UNIVERSIDADE FEDERAL FLUMINENSE

IGOR MACHADO COELHO

Hybrid and Parallel Algorithms for Single and Multi-Objective Routing Problems

NITERÓI 2015

UNIVERSIDADE FEDERAL FLUMINENSE

IGOR MACHADO COELHO

Hybrid and Parallel Algorithms for Single and Multi-Objective Routing Problems

Thesis presented to the Computing Graduate Program of the Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Doctor of Science. Topic Area: Algorithms and Optimization.

Orientador: LUIZ SATORU OCHI

Co-orientador: LUIDI SIMONETTI

> NITERÓI 2015

IGOR MACHADO COELHO

HYBRID AND PARALLEL ALGORITHMS FOR SINGLE AND MULTI-OBJECTIVE ROUTING PROBLEMS

Thesis presented to the Computing Graduate Program of the Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Doctor of Science.

BANCA EXAMINADORA

Prof. Luiz Satoru Ochi - Orientador, UFF

Prof. Luidi Simonetti - Co-orientador, UFF

Prof. Marcone Jamilson Freitas Souza, UFOP

Prof. Frederico Gadelha Guimarães, UFMG

Prof. Cristiana Bentes, UERJ

Prof. Simone de Lima Martins, UFF

Prof. Lucia Drummond, UFF

Niterói 2015

Abstract

This thesis deals with single and multi-objective routing problems by means of hybrid and parallel metaheuristics. The first routing problem in this thesis, i.e., Single Vehicle Routing Problem with Deliveries and Selective Pickups (SVRPDSP), consists in finding a route that starts from the depot and visits all delivery customers. Some pickup customers may also be visited, since the capacity of the truck is not exceeded, and there is also a revenue associated with each pickup. We develop an algorithm inspired on the Variable Neighborhood Search metaheuristic that explores the power of modern Graphics Processing Unit (GPU) to provide routes in reasonable computational time. The proposed algorithm called Four-Neighborhood Variable Neighborhood Search (FN-VNS) includes a novel high quality initial solution generator, a CPU-GPU integrated perturbation strategy and four different neighborhood searches implemented purely in GPU for the local search phase. Our experimental results show that FN-VNS is able to improve the quality of the solution for 51 instances out of 68 instances taken from the literature. We obtained speedups up to 14.49 times, varying from 17.42 up to 76.84 for each local search, measured over a set of new large size instances. We also propose a matheuristic combining a multi-objective evolutionary approach with an exact one-to-many decoder for bi-objective arc routing. This problem requires servicing a set of required edges with positive demands using a fleet of vehicles of limited capacity. The goal is to minimize: (i) the total length of all traversed edges and (ii) the length of the longest route. The evolutionary algorithm works with an indirect solution representation based on permutations of the required edges. Each permutation is mapped to a Pareto set of solutions by a multi-objective decoder based on dynamic programming. Results show that the proposed matheuristic offers a significant improvement over previous algorithms in terms of solution quality for both objectives, while also exhibiting a better Pareto front convergence. Finally, experiments with the proposed algorithm show novel convergence behaviours that can be explored in other single and multi-objective problems.

Keywords: Metaheuristics; Arc Routing Problem; Single Vehicle Routing Problem with Deliveries and Selective Pickups; Multi-Objective Problems; GPU Computing.

List of Figures

2.1	An example of a CARP instance with a central depot, five vertices, four required edges (continuous lines), six non-required edges (dashed lines) and two routes (arrows). Each edge traversal $i \rightarrow j$ is labelled by its cost c_{ij} ; in parentheses we provide the amount of provided service (0 in case of	
	deadheading)	5
2.2	Bi-objective CARP with 4 required edges considering Manhattan distance for non-required edges. Values 0 and 1 denote the two extremities of each	
	edge	9
2.3	Decoding (solid arrows) and aggregation (dashed arrows) examples. Knowl- edge from decision-maker can be used to assign objective values to input solutions s and s' based on values from the decoded sets $\mathcal{D}(s)$ and $\mathcal{D}(s')$.	15
2.4	Implicit solutions of s population $p_{\mathcal{S}}$ are decoded (solid arrows); decoded solutions are evaluated in the objective space (small dashed arrows). Based on this, a fitness assignment function $f_a : \mathcal{X} \times 2^{\mathcal{X}} \to \mathbb{R}$ evaluates and ranks the decoded solutions, generating function $\tilde{f}_s : \mathcal{S} \times 2^{\mathcal{S}} \to \mathbb{R}$ (see left-bottom section)	16
2.5	Growth of decoded solutions in relation to number of required edges. A population of 10 random permutations is decoded and the number of generated solutions in plotted. The instances were selected from sets <i>gdb</i> , <i>egl</i> and <i>val</i> , such that there are at least three different instances for each number of required edges.	20
2.6	Reduction in the number of decoded solutions as the quality of population increase for instance egl - $s3$ - A with 159 required edges. As the number of generations increase, the percentual gap from the lower bound (dotted line) is reduced together with number of unique decoded solutions in the current population (solid line). Random permutations (circles) present poor quality, so a greater number of decoded solutions is generated	21

2.7 Efficient Pareto front for instance gdb19 with 11 required edges, consisting of four non-dominated solutions: (83, 17), (71, 19), (63, 20), (55, 21) 27

- 3.3 Swap exchanges customers +4 and -1, leading the solution S = [depot, -3, -2, +5, -6, +4, +3] to become the solution S' = [depot, -3, -2, +5, -6, -1, +3, -4, +4, depot]. 42

List of Tables

2.1	Each cell represents a pair of values $(D^0(e_k, e_\ell), D^1(e_k, e_\ell))$ for all $k \in \{1, 2, 3, 4\}$ and $\ell < k + len(e_k)$. The arrows indicate the order of calculations.	10
2.2	The values $R(e_k, e_\ell)$ represent full routes for the same e_k and e_ℓ presented in Table 2.1. Note that, in this case, only a single value is reported (instead of a pair), since the vehicle always returns to the depot.	10
2.3	Final solution $sol(k)$ for $k \in [0n]$. Alg. 1 constructs this table and returns the non-marked (non-dominated) solutions listed for $k = 4$, i.e., {(26, 14), (30, 18), (32, 14)}. The next-to-last column is the fleet size; it could be used to extend $sol(k)$ to a structure indexed by both k and the number of vehicles	11
2.4	Impact of local search for gdb instances	22
2.5	Impact of local search for egl instances	23
2.6	Impact of local search for val instances	24
2.7	Comparison of gdb results from literature	26
2.8	Comparison of egl results from literature	27
2.9	Comparison of val results from literature	28
2.10	Comparison of computational time	29
3.1	Experiments with initial solution generation	48
3.2	Comparison of solution quality after local searches (including 3- Opt)	48
3.3	Comparison of computational times of local searches (including $\ensuremath{\mathcal{3}}\xspace-Opt)$	49
3.4	Results for the 68 instances from literature (Part I)	51
3.5	Results for the 68 instances from literature (Part II)	52
3.6	Speedup results for FN-VNS neighborhoods	53

3.7	Quality of the solution for the generated <i>huge</i> instances	54
3.8	Performance gains of Shared-S implementation over Global-S implemen-	
	tation.	55

List of Algorithms

1	Dynamic Programming Multi-Objective Decoder	13
2	NSGA-II	18
3	MODEMAT	19
4	Best Improvement (BI)	39
5	Initial solution generation	41
6	FN-VNS Algorithm	44
7	Kernel for Swap neighborhood	46

Contents

1	Intr	Introduction						
	1.1	Routin	ng Problems	1				
	1.2	Motiv	ation	2				
	1.3	Objec	tives	2				
		1.3.1	Main Objective	2				
		1.3.2	Specific Objectives	2				
2	Bi-C)bjectiv	ve Capacited Arc Routing Problem	3				
	2.1	Introd	luction	3				
	2.2	Bi-obj	ective arc routing: definition, representation and decoding \ldots \ldots	5				
		2.2.1	$Decoding/Indirect \ Representation \ . \ . \ . \ . \ . \ . \ . \ . \ . \ $	7				
			2.2.1.1 Dynamic Programming Bi-objective Decoder	7				
			2.2.1.2 Local search on the best routes	12				
	2.3	A gen	eral multi-objective framework	14				
		2.3.1	Aggregating the values from a multi-objective decoder $\ . \ . \ . \ .$	15				
		2.3.2	Extending the classic framework of multi-objective metaheuristics $% \left({{{\bf{x}}_{i}}} \right)$.	15				
		2.3.3	Application on the NSGA-II	17				
	2.4	Result	ΣS	20				
		2.4.1	Growth of decoded solutions	20				
		2.4.2	Impact of Local Search	21				
		2.4.3	Literature comparison	25				
	2.5	Chapt	er Conclusions	29				

3	B Parallel Algorithm in GPU for Vehicle Routing						
	3.1	Introd	uction	32			
	3.2	Relate	ed Work	34			
	3.3	The P	roblem	37			
	3.4	Four-1	Neighborhood Variable Neighborhood Search	38			
		3.4.1	Parallel Local Searches	45			
	3.5	Exper	imental Results	46			
		3.5.1	Quality of Initial Solution	47			
		3.5.2	Experiments with Local Searches	48			
		3.5.3	Comparison with Literature	49			
		3.5.4	Efficiency of GPU Acceleration	50			
		3.5.5	Memory Hierarchy	53			
	3.6	Chapt	er Conclusions	55			
4	Con	clusions	s and Future Works	57			
	4.1	Future	e Works	58			
Re	eferen	ices		60			

Capítulo 1

Introduction

1.1 Routing Problems

The distribution of goods has been the subject of a great number of researches in the last decades, mainly the Vehicle Routing Problem (VRP), since it was proposed by [23] and better formalized by [18]. The VRP consists in minimizing the total traveled distance of a set of trucks that leave a depot and must satisfy the demands of geographically dispersed customers, respecting the capacity of the trucks. This type of problem arises in many practical applications and it is also important due to the difficulty in solving large instances in reasonable time, being classified as an \mathcal{NP} -Hard problem.

Metaheuristics are handy tools to deal with these hard optimization problems, supporting the design of efficient heuristics, with many of them being inspired by ideas from nature, e.g., Genetic Algorithms, Ant Colony Optimization and Simulated Annealing. Since the problems in focus belong to the class of the \mathcal{NP} -Hard problems, no known polynomial time algorithm is able to solve these problems to optimality. On the other hand, local search techniques applied together with metaheuristics have shown to be able to provide near optimal solutions in short computational time for many complex problems like the VRP's, as can be seen in [83] and [95].

Recently, the acceleration of search algorithms by means of a Graphics Processing Unit (GPU) has also strongly emerged in the optimization field, giving birth to algorithms that are dozens of times faster than classic algorithms implemented for a Central Processing Unit (CPU). The GPU's are massively parallel processors with abundant memory bandwidth and huge number of computational cores. Dealing with the GPU architecture is challenging in many aspects, e.g., exploration of low latency and limited size memories; design of algorithms that explore the fine-grain parallelism offered by the GPU's; and development of an efficient division of computational work between the CPU and the GPU. The work of [55] successfully explores the GPU for many optimization problems with emphasis on local search techniques, and this motivates the development of a similar approach for integrated production-inventory-distribution problems.

1.2 Motivation

The motivation for this research is the great number of practical applications related to the routing and distribution of goods. Also, the validation of metaheuristic algorithms that have been shown effective to deal with large and complex optimization problems. Finally, the study of GPU acceleration techniques that can drastically reduce the computational time of the developed algorithms and to make possible obtaining good solutions quickly in practical applications.

1.3 Objectives

1.3.1 Main Objective

The main objective is to develop metaheuristic algorithms for single and multi-objective routing problem and to use recent GPU technology, in order to accelerate the most computationally expensive tasks.

1.3.2 Specific Objectives

- 1. Study the state-of-art publications related to routing problems;
- 2. Develop heuristics inspired in metaheuristic frameworks such as Variable Neighborhood Search and the Non-Dominated Sorting Algorithm to solve the problems;
- 3. Study the GPU technology and apply it on the developed algorithms in order to achieve good quality solutions quickly;
- 4. Evaluate the quality of the developed algorithms on problem instances available in literature;
- 5. Publish the achieved results in good quality international journals.

Capítulo 2

Bi-Objective Capacited Arc Routing Problem

We propose a matheuristic combining a multi-objective evolutionary approach with an exact one-to-many decoder for bi-objective arc routing. This problem requires servicing a set of required edges with positive demands using a fleet of vehicles of limited capacity. The goal is to minimize: (i) the total length of all traversed edges and (ii) the length of the longest route. The evolutionary algorithm works with an indirect solution representation based on permutations of the required edges. Each permutation is mapped to a Pareto set of solutions by a multi-objective decoder based on dynamic programming. Results show that the proposed matheuristic offers a significant improvement over previous algorithms in terms of solution quality for both objectives, while also exhibiting a better Pareto front convergence. Finally, experiments with the proposed algorithm show novel convergence behaviours that can be explored in other multi-objective problems.

2.1 Introduction

This chapter is devoted to a bi-objective version of the Capacitated Arc Routing Problem (CARP). Given a graph G(V, E), the CARP asks to find a set of *feasible routes* servicing a set of required edges $E_R \subseteq E$, under the constraint that the service amount on each route cannot exceed a given maximum (vehicle) capacity. In the single-objective version, the goal is to minimize the length of all traversed edges. The problem was first first proposed by Golden and Wong [36] and it is \mathcal{NP} -Hard. The most widespread version of bi-objective CARP was proposed by Lacomme [51] and it requires minimizing both the *total length* (of all routes), as well as the *length of the longest route*. This second objective can be very useful to give extra flexibility to the decision-maker, e.g., so as to reduce

working time inequality, comply with legislative shift-time limits, or to provide means for the decision-maker to increase fairness between routes.

We propose a Multi-Objective Decoder Evolutionary Matheuristic (MODEMAT) approach, combining two components: (i) an evolutionary algorithm (EAs) that manipulates permutations of E_R (genotype) and (ii) an exact multi-objective one-to-many decoder that maps permutations to explicit CARP solutions. The EA constitutes the main search process of MODEMAT; we chose to use Non-dominated Sorting Genetic Algorithm II (NSGA), because it is very flexible and very well-suited for problems involving permutations. Regarding the decoder, it transforms a permutation into a Pareto front of non-dominated explicit CARP solutions (the phenotype). Later, the objective values of these decoded solutions can be combined to obtain a fitness value for the input permutation using a specific fitness assignment scheme developed in this work (Sec. 2.3.2).

A permutation in the genotype space can be seen as a service order, i.e., all CARP solutions returned by the decoder provide all service in the order given by the permutation. Using this decoder, one can interpret the CARP as a *permutation problem* [14, 67], i.e., a problem for which the candidate solutions are encoded as permutations. An advantage of using a well-known mathematical object as genotype is that we can use many well-established mutation and crossover operators already available in the literature. By modifying the decoder function, the MODEMAT framework can be applied to other permutation problems with single or multiple objectives.

Based on this framework, we actually present both a standard MODEMAT version and a "memetic flavor" that includes a Local Search (LS) component applied after the decoder. From an experimental point of view, we will compare MODEMAT with two other EAs for bi-objective CARP [51, 58]. The comparison relies both on the best values of the two objectives and on the hypervolume indicator, that measure both the quality and the general spread of the Pareto fronts. Convergence issues are analysed and novel motivating discussions on the role of local search techniques are discussed.

The remaining of the chapter is organized as follows. Section 2.2 is devoted to the following aspects: the bi-objective CARP definitions, the related literature and the proposed multi-objective dynamic programming decoder. Section 2.3 presents MODEMAT as an extension of the classic multi-objective evolutionary model regarding the use of indirect representation and the presence of possibly many decoded solutions. The extended evolutionary model is applied to the NSGA-II with permutation-based operators for mutation and crossover. Section 3.5 presents the results of experiments with MODEMAT, including convergence issues and comparison with other algorithms in literature. Finally, Section 2.5 concludes the work with a discussion on the importance of the developed framework, including promising research directions for the future.

2.2 Bi-objective arc routing: definition, representation and decoding

The bi-objective CARP is defined on a graph G = (V, E), where V is a set of nodes and E is a set of edges. A subset of edges $E_R \subseteq E$, of cardinality m, represents the set of required edges, i.e., edges that must be serviced (once) using an unlimited fleet of homogeneous vehicles of capacity W. A (traversal) cost c_{ij} and a (supply) demand q_{ij} are associated to each edge $\{i, j\} \in E$ and respectively $\{i, j\} \in E_R$. A feasible trip (route) starts and ends at a special depot node $v_0 \in V$ such that the sum of serviced demands does not exceed W.

Note that the edges may be traversed multiple times; an edge traversed without service is called a *deadheaded edge*. Figure 2.1 presents an example of a CARP instance with a two-trip solution that does contain two *deadheaded edges*. The left route deadheads two edges: the non-required edge $\{3, 4\}$ and the required edge $\{v_0, 2\}$ that is also serviced by the right route.



Figure 2.1: An example of a CARP instance with a central depot, five vertices, four required edges (continuous lines), six non-required edges (dashed lines) and two routes (arrows). Each edge traversal $i \rightarrow j$ is labelled by its cost c_{ij} ; in parentheses we provide the amount of provided service (0 in case of *deadheading*).

To formally define the bi-objective CARP, let us note $\mathcal{T}_{\text{fsbl}}$ the set of *feasible trips* or *feasible routes*. The bi-objective CARP asks to find a subset of trips $T \subset \mathcal{T}_{\text{fsbl}}$ that services all required edges and that minimizes: (i) the sum of all traversed edges and (ii) the trip with the longest length (longest route). Technically, we can write these two

objectives obj_1 and obj_2 using equations below.

$$obj_1 = \min_{T \subset \mathcal{T}_{\text{fsbl}}} \left(\sum_{t \in T} \sum_{\{i,j\} \in t} c_{ij} \right)$$
(2.1)

$$obj_2 = \min_{T \subset \mathcal{T}_{\text{fsbl}}} \left(\max_{t \in T} \sum_{\{i,j\} \in t} c_{ij} \right)$$
(2.2)

We hereafter use the term single-objective CARP to refer to the CARP variant with only one objective (2.1), and the term bi-objective CARP consisting of objectives (2.1)-(2.2). The search for solutions minimizing both objectives simultaneously may lead to situations where conflicts occur between them and a choice of compromise between them must be made. In our case, we search for a set of non-dominated Pareto solutions by means of the Pareto dominance relation. It is said that a solution x dominates x' (or $x \prec x'$) when x is better or equals to x' for both objectives and it is strictly better for at least for one objective. Section 2.3 formally describes the Pareto dominance and presents more details on the development of multi-objective optimization algorithms.

Numerous heuristic algorithms have been proposed for the single-objective CARP over the last thirty years. The first work began in the 1980s with CARP-specific heuristics such as Augmented Merge [36], Path-Scanning [34] or the Ulusoy's *giant tour split* [91]. The last decades have seen a surge of interest in developing meta-heuristics, such as including Tabu Search [45, 11], Guided Local Search [8], Variable Neighborhood Search [66], Iterated Local Search [67], Genetic and Memetic Algorithms [49, 50, 86, 57]. Despite the computational complexity of the problem, exact approaches based on mathematical programming have been able to find optimal solutions for solutions with tens or even hundreds of vertices [5, 4, 10]. Recent results for the single and bi-objective CARP can be found in the annotated bibliography by CorberÃ;n and Prins [21] or in the online library *logistik.bwl.uni-mainz.de/benchmarks.php*.

There are relatively few chapters devoted to multi-objective CARP, although extra objectives can naturally be considered by many transportation companies. Amponsah and Salhi [2] propose a constructive heuristic that considers both the distance cost of waste collection and the environmental impact of the smell. Another waste collection problem is tackled by a Multi-Objective Genetic Algorithm (MOGA) minimizing the total and maximum distances of the routes [51]. This algorithm incorporates local searches for the single-objective CARP to a NSGA-II framework, evaluating individuals by means of an splitting procedure. Later, a Decomposition-Based Memetic Algorithm (D-MAENS) was applied to the same problem and some results were improved for both objectives [58]. This was achieved by a Memetic Algorithm where the population consists of diverse solution vectors with different weights for each objective.

Despite the previous good results for the first objective, there is still room to develop algorithms with better compromise between objectives. This fact also implies that singleobjective approaches cannot be adapted easily to deal with multiple conflicting objectives, what motivates the development of the present work.

2.2.1 Decoding/Indirect Representation

In terms of indirect CARP encodings, Ulusoy's *split* [91] is historically the most widespread approach. The initial version consists of splitting giant tour into a number of explicit routes. In more recent work [51], the routine is used and extended to take as input a structure defined by (i) a service order and (ii) a traversal orientation for each edge. Our decoder offers two advantages compared to Ulusoy's routine:

- 1. it only takes a simple service order as input, i.e., a permutation of E_R without explicitly indicating a traversal sense of each edge.
- 2. it returns a set of Pareto-optimal routes with respect to objectives (2.1)-(2.2), i.e., we use a multi-objective decoder.

To our knowledge, although the general idea of transforming simple sequences into permutations is not new [69, 70], *multi-objective decoders* based on dynamic programming have not been explored before.

The main decoding routine is presented below in Section 2.2.1.1. It returns a set of Pareto-optimal solutions that service all required edges in the order indicated by the input permutation. We also use a second post-processing decoding stage (Section 2.2.1.2) that is only applied on the solution of shortest total length routine from the above Pareto front. The route produced in the end might not necessarily respect the order indicated by the input permutation.

2.2.1.1 Dynamic Programming Bi-objective Decoder

Definition 1 (Implicit and Explicit Spaces)

We note S the permutation space or the space of implicit solutions and \mathcal{X} the space of decoded, explicit (or complete) CARP solutions. We consider a decoder function \mathcal{D} : $S \to 2^{\mathcal{X}}$ that maps any permutation $s \in S$ possibly many solutions from \mathcal{X} .

This decoding stage is inspired by a single-objective version for the first objective only [67]. Given $s \in S$, it creates a set of complete \mathcal{X} -solutions subject to the order of visit $s_1, s_2, \ldots s_n$. Furthermore, the decoded set $\mathcal{D}(s)$ of cardinality $\gamma(s)$ forms a *non*dominated Pareto set such that $\forall x, x' \in \mathcal{D}(s), x \not\prec x'$. A good characteristic of such algorithm is that it is always able to find the optimal solution regarding both objectives. As pointed out in literature [50], this second characteristic does not hold for all problems. In scheduling problems where the schedules are obtained by an algorithm that takes as input a list of operations in order to simplify crossover operations, for some instances the optimal solutions cannot be found [32].

Definition 2 Given an input permutation $s = (e_1, e_2, \ldots e_m)$ and some index $k \in [1..m]$, we define the following notations:

- the distance cost of the edge $e_k = \{i, j\}$ is denoted by c_{ij} , or simply c_k when a permutation of edges is given as input.
- the same idea can be applied to a demand d_{ij} , denoted as q_k .
- $len(e_k)$ is the number of edges that can be serviced from edge e_k onwards. This service has to follow order $e_k, e_{k+1}, e_{k+2}, \ldots$ such that capacity W is not exceeded.
- we denote the extremities of edge $e_k = \{i, j\}$ as e_k^0 and e_k^1 , such that e_k^0 is the node i and e_k^1 node j.
- $D^0(e_k, \delta)$ and $D^1(e_k, \delta)$ represent the shortest length of a route that services the required edges e_k , e_{k+1} , $e_{k+2}, \ldots, e_{k+\delta}$ (in any sense) and finishes at an extremity of edge $e_{k+\delta}$ (respectively, D^0 for extremity $e^0_{k+\delta}$ and D^1 for $e^1_{k+\delta}$);
- $R(e_k, \delta)$ is the cost of a route starting from depot v_0 , servicing required edges e_k , e_{k+1} , e_{k+2} ,..., e_k (in any sense) and returning to depot.

The dynamic programming scheme uses the values from the above structures to build a Pareto front of CARP solutions (sets of trips).

Let us first illustrate the above structures on the CARP example from Figure 2.2. This instance has 9 vertices and 4 required edges. Any two pairs of vertices are linked by a (required or non-required) edge with traversal cost given by the Manhattan distance. Considering an input permutation (e_1, e_2, e_3, e_4) , the proposed decoder returns three nondominated explicit solutions with the following pairs of objective values (obj1, obj2): (26, 20), (30, 18) and (32, 14). The first solution is more optimized in terms of the first objective (shortest total distance). The last solution simply corresponds to visiting each required edge in an individual route, which is optimal in relation to the second objective when an unlimited fleet of vehicles is considered.



Figure 2.2: Bi-objective CARP with 4 required edges considering Manhattan distance for non-required edges. Values 0 and 1 denote the two extremities of each edge.

Tables 2.1 and 2.2 present the computed data structures D and R, respectively, for the CARP example in Figure 2.2. Data structures D^0 and D^1 record the length of a shortest-path route that services edges $e_k, e_{k+1}, \ldots e_{\ell}$ and finishes at any end vertex e_{ℓ}^0 and e_{ℓ}^1 of e_{ℓ} , where $\ell \in [k \ldots k + \operatorname{len}(e_k) - 1]$. The calculation of these shortest paths to e_{ℓ}^0 and e_{ℓ}^1 depend on the previously-computed shortest paths to $e_{\ell-1}^0$ and $e_{\ell-1}^1$. For instance, the shortest path to e_{ℓ}^0 is determined as the sum between: (i) the cost of the edge c_{ℓ} plus (ii) the minimum between the shortest path to $e_{\ell-1}^0$ finishing at e_{ℓ}^1 and the shortest path to $e_{\ell-1}^1$ finishing at e_{ℓ}^1 . Note that the shortest path must finish at extremity 1 of the edge, in order to serve and traverse this edge, completing the process at the extremity 0. Data structure R from Table 2.2 computes the minimum cost for a complete route servicing edges $e_k, e_{k+1}, \ldots, e_{\ell}$ and returning to depot.

Table 2.1: Each cell represents a pair of values $(D^0(e_k, e_\ell), D^1(e_k, e_\ell))$ for all $k \in \{1, 2, 3, 4\}$ and $\ell < k + len(e_k)$. The arrows indicate the order of calculations.

e_k	$len(e_k)$	e_ℓ						
		e_1		e_2		e_3		e_4
$\overline{e_1}$	2	(4, 3)	\rightarrow	(16, 12)				
e_2	3			(10, 8)	\rightarrow	(15, 13)	\rightarrow	(16, 15)
e_3	2					(5, 5)	\rightarrow	(8,7)
e_4	1							(6, 5)

Table 2.2: The values $R(e_k, e_\ell)$ represent full routes for the same e_k and e_ℓ presented in Table 2.1. Note that, in this case, only a single value is reported (instead of a pair), since the vehicle always returns to the depot.

e_k	$len(e_k)$	e_ℓ					
		e_1	e_2	e_3	e_4		
e_1	2	6	18				
e_2	3		14	16	20		
e_3	2			8	12		
e_4	1				10		

The proposed multi-objective decoder is presented in Algorithm 1. The decoder receives as input a permutation of required edges and returns a Pareto front consisting of the best values for both objectives servicing all required edges in the permutation order. The first two major steps of this algorithm initialize and compute the above structures Dand L. The last step is devoted to computing complete solutions. For this, we use a table of partial solutions *sol* indexed by values $k \in [0..m]$ such that each *sol*[k] corresponds to all non-dominated partial solutions servicing all edges $e_1, e_2, \ldots e_k$ (or servicing nothing if k = 0). More technically, *sol*[k] is a Pareto set (implemented as a linked list) of pairs of objectives (*obj*₁, *obj*₂), each pair corresponding to a non-dominated partial solution. Table 2.3 provides an example of the final state of this structure *sol*.

The rows of Table 2.3 are computed by Step 3 of Algorithm 1 by generating, for instance, the following transitions:

- Row 1 (k = 0) represents a null state but it can lead to Rows 2 or 3 considering the len $(e_1) = 2$ possible future visits at e_1 and e_2 (if e_3 is included in the same route, the capacity W is exceeded); Table 2.3: Final solution sol(k) for $k \in [0..n]$. Alg. 1 constructs this table and returns the non-marked (non-dominated) solutions listed for k = 4, i.e., $\{(26, 14), (30, 18), (32, 14)\}$. The next-to-last column is the fleet size; it could be used to extend sol(k) to a structure indexed by both k and the number of vehicles.

k	f_2 -ordered routes in $sol[k]$	fleet	row
	in the form route: (f_1, f_2)	size	ID
0	$\emptyset{:}(0,0)$	0	1
1	$\{(e_1)\}:(6,6)$	1	2
2	$\{\underbrace{(e_1, e_2)}_{_{18}}\}: (18, 18)$	1	3
	$\{(e_1), (e_2)\}: (20, 14)$	2	4
3	$\{\underbrace{(e_1, e_2), (e_3)}_{18}\}: (26, 18)^*$	2	5
	$\{(e_1), (e_2, e_3)\}: (22, 16)$	2	6
	$\{\underbrace{(e_1)}_{6}, \underbrace{(e_2)}_{14}, \underbrace{(e_3)}_{8}\}: (28, 14)$	3	7
4	$\{\underbrace{(e_1)}_{6}, \underbrace{(e_2, e_3, e_4)}_{20}\}: (26, 20)$	2	8
	$\{(e_1, e_2), (e_3, e_4)\}: (30, 18)$	2	9
	$\{(e_1), (e_2, e_3), (e_4)\}: (32, 16)^*$	3	10
	$\{(e_1), (e_2), (e_3, e_4)\}: (32, 14)$	3	11
	$\{\underbrace{(e_1), (e_2), (e_3), (e_4)}_{6}\}: (38, 14)^*$	4	12

* indicates a dominated solution (never returned).

- Row 2 (k = 1) can generate Rows 4, 6 and 8 because $len(e_2)$ is 3 (see second column in Table 2.2);
- Row 3 (k = 2) can generate Rows 5 and 9;
- Row 4 (k = 2) generates Rows 7 and 11;
- Row 5 (k = 3) generates Row 10;
- Row 7 (k = 3) generates Row 12.

We observe that the complexity of Algorithm 1 (mostly due to Step 3) depends linearly on m, the maximum length of a Pareto front of a partial solution and the maximum length of a route. The current version of Algorithm 1 does not take into account the fleet size. However, it can be extended in a relatively straightforward manner by adding a second index to *sol* to represent the fleet size. We do not include this in Algorithm 1, but Table 2.3 does provide the number of vehicles of each solution, a structure with 3 dimensions, i.e., the place in the Pareto list, the number of vehicles and k. In this example, the final result of the decoder consists of the non-dominated solutions with k = 4, i.e., visiting all required edges. Since the first objective is much harder to deal with, a local search was included in order to improve the quality of the complete solutions regarding this objective.

2.2.1.2 Local search on the best routes

Given the best route (regarding obj_1) in the Pareto front returned by Algorithm 1, we consider route operators that can be computed in constant time.

The **route rotation** operator considers a route as a cycle $v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots v_r \rightarrow v_0$ (recall v_0 is the depot), that can contain both serviced and non-serviced edges. Using a similar approach as in [91, §3.3], one can compute in constant time the improvement that can be obtained by re-locating the depot v_0 before any index $i \in [2..r]$, leading to route $v_0 \rightarrow v_i \rightarrow v_{i+1} \rightarrow \ldots v_r \rightarrow v_1 \rightarrow v_2 \cdots \rightarrow v_{i-1} \rightarrow v_0$. The cost variation resulting from this operation can be computed in constant time, as one only has to compute the cost of "re-locating" points v_0, v_1, v_{i-1}, v_i and v_r . The route rotation can be applied on a linear number O(m) of points i.

The edge exchange operator swaps some edges $i, j \in E_R$. If i and j belong to the same route route, such an inversion does not change the supplied quantity of this route. If i and j belong to different routes, the inversion does change the supplied quantities of the associated routes by $q_i - q_j$ or $q_j - q_i$; however, the capacity constraints of the updated routes can be easily checked in constant time. The total cost variation induced by the swap is determined from the cost variation resulting from following operations: (a) disconnect i and j from their initial position and (b) re-connect them at the new places (computing their optimal orientation). The impact of these operations can be calculated in constant time, because (a) and (b) do not interfere with the rest of the solution. The number of potential i and j values is in $O(m^2)$.

It takes constant O(1) time to compute the interest in applying such a transformation on a fixed index *i* for *route-rotation* or fixed indexes (i, j) for *edge-exchange*. We search for such positions using one loop and we apply the transformation(s) associated to any index(es) that can lead to an improvement. We thus obtain a *steepest descent* LS that can be systematically performed after post-decoding.

Algorithm 1: Dynamic Programming Multi-Objective Decoder **Input**: V: set of nodes $\{v_0 \dots v_n\}$; E_R : set of required edges $\{e_1 \dots e_m\}$; c_k : cost of edge $e_k \in E_R$; q_k : demand for edge $e_k \in E_R$; W: vehicle capacity; s: permutation $(e_1 \ldots e_m)$ of set E_R // STEP 1: INITIALIZE len, R, D^0 AND D^1 1 len, $R, D^0, D^1 \leftarrow$ arrays with m positions, i.e., for each $k \in [1..m]$ 2 for k = 1 to m do $len(e_k) \leftarrow max\{\ell: \sum_{i=0}^{\ell-1} q_{k+i} \leq W\}$ // the maximum number of edges that can be 3 feasibly serviced from e_k on $R(e_k), D^0(e_k), D^1(e_k) \leftarrow \text{array with } len(e_k) \text{ elements} \qquad // R, D^0 \text{ and } D^1 \text{ become}$ 4 matrices 5 end // STEP 2: COMPUTE $R\,,~D^0$ And D^1 6 for k = 1 to m do $D^0(e_k,1) \leftarrow sp(v_0,e_k^1) + ck$ $// sp(v_i, v_j)$ denotes the shortest path $\forall v_i, v_j \in V$ 7 $D^1(e_k,1) \leftarrow sp(v_0,e_k^0) + ck$ 8 $R(e_k, 1) \leftarrow \min\{D^0(e_k, 1) + sp(e_k^0, v_0), D^1(e_k, 1) + sp(e_k^1, v_0)\}$ 9 for $\delta = 2$ to $len(e_k)$ do 10 $D^{0}(e_{k},\delta) \leftarrow \min\{D^{0}(e_{k},\delta-1) + sp(e^{0}_{k+\delta-1}, e^{1}_{k+\delta}) + c(e_{k}),\$ 11 $D^{1}(e_{k}, \delta - 1) + sp(e_{k+\delta-1}^{1}, e_{k+\delta}^{1}) + c(e_{k})\}$ $D^{1}(e_{k},\delta) \leftarrow \min\{D^{0}(e_{k},\delta-1) + sp(e_{k+\delta-1}^{0}, e_{k+\delta}^{0}) + c(e_{k})\}$ $D^{1}(e_{k},\delta) \leftarrow \min\{D^{0}(e_{k},\delta-1) + sp(e_{k+\delta-1}^{1}, e_{k+\delta}^{0}) + c(e_{k})\}$ $R(e_{k},\delta) \leftarrow \min\{D^{0}(e_{k},\delta) + sp(e_{k+1}^{0}, v_{0}), D^{1}(e_{k},\delta) + sp(e_{k+1}^{1}, v_{0})\}$ 12 $\mathbf{13}$ end 14 15 end // STEP 3: FIND MINIMAL ROUTES WITH BI-OBJECTIVE DYNAMIC PROGRAMMING **16** sol \leftarrow array indexed by $k \in [0..m]$ // sol(k) is a pareto set (a list) of pairs of objective values (f1, f2)// corresponding to a partial solution servicing edges $\{e_1, e_2, \ldots e_k\}$ 17

18 $sol(0) \leftarrow \{(0,0)\}$ // k = 0 means nothing serviced yet 19 for k = 0 to m - 1 do forall $(f_1, f_2) \in sol(k)$ do 20 for $\delta = 1$ to $len(e_{k+1})$ do $\mathbf{21}$ $f_1^{\text{new}} \leftarrow f_1 + R(e_{k+1}, \delta)$ // add to partial solution sol(k) a route that services 22 $e_{k+1}, e_{k+2}, \ldots, e_{k+\delta}$ $f_2^{\text{new}} \leftarrow \max\left\{f_2, R(e_{k+1}, \delta)\right\}$ $\mathbf{23}$ $\text{if } \not\supseteq (f_1^{\text{old}}, f_2^{\text{old}}) \in sol(k+\delta) \text{ such that } (f_1^{\text{old}}, f_2^{\text{old}}) \prec (f_1^{\text{new}}, f_2^{\text{new}}) \\$ $\mathbf{24}$ then $\begin{aligned} sol(k+\delta) &\leftarrow sol(k+\delta) \cup (f_1^{\text{new}}, f_2^{\text{new}}) \\ sol(k+\delta) &\leftarrow sol(k+\delta) \setminus \{(f_1^{\text{old}}, f_2^{\text{old}}) \mid (f_1^{\text{new}}, f_2^{\text{new}}) \prec (f_1^{\text{old}}, f_2^{\text{old}}) \} \end{aligned}$ 25 26 end $\mathbf{27}$ end 28 end 29 30 end 31 return sol(m)

2.3 A general multi-objective framework

We recall the general context of multi-objective problems. Given a space of explicit solutions \mathcal{X} and z minimization objectives f_1, f_2, \ldots, f_z , the goal is find a set of Pareto solutions. A Pareto set only contains *non-dominated* elements, such that there are *no* two different x and x' such that $x \prec x'$. Recall that the dominance relation \prec indicates that $f_i(x) \leq f_i(x')$ and that this inequality is strict for at least some $i \in [1..k]$.

To deal with these multi-objective problems, many specific heuristics have been proposed in literature for each different problem. Also, some general frameworks for heuristics, denominated multi-objective metaheuristics, have been developed providing more flexibility to the design of the heuristics adding a capability to escape local optima. Many multi-objective metaheuristics are composed by three main components [85]: fitness assignment, diversity preserving and elitism.

A fitness assignment scheme is necessary to guide the search and may consist of approaches based on scalar, criterion, dominance and indicator techniques. Scalar approaches basically combine the various conflicting objectives into one single objective, and after that classic single objective techniques can be applied. Examples of scalar approaches are: the aggregation method, which combines the objectives in a linear way [80, 26]; weighted metrics, or compromise programming, which can be applied by comparing the various objectives with a reference point [96]; goal programming, when the decision maker defines aspiration levels and goals for each objective, so the objective is to minimize the difference in relation to the goals [16]; achievement functions, another scalarization method based on an arbitrary point in space [97]; goal attainment method, which is a differenciable version of the weighted metrics technique [17]; and finally, the ϵ -constrained method, which optimizes each objective subject to a fixation of the others [41]; and others.

The diversity preserving techniques are responsible for keeping solutions well spread in the Pareto fronts, usually avoiding premature convergence and increasing the quality of the final solutions. Although some poor quality solutions are kept in order to balance the search process, a good search algorithm must guarantee that the best visited solutions are available in the Pareto front for the decision maker.

This strategy denominated elitism is also known as archiving, when good Pareto individuals are saved in a special pool, or archive, during the search process. The size of the archive can be limited though, since in many optimization problems it may not be feasible to store all non-dominated solutions found during the search.

2.3.1 Aggregating the values from a multi-objective decoder

Given an implicit solution $s \in S$, a multi-objective decoder procedure \mathcal{D} can generate a set of decoded solutions $\mathcal{D}(s) \subseteq \mathcal{X}$. In a classic approach, an aggregation process can be carried such that the *value* of s in the objective space corresponds to an aggregation of the values from decoded solutions $x \in \mathcal{D}(s)$, by means of an aggregation function \mathcal{G} . For instance, the aggregation function \mathcal{G} may assign a value to s according to a priority in the objective functions or by a weighted sum of the objective values in $\mathcal{D}(s)$, however this decision depends on the interests of the decision maker and it is problem specific. In this case, although there are multiple associations between s and the elements in $\mathcal{D}(s)$, an unique multi-objective value is attributed to s, which may reduce the amount of information contained in the decoded set, but with the advantage that classic multi-objective algorithms can be applied directly for the search process. This process of decoding and aggregation is presented in Figure 2.3.



Figure 2.3: Decoding (solid arrows) and aggregation (dashed arrows) examples. Knowledge from decision-maker can be used to assign objective values to input solutions s and s' based on values from the decoded sets $\mathcal{D}(s)$ and $\mathcal{D}(s')$.

When a multi-objective decoder is used, but no *a priori* information is available for the aggregation process, the three main components of the standard multi-objective metaheuristics must be adapted in order to support this new type of information.

2.3.2 Extending the classic framework of multi-objective metaheuristics

We present an extended multi-objective model for metaheuristics with indirect representation and a multi-objective decoder. Consider a multi-objective problem with a space Sof implicit solutions (genotype) and a space \mathcal{X} of explicit solutions (phenotype). We also consider a (decoder) function $\mathcal{D}: \mathcal{S} \to 2^{\mathcal{X}}$, where $2^{\mathcal{X}}$ is the power set of \mathcal{X} , i.e., \mathcal{D} maps an implicit solution $s \in \mathcal{S}$ to a set of decoded complete solutions $\mathcal{D}(s) \subseteq \mathcal{X}$.

A fitness assignment function $f_a : \mathcal{X} \times 2^{\mathcal{X}} \to \mathbb{R}$ provides a numeric fitness value for each element x of a population $p_{\mathcal{X}} \subseteq \mathcal{X}$. The fitness of x depends both on the objective values of x and the objective values of other elements in its current population. In the space of implicit solutions \mathcal{S} , we use a similar function $\tilde{f}_a : \mathcal{S} \times 2^{\mathcal{S}} \to \mathbb{R}$. This function can be defined such that $\tilde{f}a(s, p_{\mathcal{S}})$ is the *best fitness* among all the values of fa over the decoded solutions $x \in \mathcal{D}(s)$; fa can be any function at hand, i.e., well-known fitness assignment functions already presented in literature. Equation (2.3) below formalizes this implementation of \tilde{fa}

$$\widetilde{fa}(s, p_{\mathcal{S}}) = \min_{x \in \mathcal{D}(s)} fa\left(x, \bigcup_{s' \in p_{\mathcal{S}}} \mathcal{D}(s')\right)$$
(2.3)

Figure 2.4 depicts the fitness assignment process with a multi-objective decoder. Initially, the elements of a population p_S from space S are decoded into elements of space \mathcal{X} . These elements are projected in the objective space and some well-known fitness assignment function fa assigns a value to each decoded solution (a different shape represents a different value). Finally, function \tilde{fa} presented in Equation (2.3) assigns the best fitness assignment value to each implicit solution, regarding their respective sets of decoded solutions.



Figure 2.4: Implicit solutions of s population $p_{\mathcal{S}}$ are decoded (solid arrows); decoded solutions are evaluated in the objective space (small dashed arrows). Based on this, a fitness assignment function $f_a : \mathcal{X} \times 2^{\mathcal{X}} \to \mathbb{R}$ evaluates and ranks the decoded solutions, generating function $\tilde{f}_s : \mathcal{S} \times 2^{\mathcal{S}} \to \mathbb{R}$ (see left-bottom section).

Analogously, a diversity management function $dm : \{(x, p_{\mathcal{X}}) \mid p_{\mathcal{X}} \in 2^{\mathcal{X}}, x \in p_{\mathcal{X}}\} \to \mathbb{R}$ attributes a real value to each element of the input population $p_{\mathcal{X}}$. The value of dmis usually given by the distances from x to $p_{\mathcal{X}}$ in the objective space, but one can also consider the distances in the space of \mathcal{X} (see [68] for examples in the space of arrays, partitions or permutations). However, any function dm can be extended to \mathcal{S} -diversity function $dm : \{(s, p_{\mathcal{S}}) \mid p_{\mathcal{S}} \in P(\mathcal{S}), s \in p_{\mathcal{S}}\} \to \mathbb{R}$ using a similar approach as for fa above. In this case, however, a big number of similar decoded solutions may cause an excessive crowding of an area in the objective space and thus *prejudice* the original implicit solution. To avoid this issue, the search for value of dm(s) may compute the diversity measure ignoring decoded solutions that come from s. This strategy denoted *sibling avoidance* is presented in Equation (2.4), considering a function dm that assumes maximization of the diversity measure.

$$\widetilde{dm}(s, p_{\mathcal{S}}) = \max_{x \in \mathcal{D}(s)} dm \left(x, \{x\} \cup \bigcup_{s' \in p_{\mathcal{S}}/\{s\}} \mathcal{D}(s') \right)$$
(2.4)

2.3.3 Application on the NSGA-II

NSGA-II [25] is a classic evolutionary algorithm for multi-objective optimization. It is mainly based on the concepts on non-domination sorting and crowding distance and seeks to optimize a set of solutions (population), returning a Pareto front of non-dominated solutions. The idea of the non-domination sorting is to organize the population in fronts (sets) such that, every solution in front F_j is never dominated by any solution in any front F_i , with $i \leq j$. In order to avoid the evolution of very similar solutions (a common cause of premature convergence of the algorithm), a crowding distance metric evaluates how much "space" every solution occupies in the objective space. If an area is too crowded, the space occupied by each solution will be smaller, and the selection operator will prefer solutions with better fitness (smaller non-dominated front) that belong to less crowded areas. Algorithm 2 presents the classic structure of NSGA-II algorithm. An initial population P^0 of size N is given as input, and a population of children Q^0 (also with size N) is created by procedure Crossover in line 2. At each generation t, these populations are merged as a population R^t and the fitness assignment is calculated by the procedure NonDominatedSort in line 6. Since the population R^t has size $2 \times N$, the selection loop (line 9 of Algorithm 2) chooses the best N solutions (notation $F_i[1...n]$ indicates the first n solutions in front i), front by front, sorted by the CrowdingDistance (line 10 of Algorithm 2).

Algorithm 2: NSGA-II

```
Input: P^0: initial population
 1 N \leftarrow |P^0|
 2 Q^0 \leftarrow \text{Crossover}(P^0)
 \mathbf{3} t \leftarrow 1
    while not stopping criteria do
 4
          R^t \leftarrow P^{t-1} \cup Q^{t-1}
 5
          F \leftarrow \texttt{NonDominatedSort}(R^t)
 6
          P^t \leftarrow \{\}
 7
          i \leftarrow 1
 8
          while |P^t| < N do
 9
                CrowdingDistance(F_i)
10
                P^t \leftarrow P^t \cup F_i[1 \dots \min\{|F_i|, N - |P^t|\}]
11
12
                i \leftarrow i + 1
13
          end
          Q^t \leftarrow \texttt{Crossover}(P^t)
\mathbf{14}
          t \leftarrow t + 1
15
16 end
17 return P^t
```

Since the classic NSGA-II cannot handle a multi-objective decoder, a novel algorithm was developed. It is named MODEMAT (Multi-Objective Decoder Matheuristic), consisting an NSGA-II adapted with the fitness assignment and diversity management operators presented in Eqs. (2.3)-(2.4), and the dynamic programming decoder (Algorithm 1).

MODEMAT is presented in Algorithm 3. The algorithm starts with a population of N random permutations P_S^0 , which is decoded (line 2 of Algorithm 3) and stored as a secondary population P_X^0 . The local search operator improves the quality of the decoded solutions, regarding the first objective (total distance). A children population of N permutations Q_S^0 is generated (line 4 of Algorithm 3), then decoded and stored as Q_X^0 for a local search. In the main loop, both populations of permutations and decoded solutions are merged into populations R_S^t and R_X^t , and the classic NonDominatedSort procedure from NSGA-II is applied to R_X^t (procedure SuppressRepeated will be discussed later). The selection operator (line 16 of Algorithm 3) is an extension of the classic selection from NSGA-II. In this case, it searches for a set of decoded solutions. The CrowdingDistance procedure (line 17 of Algorithm 3) helps avoiding premature convergence by selecting decoded solutions which are more well-spread in the objective space, regarding Eq. (2.4).

```
Algorithm 3: MODEMAT
         Input: P_S^0: initial population
   1 \ N \leftarrow |P_S^0|
  1 \quad N \leftarrow |I_S|
2 \quad P_X^0 \leftarrow \text{Decode}(P_S^0)
3 \quad \text{LocalSearch}(P_X^0)
4 \quad Q_S^0 \leftarrow \text{Crossover}(P_S^0)
5 \quad Q_X^0 \leftarrow \text{Decode}(Q_S^0)
6 \quad \text{LocalSearch}(Q_X^0)
   7 t \leftarrow 1
  8 while not stopping criteria do

9 R_S^t \leftarrow P_S^{t-1} \cup Q_S^{t-1}

10 R_X^t \leftarrow P_X^{t-1} \cup Q_X^{t-1}

11 SuppressRepeated (R_X^t)
10
11
\mathbf{12}
                     F \leftarrow \texttt{NonDominatedSort}(R_X^t)
                  P_S^t \leftarrow \{\} \\ P_X^t \leftarrow \{\} \\
\mathbf{13}
\mathbf{14}
                     i \leftarrow 1
\mathbf{15}
                     while |P_S^t| < N do
16
                          \begin{vmatrix} \text{CrowdingDistance}(F_i, R_X^t) \\ P_S^t \leftarrow P_S^t \cup F_i[1 \dots \min\{|F_i|, N - |P_S^t|\}] \\ i \leftarrow i + 1 \end{vmatrix} 
\mathbf{17}
\mathbf{18}
\mathbf{19}
                      end
\mathbf{20}
                    \begin{array}{l} Q_S^t \gets \texttt{Crossover}(P_S^t) \\ Q_X^t \gets \texttt{Decode}(Q_S^t) \\ \texttt{LocalSearch}(Q_X^t) \end{array}
\mathbf{21}
\mathbf{22}
\mathbf{23}
                      t \leftarrow t + 1
\mathbf{24}
25 end
26 return P_S^t
```



Figure 2.5: Growth of decoded solutions in relation to number of required edges. A population of 10 random permutations is decoded and the number of generated solutions in plotted. The instances were selected from sets gdb, egl and val, such that there are at least three different instances for each number of required edges.

2.4 Results

2.4.1 Growth of decoded solutions

We have detected a growth in the number of decoded solutions, that is related to the increase of the number of required edges in problem instances. Figure 2.5 shows the number of decoded solutions from a population of 10 random permutations, for problem instances with different numbers of required edges. For the bigger instances (190 required edges) almost 100 decoded solutions are generated on average for each input permutation.

The original procedure NonDominatedSort is very sensitive to the population size since it is $\mathcal{O}(zN^2)$, where z is the number of objectives and N the size of the population. Due to this quadratic behaviour, in MODEMAT the complexity also becomes quadratic over the number of decoded solutions (for each generation t), which is very inefficient for bigger problem instances.

In order to deal with the explosion of decoded solutions in bigger instances, a method SuppressRepeated was implemented (line 11 of Algorithm 3). The idea is to remove the repeated decoded solutions, i.e., clones, from population R_X before applying the NonDominatedSort procedure. This way, it was possible to achieve a better efficiency for



Figure 2.6: Reduction in the number of decoded solutions as the quality of population increase for instance egl-s3-A with 159 required edges. As the number of generations increase, the percentual gap from the lower bound (dotted line) is reduced together with number of unique decoded solutions in the current population (solid line). Random permutations (circles) present poor quality, so a greater number of decoded solutions is generated.

bigger instances, since the number of unique solutions falls quickly as the quality of the population improves. This last observation is presented in Figure 2.6. As the population evolves, the gap from the best individuals in relation to the first objective reduces (dotted line), so the average number of decoded solutions (solid line) also decreases. For comparison purposes, random permutations were generated (circles in Figure 2.6) and the average number of decoded solutions was much higher than the evolved population, considering a population of size 30.

2.4.2 Impact of Local Search

Two versions of MODEMAT were developed for the computational experiments. The first MODEMAT version, called MM1, was tested without the LocalSearch procedure of Algorithm 3. The second version, called MM-LS, performs a local search after each decoding operation. The algorithms were compared in terms of computational time and Pareto front convergence. The hypervolume quality indicator [102] was chosen in order to evaluate the Pareto front convergence, since it considers both the quality of solutions regarding the two objectives and the diversity of the Pareto front. The hypervolume

computes the area over a Pareto front with normalized values of objectives varying from 0 to 1.0, in relation to a reference point with coordinates (1.1, 1.1). Since the CARP consists of two minimization objectives, a bigger area on the hypervolume indicates a better convergence of the Pareto front.

Tables 2.4-2.6 show the impact of the local search in the developed algorithm, for the sets of instances gdb, egl and val. It is worth mentioning that MM-LS was capable of finding Pareto fronts with better convergence (bigger hypervolume) and smaller computational time. This behavior is even clearer with bigger instances such as val and egl, where the time gains by using local search were on average 50.72% and 72.46%, respectively. These experiments indicate that, although the local search consume some computational times, a better quality permutation generates fewer decoded solutions, what in fact *reduces* the computational time of MM-LS.

Table 2.4: Impact of local search for gdb instances

Problem	Hyper	volume	Ti	me(s)	Time
	MM1	MM-LS	MM1	MM-LS	$\operatorname{Gain}(\%)$
gdb01	0.672	0.943	10.87	8.69	20.06
gdb02	0.941	1.013	16.58	14.65	11.64
gdb03	0.928	1.087	10.75	8.34	22.42
gdb04	0.939	1.039	9.55	7.67	19.69
gdb05	1.013	1.018	15.94	14.59	8.47
gdb06	0.609	0.911	10.92	8.19	25.00
gdb07	0.804	0.923	13.62	10.44	23.35
gdb08	0.744	0.870	32.65	22.95	29.71
gdb09	0.689	0.819	50.77	32.3	36.38
gdb10	0.959	0.973	19.73	17.91	9.22
gdb11	0.993	1.011	89.14	61.27	31.27
gdb12	0.807	0.974	9.74	8.92	8.42
gdb13	0.201	1.210	7.03	7.84	-11.52
gdb14	1.017	1.052	9.55	8.57	10.26
gdb15	0.824	0.824	5.35	6.49	-21.31
gdb16	0.953	0.955	11.39	12.93	-13.52
gdb17	0.899	0.988	7.11	9.28	-30.52
gdb18	0.893	0.933	35.49	27.94	21.27
gdb19	0.496	0.496	2.14	2.46	-14.95
gdb20	0.941	0.959	6.05	8.13	-34.38
gdb21	0.983	0.978	21.08	18.58	11.86
gdb22	0.927	0.945	32.27	36.97	-14.56
gdb23	0.964	1.029	43.31	40.35	6.83
Average	0.835	0.954	20.48	17.19	6.79
Median	0.927^{\dagger}	0.973^{\dagger}			

† statistical difference for medians for a Wilcoxon Rank-sum test with 95% confidence

Table 2.5: Impact of local search for egl instances

Problem	Hyper	rvolume	Time(s)		Time
	MM1	MM-LS	MM1	MM-LS	$\operatorname{Gain}(\%)$
egl-e1-A	0.748	0.907	152.38	56.11	63.18
egl-e1-B	0.671	0.995	131.75	46.29	64.87
egl-e1-C	0.536	0.964	70.04	34.57	50.64
egl-e2-A	0.831	0.996	596.78	158.87	73.38
egl-e2-B	0.711	0.964	311.01	96.39	69.01
egl-e2-C	0.826	1.091	153.09	84.87	44.56
egl-e3-A	0.644	0.911	1385.92	381.99	72.44
egl-e3-B	0.782	0.723	1115.76	268.47	75.94
egl-e3-C	0.832	1.113	633	153.73	75.71
egl-e4-A	0.738	1.002	2222.71	387.95	82.55
egl-e4-B	0.796	1.038	1234.96	237.81	80.74
egl-e4-C	0.665	0.920	762.32	176	76.91
egl-s1-A	0.781	0.875	771.51	194.01	74.85
egl-s1-B	0.767	0.994	466.57	146.61	68.58
egl-s1-C	0.708	1.076	267.54	92.81	65.31
egl-s2-A	0.687	0.938	5380.56	1039.84	80.67
egl-s2-B	0.689	0.887	3496.87	803.09	77.03
egl-s2-C	0.713	1.128	2042.35	473.12	76.83
egl-s3-A	0.706	0.990	6837.36	1306.81	80.89
egl-s3-B	0.774	1.075	4504.82	1031.72	77.10
egl-s3-C	0.764	1.090	2915.08	778.12	73.31
egl-s4-A	0.668	1.113	7200.62	1686.38	76.58
egl-s4-B	0.547	1.071	6249.38	1044.11	83.29
egl-s4-C	0.424	1.136	3868.05	979.46	74.68
Average	0.709	1.000	2198.77	485.80	72.46
Median	0.712^{\dagger}	0.996^{\dagger}			

 \dagger statistical difference for medians for a Wilcoxon Rank-sum test with 95% confidence

Problem	em Hypervolume Time(s)		Time		
	MM1	MM-LS	MM1	MM-LS	$\operatorname{Gain}(\%)$
val01A	0.968	1.069	45.65	38.71	15.20
val01B	0.798	0.948	45.37	35.3	22.20
val01C	0.657	1.085	19.13	16.55	13.49
val02A	0.833	0.948	102.56	48.83	52.39
val02B	0.928	0.910	69.67	35.2	49.48
val02C	0.328	1.082	31.58	14.05	55.51
val03A	0.816	0.909	45.05	34.42	23.60
val03B	0.633	0.848	32.72	24.35	25.58
val03C	0.396	1.188	18.83	11.95	36.54
val04A	0.852	0.986	567.17	191.54	66.23
val04B	0.790	1.018	516.41	159.1	69.19
val04C	0.849	1.080	396.87	131.53	66.86
val04D	0.559	1.144	166.16	66.76	59.82
val05A	0.970	1.030	964.15	255.66	73.48
val05B	0.870	0.967	781.17	233.13	70.16
val05C	0.871	1.014	611.18	176.23	71.17
val05D	0.616	1.086	284.44	83.12	70.78
val06A	0.894	0.958	92.53	66.14	28.52
val06B	0.890	1.010	75.48	56.59	25.03
val06C	0.596	0.662	41.17	23.93	41.88
val07A	0.975	0.996	200.47	109.17	45.54
val07B	0.841	1.012	195.32	97.26	50.20
val07C	0.668	0.951	105.23	70.08	33.40
val08A	0.892	0.947	476.76	273.35	42.67
val08B	0.847	0.978	480.88	211.23	56.07
val08C	0.805	1.006	193.85	75.07	61.27
val09A	0.931	0.937	1026.36	421.66	58.92
val09B	0.818	0.943	883.93	330.94	62.56
val09C	0.843	0.989	800.67	319.33	60.12
val09D	0.812	1.061	423.7	198.49	53.15
val10A	0.974	1.021	2496.5	701.54	71.90
val10B	0.945	1.006	1920.55	815.68	57.53
val10C	0.870	0.963	2021.18	557.2	72.43
val10D	0.751	0.944	1097.74	420.63	61.68
Average	0.828	0.988	506.78	243.91	50.72
Median	0.842^{\dagger}	0.992^{\dagger}			

 Table 2.6: Impact of local search for val instances

 lume
 Time(s)

 \dagger statistical difference for medians for a Wilcoxon Rank-sum test with 95% confidence

2.4.3 Literature comparison

The proposed MODEMAT algorithm was compared to the best-known algorithms in literature, namely the Multi-Objective Genetic Algorithm (MOGA) by Lacomme [51] and the Decomposition-Based Memetic Algorithm (D-MAENS) by Mei [58]. Results show that it is competitive for both objectives.

Computational tests were done in a Intel 2.3GHz Core i7 (single thread, no parallelism), 8GB de RAM linux kernel 3.0, stopping criterion by number of populations without improvement set to 2000, population size of 80 (permutations), including all configurations presented previously for MM-LS. The objective values are compared to lower bounds for both objectives calculated by Belenguer [5] and Lacomme [51].

Tables 2.7–2.9 present a comparison on sets of instances gdb, egl and val, for the algorithms MODEMAT, MOGA and D-MAENS. In the tables, the column *Problem* indicates the test problem used; LB_1 and LB_2 indicate the lower bounds for objectives one and two, respectively; $best_1$ and $best_2$ are the best values found by each algorithm for both objectives; $gap_1(\%)$ and $gap_2(\%)$ represent the percentual relative gap between the best value and the lower bound, for each objective. Values in bold indicate the lower bound was reached, that is, indicate optimal values. The final rows *Average* and *Hits* indicate, respectively, the average gaps for the test instances and the number of optimal solutions reached for each objective.

From Table 2.7, it is possible to observe that MODEMAT finds better solutions than MOGA for all problem instances considering the first objectives. The most expressive result is for the second objective, finding all optimal values while MOGA struggles with an average gap_2 of 20.95%. Considering instances egl and val the MOGA is more competitive, although MODEMAT always performs better on average. On the other hand, D-MAENS is very competitive with MODEMAT for the first objective since the average gaps are very close, although MODEMAT performs better than D-MAENS for the second objective. For the gdb set, only instances gdb08, gdb09 and gdb12 remain open, while optimal values were reached for all others. For the egl, no optimal value was reached for the first objective, indicating that the lower bound could be improved in future works, or more optimizations aiming the first objective could be added to the search algorithms.

The algorithms were also compared in terms of computational times. In Table 2.10, the column *Reported* indicates the computational time (in seconds) spent by the search algorithm, as reported by the authors; and column *Reduced* represents the expected com-
	MOGA D-MAENS							MOD	ЕМАТ	Г				
Problem	LB_1	LB_2	$best_1$	$gap_1(\%)$	$best_2$	$gap_2(\%)$	$best_1$	$gap_1(\%)$	$best_2$	$gap_2(\%)$	$best_1$	$gap_1(\%)$	$best_2$	$gap_2(%$
gdb01	316	63	316	0.00	63	0.00	316	0.00	63	0.00	316	0.00	63	0.00
gdb02	339	59	339	0.00	59	0.00	339	0.00	59	0.00	339	0.00	59	0.00
gdb03	275	59	275	0.00	59	0.00	275	0.00	59	0.00	275	0.00	59	0.00
gdb04	287	64	287	0.00	64	0.00	$\boldsymbol{287}$	0.00	64	0.00	287	0.00	64	0.00
gdb05	377	64	377	0.00	64	0.00	377	0.00	64	0.00	377	0.00	64	0.00
gdb06	298	64	298	0.00	64	0.00	298	0.00	64	0.00	298	0.00	64	0.00
gdb07	325	57	325	0.00	61	7.02	325	0.00	57	0.00	325	0.00	57	0.00
gdb08	344	38	350	1.74	38	0.00	348	1.16	38	0.00	348	1.16	38	0.00
gdb09	303	37	309	1.98	37	0.00	304	0.33	37	0.00	305	0.66	37	0.00
gdb10	275	39	275	0.00	54	38.46	275	0.00	39	0.00	275	0.00	39	0.00
gdb11	395	43	395	0.00	64	48.84	395	0.00	45	4.65	395	0.00	43	0.00
gdb12	448	93	458	2.23	93	0.00	458	2.23	93	0.00	458	2.23	93	0.00
gdb13	536	128	544	1.49	128	0.00	536	0.00	128	0.00	544	1.49	128	0.00
gdb14	100	15	100	0.00	17	13.33	100	0.00	16	6.67	100	0.00	15	0.00
gdb15	58	8	58	0.00	13	62.50	58	0.00	9	12.50	58	0.00	8	0.00
gdb16	127	14	127	0.00	19	35.71	127	0.00	14	0.00	127	0.00	14	0.00
gdb17	91	9	91	0.00	15	66.67	91	0.00	10	11.11	91	0.00	9	0.00
gdb18	164	19	164	0.00	27	42.11	164	0.00	24	26.32	164	0.00	19	0.00
gdb19	55	17	55	0.00	17	0.00	55	0.00	17	0.00	55	0.00	17	0.00
gdb20	121	20	121	0.00	20	0.00	121	0.00	20	0.00	121	0.00	20	0.00
gdb21	156	15	156	0.00	22	46.67	156	0.00	16	6.67	156	0.00	15	0.00
gdb22	200	12	200	0.00	20	66.67	200	0.00	18	50.00	200	0.00	12	0.00
gdb23	233	13	235	0.86	20	53.85	233	0.00	20	53.85	235	0.86	13	0.00
Average				0.36		20.95		0.16		7.47		0.28		0.00
Hits			18		12		20		15		18		23	

Table 2.7: Comparison of gdb results from literature

putational time of the algorithm running in a Intel(R) Core(TM) i7 CPU 870 2.93GHz (comparison made by memory latency times and processor gigaflops, from specialized websites). From this table, it can be concluded that MODEMAT still consumes more computational resources during the search process. However, the topic of multi-objective decoding and the management of multiple populations is still quite new in literature. More efficient techniques are likely to appear in the future as more algorithms are developed based on the novel ideas behind MODEMAT.

As a visual example of Pareto convergence, MODEMAT managed to find the efficient Pareto front for instance gdb19, presented in Figure 2.7. This comparison was possible since gdb19 consists of only 11 required edges and a brute force algorithm was able to find the efficient Pareto front in reasonable computational time. Unfortunately, this is not possible for the other instances, which consist of at least 22 required edges. Although a hypervolume comparison between the Pareto fronts of the algorithms is desirable, this comparison was not possible since this information was not provided by the authors of MOGA and D-MAENS. For future comparisons, all information related to MODEMAT is available online at the personal website of the authors.

				MO	\mathbf{GA}		D-MAENS				MODEMAT			
Problem	LB_1	LB_2	$best_1$	$gap_1(\%)$	$best_2$	$gap_2(\%)$	$best_1$	$gap_1(\%)$	$best_2$	$gap_2(\%)$	$best_1$	$gap_1(\%)$	$best_2$	9
egl-e1-A	3515	820	3548	0.94	820	0.00	3548	0.94	820	0.00	3548	0.94	820	
egl-e1-B	4436	820	4525	2.01	820	0.00	4525	2.01	820	0.00	4524	1.98	820	
egl-e1-C	5453	820	5687	4.29	820	0.00	5595	2.60	820	0.00	5615	2.97	820	
egl-e2-A	4994	820	5018	0.48	820	0.00	5018	0.48	820	0.00	5018	0.48	820	
egl-e2-B	6249	820	6411	2.59	820	0.00	6347	1.57	820	0.00	6351	1.63	820	
egl-e2-C	8114	820	8440	4.02	820	0.00	8339	2.77	820	0.00	8395	3.46	820	
egl-e3-A	5869	820	5956	1.48	820	0.00	5926	0.97	820	0.00	5982	1.93	820	
egl-e3-B	7646	820	7911	3.47	820	0.00	7801	2.03	820	0.00	7870	2.93	820	
egl-e3-C	10019	820	10349	3.29	820	0.00	10340	3.20	820	0.00	10374	3.54	820	
egl-e4-A	6372	820	6548	2.76	820	0.00	6476	1.63	820	0.00	6533	2.53	820	
egl-e4-B	8809	820	9116	3.49	820	0.00	9069	2.95	820	0.00	9138	3.73	820	
egl-e4-C	11276	820	11802	4.66	820	0.00	11774	4.42	820	0.00	11760	4.29	820	
egl-s1-A	4992	912	5102	2.20	924	1.32	5068	1.52	912	0.00	5018	0.52	912	
egl-s1-B	6201	912	6500	4.82	912	0.00	6435	3.77	912	0.00	6414	3.43	912	
egl-s1-C	8310	912	8694	4.62	912	0.00	8518	2.50	912	0.00	8518	2.50	912	
egl-s2-A	9780	979	10207	4.37	979	0.00	10117	3.45	979	0.00	10253	4.84	979	
egl-s2-B	12886	979	13548	5.14	979	0.00	13459	4.45	979	0.00	13450	4.38	979	
egl-s2-C	16221	979	16932	4.38	979	0.00	16832	3.77	979	0.00	16794	3.53	979	
egl-s3-A	10025	979	10456	4.30	979	0.00	10469	4.43	979	0.00	10501	4.75	979	
egl-s3-B	13554	979	14004	3.32	979	0.00	14082	3.90	979	0.00	14056	3.70	979	
egl-s3-C	16969	979	17825	5.04	979	0.00	17650	4.01	979	0.00	17500	3.13	979	
egl-s4-A	12027	1027	12730	5.85	1027	0.00	12602	4.78	1027	0.00	12648	5.16	1027	
egl-s4-B	15933	1027	16792	5.39	1027	0.00	16686	4.73	1027	0.00	16675	4.66	1027	
egl-s4-C	20179	1027	21309	5.60	1027	0.00	21213	5.12	1027	0.00	20979	3.96	1027	
Average				3.69		0.05		3.00		0.00		3.12		
Hits			0		23		0		24		0		24	

Table 2.8: Comparison of egl results from literature

Figure 2.7: Efficient Pareto front for instance gdb19 with 11 required edges, consisting of four non-dominated solutions: (83, 17), (71, 19), (63, 20), (55, 21)



Table 2.9: Comparison of val results from literature

				MO	GA			D-MA	AENS			MOD	EMAT	Г
Problem	LB_1	LB_2	$best_1$	$gap_1(\%)$	$best_2$	$gap_2(\%)$	$best_1$	$gap_1(\%)$	$best_2$	$gap_2(\%)$	$best_1$	$gap_1(\%)$	$best_2$	$gap_2(2$
val01A	173	40	173	0.00	58	45.00	173	0.00	40	0.00	173	0.00	40	0.00
val01B	173	40	173	0.00	42	5.00	173	0.00	40	0.00	173	0.00	40	0.00
val01C	235	40	245	4.26	40	0.00	245	4.26	40	0.00	245	4.26	40	0.00
val02A	227	71	227	0.00	90	26.76	227	0.00	71	0.00	227	0.00	71	0.00
val02B	259	71	260	0.39	78	9.86	259	0.00	71	0.00	259	0.00	71	0.00
val02C	445	71	463	4.04	71	0.00	457	2.70	71	0.00	457	2.70	71	0.00
val03A	81	27	81	0.00	31	14.81	81	0.00	27	0.00	81	0.00	27	0.00
val03B	87	27	87	0.00	27	0.00	87	0.00	27	0.00	87	0.00	27	0.00
val03C	137	27	138	0.73	27	0.00	138	0.73	27	0.00	138	0.73	27	0.00
val04A	400	80	400	0.00	92	15.00	400	0.00	80	0.00	400	0.00	80	0.00
val04B	412	80	412	0.00	83	3.75	412	0.00	80	0.00	412	0.00	80	0.00
val04C	428	80	430	0.47	80	0.00	430	0.47	80	0.00	434	1.40	80	0.00
val04D	520	80	539	3.65	80	0.00	536	3.08	80	0.00	530	1.92	80	0.00
val05A	423	72	423	0.00	96	33.33	423	0.00	76	5.56	423	0.00	72	0.00
val05B	446	72	446	0.00	86	19.44	446	0.00	73	1.39	446	0.00	72	0.00
val05C	469	72	474	1.07	80	11.11	474	1.07	72	0.00	474	1.07	72	0.00
val05D	571	72	595	4.20	72	0.00	595	4.20	72	0.00	595	4.20	72	0.00
val06A	223	45	223	0.00	56	24.44	223	0.00	45	0.00	223	0.00	45	0.00
val06B	231	45	233	0.87	50	11.11	233	0.87	45	0.00	233	0.87	45	0.00
val06C	311	45	317	1.93	45	0.00	317	1.93	45	0.00	317	1.93	45	0.00
val07A	279	39	279	0.00	59	51.28	279	0.00	41	5.13	279	0.00	39	0.00
val07B	283	39	283	0.00	51	30.77	283	0.00	40	2.56	283	0.00	39	0.00
val07C	333	39	335	0.60	40	2.56	334	0.30	39	0.00	334	0.30	39	0.00
val08A	386	67	386	0.00	87	29.85	386	0.00	71	5.97	386	0.00	67	0.00
val08B	395	67	395	0.00	79	17.91	395	0.00	69	2.99	395	0.00	67	0.00
val08C	517	67	545	5.42	67	0.00	532	2.90	67	0.00	527	1.93	67	0.00
val09A	323	44	326	0.93	68	54.55	324	0.31	47	6.82	323	0.00	44	0.00
val09B	326	44	326	0.00	58	31.82	326	0.00	47	6.82	326	0.00	44	0.00
val09C	332	44	332	0.00	51	15.91	332	0.00	44	0.00	332	0.00	44	0.00
val09D	382	44	399	4.45	44	0.00	392	2.62	44	0.00	393	2.88	44	0.00
val10A	428	47	428	0.00	91	93.62	428	0.00	62	31.91	428	0.00	47	0.00
val10B	436	47	436	0.00	77	63.83	436	0.00	60	27.66	437	0.23	47	0.00
val10C	446	47	448	0.45	66	40.43	446	0.00	58	23.40	446	0.00	47	0.00
val10D	524	47	537	2.48	54	14.89	533	1.72	51	8.51	534	1.91	47	0.00
Average				1.06		19.62		0.80		3.79		0.77		0.00
Hits			18		10		20		22		20		34	

			Time(s)
Problem	Algorithm	Reported	Reduced
	MOGA [51]	13.79^{1}	4.31*
gdb	D-MAENS [58]	8.82^{2}	4.41*
	MODEMAT	17.19^{3}	17.19
	MOGA [51]	83.06^{1}	30.74*
val	D-MAENS [58]	53.98^{2}	26.99*
	MODEMAT	243.91^{3}	243.91
	MOGA [51]	268.99^{1}	84.08*
egl	D-MAENS [58]	213.5^{2}	106.75^{*}
	MODEMAT	485.80^{3}	485.80
	MOGA [51]	115.61^{1}	36.13*
overall	D-MAENS [58]	88.42^2	44.21*
	MODEMAT	226.66^{3}	226.66

Table 2.10: Comparison of computational time

¹ Pentium IV 1.8GHz

² Intel(R) Xeon(R) CPU E5335 2.00GHz

³ Intel(R) Core(TM) i7 CPU 870 2.93GHz

* time reduced according to reported computer specifications

2.5 Chapter Conclusions

This chapter addressed bi-objective CARP in which a set of edges has to be serviced by vehicles of limited capacity, so as to minimize the sum of all traversed arcs as well as the longest tour. The CARP consisting of only the first objective is well explored in literature and it is a \mathcal{NP} -Hard problem, so many heuristic approaches have been presented before in literature. We proposed a Matheuristic approach called MODEMAT, that integrates an exact dynamic programming decoder into the classic multi-objective evolutionary metaheuristic NSGA-II. Since the decoder generates a set of explicit solutions, a novel multi-objective framework was developed in order to extend classic multi-objective components for providing fitness assignment, diversity and elitism.

Optimization techniques were presented in order to accelerate the convergence of MODEMAT and also to prevent premature convergence. The results show that the proposed algorithm is competitive with literature, being able to achieve all optimal values for the second objective of the problem. For the first objective, MODEMAT is also competitive even in the number of optimal solutions found and for the average gap compared to a lower bound from literature.

It is also worth mentioning that proposed algorithm is very generic and no specific construction heuristics were necessary in order to find good quality solutions. The algorithm can also be easily applied to other optimization problems and the ideas presented in this chapter can also be extended to improve other multi-objective search algorithms.

Acknowledgements This work was partially supported by the Brazilian funding agency CAPES (process 10381-13-9) and also by INRIA funding for the research team DOLPHIN.

Capítulo 3

Parallel Algorithm in GPU for Vehicle Routing

Environmental issues have become increasingly important to industry and business in recent days. This trend forces the companies to take responsibility for product recovery, and proper recycling and disposal, moving towards the design of sustainable green supply chains. This chapter addresses the backward stream in transportation of products, by means of reverse logistics applied to vehicle routing. This problem, called Single Vehicle Routing Problem with Deliveries and Selective Pickups (SVRPDSP), consists in finding a route that starts from the depot and visits all delivery customers. Some pickup customers may also be visited, since the capacity of the truck is not exceeded, and there is also a revenue associated with each pickup. We develop an algorithm inspired on the Variable Neighborhood Search metaheuristic that explores the power of modern Graphics Processing Unit (GPU) to provide routes in reasonable computational time. The proposed algorithm called Four-Neighborhood Variable Neighborhood Search (FN-VNS) includes a novel high quality initial solution generator, a CPU-GPU integrated perturbation strategy and four different neighborhood searches implemented purely in GPU for the local search phase. Our experimental results show that FN-VNS is able to improve the quality of the solution for 51 instances out of 68 instances taken from the literature. Finally, we obtained speedups up to 14.49 times, varying from 17.42 up to 76.84 for each local search, measured over a set of new large size instances.

3.1 Introduction

Environmental issues have received a great deal of attention over the past two decades. Organizations are adopting environmentally friendly management strategies in both operational and strategic contexts, aiming to add competitive advantages and to cope with environmental regulations [76, 22]. In addition, the evidence of environmental damage intensifies the population pressure on organizations practices [101]. In this scenario, Green Supply Chain Management (GSCM) is becoming an increasingly popular management concept because it takes environmental aspects into consideration when managing the supply chain [81, 37, 92, 93]. Different research directions for addressing GSCM arise in: product design [1, 12], manufacturing practices [98, 39], and reverse logistics [15, 74].

In this work, we focus on the reverse logistics aspect of GSCM [94]. Reverse logistics deals with the backward movement, or the product return in the supply chain [24]. Developments in reverse logistics are fundamental for GSCM, providing waste reduction and cost savings, since a value can be retrieved from returned goods [48]. Environmental conscious organizations must take responsibility for their products at end of life, including product recovery, and proper recycling and disposal. A considerable number of reverse logistics cases of study have been reported. A mail-service problem is presented by [84]. The mail is stored in a depot and later on is delivered to the customers, which can also send mail back to the depot. [71] present a practical application for the soft-drink distribution problem, which includes the delivery of bottled water to customers, that return empty bottles back to the depot. The case of European electronics industry is studied in [63]. Electrical and electronic equipment have a short life, and can be restocked and returned to the original distribution centers. The management of product return process in a timely and effective manner presented a great difficulty in Europe. [88] present the case of singleuse flash camera in Kodak. Customers take the used camera to a store, that receives a deduction for each camera returned to the factory. The used circuit boards of cameras are further recycled to produce new cameras. The combined delivery and collection of products was studied in [7] for printer cartridge recycling in Great Britain, and power tools recycling in Germany. The recovery of copy machines is proposed by [87], where used products are collected for recovery. Products are transported from plants to markets and from each market a certain amount of used products have to be collected. There are also some design considerations for product recovery networks in the literature [31, 27].

In reverse logistics, the distribution and collection of the products have to be carefully planned. Transportation activities have to be organized in order to: (i) provide efficiency in the supply chain; (ii) reduce costs, guaranteeing the economic success of reprocessing products [9]; and avoid further degradations to the environment, endorsing the environmental benefits of the backward stream [29]. The dominant design decision in the transportation activities planning is the determination of the vehicle routes.

Vehicle routing is a well-known combinatorial optimization problem that aims to find vehicle routes at minimum cost. An important variant of this problem, called Single Vehicle Routing Problem with Deliveries and Selective Pickups (SVRPDSP), fits the reverse logistics demands. This problem considers that a single vehicle delivers products to a set of customers, while some pickups are also possible during the route. Every satisfied pickup yields a positive revenue.

The SVRPDSP is a \mathcal{NP} -hard problem and this implies that there is no known polynomial algorithm that solves the problem to optimality. In this case, the use of metaheuristic frameworks and mathematical programming techniques can provide reasonably good solutions. However, they usually cannot handle efficiently large size problems. In addition, on-line vehicle routing is gaining attention because of the advances in communication technologies. The demands of customers may change quickly and the manager may need a fast routing algorithm to solve the problem. Therefore, it is indisputable that companies need a computational mechanism to provide routes in reasonable computational time, despite the size of the problem.

Recent advances in processor architectures can be explored in order to accelerate routing algorithms. We present an efficient algorithm that explores the low-cost and highly accessible Graphics Processing Unit (GPU) technology. The GPUs have evolved into a powerful massively programmable parallel environment, where a large number of cores can be assembled for solving large scale problems with lower price than multiple separated processing units. We propose an integrated heuristic, called Four-Neighborhood Variable Neighborhood Search (FN-VNS), that combines the flexibility of the metaheuristic Variable Neighborhood Search (VNS) with acceleration components implemented in a GPU. The FN-VNS also includes a novel high quality initial solution generator, integrated with two mathematical programming solvers, providing the search algorithm with good starting solutions. These solutions are further improved by local search in four different neighborhoods, that is parallelized by means of GPU programming. Finally, an integrated CPU-GPU perturbation mechanism and a novel stopping criteria introduce diversity during the search process and avoid premature convergence of the algorithm, preventing it from staying stuck in local optima. The results of FN-VNS are superior when compared to algorithms from literature and the integration with a GPU is able to achieve nearly 15 times speedup over a pure sequential version. It is also possible to achieve an acceleration of 76.84 times for the neighborhood searches, considering a new set of instances of larger size.

In the next section, we briefly review works on strategic, tactical and operational decisions for supply chain management problems, focusing on successful applications of the VNS metaheuristic. We also review the works on GPU parallelization for classic metaheuristic algorithms. In Section 3.3, we describe the problem in details, while Section 3.4 presents the proposed FN-VNS algorithm. The results are reported in Section 3.5. Finally, the Section 2.5 concludes the work and indicates directions for future researches in supply chain problems and parallel metaheuristics.

3.2 Related Work

There are three important streams of research involving a supply chain management problem: strategic, tactical and operational [30]. The strategic stream involves long-term decisions, such as facility location. The tactical stream involves decisions made in not such a long-term basis, such as transportation and fleet planning. The operational stream involves short-term decisions, usually made in real-time, such as vehicle loading and route recomputation.

Facility location is a widely studied combinatorial optimization problem, that arises in industrial engineering, with applications in manufacturing units and process plants. It seeks an efficient location of facilities to optimize production flows and minimize the costs [79]. This problem has been extensively studied since it was first formulated as a Quadratic Assignment Problem [47]. Many heuristic techniques were developed since then [89, 44] and some routing decisions were also included in the model, called the Location-Routing Problem (LRP). It simultaneously determines the location of a set of facilities (depots) and optimizes distribution in a supply chain where customers receive goods from the set of depots. Exact methods are usually based on a mathematical programming formulation [52, 6] and different heuristic approaches have been proposed to provide solutions in reasonable time [42, 100]. Literature surveys on LRP are presented in [62] and [72] with the applications in logistics and distribution management.

From a tactic and operational perspective, in order to optimize transportation costs many practical applications are modeled as a Vehicle Routing Problem (VRP). It has become a very important problem in supply chain logistics and also for computing, since it is an \mathcal{NP} -Hard problem [23]. Many heuristic algorithms have been developed for VRP variants [90, 35]. The Variable Neighborhood Search (VNS) framework proposed by [43] has been applied to many hard combinatorial problems, including those related to the VRP. Currently, most of the best techniques for VRP variants are merged with multineighborhood concepts inspired by the VNS, e.g., Hybrid Iterated Local Search [83] and Hybrid Genetic Algorithm [95].

We discuss implementations that are based on VNS and parallel metaheuristics, as they have similarities with our approach. Recently, a VNS algorithm was successfully applied to an economic lot sizing problem with product returns and recovery [78], indicating that this approach can also fit the GSCM objectives of this work. The work of [59] deals with non-linear cost for the open depots, and presents a hybrid metaheuristic consisting of Tabu Search and VNS. The work by [28] uses multiple neighborhoods to evaluate solutions regarding both depot status and customer sequence. The perturbation mechanism is composed of two neighborhoods affecting the depot, while the local search step uses classic neighborhood structures for routing, inspiring the development of the proposed FN-VNS algorithm. In these works, the routing problem considers customers to have only delivery demands. The work of [33] considers pickup and delivery demands, but the pickups are not selective. A continuous location problem is solved and routing is solved with a VNS approach. In this case, a cross-exchange neighborhood is used as perturbation and a 3-Opt operation as local search. [82] developed a parallel Iterated Local Search for a VRP with pickups and deliveries. The results indicate a tremendous decrease in the computational time by using a massive number of processing cores. Noticeable speedups are achieved by [75] for a Greedy Randomized Adaptive Search Procedure metaheuristic, while some parallel Evolutionary Algorithms on the GPU were also proposed [99, 54]. A Tabu Search implementation with GPU is presented in [46] and also in [54] for a 3-dimensional assignment problem. Finally, local search on the GPU is the subject of research of [55] and [77].

The SVRPDSP was first presented by [84]. The authors contextualized the problem with some real industry cases and developed a mathematical programming model, using the MTZ subtour elimination constraints presented by [60]. Experiments with the software CPLEX have shown that the proposed model was able to solve small randomly generated instances. However, the authors considered that customers have only delivery or pickup demands, but not both. This variant fits many real-world reverse logistic scenarios and it is closely related to the Prize Collecting Traveling Salesman Problem (PCTSP) presented by [3]. The PCTSP consists in finding a tour visiting a subset of customers and penalizing each non-visited customer. However, the PCTSP does not consider demands on customers, i.e., a scenario where the vehicle has unlimited capacity.

In [38], a heuristic based on Tabu Search was developed to solve the SVRPDSP. The idea behind the algorithm was to assign labels for each customer, *pickup only*, *delivery only*, *both pickup and delivery*, and exchange these labels in order to improve the solution quality. In terms of mathematical programming, a branch-and-cut algorithm for the SVRPDSP was developed by [40]. Efficient cuts were devised for a new mathematical programming model, which was able to solve some instances of up to 90 customers.

[19] proposed the first VNS with classic local search algorithms for the SVRPDSP. A basic initial solution was generated by randomly selecting pickup customers and adding random delivery customers in a route, merging them in a route. However, many infeasible routes may be generated with this strategy, when vehicle capacity is not respected. Also, the complicated perturbation scheme of the method involved an extra perturbation parameter, causing computational times to rise much faster for larger instances of the problem. In [20], an initial GPU parallelization of local searches for the SVRPDSP was presented, achieving modest acceleration values. According to the authors, the lack of communication between the GPU components prevented the design of a fully integrated algorithm, so no improvement in the quality of solution was achieved by means of the GPU technology. [13] improved the perturbation mechanism by using ideas from evolutionary algorithms while keeping the multi-neighborhood characteristic of VNS algorithms. The algorithm was able to improve marginally the quality of solutions and, to the best of our knowledge, these are the best results in literature for the problem at hand.

The proposed Four Neighborhood Variable Neighborhood Search (FN-VNS) algorithm unifies good characteristics of recent VNS algorithms in literature, together with a novel solution generator with a global greedy criteria that merges two exact solutions from subproblems of the SVRPDSP. This process is fast and allows the search to start from better quality initial solutions, further optimized by four different GPU local searches. The integrated perturbation scheme is a novel component that involves route recalculation in order to reduce communication time with the GPU and it is an important advance in order to design a complete metaheuristic in a CPU-GPU environment. From the managerial point of view, no parameter tuning is now involved with the proposed perturbation scheme in FN-VNS, thus allowing the design of a fast and self-adaptable algorithm for small instances and even for instances with hundreds of customers.

3.3 The Problem

In the SVRPDSP, the delivery demands of all customers must be satisfied, but the pickups are not obligatory. Consider a depot node c_0 , a set of n customers $C_{\mathcal{D}} = \{c_1, c_2, \ldots, c_n\}$ and $C_{\mathcal{P}} = \{c_{n+1}, c_{n+2}, \ldots, c_N\}$ called *delivery customers* and *pickup customers*, respectively. The delivery and pickup customers have delivery demands d_i , pickups p_i and revenues r_i such that: $\forall c_i \in C_{\mathcal{D}}, d_i > 0, p_i = r_i = 0$; and $\forall c_{n+i} \in C_{\mathcal{P}}, d_{n+i} = 0, p_{n+i} > 0,$ $r_{n+i} > 0.$

A solution S to the SVRPDSP consists of a permutation of all the delivery customers $C_{\mathcal{D}}$ and some selected pickup customers from $C_{\mathcal{P}}$. If a customer c_i precedes a customer c_j , then there is an arc connecting c_i to c_j with cost $M(c_i, c_j)$. A valid route starts and ends with the depot c_0 and respects the capacity Q of the vehicle, starting with load Q'. A load vector q is calculated such that $q_0 = Q'$ and for each customer c_i a load of $q_i = q_{i-1} - d_i + p_i$, i.e., consider the delivery as a negative value and a pickup as a positive value.

Figure 3.1 depicts a solution to the problem with vehicle capacity Q = 16, five delivery and pickup customers, where positive values denote pickups and negative values denote deliveries. The depot (square) indicates the initial load at the vehicle (Q' = 16 for this example). Note that pickup customer +8 is not visited in this route, so no revenue is gained from this pickup.



Figure 3.1: Solution S = [depot, -3, -2, +5, -6, -1, +3, -4, +4, depot] to a SVRPDSP with five delivery and pickup customers. The depot is denoted by the square. Pickups are positive values and deliveries are negative.

The evaluation of a solution consists of the sum of transportation costs minus the sum of revenues associated with the pickups. In order to guide the search only for feasible routes, all load values q_i that exceed Q are multiplied by a huge constant H. The objective function is presented in Eq. (3.1).

$$\min \sum_{(c_i, c_j) \in S} M(c_i, c_j) - \sum_{c_i \in S \mid c_i \in C_{\mathcal{P}}} r_i + H \times \sum_{c_i \in S} \max(q_i - Q, 0)$$
(3.1)

In [73], an efficient technique is used in order to reduce the complexity of solution reevaluations. We apply a similar technique for the feasible part of Eq. (3.1), but due to the presence of multiple maximum functions in the infeasible part, the complexity of the reevaluation is $\mathcal{O}(N)$. This is due to the fact that all maximum functions may change their values in the worst case, so the number of recomputed values is linear. This fact motivated the implementation of the evaluation procedure in a GPU, thus reducing the computational time by using a big number of parallel processing units.

3.4 Four-Neighborhood Variable Neighborhood Search

The proposed FN-VNS algorithm relies on the basic principles of the metaheuristic framework Variable Neighborhood Search (VNS), also incorporating a new high quality initial solution generator, an integrated CPU-GPU diversification mechanism and four different GPU searches.

The VNS is a metaheuristic framework proposed by [61], which main idea is to apply a systematic change of some selected *neighborhood structures* to explore the solution space. Let S be a solution to the optimization problem at hand and $\mathcal{N}_k(S)$ the set of neighbor solutions of S for neighborhood k. A neighbor solution S' is reached from solution S by an operation called *move*. The neighborhood can be fully explored in order to find better quality solutions. This classic heuristic, called Best Improvement (BI), is depicted in Algorithm 4.

Algorithm 4: Best Improvement (BI)
Input : S: Solution, $f(.)$: evaluation function, $\mathcal{N}_k(.)$: neighborhood structure
1 $S' \leftarrow S;$
2 $S \leftarrow arg \min_{X \in \mathcal{N}_k(S)} f(X);$
3 if $(f(S) \leq f(S'))$ then
$4 S' \leftarrow S;$
5 end
6 return S';

The schematic diagram for the FN-VNS is presented in Figure 3.2. The communications between CPU and GPU are represented in the diagram by boxes in the intersection between the CPU and GPU columns. For the manager that wishes to use only CPU technology, we provide also a fully CPU implementation of all GPU tasks. In this case, no communication between these hardwares is necessary. Steps 1-3 initialize the algorithm and copy the first generated solution to the GPU. The main loop is represented by steps 4-7, keeping the best known solution in s^* and limiting computational time to *IterMax* iterations without improvement. In steps 8 and 9 the integrated perturbation generates random move operations and updates the solutions in both CPU and GPU. Steps 10-12 represent a classic Variable Neighborhood Descent (VND) strategy proposed by [43], starting from the faster neighborhoods (k = 1) and moving to the slower ones. In step 13, given a neighborhood \mathcal{N}_k the GPU finds the best neighbor with the BI algorithm. Step 14 checks if any improvement is made on the current solution and step 15 finishes the algorithm by returning the best solution s^* .

The FN-VNS algorithm starts with the generation of an initial solution for the SVR-PDSP. In this algorithm, the problem is divided into two smaller subproblems, tackled by two exact solvers and then merged together according to a greedy criteria. Let $S_{\mathcal{T}}^*$ be an optimal solution of value $f_{\mathcal{T}}^*$ to a Traveling Salesman Problem consisting of all delivery customers plus the depot. Let $S_{\mathcal{K}}^*$ be an optimal solution of value $f_{\mathcal{K}}^*$ to a Knapsack Problem consisting of all pickup customers as items, maximizing the revenue r_i of the customers subject to the vehicle capacity Q [56]. It is worth mentioning that $f^* = f_{\mathcal{T}}^* - f_{\mathcal{K}}^*$ is a lower bound for the SVRPDSP [38]. The computation of the exact solutions $S_{\mathcal{T}}^*$ and $S_{\mathcal{K}}^*$ is also an \mathcal{NP} -Hard problem, so approximations of these solutions can be done by means of heuristics when computational time is prohibitive for pure exact methods. Finally, the solutions $S_{\mathcal{T}}^*$ and $S_{\mathcal{K}}^*$ must be merged to form a valid SVRPDSP solution S. The solution starts with the arcs from the route $S_{\mathcal{T}}^*$. In the algorithm proposed by [19], the pickup



Figure 3.2: General framework of the developed FN-VNS.

customers are inserted in S in random positions. Since this process may lead to many infeasible solutions when the vehicle capacity is exceeded during the route, we propose a global greedy strategy based on best feasible insertions. In this strategy, every non-visited pickup customer from $S_{\mathcal{K}}^*$ is considered for an insertion in best position of the route, i.e., minimizing the distance and maximizing the revenue. The best insertion is chosen such that the vehicle capacity is not violated and the process is repeated while there are pickup customers available. In the worst case, the complexity of the merge is $\mathcal{O}(N^2)$, since the route is gradually extended to $\mathcal{O}(N)$ customers with another linear calculation of $\mathcal{O}(N)$ at each iteration (for the best insertion and the feasibility test). Algorithm 5 describes this process in details.

The neighborhood structures are the basis of a VNS algorithm, so the FN-VNS was designed with four different neighborhood structures: *Swap*, 2-Opt, 1-OrOpt and 2-OrOpt. Each of them is able to induce a different set of neighbor solutions, by means of different move operations.

Algorithm 5: Initial solution generation								
Input : $C_{\mathcal{D}}$: Delivery customers, $C_{\mathcal{P}}$: pickup customers, M : cost matrix, Q :								
vehicle capacity, d : deliveries, p : pickups, r : revenues								
1 $S^*_{\mathcal{T}} \leftarrow \text{solve TSP } (C_{\mathcal{D}} \cup \{c_0\}, M);$								
2 $S_{\mathcal{K}}^* \leftarrow$ solve Knapsack Problem $(C_{\mathcal{P}}, Q, p, r);$								
$\mathbf{s} \ S \leftarrow S^*_{\mathcal{T}}$ //Initialize solution S with TSP route;								
4 while $\exists c_i \in S^*_{\mathcal{K}} \mid c_i \notin S$ do								
5 $\Delta f^* \leftarrow 0;$								
$6 \mathbf{for} \ c_i \in S^*_{\mathcal{K}} \mid c_i \notin S \ \mathbf{do}$								
7 Let Δf_i be the cheapest cost (distance) for the insertion of pickup customer								
c_i in S, respecting vehicle capacity Q;								
$\mathbf{s} \qquad \qquad \mathbf{if} \ \Delta f_i - r_i < \Delta f^* \ \mathbf{then}$								
9 $\Delta f^* \leftarrow \Delta f_i - r_i;$								
10 end								
11 end								
12 Add pickup customer c_i to S according to best insertion Δf^* ;								
13 end								
14 return S ;								

The Swap neighborhood is a restriction of the Osman Interchanges [65] and consists of

an exchange between two different customers. Figure 3.3 presents an example of a Swap move for the pickup customer +4 and delivery customer -1. It is worth mentioning that, in this case, the original solution (on the left of Figure 3.3, left) is not feasible due to the excess of one unit at customer +3. Indeed, considering Q' = 16, the load at customer +3 is: Q' - 3 - 2 + 5 - 6 + 4 + 3 = 17 > Q = 16. This situation is fixed after a swap operation in the solution (on the right of Figure 3.3). The 2-Opt, introduced by [53] for the TSP, removes two non-adjacent arcs from the route and adds two others generating a new route. In Figure 3.4, we show an example of a 2-Opt where the arcs (-1, +4) and (+3, depot) are removed, while the new arcs (-1, +3) and (+4, depot) are created. A 1-OrOpt move [64] relocates one customer to another position in the route. The 2-OrOpt is its natural extension when a block of two consecutive customers is relocated, instead of just one. Figure 3.5 presents an example of a 1-OrOpt move where the delivery customer -6 is moved to the position immediately after pickup customer +5.



Figure 3.3: Swap exchanges customers +4 and -1, leading the solution S = [depot, -3, -2, +5, -6, +4, +3, -4, -1, depot] to become the solution S' = [depot, -3, -2, +5, -6, -1, +3, -4, +4, depot].

With a view to providing diversity during the search process, a novel perturbation approach is also proposed. While the search process can get stuck in a local optimum trap from an specific neighborhood, a combination of moves from different neighborhoods can be strong enough to drive the search process into a more promising region of the solution space. This diversification cannot be too weak, otherwise the search will fall again into the same local optimum, but if it is too strong the search may skip a promising region. Due to the nature of the developed neighborhood structures, the diversity procedure must perform at least two consecutive moves, what may be enough to scape from shallow local



Figure 3.4: 2-Opt removes arcs (-1, +4) and (+3, depot); and adds arcs (-1, +3) and (+4, depot), leading solution S = [depot, -3, -2, +5, -6, -1, +4, -4, +3, depot] to S' = [depot, -3, -2, +5, -6, -1, +3, -4, +4, depot].



Figure 3.5: 1-OrOpt moves customer -6 to the position after customer +5. The arcs (-3, -6), (-6, -2) and (+5, -1) are removed, and the arcs (-3, -2), (+5, -6) and (-6, -1) are created. Solution S = [depot, -3, -6, -2, +5, -1, +3, -4, +4, depot] becomes S' = [depot, -3, -2, +5, -6, -1, +3, -4, +4, depot].

_

optima. But in order to scape from deeper local optima, the number of necessary consecutive moves can vary randomly, limited to a maximum of $\frac{|C_{\mathcal{D}}|+|C_{\mathcal{P}}|}{2}$ moves. Exceeding this limit would allow the solution to be completely rebuilt by the moves and the search process be driven to a random region in solution space, without the benefits of a high quality initial solution.

The integrated CPU-GPU search method FN-VNS is presented in Algorithm 6.

Algorithm 6: FN-VNS Algorithm
Input : <i>iterMax</i> : max. number of iterations without improvement, $f(.)$:
evaluation function, $\mathcal{N}_k(.)$: neighborhoods, $C_{\mathcal{D}}$: delivery customers, $C_{\mathcal{P}}$:
pickup customers, M : cost matrix, Q : vehicle capacity, d : deliveries, p :
pickups, r : revenues
1 $S \leftarrow \text{InitialSolution}(C_{\mathcal{D}}, C_{\mathcal{P}}, M, Q, d, p, r);$
$2 \; iter \leftarrow 1;$
3 while $iter \leq iter Max \ do$
$4 S' \leftarrow S;$
5 $l \leftarrow \text{random number [2, } \frac{ C_{\mathcal{D}} + C_{\mathcal{P}} }{2}];$
6 for $(i = 1 \ to \ l)$ do
7 $k \leftarrow \text{random number [1, 4]};$
8 $S'' \leftarrow$ random neighbor from $\mathcal{N}_k(S')$;
9 $S' \leftarrow S'';$
10 end
11 for $(k = 1 to 4) do$
12 $R \leftarrow \text{LocalSearch}(S'', f(.), \mathcal{N}_k);$
13 $S^* \leftarrow \operatorname{BI}(R, S'', \mathcal{N}_k);$
14 if $(f(S^*) < f(S))$ then
15 $S \leftarrow S^*;$
16 $k \leftarrow 1;$
17 $iter \leftarrow 0;$
18 end
19 end
20 $iter \leftarrow iter + 1;$
21 end
22 return S;

The proposed neighborhood structures are used in the diversification phase (lines 7-8) and also in the Local Search phase (line 12) in the following order (smaller to bigger neighborhoods): 2-Opt, Swap, 2-OrOpt and 1-OrOpt, defined as \mathcal{N}_1 , \mathcal{N}_2 , \mathcal{N}_3 and \mathcal{N}_4 , respectively. This block of four neighborhoods is denominated FN and gives the name to FN-VNS algorithm. The VND loop (lines 11-19) consists of: a LocalSearch procedure that explores the neighborhood \mathcal{N}_k in the GPU; a GPU feasibility test, that checks the viability of the solution; and a BI procedure, that updates the current solution. The feasibility test consists of checking every position in the solution vector if the capacity of the truck has been exceeded after a move. If an infeasible solution is found, the exceeding capacity is multiplied by a huge factor H, making the solution unattractive during the search process. The result of the Local Search process made by the GPU is stored in the auxiliary vector R. Later, if an improving move in found in R, the BI procedure applies the best move to the current solution. Whenever an improvement is made the *iter* counter is set to zero and the neighborhood counter k is also reset to the first neighborhood. When no neighborhood move is able to improve the current solution, the *iter* counter is increased. FN-VNS stops when the *iter* counter exceeds iterMax iterations without improvement.

3.4.1 Parallel Local Searches

In metaheuristics, and particularly for the FN-VNS, the computational cost of the search process is strongly dominated by the Local Search (line 12 of Algorithm 6). This is explained by the complexity of the developed neighborhood structures, which is $\mathcal{O}(N^3)$, since it is necessary to perform a $\mathcal{O}(N)$ feasibility test for each of the $\mathcal{O}(N^2)$ moves of each neighborhood. In fact, the feasible part of the objective function in Eq. (3.1) can be recalculated in constant time $\mathcal{O}(1)$, but the linearity of the feasibility test come from the penalization of infeasible solutions. In the worst case, all customers may exceed the vehicle capacity and every customer c_i may have a different penalized value $max(q_i - Q, 0) \times H$. Thus, we parallelized in a GPU this computationally demanding part of the code. The idea is to explore the massively parallel environment, the high memory bandwidth, and the efficient thread scheduling mechanism of the GPU to reduce the computational times and to allow tackling larger problem instances.

The GPU implementation of a Swap move (i, j) is presented in Algorithm 7.

Algorithm 7: Kernel for Swap neighborhood
Input : (i, j) : thread indexes, S: Solution, q: Loads, R: result vector, N: number
of customers, M : cost matrix, Q : vehicle capacity, d : deliveries, p :
pickups, r : revenues
$uvalue \leftarrow +M(S[i-1], S[j]) + M(S[j], S[i+1])$
2 $+M(S[j-1], S[i]) + M(S[i], S[j+1])$
3 $-M(S[i-1], S[i]) - M(S[i], S[i+1])$
4 $-M(S[j-1], S[j]) - M(S_j, S_{j+1});$
5 foreach customer $c_i \in S$ in the position after the possible move do
6 update evaluation with value $max(q_i - Q, 0) \times H$, if vehicle capacity is
exceeded at customer c_i
7 end
s $R[i \times N + j] \leftarrow value;$
9 return R;

The idea of the algorithm is to compute the cost of the move and to assert the feasibility of the new solution. As illustrated in Figure 3.6, the cost of the move is computed by replacing the distances from customers i - 1 to i and i to i + 1 and from customers j - 1 to j and j to j + 1, by the distances from customers i - 1 to j and j to i + 1 and from customers j - 1 to i and i to j + 1. The parallelization technique can explore the best characteristics of the CPU and the GPU, by means of a massive computational GPU algorithm that returns a cost vector R to the main flow of FN-VNS running in the CPU.

3.5 Experimental Results

We evaluate the performance and the quality of the solutions generated by FN-VNS. The results are compared against other SVRPDSP solutions in literature provided by [13]. Experiments were performed on the 68 problem instances from [38] and their respective lower bound values presented in literature. We divided the 68 instances from literature according to the instance sizes, *small*: 16 to 31 customers, *medium*: 33 to 72 customers, *large*: 76 to 101 customers. There are 28 instances in the *small* and *medium* groups; 12 instances in the *large* group. This new grouping scheme provides more insights on how FN-VNS deals with bigger problems.



Figure 3.6: Swap exchanges customers i and j by removing arcs i - 1 to i, i to i + 1, j - 1 to j, j to j + 1, and adding arcs i - 1 to j, j to i + 1, j - 1 to i, i to j + 1.

Our experimental platform is composed by a CPU Intel i7 2.67 GHz, with 8 GB of memory (only one CPU core was used), and a GPU GeForce GTX 560 Ti, with 1 GB of memory and 384 cores. The proposed algorithm was implemented in C++ and CUDA version 4.0. After preliminary tests, we fixed the algorithm to 256 threads per block and also limited the registers per block to 20, in order to achieve a better occupancy of the GPU processors.

3.5.1 Quality of Initial Solution

In order to test the quality of the solutions generated by the proposed global greedy constructive (Algorithm 5), it was compared to the randomized merge algorithm of [19]. Table 3.1 presents the results in terms of gap and time. The gap is calculated in relation to the lower bound for each instance, i.e., $gap(value) = (value-lower \ bound)/|lower \ bound|$. The column time presents the execution times in milliseconds.

From Table 3.1 it is possible to see that the proposed global greedy strategy generates solutions with much lower gaps than a pure random generation. On average a gap over 900% is obtained from a random merge, while gaps of 233% are achieved by a more elaborated greedy strategy. The running times of the proposed method are bigger, but less than one second on average, so they are still very small compared to the execution times of the complete FN-VNS.

Group of	[19]	Proposed	l constructive
Instances	gap(%)	time(ms)	gap(%)	time(ms)
small	655.61	0.007	142.64	1.606
medium	696.84	0.012	172.83	19.601
large	2286.46	0.019	589.29	192.711
Average	962.87	0.011	233.03	42.917

Table 3.1: Experiments with initial solution generation

3.5.2 Experiments with Local Searches

Since the neighborhoods 1-OrOpt and 2-OrOpt are special cases of the classic neighborhood 3-Opt, we investigated the impact of this neighborhood for the SVRPDSP. Table 3.2 presents the improvement of each local search after the constructive method. Considering all instances, the 2-Opt improves only 19.7% on average while the 3-Opt produces the best results for single local searches, with 31.5% of improvement on average. However, the FN structure consisting of neighborhoods 2-Opt, Swap, 2-OrOpt and 1-OrOpt explored as presented in Algorithm 6, lines 11-19, is capable of an improvement on initial solutions of 80.0% on average.

Group of		Improvement(%)								
Instances	2-Opt	Swap	2-OrOpt	1-OrOpt	FN	3-Opt				
small	18.0	23.7	20.4	27.1	54.2	30.6				
medium	28.0	39.2	38.9	42.1	131.3	42.8				
large	4.3	6.3	5.8	6.2	20.5	7.1				
All	19.7	27.0	25.4	29.6	80.0	31.5				

Table 3.2: Comparison of solution quality after local searches (including 3-Opt)

Table 3.3 presents the computational time of each local search after the constructive method. From this table, it is possible to see that neighborhoods 2-Opt, Swap, 2-OrOpt and 1-OrOpt are ordered in terms of computational time, from the least to the most expensive. The search process of the four neighborhoods FN combined take 274.5ms on average, while the 3-Opt takes nearly 15 times more time, 4112.2ms. This is due to the much bigger number of elements in the 3-Opt neighborhood, which is $\mathcal{O}(N^3)$, compared to the others that are quadratic. This experiment motivated us to discard the 3-Opt in our final algorithm, since the FN structure alone is already capable of big improvements, while retaining lower computational times.

Group of	Time(ms)								
Instances	2-Opt	Swap	2-OrOpt	1-OrOpt	FN	3-Opt			
small	1.8	2.0	4.6	5.3	27.5	125.5			
medium	13.0	13.5	32.1	34.1	181.5	1891.5			
large	79.4	82.3	191.1	199.5	1067.9	18596.0			
All	20.1	20.9	48.9	51.4	274.5	4112.2			

Table 3.3: Comparison of computational times of local searches (including 3-Opt)

3.5.3 Comparison with Literature

The execution time of FN-VNS depends on the parameter *iterMax*, which was set to 200 after a preliminary battery of tests. The calibration of this parameter seeks a good compromise between quality of solution and execution times. Since the loop variable *iter* is reinitialized after every improvement, the execution time grows quickly depending on the number of improvements and the shape of the solution space. This behavior is not linear and should be experimented empirically depending on the strength of the perturbation and local searches involved. The constant H was also calibrated and set to 100000. This decision considered programming language details, e.g. limitations on floating point types in C++ with very large values, also empirical tests were done to guarantee that no infeasible solution is chosen if it is possible to reach another high value feasible solution.

Tables 3.4 and 3.5 show FN-VNS results for the 68 instances from literature and more 32 new instances generated by the authors¹, considering from 150 up to 484 customers. In order to assess the quality of the solutions produced by FN-VNS, these are compared to a lower bound of the problem proposed by [38], which is presented in column LB. Column EA presents the best solutions found by the Evolutionary Algorithm of [13]. Columns best and avg presents, respectively, the best and average solutions found by FN-VNS in 10 executions. Column gap shows the gap between the best solution found by FN-VNS and the lower bound of the problem. Column var shows the variability of the average solutions in relation to the best solutions found by FN-VNS, given by:

$$var_i = \frac{avg_i - best_i}{|best_i|} \tag{3.2}$$

Finally, column time(s) $_{\times spd}$ presents the total time (in seconds) spent by the paral-

¹Instances are available online at the author's website

lel FN-VNS and the speedup in relation to the sequential FN-VNS, following the same notation of [55]. The CPU times are not presented in the tables since they can be easily deduced. Values printed in bold face show the improvements or the ties of the proposed algorithm upon the work by [13]. Unfortunately, it is not possible to compare the solution quality with the work of [38] for each instance, since in this work only general quality results are presented.

Our algorithm was able to find 51 new better solutions and 7 equal solutions, out of the 68 instances used. The average gap of 5.32% is also quiet low, since we compared the solutions with a lower bound of the problem. This shows that the best solutions achieved by the proposed algorithm are near optimal solutions. Finally, the low variability of 1.55% from the average solutions shows that the proposed algorithm is robust.

3.5.4 Efficiency of GPU Acceleration

As can be observed in Tables 3.4 and 3.5, FN-VNS improved the execution time for all the instances tested, except for one. The speedups increase with the problem size, varying from 0.93 to 14.49. So, for smaller instances the GPU time is nearly the same as the CPU time, but it is almost 15 times faster for the larger instances, showing the capability of the parallel FN-VNS to deal with larger problems.

Despite the great speedups obtained by FN-VNS, it is important to assess the efficiency of the local search separately. To perform this evaluation, we further included 32 new instances as a group *huge*, that we generated according to the methodology proposed in literature [38].

Table 3.6 shows the average and maximum speedups (avg, max) obtained by each neighborhood of the parallel FN-VNS algorithm when compared to the sequential execution of the neighborhood for each of the group of instances. The results show that our four proposed GPU neighborhoods consistently outperform the CPU counterparts, mainly for the larger instances where there is enough work to keep the threads busy in the highly parallel architecture of the GPU.

Analyzing the larger instances results in detail, we can observe that the *Swap* neighborhood presents the worst average speedup for the *large* instances. The *Swap* neighborhood handles eight arcs in the route, while k-OrOpts deal with six arcs, and 2-Opt deals with four arcs. This makes *Swap* the most expensive neighborhood in terms of computation and memory accesses. The necessary accesses to the global memory to update the solu-

Instance	LB	EA	best	avg	gap(%)	var(%)	$time(s)_{\times spd}$
016 B half	36.60	36.60	36.60	39.21	0.00	7.14	$1.91_{\times 1.03}$
016 B one	-155.41	-150.73	-150.73	-150.73	3.01	0.00	$1.33_{\times 1.06}$
016 ^B p two	130.99	135.11	132.27	134.26	0.98	1.50	$2.45_{\times 0.93}$
016 B two	-540.81	-536.13	-536.13	-536.13	0.87	0.00	$1.35_{\times 1.09}$
$021 B_{half}$	-20.16	-20.16	-18.62	-12.56	7.64	32.52	$2.78_{\times 1.77}$
$021 B_{one}$	-316.09	-301.93	-307.79	-307.79	2.63	0.00	$2.26_{\times 1.77}$
021_B_p_two	132.21	146.75	137.59	141.37	4.07	2.75	$4.03_{\times 1.69}$
021_B_two	-909.57	-901.28	-901.28	-901.28	0.91	0.00	$2.56_{\times 1.77}$
022_B_half	-64.97	-62.63	-62.63	-58.01	3.60	7.37	$2.85_{\times 1.93}$
022_B_one	-429.15	-421.04	-421.03	-421.03	1.89	0.00	$2.47_{\times 1.93}$
022_B_p_two	116.01	124.25	123.60	124.17	6.54	0.46	$2.50_{ imes 1.75}$
022_B_two	-1157.49	-1149.38	-1149.38	-1149.38	0.70	0.00	$2.47_{ imes 1.94}$
023_B_{half}	-94.06	-80.95	-80.95	-80.95	13.94	0.00	$2.75_{\times 2.05}$
023_B_one	-711.46	-698.35	-698.35	-698.35	1.84	0.00	$2.74_{\times 2.06}$
$023_B_p_two$	260.88	274.31	269.12	269.82	3.16	0.26	$2.80_{\times 1.98}$
023_B_{two}	-1946.21	-1933.10	-1933.10	-1933.10	0.67	0.00	$2.75_{\times 2.05}$
026_B_{half}	-92.47	-91.95	-88.20	-82.92	4.62	5.98	$5.79_{\times 2.54}$
026_B_one	-504.40	-497.17	-497.17	-497.17	1.44	0.00	$2.96_{\times 2.63}$
$026_B_p_two$	139.67	146.51	148.03	150.31	5.99	1.54	$4.55_{\times 2.53}$
026_B_two	-1341.49	-1334.26	-1334.26	-1334.26	0.54	0.00	$2.97_{\times 2.61}$
030_B_{half}	-382.80	-357.21	-377.12	-375.73	1.48	0.37	$8.47_{\times 3.63}$
030_B_{one}	-1156.23	-1120.33	-1150.44	-1149.35	0.50	0.09	$8.75_{ imes 3.63}$
$030_B_p_two$	81.33	82.29	82.29	86.27	1.19	4.82	$8.11_{\times 3.57}$
030_B_{two}	-2703.16	-2667.25	-2697.37	-2696.28	0.21	0.04	$8.76_{\times 3.65}$
031_B_{half}	-91.79	-85.56	-89.32	-87.15	2.69	2.43	$6.62_{\times 3.65}$
031_B_{one}	-514.05	-505.46	-510.50	-509.25	0.69	0.24	$7.26_{\times 3.60}$
031_B_p_two	115.52	123.15	123.15	125.34	6.61	1.77	$8.16_{\times 3.37}$
031_B_two	-1358.57	-1350.61	-1355.02	-1353.77	0.26	0.09	$7.27_{\times 3.61}$
033_B_{half}	-157.09	-137.29	-144.66	-144.66	7.91	0.00	$5.57_{\times 4.06}$
033_B_{one}	-778.21	-759.25	-765.76	-765.76	1.60	0.00	$5.50_{\times 4.07}$
033_B_p_two	188.44	198.44	197.01	203.55	4.55	3.32	$10.73_{\times 4.00}$
033_B_{two}	-2020.40	-2001.44	-2007.89	-2007.89	0.62	0.00	$5.87_{\times 4.08}$
036_B_{half}	-128.53	-113.44	-128.25	-126.20	0.22	1.60	$11.08_{\times 4.79}$
036_B_{one}	-624.67	-608.73	-624.41	-622.59	0.04	0.29	$10.99_{\times 4.79}$
$036_B_p_two$	121.94	139.71	132.05	134.52	8.29	1.87	$9.27_{ imes 4.19}$
036_B_{two}	-1616.90	-1596.84	-1616.63	-1614.81	0.02	0.11	$11.19_{\times 4.73}$
041_B_{half}	-186.35	-184.07	-183.46	-178.00	1.55	2.97	$13.67_{\times 5.67}$
041_B_{one}	-767.97	-755.69	-760.53	-758.88	0.97	0.22	$13.88_{\times 5.82}$
$041_B_p_two$	100.89	111.66	110.68	114.93	9.70	3.84	$11.20_{\times 5.29}$
041_B_two	-1931.31	-1922.74	-1923.88	-1922.23	0.38	0.09	$13.83_{\times 5.87}$

Table 3.4: Results for the 68 instances from literature (Part I).

Instance	LB	EA	best	avg	gap(%)	var(%)	$\texttt{time(s)}_{\times spd}$
045_B_half	-491.15	-489.47	-491.15	-491.15	0.00	0.00	$0.72_{\times 6.48}$
045_B_one	-1648.51	-1647.59	-1648.51	-1648.51	0.00	0.00	$0.73_{ imes 6.40}$
$045_B_p_two$	198.04	202.70	199.84	202.18	0.91	1.17	$12.60_{\times 6.62}$
045_B_{two}	-3963.32	-3963.32	-3963.32	-3963.32	0.00	0.00	$0.73_{\times 6.33}$
048_B_{half}	-37752.96	-36786.80	-36786.64	-36705.55	2.56	0.22	$20.42_{\times 6.96}$
048_B_one	-107058.78	-105863.00	-106625.21	-106542.29	0.40	0.08	$17.62_{\times 7.14}$
$048_B_p_two$	-4797.03	-3830.83	-4244.72	-3802.70	11.51	10.41	$18.36_{ imes 6.98}$
048_B_two	-248298.30	-247332.00	-247864.73	-247781.81	0.17	0.03	$17.59_{\times 7.15}$
051_B_{half}	-320.61	-310.02	-311.26	-309.53	2.92	0.56	$21.28_{\times 7.94}$
051_B_one	-1098.86	-1083.76	-1089.51	-1087.78	0.85	0.16	$21.28_{\times 7.93}$
$051_B_p_two$	116.58	135.34	130.99	133.85	12.36	2.18	$21.58_{\times 7.17}$
051_B_two	-2655.30	-2640.20	-2645.94	-2644.20	0.35	0.07	$21.27_{\times 7.92}$
072_B_{half}	-409.78	-388.84	-407.76	-405.92	0.49	0.45	$67.04_{\times 12.13}$
072_B_one	-1027.24	-998.14	-1024.53	-1023.20	0.26	0.13	$79.74_{\times 12.20}$
$072_B_p_two$	-39.24	-19.05	-36.20	-35.41	7.75	2.18	$58.55_{\times 12.22}$
072_B_two	-2262.21	-2232.95	-2259.32	-2257.63	0.13	0.07	$80.24_{\times 12.16}$
076_B_half	-579.52	-565.60	-574.08	-570.02	0.94	0.71	$66.57_{ imes 12.15}$
076_B_one	-1759.19	-1741.31	-1754.68	-1749.45	0.26	0.30	$79.42_{\times 12.17}$
$076_B_p_two$	14.33	34.04	38.41	38.41	168.04	0.00	$35.58_{ imes 10.87}$
076_B_two	-4118.50	-4099.12	-4114.30	-4108.92	0.10	0.13	$73.75_{\times 12.29}$
$101a_B_{half}$	-922.71	-911.68	-903.12	-896.57	2.12	0.73	$221.25_{\times 14.08}$
$101a_B_{one}$	-2552.38	-2538.99	-2531.06	-2525.65	0.84	0.21	$186.36_{ imes 13.91}$
101a_B_p_two	-66.59	-47.99	-46.79	-46.79	29.73	0.00	$99.51_{ imes 12.19}$
101a_B_two	-5811.69	-5800.66	-5790.86	-5785.43	0.36	0.09	$170.43_{\times 13.50}$
$101b_B_{half}$	-1386.67	-1380.63	-1381.01	-1379.78	0.41	0.09	$197.91_{ imes 14.49}$
$101b_B_{one}$	-3320.83	-3314.79	-3315.35	-3312.92	0.17	0.07	$187.62_{\times 13.59}$
$101b_B_p_two$	-257.18	-251.14	-248.74	-245.12	3.28	1.46	$181.00_{ imes 13.07}$
$101b_B_two$	-7188.92	-7182.88	-7183.25	-7181.20	0.08	0.03	$175.00_{\times 13.88}$
Average					5.32	1.55	$34.95_{\times 6.08}$

Table 3.5: Results for the 68 instances from literature (Part II).

Group of	Number of	Sw	Swap		2-Opt		1-OrOpt		2-OrOpt	
Instances	Instances	avg	max	avg	max	avg	max	avg	max	
small	28	1.81	3.33	2.61	4.73	3.61	6.21	3.16	6.14	
medium	28	7.63	16.41	10.36	20.49	12.81	24.69	11.28	21.00	
large	12	17.42	19.21	24.47	29.45	26.50	28.97	24.49	26.42	
huge	32	55.56	76.84	45.74	52.83	58.36	71.95	54.23	64.79	
All	100	22.51		21.21		26.45		24.33		

Table 3.6: Speedup results for FN-VNS neighborhoods.

tion vector **S**, make *Swap* slower than the others. However, the results are different for the *huge* instances. It seems that the cache memory in GPU global memory works better for larger routes, where the accesses of threads in closer blocks are stored more efficiently by the cache controller, increasing the speed of the access.

To validate the results obtained for this new set of instances, we evaluate the quality of the results obtained for these instances. Table 3.7 presents the Lower Bound (LB), the best and average solutions (best, avg) found by FN-VNS, the gap (gap), variability (var), and execution time in seconds (time). Due to the greater number of customers, the parameter *iterMax* of the algorithm was reduced to 20, but even with this small number of iterations, the sequential algorithm was not able to finish in reasonable times, so only the parallel FN-VNS results are presented.

As can be seen in the table, even for such large instances, the parallel FN-VNS algorithm succeeded in finding near optimal solutions, with a gap of 0.91% on the average, and also with a low variability of 0.40% from the average solutions. This validates the robustness of our approach. It is important to notice that the instances in the *huge* group were created for this work, making it impossible to compare their results with other authors.

3.5.5 Memory Hierarchy

The parallelization of the Local Search of FN-VNS exploits the massive computational capacity of the GPU. Nevertheless, there is still one important issue that needs to be investigated, how to deal with the GPU memory hierarchy efficiently. In terms of memory hierarchy, GPUs provides different memories with different bandwidths that can be leveraged to improve performance. The solution vector S has a linear size and we implemented two different allocation schemes for S. In the first one, called Global-S, we allocate S

Instance	LB	best	avg	gap(%)	var%)	time(s)
151_B_half	-1640.84	-1618.16	-1615.62	1.38	0.16	40.12
151_B_{one}	-4067.39	-4042.81	-4036.56	0.60	0.15	48.97
151_B_p_two	-267.52	-245.54	-236.71	8.22	3.60	34.35
151_B_two	-8920.43	-8903.88	-8890.57	0.19	0.15	51.96
200_B_half	-2259.56	-2231.61	-2224.00	1.24	0.34	122.76
$200 B_{one}$	-5402.65	-5378.39	-5361.71	0.45	0.31	143.94
200_B_p_two	-512.95	-492.01	-489.65	4.08	0.48	98.29
$200 B_{two}$	-11688.67	-11660.55	-11645.60	0.24	0.13	156.08
$256 B_{half}$	-1016.09	-1007.79	-1006.63	0.82	0.12	377.10
$256 B_{one}$	-2459.00	-2452.99	-2449.50	0.24	0.14	308.65
$256_B_p_two$	-151.77	-145.26	-142.39	4.29	1.98	390.21
256_B_{two}	-5343.33	-5335.78	-5333.55	0.14	0.04	363.09
$321 B_half$	-2031.52	-2027.10	-2021.58	0.22	0.27	522.61
$321 B_one$	-4902.92	-4900.16	-4893.26	0.06	0.14	508.53
321_B_p_two	-410.10	-403.44	-399.92	1.62	0.87	372.66
321_B_two	-10645.99	-10637.42	-10633.76	0.08	0.03	526.59
386_B_{half}	-46685.80	-46685.80	-46563.66	0.00	0.26	507.64
386_B_{one}	-102744.33	-102744.33	-102591.27	0.00	0.15	507.56
386_B_p_two	-13050.74	-13050.74	-12999.88	0.00	0.39	461.56
386_B_{two}	-214862.18	-214862.18	-214695.98	0.00	0.08	461.55
400_B_{half}	-2164.30	-2155.26	-2152.81	0.42	0.11	1025.44
400_B_one	-4999.38	-4989.82	-4986.61	0.19	0.06	1083.17
$400_B_p_two$	-463.18	-455.22	-452.02	1.72	0.70	1003.52
400_B_{two}	-10669.60	-10662.82	-10658.66	0.06	0.04	1051.67
481_B_{half}	-3988.65	-3975.52	-3967.16	0.33	0.21	2283.14
481_B_{one}	-9253.49	-9244.92	-9233.44	0.09	0.12	2576.68
481_B_p_two	-845.94	-840.97	-833.26	0.59	0.92	1512.03
481_B two	-19782.77	-19768.90	-19762.41	0.07	0.03	2573.12
484_B_{half}	-2955.05	-2944.73	-2941.37	0.35	0.11	2181.63
484_B_one	-6722.59	-6712.73	-6708.17	0.15	0.07	2181.42
$484_B_p_two$	-694.69	-686.83	-681.94	1.13	0.71	2438.10
484_B_{two}	-14256.85	-14248.10	-14243.64	0.06	0.03	2343.49
Average				0.91	0.40	883.05

Table 3.7: Quality of the solution for the generated huge instances.

Table 3.8: Performance gains of Shared-S implementation over Global-S implementation.

Group of	Number of	Average Time Decrease						
Instances	Instances	Swap	2- <i>Opt</i>	1-OrOpt	2-OrOpt			
$small^*$	28	-	-	-	-			
$medium^*$	28	-	-	-	-			
$large^*$	12	-	-	-	-			
huge	32	3.12%	3.15%	3.32%	6.26%			

*In groups *small*, *medium* and *large*, no statistical differences were observed with 95% confidence in a non-parametric test.

on the large high latency global memory and in the second one, called Shared-S, we allocate S in the low latency on-chip shared memory. The vector R was allocated in global memory due to its quadratic size.

We also made experiments to evaluate the impact of exploring the memory hierarchy of the GPU. In these experiments we compare our two different allocations of the solution vector, Global-S and Shared-S. Table 3.8 shows the percentage of the performance gains in the execution times for the four neighborhood of the Shared-S implementation over the Global-S implementation, for each group of instances. As can be observed in this table, for the *small*, *medium*, and *large* groups, the usage of the shared memory provides no substantial improvement. This occurs because the L2 cache mechanism of the GPU is effective in handling the accesses to the solution vector. In the huqe group, the usage of the shared memory improves the execution time of the neighborhoods from 3.12% to 6.26%, on the average. This improvement is achieved because in such large instances, the solution vector does not fit into the L2 cache. The misses in the L2 cache, which require accesses to the global memory, make the Global-S slower than the Shared-S. The gains of Shared-S over Global-S are, however, modest. In the Shared-S implementation, all threads in a block must copy collaboratively the vector S to the shared memory. After the vector is copied, it is accessed with a very low latency, but only a few elements are updated. So, the ratio between the cost of copying the data and the data reuse is high, what explains the small gains obtained.

3.6 Chapter Conclusions

In this chapter, we have dealt with the Single Vehicle Routing Problem with Deliveries and Selective Pickups (SVRPDSP). It is a very important and practical problem arising in reverse logistics and also in the design of green supply chains. The selective pickup component of the problem allows to include in the vehicle routing the product recovery, proper recycling and disposal. So, besides the optimization of a vehicle fleet for product deliveries, it provides waste reduction and cost savings by means of the value retrieved from returned goods. Many real-world cases illustrate the need of a selective pickup component, such as soft-drink distribution and distribution/recovery of copy machines.

Since the computation of the vehicle route is a \mathcal{NP} -hard problem, an efficient mechanism to provide routes in reasonable computational time is determinant for the success of the industry/business transportation problem. To deal with this problem we propose the FN-VNS algorithm, inspired by the metaheuristic Variable Neighborhood Search. It includes a novel high quality initial solution generator and an integrated perturbation strategy that provides diversity to four different local searches. These local searches are performed by means of modern parallel Graphical Processing Units (GPUs), which provide today the lower cost for high performance processing units.

We compared the results obtained by FN-VNS with the results of the best algorithm developed for the problem. For 68 instances proposed in the literature, containing from 16 to 101 customers, FN-VNS was able to improve the results of 51 instances and be equal in 7 instances. In terms of efficiency, our parallel version of FN-VNS outperformed the sequential version for all tested instances, except one. The speedups compared to the sequential FN-VNS varied up to 14.49 times, achieving better results as the size of the instances grows. We also evaluated the influence of the parallelization of the neighborhoods in the execution time. In this evaluation, we used a new set of instances that we created to represent larger size problems available online, containing from 151 to 484 customers. For these instances, our parallel implementation obtained remarkable average speedups ranging from 45.74 up to 76.84 for the neighborhood searches.

As future work, the proposed FN-VNS could be extended to solve related reverse logistics vehicle routing problems, for instance including time windows in the problem and solving the multi-vehicle variant of the problem. Some changes in the infeasibility function can be studied in order to allow a more efficient calculation of the objective function values in both CPU and GPU. From the supply chain manager point of view, a real-time framework for route recomputation could be incorporated to the proposed algorithm. This would require not only a communication with the navigation system of the vehicles, but also a more sophisticated parallel solution using multiple GPUs to increase the acceleration of the developed algorithm.

Capítulo 4

Conclusions and Future Works

In this thesis, we have dealt with two different routing problems, with single and multiobjective strategies. We have addressed the bi-objective CARP in which a set of edges has to be serviced by vehicles of limited capacity, so as to minimize the sum of all traversed arcs as well as the longest tour. The CARP consisting of only the first objective is well explored in literature and it is a \mathcal{NP} -Hard problem, so many heuristic approaches have been presented before in literature. We proposed a Matheuristic approach called MODEMAT, that integrates an exact dynamic programming decoder into the classic multi-objective evolutionary metaheuristic NSGA-II. Since the decoder generates a set of explicit solutions, a novel multi-objective framework was developed in order to extend classic multi-objective components for providing fitness assignment, diversity and elitism. Optimization techniques were presented in order to accelerate the convergence of MODE-MAT and also to prevent premature convergence. The results show that the proposed algorithm is competitive with literature, being able to achieve all optimal values for the second objective of the problem. For the first objective, MODEMAT is also competitive even in the number of optimal solutions found and for the average gap compared to a lower bound from literature.

We have also dealt with the Single Vehicle Routing Problem with Deliveries and Selective Pickups (SVRPDSP). It is a very important and practical problem arising in reverse logistics and also in the design of green supply chains. The selective pickup component of the problem allows to include in the vehicle routing the product recovery, proper recycling and disposal. So, besides the optimization of a vehicle fleet for product deliveries, it provides waste reduction and cost savings by means of the value retrieved from returned goods. Many real-world cases illustrate the need of a selective pickup component, such as soft-drink distribution and distribution/recovery of copy machines. Since the computation of the vehicle route is a \mathcal{NP} -hard problem, an efficient mechanism to provide routes in reasonable computational time is determinant for the success of the industry/business transportation problem. To deal with this problem we propose the FN-VNS algorithm, inspired by the metaheuristic Variable Neighborhood Search. It includes a novel high quality initial solution generator and an integrated perturbation strategy that provides diversity to four different local searches. These local searches are performed by means of modern parallel Graphical Processing Units (GPUs), which provide today the lower cost for high performance processing units.

We compared the results obtained by FN-VNS with the results of the best algorithm developed for the problem. For 68 instances proposed in the literature, containing from 16 to 101 customers, FN-VNS was able to improve the results of 51 instances and be equal in 7 instances. In terms of efficiency, our parallel version of FN-VNS outperformed the sequential version for all tested instances, except one. The speedups compared to the sequential FN-VNS varied up to 14.49 times, achieving better results as the size of the instances grows. We also evaluated the influence of the parallelization of the neighborhoods in the execution time. In this evaluation, we used a new set of instances that we created to represent larger size problems available online, containing from 151 to 484 customers. For these instances, our parallel implementation obtained remarkable average speedups ranging from 45.74 up to 76.84 for the neighborhood searches.

4.1 Future Works

As future work related to GPU computing, the proposed FN-VNS could be extended to solve related reverse logistics vehicle routing problems, for instance including time windows in the problem and solving the multi-vehicle variant of the problem. Some changes in the infeasibility function can be studied in order to allow a more efficient calculation of the objective function values in both CPU and GPU. From the supply chain manager point of view, a real-time framework for route recomputation could be incorporated to the proposed algorithm. This would require not only a communication with the navigation system of the vehicles, but also a more sophisticated parallel solution using multiple GPUs to increase the acceleration of the developed algorithm.

It is also worth mentioning that proposed MODEMAT algorithm is very generic and no specific construction heuristics were necessary in order to find good quality solutions. The algorithm can also be easily applied to other optimization problems and the ideas presented in this paper can also be extended to improve other single and multi-objective search algorithms.

References

- ALLENBY, B. Supporting environmental quality: Developing an infrastructure for design. *Environmental Quality Management* 2, 3 (1993), 303–308.
- [2] AMPONSAH, S. K.; SALHI, S. The investigation of a class of capacitated arc routing problems: the collection of garbage in developing countries. Waste management (New York, N.Y.) 24, 7 (Jan. 2004), 711–721.
- BALAS, E. The Prize Collecting Traveling Salesman Problem and Its Applications. In *The Traveling Salesman Problem and Its Variations*, D.-Z. Du, P. M. Pardalos, G. Gutin, and A. Punnen, Eds., vol. 12 of *Combinatorial Optimization*. Springer US, Boston, 2004, ch. 14, pp. 663–695.
- [4] BARTOLINI, E.; CORDEAU, J.-F.; LAPORTE, G. Improved lower bounds and exact algorithm for the capacitated arc routing problem. *Mathematical Programming 137*, 1-2 (2013), 409–452.
- [5] BELENGUER, J. M.; BENAVENT, E. A cutting plane algorithm for the capacitated arc routing problem. *Computers & Operations Research 30*, 5 (Apr. 2003), 705–728.
- [6] BELENGUER, J.-M.; BENAVENT, E.; PRINS, C.; PRODHON, C.; WOLFLER CALVO, R. A branch-and-cut method for the capacitated locationrouting problem. *Computers & Operations Research* 38, 6 (2011), 931–941.
- BETTAC, E.; MAAS, K.; BEULLENS, P.; BOPP, R. Reloop: Reverse logistics chain optimisation in a multi-user trading environment. In *Proceedings of the 1999 IEEE International Symposium on Electronics and the Environment. ISEE-1999.* (1999), IEEE, pp. 42–47.
- [8] BEULLENS, P.; MUYLDERMANS, L.; CATTRYSSE, D.; VAN OUDHEUSDEN, D. A guided local search heuristic for the capacitated arc routing problem. *European Journal of Operational Research* 147, 3 (June 2003), 629–643.
- [9] BEULLENS, P.; VAN OUDHEUSDEN, D.; VAN WASSENHOVE, L. N. Collection and vehicle routing issues in reverse logistics. In *Reverse Logistics*. Springer, 2004, pp. 95–134.
- [10] BODE, C.; IRNICH, S. Cut-First Branch-and-Price-Second for the Capacitated Arc-Routing Problem. Operations Research 60, 5 (Oct. 2012), 1167–1182.
- [11] BRANDÃO, J.; EGLESE, R. A deterministic tabu search algorithm for the capacitated arc routing problem. Computers & Operations Research 35, 4 (Apr. 2008), 1112–1126.

- [12] BRAS, B.; MCINTOSH, M. W. Product, process, and organizational design for remanufacture-an overview of research. *Robotics and Computer-Integrated Manufacturing* 15, 3 (1999), 167–178.
- [13] BRUCK, B. P.; DOS SANTOS, A. G.; ARROYO, J. E. C. Hybrid metaheuristic for the single vehicle routing problem with deliveries and selective pickups. In *Proceedings of the WCCI 2012 IEEE World Congress on Computational Intelligence* (Brisbane, Australia, 2012), pp. 10–15.
- [14] CAMPOS, V.; LAGUNA, M.; MARTÍ, R. Context-independent scatter and tabu search for permutation problems. *INFORMS Journal on Computing* 17, 1 (2005), 111–122.
- [15] CARTER, C. R.; ELLRAM, L. M. Reverse logistics-a review of the literature and framework for future investigation. *Journal of business logistics* (1998).
- [16] CHARNES, A.; COOPER, W. W. Goal programming and multiple objective optimization. European Journal of Operational Research 1, 1 (1977), 39–45.
- [17] CHEN, Y. L.; LIU, C. C. Multiobjective var planning using the goal-attainment method. Generation, Transmission and Distribution, IEE Proceedings 141, 3 (1994), 227–232.
- [18] CHRISTOFIDES, N. The vehicle routing problem. RAIRO Operations Research -Recherche OpA @rationnelle 10, V1 (1976), 55–70.
- [19] COELHO, I. M.; MUNHOZ, P. L. A.; HADDAD, M. N.; SOUZA, M. J. F.; OCHI, L. S. A hybrid heuristic based on General Variable Neighborhood Search for the Single Vehicle Routing Problem with Deliveries and Selective Pickups. *Electronic Notes in Discrete Mathematics 39*, 0 (2012), 99–106.
- [20] COELHO, I. M.; OCHI, L. S.; MUNHOZ, P. L.; SOUZA, M. J.; FARIAS, R.; BENTES, C. The Single Vehicle Routing Problem with Deliveries and Selective Pickups in a CPU-GPU Heterogeneous Environment. In 2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (June 2012), IEEE, pp. 1606–1611.
- [21] CORBERÁN, A.; PRINS, C. Recent results on arc routing problems: An annotated bibliography. *Networks* (2010).
- [22] DALÉ, L. B. D. C.; ROLDAN, L. B.; HANSEN, P. B. Analysis of sustainability incorporation by industrial supply chain in rio grande do sul state (brazil). *Journal* of Operations and Supply Chain Management 4, 1 (2013), 25–36.
- [23] DANTZIG, G. B.; RAMSER, J. H. The truck dispatching problem. Management Science 6 (1959), 80–91.
- [24] DE BRITO, M. P.; DEKKER, R. A framework for reverse logistics. Springer, 2004.
- [25] DEB, K.; PRATAP, A.; AGARWAL, S.; MEYARIVAN, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. Evolutionary Computation, IEEE Transactions on 6, 2 (2002), 182–197.
- [26] DEB, K.; SUNDAR, J.; N, U. B. R.; CHAUDHURI, S. Reference point based multiobjective optimization using evolutionary algorithms. In *International Journal of Computational Intelligence Research* (2006), Springer-Verlag, pp. 635–642.
- [27] DEKKER, R.; FLEISCHMANN, M. Reverse logistics: quantitative models for closedloop supply chains. Springer, 2004.
- [28] DERBEL, H.; JARBOUI, B.; CHABCHOUB, H.; HANAFI, S.; MLADENOVIC, N. A variable neighborhood search for the capacitated location-routing problem. In 4th International Conference on Logistics (LOGISTIQUA-2011) (2011), IEEE, pp. 514– 519.
- [29] DETHLOFF, J. Vehicle routing and reverse logistics: the vehicle routing problem with simultaneous delivery and pick-up. *OR-Spektrum* 23, 1 (2001), 79–96.
- [30] FARAHANI, R. Z.; REZAPOUR, S.; KARDAR, L. Logistics operations and management: concepts and models. Elsevier, 2011.
- [31] FLEISCHMANN, M.; KRIKKE, H. R.; DEKKER, R.; FLAPPER, S. D. P. A characterisation of logistics networks for product recovery. Omega 28, 6 (2000), 653–666.
- [32] FRENCH, S. Sequencing and Scheduling: An Introduction to the Mathematics of the Job-shop. Series in mathematics and its applications. E. Horwood, 1982. ISBN 9780853123644.
- [33] GHODSI, R.; AMIRI, A. S. A variable neighborhood search algorithm for continuous location routing problem with pickup and delivery. In *Fourth Asia International Conference on Mathematical/Analytical Modelling and Computer Simulation (AMS-2010)* (2010), IEEE, pp. 199–203.
- [34] GOLDEN, B.; DEARMON, J.; BAKER, E. Computational experiments with algorithms for a class of routing problems. *Computers & Operations Research 10*, 1 (1983), 47–59.
- [35] GOLDEN, B.; RAGHAVAN, S.; WASIL, E. The vehicle routing problem: latest advances and new challenges. Operations Research/Computer Science Interfaces Series, 43. Springer, 2008.
- [36] GOLDEN, B. L.; WONG, R. T. Capacitated arc routing problems. Networks 11 (1981), 305–315.
- [37] GREEN, K. W.; ZELBST, P. J.; MEACHAM, J.; BHADAURIA, V. S. Green supply chain management practices: impact on performance. *Supply Chain Management: An International Journal* 17, 3 (2012), 290–305.
- [38] GRIBKOVSKAIA, I.; LAPORTE, G.; SHYSHOU, A. The single vehicle routing problem with deliveries and selective pickups. Computers & Operations Research 35 (2008), Issue 9. 2908–2924.
- [39] GUNGOR, A.; GUPTA, S. M. Issues in environmentally conscious manufacturing and product recovery: a survey. *Computers & Industrial Engineering 36*, 4 (1999), 811–853.

- [40] GUTIÉRREZ-JARPA, G.; MARIANOV, V.; OBREQUE, C. A single vehicle routing problem with fixed delivery and optional collections. *IIE Transactions* 41, 12 (Oct. 2009), 1067–1079.
- [41] HAIMES, Y. Y.; LASDON, L. S.; WISMER., D. A. On a bicriterion formulation of the problems of integrated system identification and system optimization. Systems, Man and Cybernetics, IEEE Transactions on SMC-1, 3 (1971), 296–297.
- [42] HANSEN, P.; HEGEDAHL, B.; HJORTKJAER, S.; OBEL, B. A heuristic solution to the warehouse location-routing problem. *European Journal of Operational Research* 76, 1 (1994), 111–127.
- [43] HANSEN, P.; MLADENOVIĆ, N.; PÉREZ, J. M. Variable neighbourhood search: methods and applications. Annals of Operations Research 175 (2010), 367–407. 10.1007/s10479-009-0657-6.
- [44] HASSAN, M. M.; HOGG, G. L.; SMITH, D. R. Shape: a construction algorithm for area placement evaluation. *International Journal of Production Research* 24, 5 (1986), 1283–1295.
- [45] HERTZ, A.; LAPORTE, G.; MITTAZ, M. A tabu search heuristic for the capacitated arc routing problem. *Operations research* (2000), 129–135.
- [46] JANIAK, A.; JANIAK, W.; LICHTENSTEIN, M. Tabu search on GPU. Journal of Universal Computer Science 14, 14 (2008), 2416–2427.
- [47] KOOPMANS, T. C.; BECKMANN, M. Assignment problems and the location of economic activities. *Econometrica: Journal of the Econometric Society* (1957), 53– 76.
- [48] KUMAR, S.; TEICHMAN, S.; TIMPERNAGEL, T. A green supply chain is a requirement for profitability. *International Journal of Production Research* 50, 5 (2012), 1278–1296.
- [49] LACOMME, P.; PRINS, C.; RAMDANE-CHÉRIF, W. A genetic algorithm for the capacitated arc routing problem and its extensions. *Applications of evolutionary* ...2037 (2001), 473–483.
- [50] LACOMME, P.; PRINS, C.; RAMDANE-CHERIF, W. Competitive Memetic Algorithms for Arc Routing Problems. Annals of Operations Research 131, 1-4 (Oct. 2004), 159–185.
- [51] LACOMME, P.; PRINS, C.; SEVAUX, M. A genetic algorithm for a bi-objective capacitated arc routing problem. *Computers & Operations Research 33*, 12 (Dec. 2006), 3473–3493.
- [52] LAPORTE, G.; NOBERT, Y. An exact algorithm for minimizing routing and operating costs in depot location. *European Journal of Operational Research* 6, 2 (1981), 224–226.
- [53] LIN, S.; KERNIGHAN, B. W. An Effective Heuristic Algorithm for the Traveling-Salesman Problem. Operations Research 21, 2 (1973), 498–516.

- [54] LUONG, T. V.; LOUKIL, L.; MELAB, N.; TALBI, E. A GPU-based iterated tabu search for solving the quadratic 3-dimensional assignment problem. ACS/IEEE Intern. Conf. on Computer Systems and Applications 0 (2010), 1–8.
- [55] LUONG, T. V.; MELAB, N.; TALBI, E.-G. GPU Computing for Parallel Local Search Metaheuristic Algorithms. *IEEE Transactions on Computers* 62, 1 (Jan. 2013), 173–185.
- [56] MARTELLO, S.; TOTH, P. Knapsack problems: algorithms and computer implementations. John Wiley & Sons, Inc., New York, NY, USA, 1990.
- [57] MARTINEZ, C.; LOISEAU, I.; RESENDE, M.; RODRIGUEZ, S. BRKGA Algorithm for the Capacitated Arc Routing Problem. *Electronic Notes in Theoretical Computer Science 281* (Dec. 2011), 69–83.
- [58] MEI, Y.; TANG, K.; YAO, X. Decomposition-Based Memetic Algorithm for Multiobjective Capacitated Arc Routing Problem. *IEEE Transactions on Evolutionary Computation 15*, 2 (Apr. 2011), 151–165.
- [59] MELECHOVSKY, J.; PRINS, C.; CALVO, R. W. A metaheuristic to solve a locationrouting problem with non-linear costs. *Journal of Heuristics* 11, 5-6 (2005), 375–391.
- [60] MILLER, C. E.; ALBERT W. TUCKER, R. A. Z. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)* 7, 4 (1960), 326–329.
- [61] MLADENOVIĆ, N.; HANSEN, P. Variable neighborhood search. Computers & Operations Research 24, 11 (1997), 1097–1100.
- [62] NAGY, G.; SALHI, S. Location-routing: Issues, models and methods. European Journal of Operational Research 177, 2 (2007), 649–672.
- [63] NGUYEN, T. V. H. Development of Reverse Logistics-Adaptability and Transferability. Tese de Doutorado, Technische Universitat Darmstadt, november 2012.
- [64] OR, I. Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking, 1976.
- [65] OSMAN, I. H. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. Ann. Oper. Res. 41, 1-4 (May 1993), 421–451.
- [66] POLACEK, M.; DOERNER, K. F.; HARTL, R. F.; MANIEZZO, V. A variable neighborhood search for the capacitated arc routing problem with intermediate facilities. *Journal of Heuristics* 14, 5 (Oct. 2007), 405–423.
- [67] PORUMBEL, D.; HSU, T.; ALLAOUI, H.; GONCALVES, G. Arc-routing via column generation and iterated local search in a permutation set-covering framework. Submitted Paper (available as technical report cedric.cnam.fr/~porumbed/papers/ carp.pdf), 2014.
- [68] PORUMBEL, D. C.; HAO, J.-K.; KUNTZ, P. Spacing memetic algorithms. In GECCO (2011), pp. 1061–1068.

- [69] PRINS, C.; LABADI, N.; REGHIOUI, M. Tour splitting algorithms for vehicle routing problems. *International Journal of Production Research* 47, 2 (2009), 507– 535.
- [70] PRINS, C.; LACOMME, P.; PRODHON, C. Order-first split-second methods for vehicle routing problems: A review. *Transportation Research Part C: Emerging Technologies* 40 (2014), 179 – 200.
- [71] PRIVÉ, J.; RENAUD, J.; BOCTOR, F.; LAPORTE, G. Solving a vehicle-routing problem arising in soft-drink distribution. *Journal of the Operational Research Society* 57, 9 (October 2005), 1045–1052.
- [72] PRODHON, C.; PRINS, C. A survey of recent research on location-routing problems. European Journal of Operational Research 238, 1 (2014), 1–17.
- [73] PSARAFTIS, H. N. k-interchange procedures for local search in a precedenceconstrained routing problem. *European Journal of Operational Research* 13, 4 (1983), 391–402.
- [74] ROGERS, D. S.; TIBBEN-LEMBKE, R. An examination of reverse logistics practices. Journal of business logistics 22, 2 (2001), 129–148.
- [75] SANTOS, L.; MADEIRA, D.; CLUA, E.; MARTINS, S.; PLASTINO, A. A parallel GRASP resolution for a GPU architecture. In *International Conference on Metaheuristics and Nature Inspired Computing, META10* (Tunisia, 2010).
- [76] SARKIS, J.; RASHEED, A. Greening the manufacturing function. Business Horizons 38, 5 (1995), 17–27.
- [77] SCHULZ, C. Efficient local search on the GPU-Investigations on the vehicle routing problem. *Journal of Parallel and Distributed Computing* 73, 1 (2013), 14–31.
- [78] SIFALERAS, A.; KONSTANTARAS, I.; MLADENOVIC, N. Variable neighborhood search for the economic lot sizing problem with product returns and recovery. *International Journal of Production Economics 160*, 0 (2015), 133 – 143.
- [79] SINGH, S. P.; SHARMA, R. R. K. A review of different approaches to the facility layout problems. *The International Journal of Advanced Manufacturing Technology* 30, 5-6 (2006), 425–433.
- [80] SRINIVAS, N.; DEB, K. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation* 2 (1994), 221–248.
- [81] SRIVASTAVA, S. K. Green supply-chain management: a state-of-the-art literature review. International journal of management reviews 9, 1 (2007), 53–80.
- [82] SUBRAMANIAN, A.; DRUMMOND, L. M.; BENTES, C.; OCHI, L. S.; FARIAS, R. A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research* 37, 11 (2010), 1899 – 1911.
- [83] SUBRAMANIAN, A.; UCHOA, E.; OCHI, L. A hybrid algorithm for a class of vehicle routing problems. Computers & Operations Research 40, 10 (Oct. 2013), 2519–2531.

- [84] SURAL, H.; BOOKBINDER, J. H. The single-vehicle routing problem with unrestricted backhauls. *Networks* 41 (2003), 127–136.
- [85] TALBI, E.-G. Metaheuristics: From Design to Implementation. Wiley Publishing, 2009.
- [86] TANG, K.; MEI, Y.; YAO, X. Memetic Algorithm With Extended Neighborhood Search for Capacitated Arc Routing Problems. *IEEE Transactions on Evolutionary Computation 13*, 5 (Oct. 2009), 1151–1166.
- [87] THIERRY, M. C. An analysis of the impact of product recovery management on manufacturing companies. Tese de Doutorado, Rotterdam School of Management (RSM), Erasmus University, 1997.
- [88] TOKTAY, L. B.; WEIN, L. M.; ZENIOS, S. A. Inventory management of remanufacturable products. *Management Science* 46, 11 (2000), 1412–1426.
- [89] TOMPKINS, J. A.; REED JR., R. An applied model for the facilities design problem. The International Journal Of Production Research 14, 5 (1976), 583–595.
- [90] TOTH, P.; VIGO, D., Eds. The vehicle routing problem. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.
- [91] ULUSOY, G. The fleet size and mix problem for capacitated arc routing. European Journal of Operational Research 22, 3 (Dec. 1985), 329–337.
- [92] VALIDI, S.; BHATTACHARYA, A.; BYRNE, P. A case analysis of a sustainable food supply chain distribution system – a multi-objective approach. *International Journal of Production Economics* 152 (2014), 71–87.
- [93] VALIDI, S.; BHATTACHARYA, A.; BYRNE, P. Integrated low-carbon distribution system for the demand side of a product distribution supply chain: a doe-guided mopso optimiser-based solution approach. *International Journal of Production Re*search 52, 10 (2014), 3074–3096.
- [94] VAN HOEK, R. I. From reversed logistics to green supply chains. Supply Chain Management: An International Journal 4, 3 (1999), 129–135.
- [95] VIDAL, T.; CRAINIC, T. G.; GENDREAU, M.; PRINS, C. Heuristics for multiattribute vehicle routing problems: A survey and synthesis. *European Journal of Operational Research* 231, 1 (2013), 1–21.
- [96] WIENKE, D.; LUCASIUS, C.; KATEMAN, G. Multicriteria target vector optimization of analytical procedures using a genetic algorithm: Part i. theory, numerical simulations and application to atomic emission spectroscopy. *Analytica Chimica* Acta 265, 2 (1992), 211 – 225.
- [97] WIERZBICKI, A. The use of reference objectives in multiobjective optimization. In Multiple Criteria Decision Making Theory and Application, G. Fandel and T. Gal, Eds., vol. 177 of Lecture Notes in Economics and Mathematical Systems. Springer Berlin Heidelberg, 1980, pp. 468–486.

- [98] WINSEMIUS, P.; GUNTRAM, U. Responding to the environmental challenge. Business Horizons 35, 2 (1992), 12–20.
- [99] WONG, M.; WONG, T. Implementation of parallel genetic algorithms on graphics processing units. In *Intelligent and Evolutionary Systems* (2009), pp. 197–216.
- [100] YU, V. F.; LIN, S.-W.; LEE, W.; TING, C.-J. A simulated annealing heuristic for the capacitated location routing problem. *Computers & Industrial Engineering* 58, 2 (2010), 288–299.
- [101] ZHU, Q.; SARKIS, J. The moderating effects of institutional pressures on emergent green supply chain practices and performance. *International Journal of Production Research* 45, 18-19 (2007), 4333–4355.
- [102] ZITZLER, E.; THIELE, L. Multiobjective optimization using evolutionary algorithms - a comparative case study. In *Parallel Problem Solving from Nature - PPSN* V, A. Eiben, T. Back, M. Schoenauer, and H.-P. Schwefel, Eds., vol. 1498 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 1998, pp. 292–301.