with respect to the overall average and maximum deviations from lower bound. Next, the full GRASP with PR was compared with three other metaheuristics: a tabu search (Brandão and Eglese 2008), a variable neighborhood search (Polacek et al. 2008), and an ant colony optimization (Santos et al. 2010). The GRASP outperformed these metaheuristics with respect to the overall average deviation from lower bound and number of best solutions, in spite of additional execution time.

On the exact approach.

An exact approach has also been atempted to solve OCARP, despite it being NP-hard. The proposed algorithm used an equivalence between the OCARP and a problem related to minimum spanning tree, called *capacity and degree constrained minimum spanning forest* (CDCMSF). Deriving lower bounds to the CDCMSF seemed more intuitive, and this motivated the approach. To guarantee optimality, the algorithm relied on the branch-and-bound paradigm. As expected, the complexity of the proposed branch-and-bound turned out to be exponential. Nonetheless, if not to solve all instances to optimality, the method at least improved the known trivial lower bounds, thus reducing the gaps in average.

Three lower bounding procedures to the CDCMSF were implemented: the *minimum cost edges* (LB_1) relied on the number of missing edges to fulfill a solution; the *minimum cost spanning forest* (LB_2) computed adapting the Kruskal's algorithm; and the *degree constrained minimum spanning forest* (LB_3), which is NP-hard itself, but has a Lagrangian relaxation/subgradient method that efficiently computes good lower bounds. Each of these three lower bounds has a trade-off between the tightness of the bound and the computational effort to obtain it. The branch-and-bound stopping criterion is defined by three conditions: optimality, execution time, and memory use. After meeting with one of these criteria, the algorithm returns the best lower bound encountered in the process.

Computational experiments with the exact algorithm were conducted using the same OCARP instances tested in the GRASP with PR study cases. The lower bounds obtained by the method proposed in this work were compared with the bounds generated solving an OCARP model with a high-end ILP solver, both equally time limited. The results showed that the branch-and-bound algorithm outperformed CPLEX in the overall average deviation from lower bounds and in the number of best lower bounds.

On the future.

The methods proposed to solve OCARP, heuristic and exact both, have contributed to improve the upper and lower bounds for the instances tested in the computational experiments. The OCARP average gap is 3.94% using the GRASP with PR upper bounds and the branch-and-bound lower bounds. This is relatively high comparing to the CARP average gap of 0.79%. Therefore, a large room for improvements is still available, leaving for future researches to focus on the design of algorithms that can tighten the bounds here introduced, especially the lower bounds. Two interesting fields that should be investigated are exact algorithms using column generation and cutting planes approaches.

Conclusões

Sobre o problema.

Este trabalho introduziu o problema de roteamento em arcos capacitado e aberto (OCARP), um problema de otimização combinatorial NP-difícil de interesse teórico e prático, pertencente à família de problemas de roteamento em arcos. Pelo menos duas aplicações reais podem ser modeladas como um OCARP, o problema de roteamento de leituristas (Stern and Dror 1979) e o problema de determinação do caminho de corte (Moreira et al. 2007).

O OCARP pode ser formulado como um problema de programação linear inteira, com um número exponencial de variáveis e restrições. Ainda é incerto se há uma formulação compacta para o OCARP, ou se há algum algoritmo polinomial para separar as restrições de eliminação de subciclos ilegais. Algumas propriedades interessantes podem ser derivadas para o OCARP, no que se referem ao número de veículos requeridos para uma solução, a localização do início e o fim de cada rota, e as características de uma rota cíclica.

Através de uma redução polinomial do CARP para o OCARP, foi possível demonstrar que o último é NP-díficil. Também foi possível demonstrar a redução reversa do OCARP para o CARP. Essa redução pode ser utilizada por algoritmos desenvolvidos para o CARP de modo que também resolvam o OCARP. Contudo, os mais recentes algoritmos e heurísticas CARP da literatura supõem que o número de veículos é uma variável de decisão, tornando-os inapropriados para o OCARP, que considera um número fixo de veículos.

Sobre o método heurístico.

Uma metaheurística de alto desempenho, baseada no método GRASP (*greedy randomized adaptive search procedure*) com reconexão por caminhos evolutiva (*evolutionary path-relinking*), foi concebida para tratar tanto CARP quanto OCARP.

A heurística construtiva GRASP (GCH) foi baseada em uma eficiente heurística de varredura de caminhos (*path-scanning*) para o CARP (Santos et al. 2009). Essa heurística utiliza uma regra da elipse que é acionada quando a capacidade do veículo se encontra abaixo de um certo limiar definido por um parâmetro β. Uma vez ativa, a regra da elipse encaminha a rota em direção ao depósito, reduzindo o potencial custo de retorno. Apesar de o OCARP não possuir um depósito, verificou-se empiricamente que a regra da elipse, ao encurtar as rotas, reduz os custos

das soluções, em média. Outras características da GCH são: (i) uma regra de seleção de arestas baseada em custo-demanda, que se auto-regula de acordo com a dificuldade em obter soluções factíveis para a instância; (ii) uma lista restrita de candidatos (*restricted candidate list*, RCL), contendo as arestas candidatas a entrar na solução que não sejam $\alpha(\%)$ mais custosas do que a aresta mais econômica; (iii) ajustes reativos dos parâmetros α e β, o que significa que seus valores são selecionados de um conjunto pré-definido a cada iteração, com viés aos valores que induziram às melhores soluções em média.

Após uma solução factível inicial ser criada pela GCH, uma busca local explora soluções vizinhas em busca de uma solução de menor custo. Quatro tipos de vizinhanças são definidas: *inserção única*, *dupla inserção*, *dupla-troca* e *inserção em bloco*. A estratégia de *melhor vizinho* é adotada para selecionar o próximo movimento de busca local a ser executado. Um filtro foi concebido com a função de prevenir que soluções de baixa-qualidade passem pela busca local. Esse filtro utiliza um limiar estatístico com alta probabilidade de não descartar uma solução que de outra maneira superaria a melhor solução incumbente. Foi mostrado a partir de gráficos de distribuição de tempos (*time-to-target plots*) que o filtro proposto reduz, em média, as durações de execução do GRASP.

Outro atributo incorporado à metaheurística consiste na reconexão por caminhos evolutiva, que utiliza um conjunto de soluções de alta-qualidade (soluções de elite), armazenadas ao longo da execução. A ideia por detrás da reconexão por caminhos é a de explorar regiões do espaço de soluções circundando a trajetória que liga um par de soluções, inicial e guia, uma delas pertencente ao conjunto elite. Essa trajetória não precisa estar restrita ao espaço factível de soluções. Foram incorporadas, neste trabalho, trajetórias alternativas que percorrem as fronteiras factíveis/infactíveis do espaço de soluções. Isso se concretizou por via de uma busca local infactível, que produz soluções infactíveis de baixo custo e que são oportunamente utilizadas na reconexão por caminhos como soluções iniciais. A eficácia da reconexão por caminhos evolutiva na redução do tempo de execução também foi atestada por gráficos de distribuição de tempos.

Os experimentos computacionais voltados ao OCARP utilizaram 81 instâncias do CARP (23 instâncias *gdb* (Golden et al. 1983), 34 instâncias *val* (Benavent et al. 1992), e 24 instâncias *egl* (Li and Eglese 1996)). O depósito original foi considerado como um nó comum, e os grupos de instâncias renomeados para *ogdb*, *oval*, e *oegl*. Esses grupos de instâncias foram também divididos em três classes de acordo com o número de veículos disponíveis, $M = M^*$, $M = M^* + 1$ e $M = M^* + 2$, tal que M^* representa o número mínimo de veículos necessário para obter uma solução factível da instância em estudo. Resultados mostraram que a GCH obteve um desempenho superior ao de outras duas heurísticas de varredura de caminhos da literatura. Justifica-se a boa performance da GCH pelo fato dela construir rotas com atenção ao número limitado de veículos, aspecto este não considerado pelas heurísticas da literatura.

Experimentos adicionais revelaram a performance da metaheurística na solução do CARP. A GCH foi comparada com outras duas heurísticas de varredura de caminhos da literatura

(Belenguer et al. 2006; Santos et al. 2009). Revelou-se que a primeira superou as heurísticas concorrentes em dois aspectos, desvios médios e máximos entre limitantes. A metaheurística GRASP completa foi comparada com três metaheurísticas: uma busca tabu (Brandão and Eglese 2008), uma busca em vizinhança variável (Polacek et al. 2008), e otimização em colônia de formigas (Santos et al. 2010). O GRASP obteve um desempenho superior ao das três metaheurísticas com respeito aos desvios médios entre limitantes e quanto ao número de melhores soluções. Esse bom desempenho do GRASP foi, contudo, às custas de um tempo de execução superior ao das demais metaheurísticas.

Sobre o método exato.

Um algoritmo exato foi elaborado na tentativa de obter a solução ótima para o OCARP, apesar deste ser um problema NP-difícil. O algoritmo proposto utilizou-se da equivalência entre o OCARP e um problema relacionado com árvore geradora mínima, denominado problema da floresta geradora mínima com restrições de grau e capacidade (*capacity and degree constrained minimum spanning forest*, CDCMSF). A concepção do método exato foi motivado pela facilidade em derivar limitantes inferiores para o CDCMSF. A garantia de otimalidade do método se fundamentou no paradigma branch-and-bound. Como esperado, a complexidade do branch-and-bound proposto mostrou-se exponencial no pior caso. Entretanto, se não para resolver instâncias até a otimalidade, ao menos o método pôde aperfeiçoar os limitantes inferiores conhecidos, reduzindo dessa forma os desvios médios.

Três procedimentos para a obtenção de limitantes inferiores foram implementados para o CDCMSF: o limitante de custo mínimo de arestas (LB_1), que se baseia no número de arestas faltantes para completar uma solução; o limitante de floresta geradora de custo mínimo (LB_2), que é computado por uma adaptação do algoritmo Kruskal; o limitante de floresta geradora de custo mínimo com restrição de grau (LB_3), obtido resolvendo-se um problema que, apesar de NP-difícil, pode ser resolvido eficientemente pelo uso da relaxação lagrangeana e método dos subgradientes para a obtenção de bons limitantes inferiores. Cada um desses três limitantes inferiores apresenta uma relação de compromisso entre a qualidade do limitante e o esforço computacional para obtê-lo. O critério de parada do algoritmo branch-and-bound é definido por três condições: otimalidade, tempo de execução e uso de memória. Ao atingir um desses critérios, o algoritmo retorna o melhor limitante inferior encontrado ao longo da execução.

Experimentos computacionais com o algoritmo exato foram conduzidos utilizando as mesmas instâncias OCARP dos estudos de casos da metaheurística GRASP. Os limitantes inferiores obtidos pelo método proposto neste trabalho foram comparados com os limitantes gerados pelo *solver* comercial CPLEX, após solução do modelo OCARP. Os resultados mostraram que o algoritmo branch-and-bound superou o CPLEX no desvio médio entre limitantes e no número de melhores desvios.

Sobre o futuro.

Os métodos propostos para resolver o OCARP, tanto heurístico quanto exato, contribuíram na melhoria dos limitantes superiores e inferiores das instâncias utilizadas nos experimentos computacionais. O desvio médio para o OCARP resultou em $3,94\%$ utilizando os limitantes superiores da metaheurística GRASP e os limitantes inferiores do algoritmo branch-and-bound. Esse desvio ainda está relativamente alto comparado com o desvio de $0,79\%$ do CARP. Portanto, ainda há uma ampla margem de aperfeiçoamentos à disposição, deixando para futuras pesquisas a possibilidade de estreitar os limitantes aqui introduzidos, especialmente os inferiores, por meio da idealização de novos algoritmos. Dois novos campos que devem ser investigados são algoritmos exatos utilizando técnicas de geração de colunas e de planos de corte.

References

Aiex, R. M., Resende, M. G. C., Ribeiro, C. C., Celso and Ribeiro, C.: 2000, Probability distribution of solution time in grasp: An experimental investigation, *Journal of Heuristics* **8**(3), 200–2.

Andrade, R., Lucena, A. and Maculan, N.: 2006, Using Lagrangian dual information to generate degree constrained spanning trees, *Discrete Applied Mathematics* **154**, 703–717.

Assad, A. A. and Golden, B. L.: 1995, Arc routing methods and applications, *in* C. M. M.O. Ball, T.L. Magnanti and G. Nemhauser (eds), *Network routing*, Vol. 8, Elsevier, pp. 375 – 483.

Assad, A., Pearn, W. L. and Golden, B. L.: 1987, The capacitated chinese postman problem: lower bounds and solvable cases, *American Journal of Mathematical and Management Science* **7**, 63–88.

Beasley, J. E.: 1993, *Lagrangian relaxation*, John Wiley & Sons, Inc., New York, NY, USA, pp. 243–303.

Belenguer, J. M. and Benavent, E.: 2003, A cutting plane algorithm for the capacitated arc routing problem, *Computers and Operations Research* **30**, 705–728.

Belenguer, J. M., Benavent, E., Lacomme, P. and Prins, C.: 2006, Lower and upper bounds for the mixed capacitated arc routing problem, *Computers and Operations Research* **33**, 3363–3383.

Benavent, E., Campos, V., Corberán, A. and Mota, E.: 1992, The capacitated arc routing problem: lower bounds, *Networks* **22**, 669–690.

Beullens, P., Muyldermans, L., Cattrysse, D. and Oudheusden, D. V.: 2003, A guided local search heuristic for the capacitated arc routing problem, *European Journal of Operational Research* **147**, 629–643.

Bodin, L. and Levy, L.: 1989, The arc oriented location routing problem, *INFOR* **27**, 74–94.

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

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

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

Cormen, T. H., Stein, C., Rivest, R. L. and Leiserson, C. E.: 2001, *Introduction to Algorithms*, 2nd edn, McGraw-Hill Higher Education.

Dror, M.: 2001, *Arc routing: theory, solutions and applications*, 1st edn, Kluwer Academic Press.

Edmonds, J. and Johnson, E. L.: 1973, Matching, Euler tours and the Chinese postman, *Mathematical Programming* **5**, 88–124.

Eglese, R. W. and Li, L. Y. O.: 1996, *A tabu search based heuristic for arc routing with a*

capacity constraint and time deadline. In Osman I. H., Kelly J. P., Metaheuristics: theory and applications, Kluwer.

Eiselt, H. A., Gendreau, M. and Laporte, G.: 1995a, Arc Routing Problems, Part I: The Chinese Postman Problem, *Operations Research* **43**(2), 231–242.

Eiselt, H. A., Gendreau, M. and Laporte, G.: 1995b, Arc routing problems, part II: the rural postman problem, *Operations Research* **43**(3), 399–414.

Feo, T. A. and Resende, M. G. C.: 1995, Greedy randomized adaptive search procedures, *Journal of Global Optimization* **6**, 109–133.

Feo, T. A., Resende, M. G. C. and Smith, S. H.: 1994, A greedy randomized adaptive search procedure for maximum independent set, *Operations Research* **42**(5), 860–878.

Frederickson, G. N.: 1979, Approximation algorithms for some postman problems, *J. ACM* **26**, 538–554.

Glover, F.: 1996, Tabu search and adaptive memory programming - advances, applications and challenges, *Interfaces in computer science and operations research*, Kluwer, pp. 1–75.

Glover, F.: 2007, Infeasible/feasible search trajectories and directional rounding in integer programming, *Journal of Heuristics* **13**, 505–541.

Golden, B. L., DeArmon, J. S. and Baker, E. K.: 1983, Computational experiments with algorithms for a class of routing problems, *Computers and Operations Research* **10**(1), 47–59.

Golden, B. L. and Wong, R. T.: 1981, Capacitated arc routing problems, *Networks* **11**(3), 305–315.

Gouveia, L., Mourão, M. C. and Pinto, L. S.: 2010, Lower bounds for the mixed capacitated arc routing problem, *Computers and Operations Research* **37**(4), 692–699.

Greistorfer, P.: 2003, A tabu scatter search metaheuristic for the arc routing problem, *Computers and Industrial Engineering* **44**, 249–266.

Hertz, A.: 2005, *Recent trends in arc routing. In Sharda R., Voß S., Golumbic M. C., Hartman I. B. A., Graph theory, combinatorics and algorithms*, Springer US.

Hertz, A., Laporte, G. and Mittaz, M.: 2000, A tabu search heuristic for the capacitated arc routing problem, *Operations Research* **48**(1), 129–135.

Hertz, A. and Mittaz, M.: 2001, A variable neighborhood descent algorithm for the undirected capacitated arc routing problem, *Transportation Science* **35**(4), 425–434.

Hirabayashi, R., Saruwatari, Y. and Nishida, N.: 1992, Tour construction algorithm for the capacitated arc routing problem, *Asia-Pacific Journal of Operational Research* **9**, 155–175.

Labadi, N., Prins, C. and Reghioui, M.: 2008, GRASP with path relinking for the capacitated arc routing problem with time windows, *in* A. Fink and F. Rothlauf (eds), *Advances in computational intelligence in transport, logistics, and supply chain management*, Vol. 144, Springer Berlin / Heidelberg, pp. 111–135.

Lacomme, P., Prins, C. and Ramdane-Chérif, W.: 2004, Competitive memetic algorithms for arc routing problems, *Annals of Operations Research* **131**(1), 159–185.

Li, L. Y. O. and Eglese, R. W.: 1996, An interactive algorithm for vehicle routing for winter-gritting, *Journal of the Operational Research Society* **47**(2), 217–228.

Longo, H., de Aragão, M. P. and Uchoa, E.: 2006, Solving capacitated arc routing problems using a transformation to the CVRP, *Computers and Operations Research* **33**(6), 1823–1837.

Maniezzo, V. and Roffilli, M.: 2008, *Algorithms for Large Directed Capacitated Arc Routing Problem Instances*, Vol. 153, Springer Berlin / Heidelberg.

Martello, S. and Toth, P.: 1990, *Knapsack Problems: Algorithms and Computer Implementations*, John Wiley & Sons.

Mei-ko, K.: 1962, Graphic programming using odd or even points, *Chinese Mathematics* **1**, 273–277.

Moreira, L. M., Oliveira, J. F., Gomes, A. M. and Ferreira, J. S.: 2007, Heuristics for a dynamic rural postman problem, *Computers and Operations Research* **34**(11), 3281–3294.

Mourão, M. C. and Almeida, M. T.: 2000, Lower-bounding and heuristic methods for a refuse collection vehicle routing problem, *European Journal of Operational Research* **121**(2), 420 – 434.

Negreiros, M., Coelho Júnior, W. R., Palhano, A. W. d. C., Coutinho, E. F., de Castro, G. A., Gomes, F. J. N., Barcellos, G. C., Rezende, B. F. and Pereira, L. W. L.: 2009, O problema do carteiro chinês, algoritmos exatos e um ambiente MVI para análise de suas instâncias: sistema XNÊS (in portuguese), *Pesquisa Operacional* **29**, 323 – 363.

Papadimitriou, C. H.: 1976, On the complexity of edge traversing, *Journal of the ACM* **23**(3), 544–554.

Papadimitriou, C. H. and Steiglitz, K.: 1982, *Combinatorial optimization: algorithms and complexity*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Pearn, W. L.: 1988, New lower bounds for the capacitated arc routing problems, *Networks* **18**, 181–191.

Pearn, W. L.: 1989, Approximate solutions for the capacitated arc routing problem, *Computers and Operations Research* **16**(6), 589–600.

Pearn, W. L.: 1991, Augment-insert algorithms for the capacitated arc routing problem, *Computers and Operations Research* **18**(2), 189–198.

Polacek, M., Doerner, K., Hartl, R. and Maniezzo, V.: 2008, A variable neighborhood search for the capacitated arc routing problem with intermediate facilities, *Journal of Heuristics* **14**, 405–423.

Prais, M. and Ribeiro, C. C.: 2000, Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment, *INFORMS Journal on Computing* **12**, 164–176.

Prins, C.: 2009, A GRASP x evolutionary local search hybrid for the vehicle routing problem, *in* F. Pereira and J. Tavares (eds), *Bio-inspired algorithms for the vehicle routing problem*, Vol. 161, Springer Berlin / Heidelberg, pp. 35–53.

Prins, C. and Calvo, R. W.: 2005, A fast GRASP with path relinking for the Capacitated

Arc Routing Problem, *Proceeding of INOC 2005 (3rd International Network Optimization Conference)*, University of Lisbon, pp. 289–295.

Resende, M. G. C., Martí, R., Gallego, M. and Duarte, A.: 2010, GRASP and path relinking for the max-min diversity problem, *Computers and Operations Research* **37**(3), 498 – 508. Hybrid Metaheuristics.

Resende, M. G. C. and Ribeiro, C. C.: 2005, GRASP with path-relinking: Recent advances and applications, *Metaheuristics: Progress as Real Problem Solvers*, Springer, pp. 29–63.

Resende, M. G. C. and Werneck, R. F.: 2004, A hybrid heuristic for the p-median problem, *Journal of Heuristics* **10**(1), 59–88.

Ribeiro, C. C. and Resende, M. G. C.: 2011, Path-relinking intensification methods for stochastic local search algorithms, *Journal of Heuristics* **18**(2), 193–214.

Santos, L., Coutinho-Rodrigues, J. and Current, J. R.: 2009, An improved heuristic for the capacitated arc routing problem, *Computers and Operations Research* **36**(9), 2632–2637.

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

Stern, H. I. and Dror, M.: 1979, Routing electric meter readers, *Computers and Operations Research* **6**(4), 209–223.

Usberti, F. L., França, P. M. and França, A. L. M.: 2011a, Grasp with evolutionary path-relinking for the capacitated arc routing problem, *Computers and Operations Research* . doi: 10.1016/j.cor.2011.10.014.

Usberti, F. L., França, P. M. and França, A. L. M.: 2011b, The open capacitated arc routing problem, *Computers and Operations Research* **38**(11), 1543 – 1555.

Villegas, J. G., Prins, C., Prodhon, C., Medaglia, A. L. and Velasco, N.: 2011, A GRASP with evolutionary path relinking for the truck and trailer routing problem, *Computers and Operations Research* **38**(9), 1319–1334.

Welz, S. A.: 1994, *Optimal solutions for the capacitated arc routing problem using integer programming*, PhD thesis, University of Cincinnati, United States.

Wøhlk, S.: 2006, New lower bound for the capacitated arc routing problem, *Computers and Operations Research* **33**(12), 3458–3472.

Wøhlk, S.: 2008a, An approximation algorithm for the capacitated arc routing problem, *The Open Operational Research Journal* **2**, 8–12.

Wøhlk, S.: 2008b, A decade of capacitated arc routing, *in* R. Sharda, S. Voß, B. Golden, S. Raghavan and E. Wasil (eds), *The vehicle routing problem: latest advances and new challenges*, Vol. 43, Springer US, pp. 29–48.

Wunderlich, J., Collette, M., Levy, L. and Bodin, L.: 1992, Scheduling meter readers for southern california gas company, *Interfaces* **22**(3), 22–30.