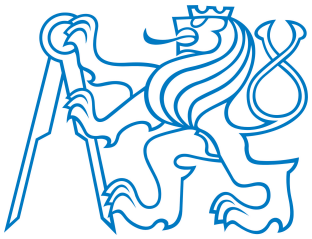
Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Cybernetics



Agent-based Algorithms for the Vehicle Routing Problem with Time Windows

Doctoral Thesis

Petr Kalina

Prague, August 2014

Ph.D. Programme: Electrical Engineering and Information Technology
Branch of study: Artificial Intelligence and Biocybernetics

Supervisor: Prof. Ing. Vladimír Mařík, DrSc.
Supervisor-Specialist: Doc. Ing. Jiří Vokřínek, Ph.D.

Dedicated to my family.

Acknowledgments

First and foremost I would like to thank Ladislav Janiga, who was the original inspiration for me to take this path. This thesis would also never come to being without the support of my supervisors Vladimír Mařík and Jiří Vokřínek. I'd further like to thank all my present and past colleagues from the Agent Technology Center and the Intelligent Systems Group here at the CTU Prague, the School of Computing and Engineering at the University of Huddersfield, England and the Optimization Research Group of the NICTA, Sydney, Australia, for feedback, advice, inspiration and friendship. Lastly I would like to express my gratitude towards all my family and friends who supported and motivated me throughout this effort.

Abstract

The vehicle routing problem with time windows (VRPTW) is one of the most important and widely studied transportation optimization problems. It abstracts the salient features of numerous distribution related real-world problems. It is a problem of finding a set of routes starting and ending at a single depot serving a set of geographically scattered customers, each within a specific time-window and with a specific demand of goods to be delivered.

The real world applications of the VRPTW can be very complex being part of higher level systems i.e. complex supply chain management solutions. For a successful deployment it is important for these systems to be flexible in terms of incorporating the problem specific side-constraints and problem extensions in an elegant way. Also, employing efficient means of addressing the dynamism inherent to the execution phase of the relevant operations is vital.

The multi-agent systems are an emerging architecture with respect to modeling multi-actor heterogenous and dynamic environments. The entities within the system are represented by a set of agents endowed with autonomic as well as social behavioral patterns. The behavior of the system then emerges from their actions and interactions. The autonomic nature of such a model makes it very robust in highly uncertain dynamic environments. The problem specific side-constraints can then be embedded within the agents' local subproblem solving. Empowered with efficient behavioral and communication patterns the multi-agent paradigm provides an intriguing alternative to the traditional optimization methods.

In this work we present a reformulation of the VRPTW as a multi-agent optimization problem within a society of agents representing individual vehicles being part of the problem. Alternative local planning strategies used by the agents as well as efficient interaction patterns enabling for finding efficient solutions to the global problem are introduced, providing for several incremental versions of the resulting VRPTW algorithm. A detailed experimental assessment of the alternative algorithm versions is presented including the comparison to the traditional centralized as well as previous agent-based algorithms. Such a comparison was missing from most previous comparable agent-based works. An in-depth analysis of the underlying solving process is provided as well revealing interesting future research opportunities.

Over the relevant benchmarks, the algorithm equals the best-known solutions achieved by the state-of-the-art traditional algorithms in 90.3% of the cases, resulting in a 0.3% overall relative error. This represents a significant improvement over the previous comparable agent-based algorithms. A parallel version of the algorithm is presented as well, boasting very good anytime attributes, arguably outperforming even the traditional algorithms in this respect. Underlying the presented algorithm is the introduced abstract task allocation framework, significantly extending the previous similar concepts. The core contribution of this thesis is the deeper understanding of the implications of adopting an agent-based approach to solve the VRPTW and complex transportation optimization problems in general.

Abstrakt

Vehicle routing problém s časovými okny (VRPTW) je jedním z nejdůležitějších a nejvíce zkoumaných problémů v oblasti dopravy. Matematický model tohoto problému vystihuje klíčové vlastnosti společné celé řadě dalších dopravních problémů řešených v praxi. Jádrem problému je hledání množiny tras začínajících a končících v jediném depu, které obsahují zastávky u množiny zákazníků. Pro každého zákazníka je pak definováno konkrétní množství zboží, jež je třeba doručit a časové okno, ve kterém je požadováno dodání tohoto zboží.

Reálné aplikace tohoto problému jsou zpravidla výrazně bohatší, napojené na nadřazené logistické systémy. Klíčovým faktorem pro uspěšné nasazení odpovídajících algoritmů je proto jejich flexibilita vzhledem k dodatečným rozšířením základního matematického modelu spojeným s nasazením v reálném světě. Dalším podstatným faktorem je schopnost systému reagovat na nepředvídané události jako jsou dopravní zácpy, poruchy, změny preferencí zákazníků atd.

Multi-agentní systémy reprezentují architekturu a návrhový vzor vhodný pro modelování heterogenních a dynamických systémů. Entity v systému jsou v rámci multi-agentního modelu reprezentovány množinou agentů s odpovídajícími vzorci autonomního jako i společenského chování. Chování systému jako celku pak vyplývá z autonomních akcí jednotlivých agentů a jejich interakcí. Díky autonomní povaze tohoto modelu je výsledný systém velice robustní v prostředích s velkou mírou nejistoty. Zmíněná rozšíření problému i dodatečná omezení mohou být zakódována do lokálních vzorců chování agentů včetně autonomních vzorců chování pro řešení lokálně pozorovaných dynamických změn v systému. Jako takové, multi-agentní systémy představují zajímavou alternativu tradičních centralizovaných optimazlizačních metod.

V této práci představujeme reformulaci VRPTW jako multi-agentního optimalizačního problému ve společnosti agentů reprezentujících jednotlivá vozidla, jež jsou součástí problému. V centru zájmu jsou pak alternativní přístupy k lokálnímu plánování agentů, stejně jako různé modely interakce mezi agenty umožňující efektivně kooperovat směrem k efektivnímu řešení problému. Na jejich základě je představeno několik inkrementálních verzí výsledného algorithmu pro VRPTW. V experimentální části této práce pak přdstavujeme srovnání výsledého algoritmu s úspěšnými tradičními centralizovanými algoritmy, stejně jako i s předchozími agentními algoritmy. Takovéto relevantní srovnání chybí ve většině předešlých srovnatelných agetních prací. Experimentání část práce navíc obsahuje i řadu dodatečných experimentů odhalujících hlubší povahu řešicího procesu.

Srovnání s existujícími algoritmy využívá množiny testovacích problémů, které jsou dobře známé v oblasti operačního výzkumu. V 90.3% řešených případů dokázal algoritmus nalézt řešení stejné kvality jako nejlepší centralizované algoritmy s výslednou celkovou relativní odchylkou 0.3%. Tento výsledek je výraznýcm zlepšením oproti předchozím srovnatelným agentním studiím. Dalším pozoruhodným výsledkem je konvergence paralelní verze představeného algoritmu, která v tomto ohledu překonává i zavedené centralizované metody. Hlavním přínosem této práce je hlubší porozumění důsledkům vyplývajícím z použití multi-agentního přístupu k řešení VRPTW a komplexních transportních problémů obecně.

Contents

Chapter 1

Introduction

The multi-agent systems are an emerging architecture and paradigm with respect to modeling heterogenous, potentially noncooperative systems and environments [125]. The individual agents represent autonomous entities being part of the system with potentially diverging information about the global state of the system as well as potentially diverging interests or goals. The two fundamental components underpinning the operation of the multi-agent systems are (i) the specific planning strategies and autonomic behavioral patterns employed locally by the agents and (ii) the interactions between the individual agents being part of the system. The overall behavior of the system as a whole then emerges as a result of the agents' behavior and interactions. [103].

In many cases it is desirable to quantify and optimize the performance of the society of agents when dealing with a specific task. For self-interested agents pursuing their own private goals their local profits can be quantified in order to evaluate the alternative strategies the agents can use in order to achieve these goals. On the other hand, the performance of the agents' society as a whole when dealing with a common problem can be analyzed as well and efficient means of coordinating the agents in the joint problem solving effort can be researched. Lastly, the tradeoff between the two — the agents' private profits and the social welfare can be studied as well. Similar questions are at the center of interest for the area of multi-agent optimization representing also the core framework for this work. Several illustrations of the cooperative societies of agents found in nature as well as in human operations are illustrated by Figure 1 [1].

[1]The images are part of publicly available image libraries with no royalties imposed on non-profit usage

Figure 1.1: Cooperative multi-agent systems in nature and human society (left to right): autonomic systems in car transportation, the society of bees cooperating on feeding the offspring, the society of ants cooperating on traversing an obstacle

Multi-agent optimization is at the forefront of modern optimization theory, lying at the intersection of classical optimization, game theory and social economics. In optimization, the multi-agent paradigm is used to model optimization problems with high level of heterogeneity, dynamism, antagonistic interests of the participating parties or constraints concerning the local information available to the parties. The overall performance of the society can be expressed in number of forms e.g. social welfare, pareto efficiency, fairness of the allocation of resources etc. while game theoretic features like the Nash dynamics, worst and best case equilibria etc. can be analyzed as well [79, 12, 84, 63]. In order to optimize the performance of the system with respect to similar metrics the alternative local behavioral patterns of the agents' can be researched as well as the structure of agents' interaction and coordination mechanisms.

In this thesis we present a multi-agent optimization model for the vehicle routing problem with time windows (VRPTW). The VRPTW is a well known optimization problem. It is one of the most important and widely studied problems within the operations research (OR) domain, featuring in many distribution and logistic systems. The VRPTW consists of finding a set of routes starting and ending at a single depot serving a set of geographically scattered customers, each within a specific time-window and with a specific demand of goods to be delivered. In the presented multi-agent algorithm, the individual vehicles contributing to the overall solution to the problem are represented as autonomously planning agents each employing a specific local planning strategy. The optimization criterium of the problem then coincides with the social welfare of the society of these agents. The overall solution to the problem maximizing the social welfare is then sought by means of intelligent coordination of vehicles' respective plans by means of negotiation, resulting in advantageous customer allocations and relocations being performed.

While the VRPTW admits an efficient application of a centralized approach the related real-world problems are typically very complex making the adoption of a multi-agent approach an interesting alternative. The relevant extensions include, for example, heterogenous fleets with varied operational costs [71], specific local constraints e.g. loading constraints [124], driver shift times [93] or uncertainty and dynamism induced by real-world challenges e.g. unexpected traffic conditions, vehicle breakdowns and the corresponding execution requirements i.e. real-time replanning etc. The presented problem decomposition is very robust with respect to similar requirements. All the problem specific logic is embedded within the agents' local planning and can easily encapsulate any additional local constraints. On the other hand, the coordination mechanism is abstract and problem independent. Moreover, the customer allocations are represented as agents' social commitments [62]. Such a representation presents a powerful tool for task execution stability and performance in dynamic and/or uncertain environments, further contributing to the expressivity of the presented multi-agent model.

Core to the presented effort is also the focus on the resulting algorithm's performance when compared to the traditional centralized algorithms. Considering the previous similar studies, both the breadth of provided experimental evidence as well as the resulting solution quality represent a significant improvement over the existing state-of-the-art. The importance of the presented findings is further accentuated by the fact that the abstract coordination framework, being part of the presented algorithm, can be easily reused to address not only the relevant VRPTW extensions, but also other general task allocation problems, representing a significant extension of the previous similar abstract framework presented in [121].

1.1 Goals of this Thesis

This thesis aims at reformulating the VRPTW as a multi-agent optimization problem and researching the alternative behavioral and interaction patterns in an effort to provide efficient multi-agent VRPTW algorithm. It further aims at (i) discussing the implications of adopting a multi-agent approach to solve the VRPTW and (ii) rigorously assessing the performance of the thus developed incremental algorithm versions including the sound comparison to the state-of-the-art traditional methods missing from most previous similar agent-based studies. The goals of this thesis can be summarized as follows:

1. Reformulate the VRPTW as a multi-agent optimization problem.
2. Research alternative ways the local agents' subproblems can be efficiently addressed.
3. Research alternative ways the global coordination within the society of the agents can be addressed enabling the society to address the joint optimization problem in an efficient way
4. Develop an algorithm for the VRPTW that is competitive in its performance. In particular, the aim is to achieve a performance that is within 1% in terms of the overall relative error with respect to the state-of-the-art traditional algorithms that is usually considered to be separating the efficient methods from the less successful ones.
5. Provide an in depth analysis if the underlying solving process addressing the problematic areas within the corresponding research area of agent-based VRPTW algorithms, namely (i) the lack of formal analysis, (ii) the lack of relevant comparison to the state-of-the-art traditional centralized methods and (iii) the unsatisfactory performance of the previous agent-based VRPTW algorithms where such a comparison was provided.

1.2 Organization of the Thesis

This thesis thus aims at researching the agent-based algorithms for efficiently solving the VRPTW in an effort to provide an alternative to the existing centralized algorithms. Central to this effort is the introduction of a fitting agent-based problem decomposition and featuring a problem independent abstract coordination layer used to coordinate the fleet of agents contributing to the overall problem solution. Alternative semantics of the coordination process are discussed as well as the alternative local planning strategies used by the agents. The presented concepts are then assessed in an extensive experimental evaluation, including the comparison of the resulting algorithm to the state-of-the-art centralized VRPTW algorithms. The work is thus positioned on the border of optimization and operations research on one hand and the multi-agent systems and multi-agent planning on the other hand.

The organization of the thesis is as follows:

- Chapter 2 introduces the VRPTW and discusses alternative approaches to solving the problem as well as the relevant related research.
- Chapter 3 provides a brief introduction to the fields of social choice and multi-agent optimization.
- Chapter 4 presents an agent-based VRPTW algorithm representing the core contribution of this thesis. An abstract, problem independent task-allocation framework is presented and its adaptation for the VRPTW is discussed. Several incremental versions of the resulting VRPTW algorithm are introduced.
- Chapter 5 finally provides an extensive experimental evaluation of all the introduced concepts.

Chapter 2

Vehicle Routing Problem With Time Windows

Transportation is one of the core human activities. Irrespective whether the transported items are people, packets in a network or we're just planning a trip, there's always some *routing algorithm* involved in order to perform the corresponding activity in an efficient and timely manner. In [57] the losses incurred by excess travel in U.S. fright transportation are analyzed with the resulting loss estimate being USD 45 billion. Such a result highlights the immense relevance of developing and deploying appropriate and efficient routing algorithms. The magnitude of the related real-world problems is further highlighted by Figure 2.1.

The Vehicle Routing Problem with Time Windows is one of the most widely studied problem in operations research. In essence, the VRPTW abstracts the salient features of the notoriously common real-world problem of distributing goods across a set of customers using a set of capacity constrained transportation units and with temporal constraints being imposed at deliveries to the individual customers. As such it provides a simple model to study the outstanding features of the related real-world problems and test and compare the efficiency of various solving methods to tackle these problems. In this chapter we provide the formal problem definition and the formal framework used throughout this work as well as an overview of the problem and the known VRPTW solving methods.

Figure 2.1: Merchant shipping — 90% of international trade passes this way, with over 100,000 commercial ships operating worldwide. A freight transportation port (left). A fully loaded freight ship (right).

2.1 Problem Statement and Formal Framework

The Vehicle Routing Problem With Time Windows (VRPTW) is a problem of finding a set of routes from a single depot to serve customers at geographically scattered locations, delivering to each customer a specified amount of goods. The problem specific constraints require that (i) each customer is served within a given time interval and (ii) the capacity of neither of the vehicles serving the individual routes is exceeded. Formally the problem can be defined as follows:

Definition 1 (Vehicle Routing Problem with Time Windows)**.** The VRPTW is given by a tuple (D, C, T). D is the vehicle capacity (a homogeneous fleet is assumed). $C = \{1 \dots N\}$ is the set of customers to be served. For each customer the earliest/latest service start times e_i, l_i are given (the *time window*) as well as the corresponding service time s_i and demand $d_i, i = 1 \dots N$. The $T = \{t_{i,j}\}, i, j = 0 \dots N$ are the mutual travel times between the customers with $t_{i,j} = t_{j,i}, i, j \in 1..N$ denoting the travel times between the customers i and j and $t_{0,i} = t_{i,0}, i = 1 \dots N$ being the travel time between the depot and the customer i.

A feasible solution σ to the VRPTW is a set of feasible routes $\{r_i\}, i = 1 \dots k$ serving all the customers $1 \dots N$. A route corresponds to a sequence of customers $\langle c_0, c_1, ..c_m, c_{m+1} \rangle, c_i \in 0..N$. Each route has to start and end at the depot i.e. $c_0 = c_{m+1} = 0$ and no backhauls are allowed i.e. $\forall i \notin \{0, m+1\}, c_i \neq 0$. Moreover, each customer has to be served by exactly one route i.e. for a pair of routes $r_1 = \langle c_0^1 \dots c_{m+1}^1 \rangle$ and $r_2 = \langle c_0^2 \dots c_{n+1}^2 \rangle$, $c_i^1 \neq c_j^2, i \in 1 \dots m, j \in 1 \dots n$.

A route is feasible when both the time-window constraints and the capacity constraint are satisfied. Let $E_i, i \in 1..m$ be the earliest possible service start time at the customer c_i on the route. For c_1 the E_i is trivially computed as $E_1 = max(s_1, t_{0,1})$. For the remaining customers it can be computed recursively as $E_j = max(s_j, E_{j-1} + s_{j-1} + t_{j-1,j})$. The time-windows constraints are thus satisfied whenever $E_j \leq l_j, j = 1 \dots m$, i.e. the vehicle arrives to each of the customers prior the expiry of the corresponding time-window i.e. $\forall i = 1..m, E_i \leq l_i$ The capacity constraint is satisfied when $\sum_1^m d_i \leq D$ i.e. the sum of demands of the served customers does not exceed the vehicle capacity.

The VRPTW solution σ corresponds to a set of routes i.e $\sigma = \{r_1 \dots r_k\}$. A feasible solution is a solution in which all the routes $r_i \in \sigma$ are feasible. A complete solution is a solution in which all the customers are served i.e. $\forall i \in C, \exists r_j \in \sigma$ s.t. $i \in r_j$. The VRPTW is then defined as finding a minimal feasible complete solution σ i.e. a minimal set of feasible routes such that all the customers are served.

In the definition above we introduced the route and solution feasibility and solution completeness. Below we define the complementary solution attributes:

Definition 2 (Infeasible and Partial Solution)**.** A solution $\sigma = \{r_1 \dots r_k\}$ is infeasible whenever any of the routes $r_i \in \sigma$ is infeasible i.e. violates any of the problem specific constraints. A solution is partial whenever there exist a customer i not being served by any of the routes in $r_j \in \sigma$ i.e. $\exists i \in C$ s.t. $\forall r_j \in \sigma, i \notin r_j$.

We define the closely related pickup and delivery problem with time windows (PDPTW) informally. The PDPTW differs from the VRPTW in that the tasks are given in pairs — the *pickup* task and the corresponding *delivery* task corresponding to transporting a certain amount of goods between the two corresponding customer locations. Thus, naturally, both these tasks have to be served within a single route, the pickup preceding the delivery (pairing constraint). The vehicles leave the depot empty an pickup and unload along the way based on the way in which the individual pickups and deliveries are distributed along the corresponding route. The load of the vehicle at a given point of the corresponding route therefore depends on the actual

interleaving of individual pickups and deliveries on that route. The capacity constraint for the PDPTW is thus satisfied when the vehicle capacity D is not exceeded at *any point* on the route.

2.1.1 VRPTW Complexity

The VRPTW is in fact an extension/interdigintation of several well known problems which are known to be *NP-Complete*. Consider a specific VRPTW instance with infinitely benevolent time windows. In such a case the minimal fleet objective corresponds to solving the decision variant of the multiple bin-packing problem. For a problem instance with all customers sharing identical location and considering an infinite vehicle capacity, the problem resembles parallel multi-machine scheduling with release times and deadlines which has been proved to be strongly *NP-complete* [19]. Lastly, consider a problem instance with all the customers having identical time-windows, considering an infinite vehicle capacity as well. An efficient solution to such a problem naturally requires the vehicles to minimize the traveled distances in order to be able to serve maximum customers within the time-frame given by the time-windows. In this respect the problem resembles the multiple traveling salesmen problem. Based on these assumptions the problem is obviously *NP-hard*. A formal proof of *NP-hardness* in the strong sense of the VRPTW can be found in [66].

2.1.2 Upper and Lower Bounds

The VRPTW admits the identification of the lower bound number of vehicles admissible for the particular problem instance being solved. Two alternative values for the lower bound can be computed based on the underlying multiple bin packing problem denoted the *capacity based lower bound number of vehicles* (NV_{cap}) and based upon the mutual incompatibilities of the individual customers based upon their distance and their time-windows denoted the *time-windows based lower bound number of vehicles* (NV_{tw}).

Consider problem $P = (D, C, T)$. Given the vehicle capacity D and the demands of all the customers $d_i, i \in C$ it is obvious that no feasible solution can use a number of vehicles lower than the number of bins of a size D in the optimal solution to the corresponding multiple bin packing problem. In case of a fleet smaller than this number is used, this does not permit for the fleet to accommodate simultaneously all the customer deliveries. As no backhauls are allowed, this would prevent some of the customers from being served.

As mentioned, an alternative NV_{tw} lower bound can be computed based on the mutual customer incompatibilities. Two customers are incompatible if they cannot be served by a single vehicle, that is when it is impossible to start the service at either of the customers at the earliest possible moment and reach the other customer within the corresponding time window. The situation is illustrated by Example 2.1.1.

Example 2.1.1. Imagine a situation depicted on Figure 2.2 i.e. serving a customer in Sydney and another one in Prague delivering a package issued from a delivery central in Hong Kong, delivered by an agent traveling by plane. Imagine both deliveries have to be carried out between 1pm and 2pm on a same day. As the flight from Sydney to Prague or vice versa takes considerably more than 1 hour, the solution to any delivery problem being a superset of these two customers doesn't admit a solution with less than two agents.

The time-windows based lower bound number of vehicles NV_{tw} necessary for solving the instance is thus bound by the size of the maximal set of mutually incompatible customers. The customer mutual incompatibilities can be easily captured within a graph. Consider a graph $I = (C, E)$ with the vertices $C = 1..N$ corresponding to the individual customers $E = \{(i, j)|i$

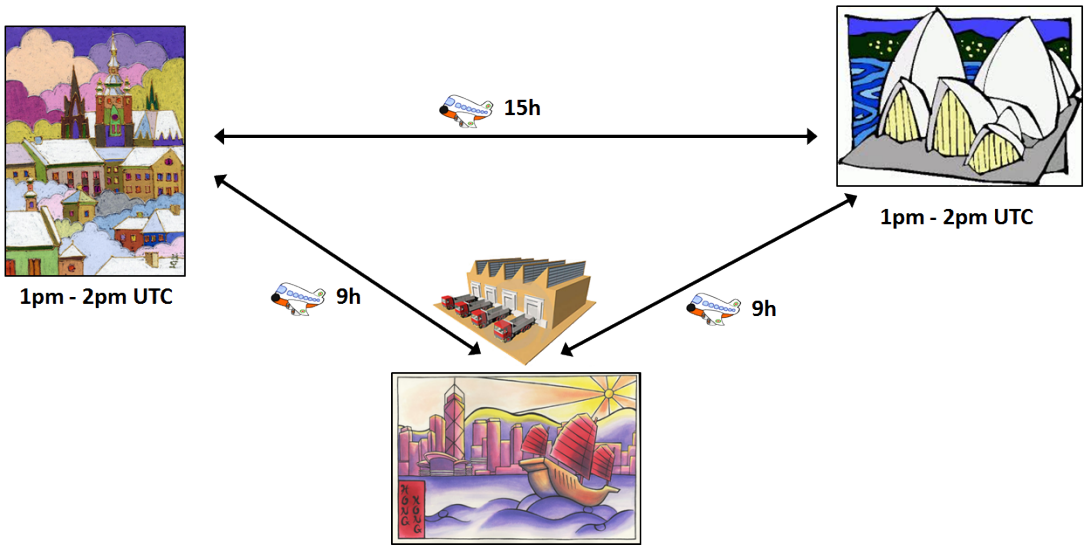


Figure 2.2: Example of temporally incompatible tasks.

is incompatible with j)}. Then identifying the maximal set of mutually incompatible customers corresponds to identifying the size of the maximal clique within the graph I.

Obviously, the evaluation of both of these bounds is *NP-complete* and therefore not suitable to be embedded within an efficient VRPTW algorithm. As discussed later in this work, however, efficient polynomial approximations can be used to compute the bounds with sufficient accuracy.

2.1.3 Benchmarks for the VRPTW

The two most influential benchmarks used widely for evaluating the VRPTW algorithms are the Solomon's benchmark [107] and the Homberger's benchmark [39]. The Solomon's benchmark [107] consists of 56 problem instances with 100 customers each. Two additional sets are provided as well with 50 and 25 customers per instance. These two additional sets are used mostly to evaluate the performance of the exact algorithms that cannot cope with the larger problem sizes and do not typically feature in the evaluation of the more efficient approximate methods. The Homberger's benchmark is an extension of the Solomon's benchmark in which the problems are generated using the same patterns concerning the customer locations, demands and time-windows, however, the problems presented are of considerably bigger size with 5 sets of 60 customers being provided with sizes ranging from 200 to 1000 customers.

Thus both the benchmarks share the core basic characteristics. There are 6 instance types provided — the R1, R2, RC1, RC2, C1, and C2 type. For each instance type there is a slightly different topology with respect to customer placement on the map as well as the time windows properties. The road network is considered being a complete graph between the customer location in an Euclidian space (i.e. respecting the triangle inequality). For C1 and C2 types the customer locations are grouped in clusters. For R1 and R2 classes, on the other hand, the customers are randomly placed. The RC1 and RC2 instance types then combine the previous two types with a mix of both random and clustered customer locations. The C1, R1 and RC1 also differ from C2, R2 and RC2 in terms of the scheduling horizon. The C1, R1 and RC1 instances feature a

shorter horizon resulting in routes of about 10 customers on the average while the C2, R2 and RC2 problems have a longer horizon providing for routes of around 50 customers each.

Together, omitting the 25 and 50 customer instances from the Solomon's benchmark, these sets provide for a total of 356 problem instances. When evaluating the efficiency of a VRPTW algorithm, it is common to provide the *cumulative number of vehicles* (CNV) over these two benchmarks defined as:

$$CNV(\Sigma) = \sum_{\sigma \in \Sigma} vn(\sigma) \tag{2.1}$$

Where Σ is the set of problem solutions over the corresponding set of problem instances and $vn(\sigma)$ is the number of vehicles within a particular problem solution. The reference results used within the experimental evaluation of this work correspond to one of the state-of-the-art algorithms presented in [76] with a CNV of 405 over the Solomon's 100 customer benchmark problems and 10290 over the whole Homberger's benchmark.

2.2 Algorithms for the VRPTW

The VRPTW is one of the classical problem in the domains of operations research and logistics occurring in many distribution systems. It has been extensively studied for both heuristic and exact optimization approaches for almost thirty years with the classical Solomon's [107] article dating back to 1987. Due to the high complexity level of the VRPTW and its wide applicability to real-world scenarios the emphasis has been gradually shifting from the exact methods towards approximate techniques capable of producing high-quality solutions in limited time. In this section we review some of the historical as well as the state-of-the-art algorithms in order to illustrate the wide range of available solving approaches and their relation towards the current study.

2.2.1 Exact Algorithms

Solving *NP-hard* optimization problems to optimality is a fundamental topic in general optimization. Arguably the two most influential methods in this respect are the constraint optimization programming (COP) and mathematical programming (MP). Advances in these methods allow for addressing problem instances of considerable size for some specific problems. The vehicle routing problem variants, however, have proved to be one of the more difficult problems to be solved optimally and therefore only moderately sized problem instances can be solved to optimality consistently. In this section we provide a brief overview of some of the exact methods and discuss the relevant results.

The VRPTW and routing problems in general admit several alternative mathematical formulations. In a departure from the model introduced previously, the definition typically considered with exact VRPTW methods aims at minimizing the *cost* of the solution (i.e. the cumulative travel duration of all routes combined) rather than the fleet size, assuming the fleet size is given for the problem. On the other hand, as outlined later in this work as well, such a model is not too restrictive as the fleet minimization objective can be indirectly addressed by solving the above problem for increasingly big fleet sizes until the first feasible solution is found.

Let Σ^k be the space of all feasible complete solutions with k routes. The problem is then defined as minimizing the cost of the solution

$$min(cost(\sigma)) \tag{2.2}$$

subject to

$$\sigma \in \Sigma_k. \tag{2.3}$$

The set Σ_k can be characterized in a number of ways. The *set-partitioning* model considers the space of all possible feasible routes R and binary variables $x_r, r \in R$ specifying whether or not a particular route features in the resulting optimal solution. This approach to formalizing the VRPTW is used for example in [96]. An alternative *multi-commodity network flow* [20] formulation outlined for example in [37] uses binary variables $x^l_{i,j}, i, j = 0 \dots N, l = 1 \dots k$ being equal to 1 whenever the pair of customers (i, j) is visited by the route l in immediate succession (and 0 otherwise). The temporal aspects of the solution are characterized by additional real variables s^l_i determining the start of service at customer i on the route l. Corresponding constraints are introduced addressing the feasibility of the solution. In this model, each vehicle is modeled as a separate commodity. A modified *two-index* flow formulation omitting the index l in the variable declarations is possible as well.

The approaches used to solve the VRPTW to optimality are mostly based on a branch-and-bound approach [64]. The upper and lower bounds are typically computed using linear

programming techniques e.g. lagrangian relaxation, column generation based decompositions, cutting planes and valid inequalities based on the above formal models. As mentioned, the problem proves to be difficult to solve using these exact methods. This is in particular due to the fact that (i) the bounds provided by the known relaxations are not very strong and (ii) the corresponding linear programs are difficult to solve. An in-depth analysis of the structure of the problem and the available known formalizations and solving techniques can be found in [52].

Various algorithms based on the above concepts have been presented. In [17] an algorithm based on column generation approach using the three-index commodity flow model is presented with the subproblem used for evaluating the columns to be added within the master problem being based on shortest path problem with resource constraints (SPPRC) computation being solved by a pseudo-polynomial labeling algorithm introduced previously in [25]. A branch and price [5] scheme is introduced with the branching being based on presence or absence of a specific arc within the resulting solution. The algorithm was assessed using the Solomon's benchmark problems [107] with the optimal solutions being consistently found in instances of up to 50 customers. Very similar scheme is used in [42], however, with a slightly different variant of the underlying subproblem is used — the *SPPRC-k-cyc* in which cycles of length lower than the value of a parameter k are prohibited. Another variation is presented by [95], where the SPPRC is solved using a constraint programming based algorithm.

In [58] a strong valid inequality is presented denoted the 2-path cut, which is shown to produce a better lower bound for the VRPTW as well as the corresponding separation algorithm. The same branch-and-price scheme is used as above with the SPPRC being used as the underlying subproblem in column generation. Similarly as above, the algorithm consistently copes with problem sizes of up to 50 customers, however, it admits optimal solutions to several previously unsolved problems with up to 150 customers as well.

The algorithm presented in [53] is based on the Lagrangian relaxation of the three-index flow problem formulation. A cutting-plane algorithm [55] based on some strong valid inequalities for the relaxed problem is introduced and embedded within a branch-and-bound framework, resulting in a technique denoted as Lagrangian branch-and-cut-and-price. The algorithm copes with the similar problem sizes as the previously mentioned algorithms and proves to provide for improved computation times for some of the known benchmark problems from the Solomon's benchmark.

As already mentioned, a comprehensive survey of both the formal models for solving the VRPTW optimally as well as the review of relevant relevant literature is available in [52] complementing the brief analysis presented above.

2.2.2 Approximate Algorithms

As already mentioned, due to the high complexity of the problem as well as the complexity of the problem instances solved in the industry (of up to thousands of customers [14]) the main focus within the VRPTW research was in developing algorithms providing good quality solutions within reasonable time frame i.e. various approximate heuristic and metaheuristic methods. In this section we survey some of the known techniques underpinning these methods and survey the contemporary state-of-the-art algorithms in order to provide future reference for the results presented as part of this thesis.

The state-of-the-art VRPTW approximate algorithms are usually quite complex, combining several known techniques within a single algorithmic wrapper. To illustrate this fact, consider the algorithm presented in [76]. Initially a baseline solution with each customer served by a separate route is created. Follows an attempt to remove some of the routes and allocate the corresponding customers to the remaining routes. Firstly, the currently processed customer c is

allocated within the solution albeit at a price of violating some of the constraints. In case the solution thus becomes unfeasible, a specific local search is executed in which the neighborhood of σ is traversed in order to recover its feasibility. Within the search a specific parameter affecting the measure of unfeasibility of the intermediate solutions within this search is gradually tightened until a feasible solution is found or the process fails. In the latter case, the customer c is allocated to specific using a specific *ejection-pools* approach i.e. at the price of ejecting some other customers from σ. Once the route minimization process finishes i.e. no more routes can be eliminated, a genetic algorithm is invoked in order to minimize the travel time within σ.

Within the next section we'll outline some of the known algorithmic concepts for the VRPTW. These include some trivial insertion heuristics, the neighborhood generation operators used by the local search methods as well as some of the successful metaheuristics.

2.2.2.1 Construction Heuristics

A number of heuristics have been proposed in the relevant literature for the VRPTW. In [106] the authors propose a so-called *giant-tour* heuristic. Initially a one giant route containing all the customers is created, not taking into account the time window and capacity constraints. Subsequently this route is divided into a number of smaller routes until a feasible (in terms of the constraints) solution is obtained.

Within the classical Solomon's study [107] several *insertion* heuristics were presented. The most successful of these heuristics denoted as the *I1* heuristic is based on weighted combination of travel time increase implied by the insertion of a new customer and the *push forward* — the delay in service start at the customer visited immediately after the newly inserted customer. Importantly, the Solomon's experiments proved that heuristics addressing both the temporal as well as the spatial aspects of the problem prove to be more successful than heuristics addressing only the shortest-path subproblem.

In the Solomon's original work, the I1 heuristic is used in a combination with a function enabling the determination of the customer which is to be allocated next out of all the yet unallocated customers. In [89] an alternative customer selection criterion is used. The customer to be allocated is thus selected based on a *regret* measure — the cheapest and the second cheapest insertion points within the solution are determined and the customer presenting the biggest gap between the two corresponding costs is selected to be inserted next. A parallel implementation of this process is then presented by [30].

An improved sequential insertion scheme is presented by [41]. A new criteria for customer selection and insertion is presented motivated by the minimization function of the greedy look-ahead solution approach of [3]. The core idea is that the customer c selected for insertion into a route should minimize the impact of the insertion on the customers already within σ and the time window of the customer himself.

2.2.2.2 Local Search Methods and Neighborhood Generation

The construction methods are useful in creating solutions that can be used as initial starting points for the more elaborate solution improvement methods. Alternatively, the solution creative steps can be interleaved with the solution improving steps as well. Within the corresponding local search process, when traversing the neighborhood of the intermediate solution σ a number of alternative *neighborhood generation operators* can be used.

The individual operators (*2-opt* [70], *Or-opt* [82], *2-opt** [90], *relocate* [98], *exchange* [98], λ-*exchange* [83], *GENI-exchange* [35] and cyclic-transfer [113]) are outlined by figures 2.3 – 2.10. The round dots represent the customers while the boxes correspond to the depot. We credit [13] for the illustrations.

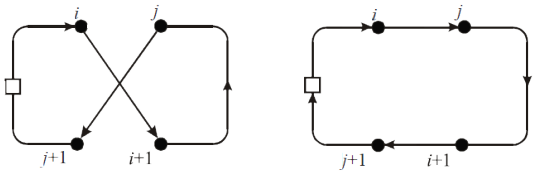


Figure 2.3: *2-opt* exchange operator. The edges $(i, i+1)$ and $(j, j+1)$ are replaced by edges (i, j) and $(i+1, j+1)$, reversing in effect the direction of customers between $i+1$ and j.

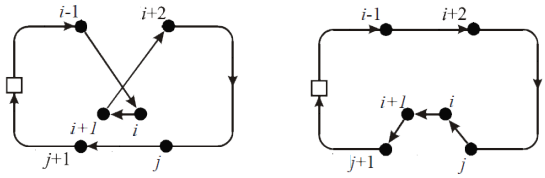


Figure 2.4: The *Or-opt* operator. Customers i and $i+1$ are relocated to be served between two customers j and $j+1$ instead of customers $i-1$ and $i+2$. This is performed by replacing 3 edges $(i-1, i)$, $(i+1, i+2)$ and $(j, j+1)$ by the edges $(i-1, i+2)$, (j, i) and $(i+1, j+1)$, preserving the orientation of the route.

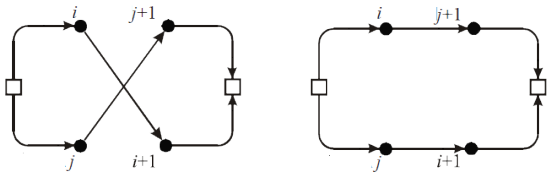


Figure 2.5: The *2-opt** operator. The customers served after customer i on the upper route are reinserted to be served after customer j on the lower route and customers after j on the lower route are moved to be served on the upper route after customer i. This is performed by replacing the edges $(i, i+1)$ and $(j, j+1)$ with edges $(i, j+1)$ and $(j, i+1)$.

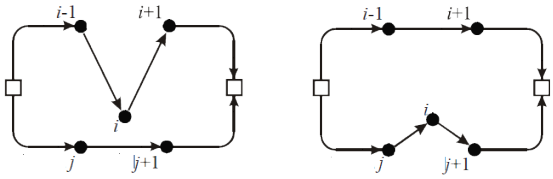


Figure 2.6: The relocate operator. The edges $(i-1, i)$, $(i, i+1)$ and $(j, j+1)$ are replaced by $(i-1, i+1)$, (j, i) and $(i, j+1)$ i.e. the customer i from the original route is placed into the destination route.

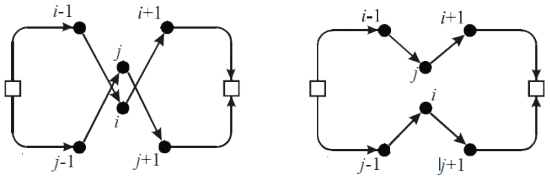


Figure 2.7: The exchange operator. The edges $(i-1,i)$, $(i,i+1)$, $(j-1,j)$ and $(j,j+1)$ are replaced by $(i-1,j)$, $(j,i+1)$, $(j-1,i)$ and $(i,j+1)$ i.e. two customers from different routes are exchanged between the two routes.

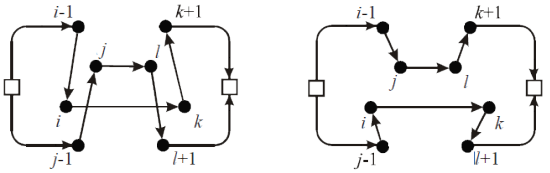


Figure 2.8: The λ-*exchange* operator. The operator is instantiated given two bounds (i,j). Given an intermediate solution $\sigma = (r_1 \dots r_k)$ the λ neighborhood consists of solutions that can be obtained by (i) selecting a pair of routes (r_p, r_q) then (ii) selecting a segment of i customers from r_p and j customers from r_q and (iii) exchanging them between the two routes, preserving the orientation of the routes. In the figure a $\lambda(k-i, l-j)$ is depicted. Segments (i,k) on the upper route and (j,l) on the lower route are simultaneously reinserted into the lower and upper routes, respectively. This is performed by replacing edges $(i-1,i)$, $(k,k+1)$, $(j-1,j)$ and $(l,l+1)$ by edges $(i-1,j)$, $(l,k+1)$, $(j-1,i)$ and $(k,l+1)$, preserving the orientation of the edges.

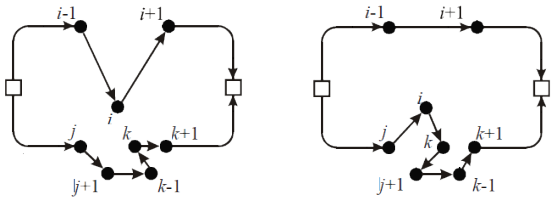


Figure 2.9: The GENI-exchange operator. Customer i on the upper route is inserted into the lower route between the customers j and k closest to it by adding the edges (j,i) and (i,k). Since j and k are not consecutive, the lower route has to be reordered. Here the feasible tour is obtained by deleting edges $(j,j+1)$ and $(k-1,k)$ and by relocating the path $(j+1 \dots k-1)$.

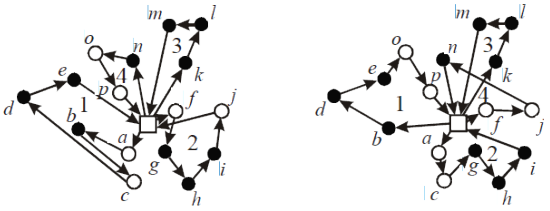


Figure 2.10: The cyclic transfer operator. The basic idea is to transfer simultaneously the customers denoted by white circles in cyclical manner between the routes. More precisely here customers a and c in route 1, f and j in route 2 and o and p in route 4 are simultaneously transferred to routes 2, 4, and 1 respectively, while route 3 remains untouched.

Following the well known local-search strategies, these operators can be used in a *first-fit* or *best-fit* approach (either the first solution presenting an improvement over the original solution is adopted or the whole neighborhood is traversed first and the best overall solution is adopted). In [90] several alternative operators are compared and a hybrid strategy alternating between these in case of local optimum being found is introduced as well, proving particularly successful. Moreover, as outlined within the next section, most of the metaheuristics use the local search neighborhood generating operators for traversing the search space as well in some phases of the solving process.

Another important factor affecting the success of a particular local search method is whether or not this method is able to traverse also the infeasible solution space i.e. whether or not intermediate infeasible solutions are considered. For the tightly constrained problems with short time-windows, long scheduling horizons or with a non-trivial bin-packing subproblem the number of possible infeasible solutions is considerably bigger than the number of the feasible ones and therefore including them to the search potentially adds to the complexity of the resulting method. On the other hand by doing so this increases significantly the chance of the algorithm avoiding being trapped in a local extreme. This, in particular, will be discussed in detail later within the work when the backtracking mechanism embedded within the presented multi-agent algorithm is discussed.

Multiple ways of introducing infeasible search space search were considered in literature. In [69] a heuristic is presented considering also the measure of infeasibility within the resulting solution as an additional parameter when evaluating the solutions within the local search. The parameter affecting the maximal allowed infeasibility measure is then adapted throughout the solving process in order to guide the search towards feasible solutions. The approach is extended by the genetic algorithm presented in [76] where similar measure is embedded within the *maturing* phase of the individuals representing the alternative intermediate solutions.

An alternative way of enabling temporal infeasibility is to allow for the solutions to be temporally incomplete i.e. for some of the customers to be temporally ejected from the solution. This enables some advantageous local moves to be performed while maintaining the feasibility of the solution at the expense of its incompleteness. Such an approach is used for example by [77] where the customers are allocated to the intermediate solution albeit at the expense of ejecting some other customers from the solution, which re-enter the allocation process. In [4] similar appraoch is used, however the individual local moves and the resulting ejections are captured within a graph. This graph is then analyzed using a variant of a maximal-pairing algorithm in order to identify a fitting subset of these moves transforming the solution (through eventual temporal infeasibility) towards a new improved state.

The above mentioned neighborhoods are of relatively small sizes. Therefore, in a best-first

approach, the whole neighborhood can be generated and traversed. However, neighborhoods of considerably bigger size (i.e. exponential in the size of the problem instance) have been considered as well providing for some successful VRPTW algorithms. These neighborhoods are typically denoted as *large* neighborhoods and the corresponding local search methods denoted as *Large Neighborhood Search* (LNS) methods.

The concept was introduced by [97] or later by [101]. In this work the neighborhood was generated by removing and reinserting some of the customers to the intermediate solution σ. The set of customers to be removed is chosen based on the spatial closeness of these customers or their presence in the same route, with the underlying hypothesis being that by removing and reinserting *related* customers the chance of improving the solution increases. The subsequent reinsertion process explores exhaustively all the possible feasible insertions for all the removed customers. The underlying search process is based on constraint programming with specific propagation rules being introduced concerning the temporal and capacity-based feasibility of the affected routes and bounding on the solution cost of the best-so-far solution. Similar concept is used by [100], denoted as the *Ruin and Recreate* approach emphasizing the larger scale of the modifications to the emergent solution being made within a single algorithm's iteration, or by [74]. In [91] a similar approach is used. In this instance, for example, four alternative criterions for selecting the customers to remove and reinsert within the emerging solution are introduced in order to diversify the search that are being alternated throughout the solving process. The reinsertion procedure is then based on a branch-and-price linear programming approach being a modification of the exact algorithm presented in [24].

2.2.2.3 Metaheuristics and Hybrid Algorithms

In this section we arrive to the state-of-the-art algorithms that contribute to the best known solutions for the benchmarks used also to evaluate the algorithms presented in this thesis. These algorithms typically use sophisticated means of searching the search space, integrating one or more of the above local search methods within a higher level solving approach. We refer to these algorithms as metaheuristic or hybrid algorithms.

A simple tabu search based such extension is presented in [32]. An initial solution is obtained using the Solomon's I1 heuristic. Follows a local-search based improvement phase based on the alternation of the *2-opt** and the *Or-opt* operators while a tabu list is used to prevent visiting already visited solutions.

An improved tabu based algorithm is presented by [110]. A specific diversification approach denoted as the *adaptive memory* is introduced. A pool of routes taken from the best solutions visited during the search is maintained and used to initiate new starting solutions. The underlying tabu search process is thus restarted several times using initial solutions constructed randomly by reusing the solution fragments from the adaptive memory. A diversification strategy based on penalizing the frequent customer exchanges is introduced as well. A specific post-optimization method is introduced based on an adaptation of the GENIUS heuristic [34] applied to each individual route of the final solution. The tabu search metaheuristic is used also in [31].

Another particularly successful family of metaheuristic algorithms are the genetic or evolutionary algorithms. The algorithm presented by [112] represents the first effort in this respect. The individuals within the genetic evolutionary process correspond to a specific clustering of the set of routed customers. The traditional bit-string encoding is used for encoding the individuals. The fitness of the individuals is determined by constructing the routes within each cluster using a local-search based approach using the λ-*interchange* operator. Unfeasible solutions are also considered at a specific penalty being applied for the fitness value.

An improved genetic algorithm was presented by [39] based on *evolution strategies* — a

genetic inspired optimization technique introduced by [92] applied to optimization of real-value engineering design problems. The evolution strategies are based on adding a specific vector to the encoded individual consisting of *strategy parameters* affecting the neighborhood generation mechanism and the overall strategy for the next algorithm's iteration. These parameters are also evolved throughout the genetic process in an effort to provide additional means of search diversification and intensification. The individuals are encoded as integer strings corresponding to sequences of individual customers with individual routes being separated by a delimiter. The crossover and mutation operators are being applied directly to thus encoded solutions. The used crossover operator is based on the traditional order-based operators based on a precedence relationship among the individual genes. The used mutation and maturing operators are based on the *Or-opt*, *2-opt*, *λ-interchange* and a special route eliminating *Or-opt* operators.

Similar concepts are used by the algorithm introduced by [73]. However a different initialization procedure is used with all customers being initially served by separate routes. A set of six initial solutions is created based on reinsertions of individual customers using varying insertion criteria. The best thus obtained solution is selected as the starting point. The modified mutation operator additionally uses the *GENIUS* heuristic of [34] in combination with the so called multi-parametric mutation consisting of randomly removing and reinsertion a set of customers from the solution based on different criteria. Another example of successful application of the approach is the algorithm presented in [74]. The algorithm uses a development of a similar evolutionary approach denoted *Active Guided Evolutionary Strategies* (AGES). The AGES alternates between two phases. The first phase employs the Guided Local Search (GLS) [123] approach in an effort to guide the local search process traversing neighborhoods generated using some the mentioned operators to eliminate features of the emerging solution unlikely to feature within good quality solutions i.e. very long arcs. The second phase employs a ES based algorithm with the new individuals being generated using a mutation operator based on an adaptive LNS approach.

One of the most successful algorithms to date presented in [76] also uses a genetic approach. Initially a trivial solution with each customer being served by a separate vehicle is instantiated. Follows an ejection-pools based route minimization procedure employing also a specific feasible and unfeasible local search phases (as was outlined above when discussing the unfeasible local search neighborhoods). Follows a genetic algorithm used to minimize the traveled distance. The algorithm uses a specific crossover operator denoted *Edge Assembly Crossover* operator introduced previously in [75] and a fitness function incorporating elements to the fitness value corresponding to the infeasibility of the solution.

The simulated annealing (SA) metaheuristic was also successfully employed within the context of the VRPTW e.g. by [18], [68] or [6]. The last mentioned algorithm uses a hybrid approach combining the simulated annealing based route minimization phase with a LNS based final phase aimed at minimizing the total traveled distance within the solution. The underlying local search procedures are similar to those described above based on the known neighborhood generation operators. A specific evaluation criterium is used within the route minimization phase, based on the route size, minimal delay and the sum of squares of route sizes. The last factor is used to favor inserting customers from short to larger routes. The second phase consists of removing and reinserting of up to 30% of the routed customers in a LNS procedure.

A number of other VRPTW implementations was presented throughout the three decades the problem has been studied. The field is extremely competitive. The existence of the acknowledged benchmark problem sets over which the performance of the corresponding algorithms can be assessed further promotes the competition. As an improvement in even a single problem instance of these sets is considered a valuable contribution to the field, there has been many increments to the already significantly complex solving methods being published resulting — as outlined above — in an array of very efficient but increasingly complex algorithms. In overall, the contemporary

approximate algorithms are able to solve majority of problems within a 1% deviation from the optimal solution whenever such a solution is known, however with a vastly superior convergence. Such a difference, especially given the significantly better response times, is not too significant from the point of view of the industry. Due to these reasons, as already mentioned in the beginning of this section, the focus has been gradually shifting from the exact mathematical methods towards the efficient approximate methods.

A very interesting extension of the known approaches is the inclusion of stochastic optimization methods and machine learning strategies within the optimization process. The experience with industrial applications of the routing algorithms shows that the real-world problem instances tend to feature specific regularities (both in the spatial as well as the temporal domain) as well as uncertainties [37, 56, 108] which can be efficiently exploited using these techniques. A somewhat similar approach is adopted by one of the incremental versions of the presented algorithms, as will be outlined later.

Another aspect of the algorithms relevant for the industry is the extensibility and flexibility of the algorithms in terms of (i) adopting additional problem constraints and variants and (ii) the execution stability when deployed in uncertain and dynamic environments. In [56] a specific approach is introduced with the separation of the core routing algorithm relying on a heuristic algorithm from the vehicle specific constraints that are embedded within the solver using specific constraints and corresponding propagators. Such an approach enables for the resulting system to be flexible in terms of solving the real world problems where the correct handling of the side constraints is essential, narrowing efficiently the gap between the theory and application and thus increasing the potential impact of such a technology on the industry.

In this respect, the agent based approaches promise to provide for a very interesting alternative to the established methods. As will be outlined in greater detail later, the agent-based framework presented in this work features a clear separation between the vehicles' local planning (encapsulating *all the problem specific logic and constraints*) and the global optimization core based on agent negotiation over scaleless cost estimates provided by the vehicle agents, which thus remains completely abstract and problem independent. This enables a very efficient handling of the heterogeneities and additional constraints within the resulting system. Moreover, due to the autonomic nature of such a system, autonomic behavioral patterns can be encoded within the used agent local behavior that can be used to capture socially advantageous agent's reactions to the unexpected events observed within the environment. In the context of i.e. strategic mission planning in uncertain environments [62, 122] such a possibility provides for an intriguing extension potentially boosting the task execution stability of the overall routing system.

2.2.3 Agent-based Routing Algorithms

As mentioned at the end of the previous section, in context of the contemporary routing algorithms, the agent-based methods represent an intriguing alternative especially due to the inherent flexibility of the typical multi-agent problem decomposition in terms of encapsulating additional side constraints as well as the opportunity to exploit the autonomic nature of the solving process in order to improve the execution stability of the solution. An obstacle which, in our opinion, prevents a wider adoption of the multi-agent paradigm in the transportation routing area is the lack of relevant comparison of the agent-based algorithms to the state-of-the-art algorithms as well as the significantly inferior quality reported by the previous multi-agent studies whenever such a comparison was presented. We will outline these deficiencies later within this section when reviewing the previous significant contributions in this area. This work thus represents an effort to address these problems by: (i) providing the relevant comparison using the established benchmarks and (ii) improving the overall efficiency of the algorithm to within a 1% overall

relative error gap in terms of the cumulative number of vehicles (CNV) between the presented algorithm and the state-of-the-art algorithms over these benchmarks. Both of these targets have been achieved, representing the core contribution of this thesis.

A number of algorithms have been suggested for solving routing problems in general by means of negotiation-based decentralized planning. An interesting survey of the relevant works is provided for example by [22]. Also, as discussed in the previous chapter, many of these works extend the agent-based methods for general task allocation problems which have been well studied in the past [121, 99, 102, 87]. In this section we review some of the significant contributions in the field.

In [121] a general agent-based task allocation framework is presented. The framework essentially uses an agent architecture with a central coordinating agent and a set of agents representing individual resources. Each of the resources is equipped with a particular local planning strategy enabling it to (i) estimate the costs of accommodating a specific task, (ii) estimate the gain inherent to dropping a task and (iii) identify the most costly tasks within its respective plan. The allocation process then consists of a series of auction-like negotiation interactions between the coordinating agent and the individual resources based on these cost estimates. The framework was assessed in multiple applications ranging from strategic mission planning and execution, order based manufacturing scheduling to application to the basic VRP. Most importantly, this work also provides an insight into the advantages of adopting an agent-based approach to task allocation in terms of the improved execution stability in uncertain environments, including relevant experimental assessment. Such a topic is relevant also in scope of this thesis. However, it has not yet been addressed as part of the presented research and remains therefore in the future work department.

The VRPTW algorithm presented in [28] is built around the concepts of Shipping Company Agents (SCAs) representing individual shipping companies and their respective fleets of Truck Agents (TAs). After registering a customer, the SCA negotiates with his fleet of TAs to estimate the price of serving the customer. The other SCAs are contacted as well and based on the corresponding cost estimates the SCA may assign the customer to one of its trucks or decide to cooperate with another company. The planning within each fleet is done dynamically and is based on the well known contract net protocol (CNP) [23] accompanied by a *simulated trading* improvement strategy based on finding the optimal customer exchanges by solving a modification of the maximal pairing problem on a graph representing the proposed exchanges originally presented in [4]. Both cooperative and competitive models are explored with respect to the cooperation of individual SCAs. Also a specific model for simulating traffic jams is presented. The algorithm achieves a CNV of 173 over the 12 Solomon's R1 problem instances — corresponding to a 21.0% relative error compared to the state-of-the-art algorithm of Nagata [76].

The algorithm for the closely related pickup and delivery problem with time windows presented by [59] is essentially a parallel insertion procedure based on CNP with subsequent improvement phase consisting of relocating some randomly chosen tasks from each route. The customer insertion costs are calculated using the well known Solomon's I1 insertion heuristic [107]. The performance is assessed on an ad-hoc dataset which is not made available to the public.

The algorithm for VRPTW presented by [67] is based on agents representing individual customers, individual routes and a central planner agent. The featured problem decomposition is unique in considering not only the agents representing individual routes, but also agents representing individual customers. Initial solution is created using a sequential insertion procedure based on the Solomon's I1 heuristic. Follows an improvement phase in which the vehicle as well as the customer agents propose advantageous moves based on their local processing. These move proposals are gathered in a *move pool*. The improvement phase thus consists of iteratively gath-

ering these proposals and executing the most advantageous ones. The experimental assessment is based on Solomon's 100 customer instances with a CNV of 436 corresponding to a 7.7% relative error compared to Nagata [76].

In [21] the authors propose a VRPTW algorithm based on Order agent — Scheduling agent — Vehicle agent hierarchical architecture. The algorithm is based on a modified CNP insertion procedure limiting the negotiation to agents whose routes are in proximity of the task being allocated in an effort to minimize the number of negotiations. The algorithm thus focuses on optimizing the efficiency of the distributed negotiation process rather than outright performance of the algorithm per-se and thus no relevant performance information is provided.

In general, however, the number of successful published agent-based algorithm for the VRPTW is relatively low. We argue that this is in particular due to the fact that the previous agent-based methods have not proved to be particularly successful. As outlined in the previous sections, the VRPTW is a very complex problem and the state-of-the-art algorithms represent top of the shelf optimization algorithms using the latest developments in a very competitive field — something that cannot be said with respect to the previous VRPTW multi-agent algorithms. The preliminary iterations of the algorithm presented as part of this thesis were presented in [47, 50, 51, 46, 48, 49], each time being benchmarked by the relevant problem sets and each time improving on the previous best known results over these benchmarks achieved by comparable agent-based algorithms. The concepts used by the resulting algorithm and its various configurations corresponding to these versions will be extensively discussed later in this work.

Chapter 3

Optimization in Multi-agent Systems

This work is concerned with reformulating a specific optimization problem as a social welfare maximization problem within a society of agents. The implications and opportunities and relevant refinements of such an approach are explored and assessed. In this chapter we provide an elementary introduction to the relevant topics from the fields of social choice theory and multi-agent systems in an effort to provide background for understanding the specific positioning of this work. An introductory word-cloud is depicted in Figure 3.1.

Figure 3.1: Multi-agent optimization key words cloud [127]

3.1 Social Choice and Welfare

The theory of social choice is concerned with the evaluation of alternative methods of collective decision-making, as well as with the logical foundations of welfare economics [2]. In this theory, the social welfare captures the relative desirability of alternative social states with respect to the preferences of the participating individuals. In general, the determination of a sound way of representing the preferences of the members of the society as well as a mechanism in which these should be aggregated in order to evaluate the alternative social states is far from trivial. In particular, it is difficult to quantify these preferences as well as provide a calculus in which conclusions can be drawn upon manipulating these e.g. compare them or merge alternative opinions of multiple individuals into a collective one. An example of a widely used computationally sound such approach to this dilemma is briefly discussed in Example 3.1.1.

Example 3.1.1. A widely used approach to comparing social states in terms of social welfare is the concept of pareto principle introduced in [84]. According this principle the change from one social state to another social state can be judged as socially advantageous if at least one individual benefits from the change while all the remaining individuals are indifferent to the change. Obviously, the applicability of the principle is quite limited as it only provides a comparison between specific social states with all but one individual being indifferent. In general, as with most economic policies, there is a tradeoff between favoring some individuals on the expense of disfavoring some other ones. Such a tradeoff cannot be efficiently expressed using the pareto principle. In such a case, the aggregation (i.e. substraction) of the gains and loses in subjective preferences of the individuals would have to be performed.

The problem of aggregation of social preferences and determination of the most suitable social states can also be viewed as a specific optimization problem. The traversed state-space corresponds to the space of alternative social states and the optimization objective is the desired feature of the sought social state e.g. social welfare, envy-freeness, equilibrium, pareto optimality etc. Thus formulated, the solving methods used to solve these problems range from the methods based in social choice theory e.g. voting, auction or negotiation mechanisms to traditional optimization techniques e.g. constraint and linear programming, sat solving, heuristics etc. Obviously, using a complementary approach, specific optimization problems can be reformulated as social choice or welfare maximization problems. For specific problems e.g. problems where multiple actors contribute to the overall joint solution of the problem potentially featuring game theoretic aspects, constraints and/or uncertainty concerning the local information available to the individual actors such an approach can represent an interesting alternative to the traditional optimization methods. On the other hand, specific means of solving the thus formulated optimization problem are required. An elementary introduction to multi-agent optimization is presented within the next section.

3.2 Optimization in Multi-agent Systems

Multi-agent systems are concerned with modeling societies in which the individuals correspond to autonomously acting entities. Thus in addition to capturing the social preferences and social states within the society, the multi-agent systems aim at providing a model for representing, simulating and controlling the behavior of the society over time. As such, they are imminently concerned with typical problems inherent to social dynamics e.g. local decision making of the individuals, limited availability or uncertainty concerning the overall state of the society or the modeled system and the dynamism of the environment. Apart from addressing the social choice problems inherent to diverse societies, the multi-agent systems are further concerned with addressing the planning aspects inherent to reaching and maintaining the desired social states over time.

As outlined in the previous section, by finding a parallel between the desired social states and relevant optimization criteria the multi-agent systems provide an interesting alternative for modeling and solving various optimization problems. In the center of interest of multi-agent optimization is thus (i) the finding of efficient solutions to problems where multiple individuals with potentially diverse interests and local knowledge contribute to reaching some overall state of the modeled system and (ii) efficiently addressing the dynamics inherent to such systems e.g. developing efficient strategies enabling the society to efficiently reach the desired social state and/or to react to the dynamic changes of the environment.

The multi-agent optimization is on the forefront of the contemporary optimization research and represents a rapidly developing field. One reason for this is that the optimization models throughout the industry are becoming increasingly complex, requiring multiple autonomous entities being represented as part of the optimization model as well as the dynamic of their interaction with the rest of the environment. Also, the contemporary developments in industry and manufacturing have seen a shift towards smart objects and the internet of things, further supporting the adoption of decentralized optimization techniques [36]. However, this new paradigm also creates the need for relevant representations of these objects and optimization methods to be developed making use of the novel paradigm, further supporting the relevance of multi-agent optimization research field.

The relevant multi-agent extensions exist for many well known optimization problems. As an illustration, consider Example 3.2.1 presenting a simple multi-agent scheduling game.

Example 3.2.1. Consider a scheduling problem known as the N machine scheduling problem with precedence constraints. The problem consists of scheduling a set of tasks on N identical machines in such a way as to minimize the overall completion time (a number of other criteria can be considered as well). Consider moreover that some tasks cannot be scheduled until some other tasks are completed. Now consider a multi-agent extension of this problem with several self-interested agents contributing to the overall schedule, each owning a subset of the overall set of tasks to be scheduled.

Similar problems have been studied for example as part of the computational disaster management [38] when multiple parties contribute to the restoration of shared interdependent infrastructures and have proved to have a significant impact on mitigating the impact of natural disasters on human society. The game theoretic properties of such a system immediately emerge. Is there a common schedule such that neither of the agents has an incentive to reschedule its tasks? Is there a schedule in which the social welfare is maximized (i.e. the makespan of the schedule — the completion time of the last scheduled task — is minimized)? Is there a schedule that is fair e.g. none of the agents is favored at the expense of others? Can a set of agents form a coalition in order to improve their utilities on the expense of the remaining agents? Is there

a mechanism potentially involving payoffs between the agents that could motivate the agents to collaborate in order to collectively arrive to a globally satisfactory or optimal solution?

The above example illustrates the alternative desired social states that can be considered and the tradeoff between the social welfare and the selfish interests of the participating parties. In [63] two interesting properties characterizing the way the selfish behavior affects the overall system efficiency in terms of social welfare are introduced. The *price of anarchy* (PoA) corresponds to the ratio between the social welfare of the worst nash equilibrium and the globally optimal solution. The *price of stability* is than the same ratio, however computed for the best welfare equilibrium. In case of the above scheduling problem, for example, the PoA is unbounded in general, meaning that using selfish strategies the players may reach a state where neither of the players has an incentive to reschedule any of its tasks yet the resulting overall schedule is seriously ineffective. In order to address this, the game theoretic properties of the system can be modified by means of social engineering e.g. by introducing a relevant *mechanism* [103]. In most general terms, a mechanism introduces payoffs between the players compensating them for choosing cooperative strategies instead of selfish ones. We refer the reader to [103] for further details. The above example serves to highlight the aspects typically associated with multi-agent optimization. Within this work we present a multi-agent optimization system that is based on a society of fully collaborative agents and therefore some of these aspects are not addressed as part of this work.

A number of formalisms and solving methods have been proposed to model multi-agent optimization problems, to solve these problems and to implement the autonomic societies of agents underlying the solving processes. Core to these formalisms and methods is the separation of (i) the processing inherent to the agents' local problem solving and behavior modeling and (ii) the agent interactions corresponding to the global coordination level of the overall solving process. Below we briefly introduce some of these formalisms.

3.3 Formalizing Multi-agent Problems

The multi-agent optimization problems, systems and social behaviors can be formalized in a number of formalisms stemming from various adjoining research fields. By no means being an extensive list, these include multi-agent extensions of the classical planning formalisms, the constraints programming approaches well known from classical optimization as well as the temporal logic based formalisms inherent to the traditional multi-agent research. While not all of these formalisms are directly relevant to our work, they highlight the fundamental opportunities and challenges relevant to multi-agent optimization.

3.3.1 Classical Planning

The classical planning (denoted usually as STRIPS planning) is based on exploring the state-space generated by applying a set of *actions/operators* starting from initial *states* in order to determine efficient *plans* reaching the set of *goal* states. For a thorough introduction to the classical as well as alternative approaches to automated planning please refer to the excellent book [78].

In planning, alternative approaches to instantiating plans can be considered. The classical representation uses predicate logic. Objects in the environment are substituted by constant symbols, predicates describe the relations among the objects, and a particular state is a set of ground atoms. The possible actions within the environment are represented by a set of operators. The operators are abstract with the preconditions allowing them to be applied as well as the effects within the environment resulting from their application being expressed over a set of variables. By substituting these variables with particular objects within the environment the particular actions are instantiated.

An operator o is thus defined as the triple

$$(\mathsf{name_o}(\mathsf{x_1}, \ldots, \mathsf{x_k}), \mathsf{precond_o}, \mathsf{effects_o}). \tag{3.1}$$

When instantiated, the operator variables $x_1, \ldots, x_k$ are substituted with a set of objects within the environment such, that all of the predicates within the set $\mathsf{precond_o}$ are satisfied for these objects. By applying thus instantiated operator, the corresponding changes within the environment are represented by the set of predicates within $\mathsf{effects_o}$ becoming true.

Apart from the classical representation two other representations are typically considered — the set theoretic representation and the state-variable representation. While equivalent in their ultimate expressive power, these representations differ in what algorithms can be used for solving the correspondingly defined planning problems. The solution of a planning problem is a *plan*, corresponding to an ordered list of actions transforming the environment from the initial state s_0 to one of the goal states $s_g \in S_g$.

The multi-agent problems are inherently distributed. The collective plan for a set of agents doesn't require to be fully ordered as some actions may be executed in parallel ba different agents. Instead, the plan features only a partial ordering with the ordered sequences corresponding to either (i) chaining of the actions within the individual agents' plans or (ii) the mutual precedence coordination requirements on the plans of different agents.

An analysis of extending the classical planning approaches to distributed domains is presented in [11]. In a distributed domain, the actions are partitioned between the agents. The two fundamental approaches to solving such a problem are to (i) compile it to a single-agent classical planning problem or (ii) try to solve it in a two-level process addressing the coordination problem separately from the local planning problems of the individual agents. In case of the first approach, however, the structure originally characterizing the planning domain is lost.

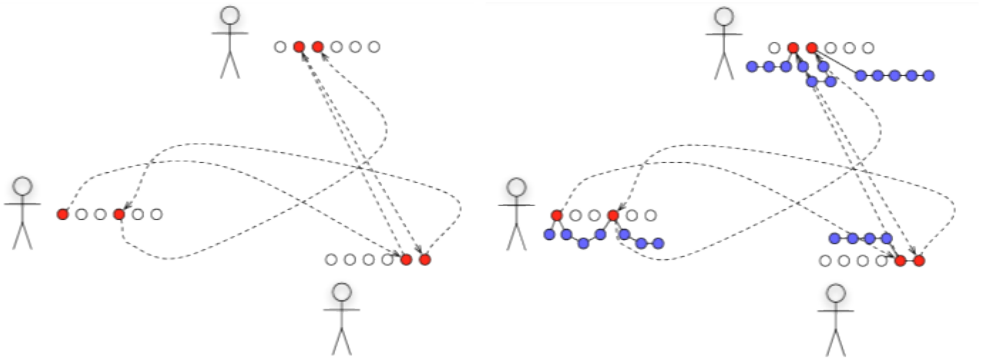


Figure 3.2: Multi-agent domain — non-private coordination and private actions. Credits [11].

The inherent optimization process cannot be efficiently parallelized or distributed amongst the agents and the approach also prohibits development of autonomic agents' strategies addressing e.g. dynamic changes within the environment. On the other hand, by separating the coordination process from the local planning processes the autonomic nature of the problem can be exploited. In the previously mentioned work the actions (operators) within the domain are characterized as either *private* or *non-private*. The non-private agent's actions correspond to the coordination interactions changing the state of the environment as observed by the other agents. The private action, on the other hand, only affect the part of the environment locally observable by the agent. The two-level optimization process then consists of (i) creating a feasible coordination plan (or a set of such plans) by solving the global coordination problem and (ii) for each such solution solving (in parallel) the corresponding set of local planning/optimization problems. We illustrate the decentralized approach in Exmaple 3.3.1.

Example 3.3.1. In a simple logistics domain the non-private actions correspond to loading and unloading of a particular vehicle at a particular depot facility, where other vehicles can load the previously unloaded goods. On the other hand, the actual transportation and distribution of goods to customers assigned to the particular agent can be considered as agent's private actions. The actual assignment of the customers to the vehicles can be static for the domain or can be part of the non-private subproblem as well. The overall planing process then consists of identifying a suitable global plan composed of non-private coordination actions and then having each agent independently try to accomplish the constraints corresponding to the examined global plan by interleaving his coordination actions with arbitrarily long lists of his private actions. The situation is illustrated on Figure 3.2.

The figure depicts three agents. The red circles correspond to the coordination interactions within the individual agents' plans which can for example correspond to loading and unloading goods in specific nodes (depots) within the environment. The dotted arrows denote the precedence constraints between the coordinated actions within the global coordination plan. Such a coordination plan can be obtained as the solution of the relaxed global coordination problem and corresponds to "guessing" the correct coordination structure of the problem. Specific heuristics can be used to identify such a structure correctly [11]. The blue circles then correspond to individual agents' local actions enabling them to satisfy the constraints resulting from their local view of the global coordination plan. The inherent planning/optmization processes can be run in parallel as the soundness of the overall plan is guaranteed by the presence of the coordination constraints.

The above approach illustrates the fundamental extension of classical STRIPS planning to the multi-agent domain. Several implementations extending the well known planning heuristics for the multi-agent domain have been proposed as well. In [11] an approach based modeling the coordination problem as a constraint satisfaction problem while using known planning heuristics for solving the local subproblems is presented. In [109] a multi-agent version of the well known Fast Forward planning heuristic [8] is outlined while [80] discusses the multi-agent version of the planning algorithm based on the known A^* algorithm. An interesting extension to multi-agent planning approaches in dynamic environment is introduced in [60] where a specific multi-agent dynamic plan repair algorithm is introduced.

3.3.2 Constraint Programming

The constraint satisfaction and optimization techniques can be efficiently used to formalize and solve a number of multi-agent optimization problems. In scope of this work, the distributed constraint optimization (DCOP) formalism and algorithms are of specific relevance. The DCOP refer to the situation when a set of agents collaborate in order to optimize their performance over a collective problem, however, specific constraints exist that have to be satisfied as well. The search space of the particular problem being solved is formalized as a set of variables, their corresponding domains and their relations expressed in form of constraints. A constraint represent a relation over two or more variable domains, specifying the value combinations admissible within the feasible solution of the problem. The performance of the system is characterized by an objective expressed as a function of these variables. Thus the DCOP multi-agent optimization problem is defined as finding an assignment for these variables such that it is feasible (all the constraints are satisfied) and the value of the value of the objective function is maximized.

Most importantly, the DCOP refers to such situations, where each of the variables is *owned* by a single agent. The goal is still to find an efficient feasible variable assignment that meets the constraints, but each agent decides on the value of his own variables with relative autonomy and may be also privacy conscious concerning it's local subproblem. While he does not have a global view of the system, as part of the solving process, he is informed about the value choices made by the agents owning variables that are bound to his variables by existing constraints. The solving process therefore requires each agent engage in some protocol that combines local problem solving with communication with the other agents within the society.

An example of a distributed constraint optimization problem is the following. Imagine a group of people want to schedule a set of meetings, however, all of them are busy and have their duties to attend to. Therefore, they have various potentially conflicting views concerning the time the common meeting takes place and potentially are not willing to communicate these preferences explicitly. Imagine further that for each of the common meetings the importance of the presence of each participant is assigned a specific weight. The common objective is thus find a schedule for the meetings such that the combined weight of the attending people is maximized. Thus each of the participants is presented with a local scheduling subproblem corresponding to efficiently schedule the set of meetings within his schedule. On the other hand, the collective choice has to be suitable all the participants.

This is a trivial example, however, it shows that an efficient method for solving such a problem requires the agents to communicate in an efficient manner over the variable choices for the shared constraints i.e. the schedule of the meetings. However fundamentally different, the constraint based formalism again requires efficient means of multi-agent coordination to be used. Also, the global coordination part of the problem as well as the local subproblem solving, while being bound by the common formalism, emerge as the two inherently separate parts of the process.

The global coordination communication semantics has to generally address two aspects of

the overall optimization problem. On one hand, the communication can be used to exploit the relations between the variables given by the constraints to trim the corresponding admissible domains of the variables. On the other hand, along the same lines the bounds for the cost of the overall emerging solution have to be communicated as well in order to possibly exploit the cost structure inherent to the solved problem in order to accelerate the search. In both cases the communication can follow some inherent structures of the problem given e.g. by the constraints and cost relations between the variables owned by the communicating agents. For a comprehensive introduction to DCOP and constraint programming in general refer to [94, 65].

3.3.3 Markov Decision Processes

When modeling social behavior of individuals within the society and dynamic environment, the ability of the individuals to learn naturally comes to mind. One of the widely used formalisms used to model continuous learning is presented by the Markov Decision Processes (MDP) [40]. The context in which the MDPs are defined is somewhat similar to the classical planning context. Just like in planning, the agent's state-space is generated by sequential application of its actions. In case of the classical planning the optimization criterium of the agent is the minimization of the size of the plan enabling the agent to reach a specific goal state. In case of the MDPs, the considered utility structure of the problem is fundamentally different. For each state and a possible action a reward associated with performing the action in this state is defined. Formally this corresponds to a function $r : S \times A \to \mathbb{R}_0^+$. The agent's utility is then accumulated for every performed action. The optimization criterium is therefore to select such a sequence of actions as to accumulate maximal reward. The classical planning problems can be formalized in this model as problems where the reward is 0 for all states and actions but the states and actions that result in immediately reaching the goal state.

Within the thus defined environment, various rewards may result as a result of the agent adopting alternative policies. A policy is a function $\pi : S \to A$ assigning an action $a \in A$ to be performed to each state $s \in S$ and corresponds to a particular behavioral pattern adopted by the agent. Assuming the value of the reward function is known and static for all states, the identification of the optimal policy is quite straightforward. More interesting is the situation when the reward function values are not known to the agent or dynamically change over time e.g. due to the dynamic changes within the environment or as a result of increasing accuracy of their estimates resulting from agent's exploration of the environment. In such a case, the policy of the agent can be updated throughout the agent's exploration of the environment based upon the rewards encountered by the agent when visiting various states.

The policy learning in MDPs has been extensively studied. From the optimization standpoint this corresponds to *training* the agent (or, as outlined below, the whole society of agents) to adopt efficient behavioral patterns within the environment or modify these according to the changes within the environment observed over time. This enables the agent to increase its performance within the environment in the long run. The underlying model can be further enriched by considering the transition function $(s_1, a, s_2), s_1, s_2 \in S, a \in A$ inherent to applying specific actions in specific states is also not fully known to the agent. In such a case instead of considering a deterministic function the probability distributions $T(., a, .)$ over S can be considered instead [10]. In such a case, the learning process can be extended to the state transition distributions as well. In dynamic environments, the learning process can further address only a limited planning horizon or favor assigning greater weight to the more recent information [45].

In a multi-agent environment, the exploration and learning of the environment can be done cooperatively by a society of agents. The relevant model then has to be updated by introducing efficient means of coordination addressing (i) the sharing of information about the environment

and (ii) formalizing the cooperative and conflicting behaviors within the framework. This can be done by extending the MDP considering the states to correspond to the states of the whole multi-agent system and the transitions between these states to be generated by all agents performing an action simultaneously i.e. a single collective joint action being performed. Similar model is presented and analyzed in [9].

3.3.4 BDI Architecture and Temporal Logic

The Belief-Desires-Intentions (BDI) architecture refers to a specific way of formalizing agent societies. Fundamentally, this approach is inspired by logic and psychology, addressing within it's formal framework the mental states of the participating agents — the beliefs, desires, and intentions. The architecture enables for the agents to maintain and periodically update information constituting it's *mental-state* corresponding to (i) the agent's contemporary local knowledge of mental states of the other members of the society and the surrounding environment (beliefs), its long term ultimate goal states (desires) and the immediate states that the agent commits to achieving (intentions) in its course to achieve the desired goal states.

By introducing the abstraction of mental state the immediate planning of the agent is decoupled from it's ultimate desired states. Thus, instead of formulating long-horizon plans aiming at achieving the desired goals, the agent can instead operate in a two-level process by first revising the set of shorter time intentions based upon the updates to its beliefs and then planning in order to achieve these. The mental process corresponding to formulating the intentions based upon the belief database is referred to as deliberation. The deliberation module enables to encode sophisticated higher-level heuristic behavioral patterns within the agent's behavior. On the other hand, the immediate planning horizon corresponds to the agent's intentions and is therefore much smaller than if planning towards the desires. Such an approach is motivated primarily by (i) enabling the agent to efficiently operate in a dynamic environment with limited computational resources available to the agent by (ii) significantly reducing the complexity of agent's immediate planning. Formally, the agents' belief, desire and intention repositories are formalized using the modal logic and its variants. We refer the reader to [128] for the introduction of this formal apparatus.

An intention can be viewed as reflecting an agent's commitment to perform an action. Therefore, the formalizations of agents' commitments has a significant role in the BDI architecture. In general, commitments represent an alternative way of representing agents' intentions and plans. By doing so, specific behavioral patterns can be associated with specific commitments. Most significantly, in case the agent no longer believes a particular commitment is feasible, it can react to such an event in a specific way i.g. by informing the contracting parties, trying to find a suitable delegate agent which can undertake the commitment etc. Viewed from the perspective of classical planning discussed previously, the commitment-based plan representation and corresponding behavioral patterns associated with the commitment life-cycle can be used for embedding autonomic plan-repair strategies within the overall planning process. Similar strategy has been efficiently exploited to provide for an increased execution stability in dynamically changing, uncertain environments [62]. We introduce the commitments formally in the next section.

3.3.4.1 Social Commitments

As mentioned, the social commitments and the behavioral patterns associated with the commitment life-cycle can be used to address the unexpected changes within the environment observed by individual agents leveraging the execution stability of the multi-agent system. In case of such

an event is encountered, the affected agent(s) can trigger the reconfiguration of the system in order to address the event by i.e. re-negotiating some of the existing commitments. In context of multi-agent optimization the social commitments represent an efficient method of addressing dynamic optimization problems, providing means of recovering the intermediate problem solution in reaction to unexpected and potentially locally observable events increasing the robustness of the resulting system.

Within the context of this work the social commitments are of particular relevance. While this study is concerned with the static variant of the VRPTW in an effort to provide a sound comparison with the existing centralized optimization methods the resulting multi-agent model can be easily extended to adopt the corresponding autonomic commitment life-cycle behavioral patterns. In [118] such extensions of a task allocation framework that is similar to the framework presented later within this study are introduced and evaluated against a scenario including a highly dynamic environment and requiring near real-time response to unexpected events. The results indicate that the social commitments provide for an increased execution stability of the resulting system by leveraging its autonomic features.

In classical literature [126] the social commitments are formalized within the scope of the BDI formal model as a tuple

$$(\mathsf{Commit}\ A\ \psi\ \varphi\ \lambda). \tag{3.2}$$

In the definition, the A denotes the committing agent, ψ is the commitment activation condition, φ is the commitment goal and $\lambda = \{(\rho_1, \gamma_1), (\rho_2, \gamma_2), \ldots, (\rho_k, \gamma_k)\}$ is the set of decommitment rules, also referred to as the *convention*. In general, the above definition corresponds to the agent A's intention to contribute to satisfy the goal condition φ given that the agent believes the ψ to hold and no $\gamma_i, i \in 1 \ldots k$ to have yet been made true. In the opposite case i.e. the activation condition ψ doesn't hold or some $\gamma_i, i \in 1 \ldots k$ becomes true the commitment is invalidated. The decommitment rules become activated by some decommitment condition $\rho_i, i \in 1 \ldots k$ becoming true. In such a case, the original commitment is dropped and is replaced by the agent's intention to satisfy the corresponding γ_i.

The definition can be further refined using the BDI temporal logic formalism [104, 126]. The AG operator denotes the temporal inevitability while the $\curvearrowleft$ operator denotes the temporal until. The *Bel* predicate corresponds to the agent's belief in the truthfulness of the corresponding condition and the *Int* predicate to its intention to contribute to satisfying the corresponding condition. The above definition can thus be expressed as:

$$\begin{aligned}
&(\mathsf{Commit}\ A\ \psi\ \varphi\ \lambda) \equiv \\
&\quad ((\mathsf{Bel}\ A\ \psi) \Rightarrow \mathsf{AG}((\mathsf{Int}\ A\ \varphi) \\
&\qquad \wedge(((\mathsf{Bel}\ A\ \rho_1) \Rightarrow \mathsf{AG}((\mathsf{Int}\ A\ \gamma_1))) \curvearrowleft \gamma_1) \\
&\qquad \ldots \\
&\qquad \wedge(((\mathsf{Bel}\ A\ \rho_k) \Rightarrow \mathsf{AG}((\mathsf{Int}\ A\ \gamma_k))) \curvearrowleft \gamma_k) \\
&\quad) \curvearrowleft \bigvee_i \gamma_i).
\end{aligned} \tag{3.3}$$

As mentioned, by representing the agent's obligations as social commitments it is possible to embed autonomic behavioral patterns within the agent's plan which can be activated in case of dynamic changes within the environment. The decommitment rules $(\rho_1, \gamma_1) \ldots (\rho_k, \lambda_k)$ thus represent the agent's reactions to specific events within the environment should these occur. Such a reaction may correspond for example to an effort on the agent's behalf to find another agent or a set of agents to which the commitment can be delegated, the negotiation about relaxation of the commitment conditions or communicating the failure and the reason for the failure to other system components. In this way, the resulting agent-based optimization system can be empowered with potentially complex and powerful recovery strategies triggered autonomously

by the agents based on their observations of the environment increasing the execution stability of the system in uncertain environments.

In [62] and [118] successful applications of this approach to highly dynamic task allocation and planning problems have been showcased as well as some interesting formal extensions of the logical model for commitments useful in such applications. In this work therefore we use the same formalism to denote the allocations of customers to the agents representing the vehicles. As this work is primarily concerned with the static VRPTW problem and the research of alternative agents' local planning strategies and coordination semantics, we do not explore or experimentally assess this aspect of the presented algorithm in great detail. In our opinion, however, this represents an intriguing future research opportunity.

3.4 Multi-agent Task Allocation

The task allocation problems represent a wide family of hugely relevant optimization problems. In fact, many scheduling, transportation as well as planning problems can be formulated as task allocation problems, including the VRPTW and other routing problems. In general a task allocation problem can be defined as follows. Given a set of tasks $T = \{t_1 \dots t_{|T|}\}$ and a set of resources $R = \{r_1 \dots r_{|R|}\}$, find a set of task allocations $A = \{[t_i, r_j], i = 1 \dots |T|, j \in 1 \dots |R|\}$ such that each task is assigned to exactly one of the resources and the cost of the assignment $Cost(A)$ is minimized. Additional constraints may be imposed e.g. constraints concerning the resources' capabilities to accommodate specific tasks, capacity constraints, temporal or precedence constraints concerning the tasks execution etc. The cost function can also have a number of forms. The simplest alternative is a constant function, corresponding to problems where any feasible allocation of the tasks is also the optimal one. A common alternative is the $Cost(A) = \sum_{r \in R} cost_r(T_r)$ form where $cost_r(T_r)$ corresponds to the cost of resource r accommodating all the tasks T_r assigned to it.

The multi-agent approaches to task allocation optimization problems have been extensively studied for example in [121, 99, 102, 87]. The motivation behind this is quite straightforward. On one hand such problems are inherently bound to numerous human activities e.g. transportation, manufacturing or operations planning. On the other hand, as discussed, the inherent autonomic nature of multi-agent systems may be beneficial for specific target environments in which these problems are solved, including dynamic and uncertain environments. The successful applications of multi-agent planning to task allocation problem include for example strategic mission planning [61], frontiers exploration [119] or order based production planning [85].

The task allocation problems can be reformulated as multi-agent optimization problems in a very straightforward manner. The resources being part of the original problem formulation can be represented by a society of agents. The objective function can then be represented as the social welfare within such society. The problem is then solved by combining the agents' local decision making algorithms (e.g. local planning, scheduling and cost processing) with efficient coordination mechanisms enabling the tasks to be efficiently distributed and traded in between the agents. By assuming a fully collaborative society of agents this approach allows to capture the above mentioned traditional scenario. However, as already mentioned, such a model can be also used to capture problems with competitive elements. An example of a competitive N-machine scheduling problem with precedence constraints was already discussed previously in Section 3.2 — a problem that can be easily reformulated as a specific multi-agent task-allocation game.

This work is concerned with presenting a collaborative multi-agent model for the VRPTW. The problem is formulated as a social welfare maximization problem on a society of agents representing the vehicles or resources contributing to the overall problem solution. As such, the agents within the society attempt to efficiently address their local subproblems by (i) efficiently accommodating the tasks assigned to them and (ii) reacting appropriately to events inherent to the dynamism and uncertainty within the environment. The overall problem is than solved by introducing an efficient coordination mechanisms enabling the agents to cooperate on performing socially advantageous moves i.e. trade the customers between themselves in order to improve the intermediate problem solution. Interestingly, as will be discussed in greater extent later within the work, all the VRPTW specific logic is embedded within the agents' local planning strategy. The presented coordination mechanism, on the other hand, remains abstract and applicable to a wide range of general task allocation problems. In this sense, the coordination mechanisms and semantics in multi-agent task allocation are an interesting research domain in its own right.

In [121] an approach is introduced to the abstract coordination in multi-agent task allocation inspired by the market-based contract nets mechanisms and negotiation protocols [23]. In this

approach a contract net is established between the cooperating agents and a single manager agent. The interactions are based on the well known CNP protocol [23]. Alternative task allocation problems are then addressed by developing efficient resource local planning strategies for these problems. By using such fitting local allocation algorithms individual resource agents are able to estimate the (scaleless) costs of accommodating particular tasks. The overall problem is then solved by having the resource agents trade the tasks in between themselves based on these estimates following the above mentioned abstract negotiation-based coordination mechanism. In this approach, the local as well as global constraints are addressed within the agents local processing. As shown, the local processing can further address i.e. the hierarchical structure of the allocated tasks by further decomposing these tasks into subtask and coordinating the allocation of these subtasks. In this sense the agents' society can mirror the symmetries within the problem know to be exploited to a good effect by hierarchical planning approaches e.g. HTN networks [26]. The coordination mechanism then consists of a series of CNP auction steps in which advantageous task allocations to the resources and advantageous delegations of the tasks between the resources are identified and performed in order to construct a good feasible solution to the problem.

A similar approach is used within this work, however, with some key differences. We refer to the structure of the trading based coordination process as a specific *coordination semantic* being used. In case of the coordination semantic used in the previously mentioned work the only admissible moves considered within the coordination process are the feasible task delegations between a pair of resource agents. Similar coordination semantics have been used as well in [28, 59] or [67]. On the other hand, as outlined for example in [4] in some cases the simple delegation trading steps are not sufficient to significantly modify the contemporary solution of the overall optimization problem. In the above study, instead of considering an iterative process based on single task delegations, an optimal set of trading steps of a greater size to be executed simultaneously is evaluated in an approach denoted as *simulated trading*. The agents' reasoning thus goes along the lines:

- Agent A: "I could accommodate the task x very well but then I'd have to drop the tasks y and z"
- Agent B: "I could accommodate the task z very well but then I'd have to drop the task r"

...

and a possible networks of these operations are evaluated using a centrally executed specific graph algorithm. Similar to the presented study, this study targets the VRPTW. The results prove that for the VRPTW and problems with complex constraints e.g. sequential or temporal constraints the corresponding coordination semantic yields significantly better results.

When abstracted from the autonomic and execution specific properties the multi-agent task allocation can be viewed simply as a specific local search algorithm traversing the search-space of relevant task allocations. By comparing the known heuristics with the corresponding possible coordination semantics and local problem solving extensions, important lessons can be learned. For example, the *simulated trading* mechanism outlined above in fact corresponds to a simple ejection based backtracking trading mechanism allowing the agents to trade some tasks for some other tasks if this is found advantageous. Similar lessons can be learned by analyzing other local search or meta-heuristic methods as well. In [76] a genetic algorithm is presented for the VRPTW. Within the crossover phase a specific *edge assembly crossover* operator is used affecting a pair of routes within the two solutions being combined in which the customers served by these two routes are redistributed between them in a specific manner. Such an operation can be again reinterpreted as a complex trading move negotiated between the two agents representing the corresponding pair of routes. Other similar parallels are discussed later within this work.

As mentioned, a significant part of the contribution presented as part of this thesis deals with researching the efficient coordination semantics for the VRPTW and general task allocation problems. Elements of the presented coordination semantics are inspired also by the market-based optimization methods [54]. In particular, relevant for this work are the negotiation protocols surveyed within the next section.

3.4.1 Agent Negotiation and Interaction Protocols

The market-based multi-agent optimization and task-allocation approaches are based on structured negotiation between the participating agents. Similar approaches have been extensively studied within the field of economics [54]. As part of the corresponding research, number of alternative agent negotiation and interaction protocols were proposed providing a formal framework for the underlying communication.

These agent negotiation and interaction protocols enable efficient coordination of a group of cooperating agents by providing communication primitives enabling them to efficiently address the coordination requirements inherent to the particular method of solving the solved problem. In case of the presented task allocation framework this corresponds to the communication inherent to the commitment life-cycle i.e (i) the commitment establishment (ii) commitment execution and (iii) commitment conclusion/termination.

We provide a brief overview of the *contract net protocol* (CNP) introduced by Smith [105] and has later been adopted by the FIPA [27] initiative and standardized as on of the provided alternative FIPA agent communication languages (FIPA ACL). It forms the basis for the communication semantic used by the presented agent framework. The protocol focuses on single-round task allocation. As such it does not address the task execution uncertainty, multi-round negotiations, canceling, repairing or re-negotiation mechanisms inherent to the full commitment life-cycle. On the other hand it provides a basis for a number of extensions some of which provide means to address these aspects. We briefly outline these extensions as well.

3.4.1.1 Contract Net Protocol and its Extensions

As mentioned, the CNP was originally introduced by Smith in early 80's [105], The CNP is one of the basic agents' interaction protocols. It forms the basis for more advanced and complex protocols and captures the fundaments of distributed coordination. The message flow diagram inherent to the CNP is outlined by Figure 3.3. Thus an initiator agent introduces a task to a group of participants using the *call-for-proposal* message (*cfp*) and each participant responds either with *refuse* message if it is not interested in undertaking the task or with *propose* message. The initiator compares all obtained proposals and determines the winning proposal for the task. The winning participant gets *accept-proposal* message representing the confirmation of task assignment from the initiator and the others get *reject-proposal* message. The winning participant then executes the task and informs the initiator abut success of the execution (*failure* or one of the *inform* messages). The particular responders' bids are computed using the agent's local processing algorithm which is private and hidden from the other agents. As such, it can encapsulate the agent's local constraints as well as the agent's strategy used for bidding. Likewise the initiator's evaluation function used to determine the winning agent is private and hidden as well, not being a part of the protocol.

As mentioned a number of extensions to the basic CNP was introduced as well. In [1] an extension of the CNP is presented denoted the *Extended Multi-agent Negotiation Protocol* (EMNP), being inspired by a similar protocol introduced previously in [29]. The protocol targets both collaborative and competitive environments. The EMNP introduces new speech acts such

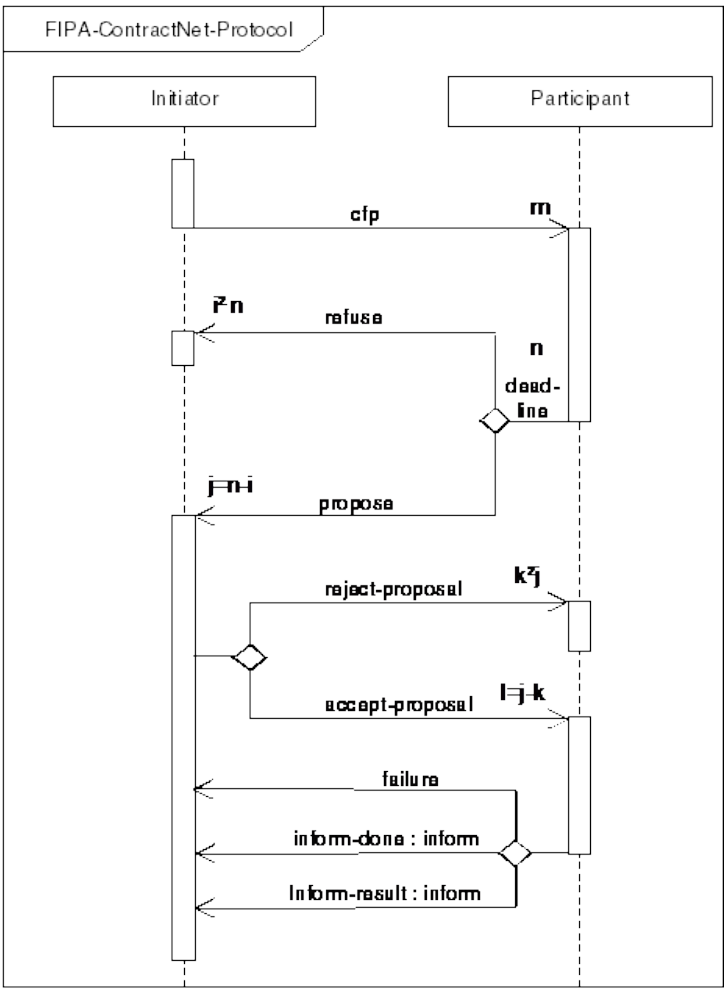


Figure 3.3: FIPA Contract Net Interaction Protocol.

as: *Announce*, *PreBid*, *PreAccept*, *PreReject*, *DefinitiveBid*, *DefinitiveAccept* and *DefinitiveReject* which allows for multi-round negotiation while also enabling conducting multiple negotiations in parallel (more ECNP negotiations running in parallel are enabled due to added flexibility inherent to the preliminary accept/reject actions). This provides an increased flexibility in the commitment establishment phase. On the other hand, the execution and termination phases of the commitment lifecycle are not addressed at all. The commitments are considered final and and no means for reflecting changes in attitudes of the participants once committed are not provided.

In [120] the *Competitive Contract Net Protocol* (CCNP) is introduced, designed for flexible contracting in competitive environments. The protocols provides a greater coverage of the commitment lifecycle specifically (i) the contract conclusion phase, (ii) the optional decommitment phase and (iii) the contract termination phase. During the execution phase, in case any of the committed parties may decide to decommit from the commitment contract a multi-round decommitment negotiation on conditions for dropping the commitment occurs. This may result in

(i) renewing the original commitment (the negotiation fails) or (ii) in dropping the commitments under a payment of a specific penalty. Finally, in the termination phase, the protocol offers the possibility of inflicting penalties on agents not observing the compliance with their respective commitments. The protocol provides support for multi-round negotiations allowing for multiple negotiations to run in parallel.

A further extension of the CCNP is provided by the *Renegotiable Competitive Contract Net Protocol* (RCCNP) [7]. It extends the CCNP by renegotiation phases and provides means for fully flexible contracting in competitive environments. It covers the full commitment lifecycle i.e. the commitment creation, commitment adaptation or breach (even partial breach, with the option of commitment compliance checking and eventual penalty being inflicted in case of contract breach) and commitment termination for the unbreached contracts. On the other hand, the negotiation scheme does not provide the means for capturing the evolution of commitments i.e. the commitments cannot be partially dropped, extended or delegated.

For specific purposes a number of additional extensions have been proposed. In [86] the *Legal Agreement Protocol* was introduced based on Australian contract law. The protocol allows an $M : N$ negotiation which is split into several phases. The first phase allows a not-binding negotiation (the agreed conditions do not imply any commitment for any of the parties) which enables the parties to reach a mutually advantageous compromise. The next phase consists of a binding negotiation over a binding offer (which can be accepted or rejected). Once a contract is established, it is possible to terminate it in several ways as well — by fulfilling the contract, by unilateral decommitment under agreed penalties given by the agreement, by a mutual agreement about canceling the contract without penalties or by contract breach. Another such extension is represented by the *Request-based Virtual Organization Formation Protocol* [117, 116] aimed at automated or semi-automated negotiations mainly in the creation part of the virtual organization life cycle using the concepts of service level agreements.

Chapter 4

Algorithm for VRPTW Based on Agent Negotiation

In this chapter we introduce the agent-based VRPTW algorithm representing the main contribution of this thesis. The algorithm is built around a hierarchical agent architecture and problem decomposition and an agent-based planing process. As outlined in [11], the outstanding aspect of the multi-agent planning is the clear separation between the planning of individual agents and the coordination of their respective plans within a global coordination part of the overall planning process. The presented architecture goes even farther. The top level coordination layer is actually abstract and problem independent as far as the general task allocation problems go. On the other hand, all the problem specific logic inherent to VRPTW is embedded within the agents' local planning layer.

The significance of a big part of the presented work is therefore two-fold. On one hand — this being the main interest of this study — an agent architecture and an underlying agent-based algorithm are presented aimed at efficiently solving the VRPTW. On the other hand, the abstract layer of the presented framework presents a significant extensions of similar concepts known from the multi-agent literature [121] and can be easily reused to solve a variety of general task allocation problems. The two conceptual layers are presented in separate sections to underline the outstanding aspect of the introduced agent-based algorithm. A cloud with the key words for this chapter is depicted by Figure 4.1.

Figure 4.1: Key words for this chapter in a word cloud [127]

4.1 VRPTW as a Task Allocation Problem

As discussed in the previous chapter, the task allocation problem can be described as the problem of assigning a set of tasks to a set of resources in a way such as a particular global objective is maximized. A number of relevant optimization problems, including e.g. various multiple-machine scheduling problems as well as the variants of routing problems all fall into this category. A corresponding multi-agent view of the problem can be achieved by representing the resources as agents. As mentioned, the multi-agent approaches to task allocation have been extensively studied e.g. by [121, 99, 102, 87].

The VRPTW can also be viewed as a task allocation problem. In this case, the resources correspond to the individual vehicles and the tasks to be allocated to the customers these vehicles are serving. While this thesis concentrates in particular on developing efficient multi-agent algorithms for the VRPTW, it turns out that a great part of the algorithmic mechanisms underlying the VRPTW case can be easily generalized to the general task allocation problems.

As discussed in [11], the outstanding aspect of multi-agent planning (or problem solving in general) when viewed in context of distributed problem solving and optimization is the clear separation of the local subproblem solving of individual agents from the coordinations of their respective solutions on the global level of the overall solving process. The presented multi-agent problem decomposition adheres to the aforementioned principle. A global coordination mechanism is presented, that is used to coordinate the distribution of the tasks — the customers to be served — among the vehicles in an efficient manner. The vehicles — the resources — on the other hand, maintain their respective routes independently including addressing also all the VRPTW specific constraints inherent to the feasibility of the routes.

As mentioned, all the problem specific logic e.g. the efficient route planning (the traveling salesmen subproblem), the vehicle capacity constraints (the bin-packing subproblem) or the handling of time-windows constraints is managed by the vehicles locally as part of their local planning strategy. Based on the local processing, the vehicles' only interface with the global coordination mechanism are the cost estimates of committing to serving a particular customer or decommitting from the service (in case the particular vehicle already committed to the task previously) or, in case autonomous behavioral patterns are considered, interactions with the rest of the fleet in order to address dynamism of the environment. Similar local planning strategies, however, can be developed for other task allocation problems as well, enabling the global coordination envelope to be reused for these problems. We consider this inherent flexibility to be one of the outstanding aspects of the presented algorithm.

Also interestingly, in this light, the experimental results and the detailed analysis of the solving process can also serve to outline the limits of the presented abstract coordination framework in general. As outlined previously, due to its huge real-world impact the VRPTW as well as the competitive nature of the field the state-of-the-art VRPTW algorithms represent top of the shelf optimization methods with respect to dealing with complex constrained NP-hard problems. On the other hand, as will be explained later, the used problem specific local planning strategy employed by the vehicles is actually relatively simple. Therefore, the presented comparison can arguably highlight also the impact of the strong agent-to-agent interactions within the abstract task allocation coordination framework to the success of the solving process, providing a useful pointer towards its wider applicability.

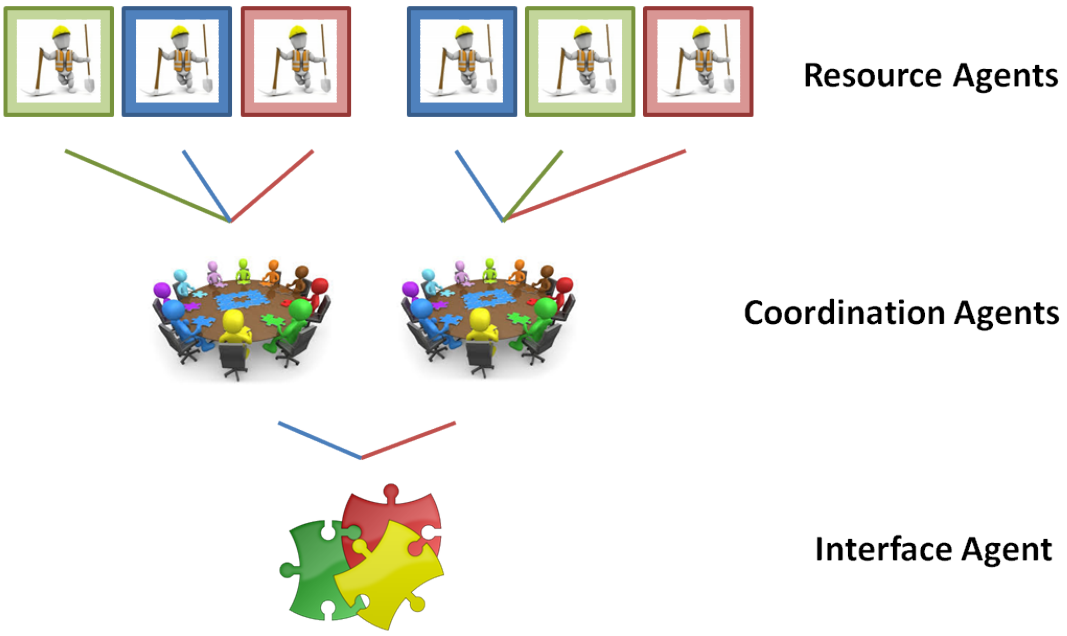


Figure 4.2: Agent Hierarchy Illustration

4.2 The Abstract Agent Architecture for Task Allocation

Within this chapter we describe in detail the agent architecture underlying the presented VRPTW algorithm. It is abstract as it can accommodate various general task allocation problem. The presented agent architecture is a generalization of our previous works [51, 46, 49] and represent an extension of the similar abstract task allocation framework presented in [121].

The architecture is illustrated by Figure 4.2. The three level architecture thus features a top layer consisting of an Interface Agent, middle layer represented by a set of Coordination Agents, each coordinating a set Resource Agents present at the bottom level of the architecture. The architecture can accommodate multiple solving processes being run in parallel. The parallel solving processes may correspond to various problem instances being solved in parallel or to multiple instances of the coordination algorithm with different configurations being run on the same problem instance in a manner similar to portfolio-based planning [114]. This approach is discussed in detail later in Section 4.4.3.

Interface Agent acts as an interface between the algorithm's computational core and the surrounding infrastructure. It is the entry point through which the real problems to be solved are communicated to the system, decomposed into tasks to be allocated which are subsequently submitted to the underlying computational core. It also acts as a controller of the computational core, managing the underlying solving processes represented by the Coordination Agents and communicating the results back to the clients.

Coordination Agents are responsible for coordinating individual solving processes. Each solving process — we use the term *algorithm instance* to denote a single solving process —

relates to a particular problem instance and a particular set of resources represented by Resource Agents assigned to solve the problem instance. In order to solve the problem instance, the Coordination Agent communicates with the Resource Agents about accommodating individual tasks. Various trading moves of tasks between the Resource Agents are explored in an effort to find a quality solution to the problem, corresponding to a particular *coordination semantic* being used. The Coordination Agent is therefore responsible for addressing the *global coordination* part of the overall solving process.

Resource Agents represent individual resources assigned to accommodate the individual tasks within a particular problem instance. Each resource plans and executes its plan independently based on the tasks being assigned to it, using some specific *local planning strategy* to maintain the plan. Based on the particular coordination semantic being used by the corresponding Coordination Agent, the resource agents may be requested to provide specific information to their Coordination Agent e.g. estimate of additional costs inherent to accommodating some additional task or the gains resulting from dropping some task already in the agent's plan. In the remainder of the text we will interchange the terms resource and Resource Agent freely, always referring the corresponding Resource Agent.

```
Input: Ordered list of tasks C, initial solution σ, max no. of backtracks backtrackLimit
Output: Solution σ — complete or partial based on success of the process

Procedure COORDINATE(C, σ)
 1:   backtracks := 0;
 2:   while C ≠ ∅ and backtracks ≤ backtrackLimit
 3:     Extract first c̄ ∈ C;
 4:     Cheapest ≡ {v ∈ Feasible(σ, c̄), v.costCommit(c̄) is minimal};
 5:     if (Cheapest ≠ ∅) then
 6:       Randomly select v̄ ∈Cheapest;
 7:       v̄.commit(c̄);
 8:     else
 9:       PUSH(c̄, σ);
10:     end if
11:     if c̄ ∉ σ and backtrackLimit ≠ 0 then
12:       C_ej := BACKTRACK(c̄, σ);
13:       Add C_ej to the start of C (LIFO stragey)
14:       backtracks++;
15:     end if
16:     SHAKEDYNAMIC(σ);
17:   end while
18:   if C = ∅ then
19:    SHAKEFINAL(σ);
20:   end if
End
```

Figure 4.3: The abstract global coordination process

4.3 The Abstract Global Coordination Algorithm

This section is dedicated to describing the abstract global coordination algorithm in which the Coordination Agent communicates with the Resource Agents about accommodating individual tasks. The coordination algorithm is abstract as it does not address any of the problem specific logic. Also, it can be configured in a number of ways, affecting the complexity and efficiency of the resulting particular coordination algorithm instances. It is a generalization of concepts presented in our previous works [51, 46, 49].

In overall, the coordination algorithm is based on (i) allocating individual tasks to resources and (ii) performing advantageous trading moves shifting the tasks between the resources in order to improve the emerging problem solution. The allocations and trading moves are based on commitment and decommitment cost estimates computed locally by the resources, encapsulating the problem specific logic. On the other hand, the coordination algorithm remains abstract. As already discussed, the separation of agents' local planning and the global coordination is one of the main outstanding aspects of multi-agent planning or problem solving in general.

The coordination algorithm is outlined by Figure 4.3. The COORDINATE function illustrates the allocation and coordination process implemented by the Coordination Agent, corresponding to the global coordination part of the overall solving process. The parameter σ represents the initial solution. The initial solution can either be *empty*, corresponding to a set of particular resources with no tasks having been assigned to them or it can be *partial* or *complete*, with some or all tasks having been already assigned the resources. The C parameter is the list of all unallocated tasks ordered using some particular ordering.

The process begins with identifying the resources that can feasibly accommodate the task being allocated $\bar{c}$ (the set $Feasible(\sigma, \bar{c})$) at the lowest possible cost (the set $Cheapest \subseteq Feasible(\sigma, \bar{c})$) based on the provided cost estimates (line 4). Such a resource (i) has to have the capability to process the particular task and (ii) the capacity to accommodate it — e.g. it has to be able to add it to its plan in such a way as to no not to violate any of the problem specific constraints. For the VRPTW case this for example means, that the vehicle capacity must not be exceeded and the push-forward in the vehicle's schedule resulting from inserting the task $\bar{c}$ (the service of a particular customer in this case) to the corresponding route must not render the service at any of the subsequent customers within the route late. In case $Cheapest \neq \emptyset$ a random resource from the set is chosen and commits to the task (lines 6–7).

In case a straightforward feasible allocation is not possible, the PUSH function is invoked (line 9). In essence the PUSH function traverses the neighborhood of σ attempting to find a solution σ' accommodating all customers from σ such that $Feasible(\sigma', \bar{c}) \neq \emptyset$ — and allocating $\bar{c}$ to this solution. The PUSH function is discussed in detail in Section 4.3.1. In case even this attempt fails (i.e. $\bar{c} \notin \sigma$), the coordination algorithm backtracks using the BACKTRACK function (lines 11–15). The backtracking mechanism is based on allocating the task $\bar{c}$ to the solution σ at the expense of ejecting some other tasks from σ. Most importantly, the criteria for selecting the resource r_{ej} and the particular tasks $c_{ej_i} \in C_{ej} \subseteq r_{ej}$ to be ejected in order for r_{ej} to be able to accommodate the task $\bar{c}$ are abstract and problem independent. The backtracking mechanism including the abstract selection criteria for selecting the tasks in the set of ejected customers C_ej is described in detail in Section 4.3.2.

The ejected tasks $c_{ej_i} \in C_{ej}$ are then added to the start of the list of remaining tasks to be allocated C thus following a LIFO strategy with respect to allocating the ejected customers. This is motivated by the assumption that in case the emergent solution σ cannot accommodate the task $\bar{c}$ (including the possible chain of backtracking steps and the resulting attempts to re-allocate all the ejected customers to the solution σ), it is very unlikely that it will be able to accommodate $\bar{c}$ later within the solving process. The failure to allocate $\bar{c}$ to σ arises typically towards the end of the solving process as the plans of the individual resources get more tightly constrained. For example, for scheduling problems with task completion time constraints, this corresponds to the fact that the schedules get denser and most insertion slots are rendered unfeasible due to the fact that the resulting push-forward in the schedule would render some of the later tasks in the schedule late. Such is often the situation with the VRPTW where, however, additional vehicle capacity constraints apply as well.

Follows an attempt to improve the emerging partial solution σ by using the SHAKEDYNAMIC function (line 16). The SHAKEDYNAMIC function traverses the neighborhood of σ in an effort to find a *better* solution σ'. The intended *improvement* to σ may be twofold — (i) the solution may be modified in a way as to maximize the social welfare and (ii) it can also aim at transforming the solution in a way as to increase the chance of success for the later task allocation e.g. by aiming at relaxing the constraints within the solution σ. In case of the VRPTW, for example, an effort can be made to increase the *slackness* of the individual routes to create space for possible advantageous detours to be made to serve the customers encountered later within the coordination process. The SHAKEDYNAMIC function and its alternative implementations are discussed in detail in Section 4.3.3. Again, like in the case of the BACKTRACK function, the semantic of the SHAKEDYNAMIC function is abstract and does not relate to any particular allocation problem.

The task allocation loop (lines 2–17) concludes by either (i) all tasks from $c \in C$ being allocated to σ (i.e. $C = \emptyset$) or by (ii) the upper limit on the number of backtracking steps being reached (i.e. $backtracks = backtrackLimit$). In the latter case the coordination process is considered failed. The returned solution σ is not complete as some of the tasks were not allocated. In the opposite case, after all tasks have been allocated the SHAKEFINAL function is

```
Input: Resource r, task t, solution σ
Output: Solution σ

Procedure RELOCATE(r, t, σ)
1:    r.decommit(t);
2:    Cheapest := {r' ∈ Feasible(σ, t), r'.costCommit(t) is minimal};
3:    Randomly select r̄ ∈ Cheapest;
4:    r̄.commit(t);
5:    return σ;
End
```

Figure 4.4: A relocation of a single task

invoked (line 19). It is based on the same concepts as the SHAKEDYNAMIC fuction, being an abstract, problem independent method aimed at improving the solution σ. However, in case of the SHAKEFINAL function, unlike the SHAKEDYNAMIC function, all tasks $c \in C$ have already been allocated to σ. We mentioned previously, that in case of the SHAKEDYNAMIC function, the intended improvement may be aimed at transforming the solution in a way as to increase the chance of future task allocations. For the SHAKEFINAL function, all tasks have already been accommodated by the resources in σ and therefore no such need exists. Therefore it is solely concerned with improving the social welfare and thus finalizing the coordination process. The abstract global coordination algorithm concludes by returning the solution σ (line 22).

The abstract global coordination algorithm provides a framework which can be used to solve a number of general task allocation problems. In [121] a similar approach is used to solve a number of diverse task allocation problems with good results. Also, it can be parameterized in a number of ways, providing for alternative *coordination semantics*. For example, the backtracking mechanism can be disabled by setting the *backtrackLimit* parameter to 0, or a dummy implementation can be used for any of the SHAKEDYNAMIC, SHAKEFINAL and PUSH functions, reducing the complexity and potentially also the efficiency of the resulting algorithm. As mentioned above, within the presented agent architecture each Coordination Agent is responsible for running a single coordination process. This process corresponds to solving a particular allocation problem given by an ordered list of tasks C by allocating these tasks to a corresponding set of resources given as part of the initial solution σ using a particular coordination semantic. We refer to a single such process as a single *algorithm instance*. In the following sections the particular implementation of the PUSH, SHAKE.. and BACKTRACK functions are discussed including the ways in which these can be parameterized providing for alternative coordination semantics.

As outlined previously, in order to leverage the autonomic nature of the presented task allocation process, the allocations of tasks to the agents are represented by means of social commitments. While this work does not directly address the benefits such a representation provides in terms of increased execution stability in adynamic environments, the experimental assessment presented in [121] or [122] suggests that in specific scenarios it can provide a significant advantage making the adoption of a multi-agent approach an intriguing alternative to traditional centralized optimization approaches.

4.3.1 The PUSH and SHAKE.. Functions

Within the PUSH and SHAKE.. functions a series of trading moves is performed. By providing the particular implementation of these functions a particular *coordination sematic* of the abstract coordination process is specified. In the remainder of this section we discuss the particular

```
Input: Resource r, relocated task t, pushed task p, solution σ
Output: Solution σ

Procedure RelocatePush(r, t, p, σ)
 1:     r.decommit(t);
 2:     β := null;
 3:     if r.feasible(p) ≠ ∅ then
 4:       Φ := {r' ∈ Feasible(σ, t) \ {r}};
 5:       forall (γ ∈ r.feasible(t))
 6:         r.commit_γ(t)
 7:         if r.feasible(p) ≠ ∅
 8:           Φ := Φ ∪ {r};
 9:           β := γ;
10:           break forall;
11:         end if
12:         r.decommit(t)
13:       end forall
14:       if Φ ≠ ∅ then
15:         Randomly select r' ∈ Φ;
17:         if r' = r then r'.commit_β(t) else r'.commit(t) end if;
16:         r.commit(p)
18:         return σ;
19:       end if
20:     end if
21:     Cheapest := {r' ∈ Feasible(σ, t), r'.costCommit(t) is minimal};
22:     Randomly select r̄ ∈Cheapest;
23:     r̄.commit(t);
24:     return σ;
End
```

Figure 4.5: A relocation of a single task with a push effort

implementations used within the presented framework and discuss the ways in which these can be further parameterized.

4.3.1.1 Task Relocations

In this work we considered trading moves based on *task relocations*. The relocation process is outlined by the Figure 4.4. For a single task t thus the relocation process consists of the resource r whom the task is assigned to decommiting from the task (line 1). Then all the agents $r' \in \sigma$ that can feasibly accommodate the task t (the set $Feasible(\sigma, t)$) estimate the commitment cost $r'.costCommit$ and the resources providing for the lowest commitment cost are identified (line 2). Then a randomly chosen one of these resources $\bar{r} \in Cheapest$ commits to the task (lines 3,4). Note that the original resource r is also included in the process of selecting the most suitable resource for t (line 2). Thus $Feasible(t) \neq \emptyset$ and as a special case of relocation the task t may be reassigned to r. Considering the schedule or plan of r, this can result in t being inserted to exactly the same position within the plan or a more advantageous position is identified.

An alternative variant of the relocate function is used within the Push function. In that situation, a task p was identified within the solving process such that $Feasible(\sigma, p) = \emptyset$. Then, for every examined relocation of the task t we examine whether the resource r's decommitment from t doesn't create an opportunity for r to accommodate p. If so, we further examine whether

there exists a way to accommodate the relocated task t by any other resource, or by the resource r itself (i.e r may be able accommodate t in a different way then the one associated with the original commitment, enabling both the tasks t and p to be allocated to the resource r). In case this is possible, task p has been successfully *pushed* within the solution σ. Let $r.feasible(t)$ denote the set of all the *particular commitments* of the resource r to the tasks t — i.e. all the possible ways in which r can accommodate task t. Then $r.feasible(t) = \emptyset$ means r cannot feasibly accommodate t. Let further for any $\gamma \in r.feasible(t)$ the $r.commit_\gamma(t)$ denote the corresponding particular way the resource r can accommodate the task t.

The alternative relocation process is outlined by Figure 4.5. The process starts by resource r decommitting from the relocated task t (line 1) and resetting the variable β discussed later. As no other resource can accommodate p (as in the opposite case it would have been allocated to that resource and the RelocatePush function would not be invoked) the resource r after decommitting from t is the only resource that can possibly accommodate p. If this is not possible, it means that the task p cannot be allocated to σ in the current iteration. Thus the process continues as a standard relocation (lines 21 – 24). In the opposite case, the process continues by trying to relocate the task t to another resource or to allocate it back to r but in a different way, so that both p and t can be feasibly accommodated by the resource r. The set of resources that can feasibly accommodate t other than r denoted Φ is identified first (line 4). Follows the evaluation of all possible alternative ways of r accommodating t (line 5) in an effort to identify some that would allow p to be accommodated as well. For each of them, the corresponding commitment is performed (line 6) and the feasibility of allocating p to r is evaluated. In case of success (line 7), the resource r is also added to the set Φ (line 8) and the particular commitment is stored within the variable β. If even after this effort the set Φ is empty it means that the process of allocating both the tasks p and t has failed and the process continues as a standard relocation (lines 21–24). In the opposite case a random resource r' is selected from the set Φ and commits to the task t (line 16). If the selected resource is the original resource r, the identified particular commitment stored within the variable β is used. In the opposite case the resource uses the least costly way of accommodating the task t based on its local processing. Follows the commitment of the resource r to the task p (line 17). At this point the task p was successfully accommodated within the solution σ and the modified solution is returned (line 18).

The presented implementation of the RelocatePush function corresponds to a simplified version of the actual process. Within the full implementation the process also considers the costs of the possible commitments and strives to return a solution σ accommodating both the tasks with the lowest possible cost thus trying to choose such a solution that enables the allocation of p but as well maximizes the social welfare. We present the simplified version for the sake of clarity. The asymptotic complexity of the process remains the same and the relevant extension is not difficult to implement.

4.3.1.2 Trading Methods based on Task Relocations

We considered three general trading methods based on task relocations. For all three methods all resources within the solution σ are affected. The order of processing the individual resources follows any fixed ordering of the resources within σ the and may be subject to problem-specific arrangements. The three methods differ in the way the tasks to be relocated for each resource are selected. The methods are:

- ***RelocateAll***: For each resource all of its tasks are relocated. The order of processing is not exactly specified and may correspond to the particular problem being solved.
- ***ε-RelocateRandom***: For each resource a portion of its tasks corresponding to $\varepsilon \in (0, 1\rangle$ is relocated. For $\varepsilon = 1$ this corresponds to the previous method but for an individual resource

```
Input: Solution σ, max. no of iterations loopLimit, trading method instance m
Output: Solution σ

Procedure SHAKE(σ, loopLimit, m)
 1:     repeat loopLimit times
 2:       shaking := false;
 3:       for r ∈ σ
 4:         R := {t ∈ r selected based on m}
 5:         forall t ∈ R
 6:           σ' := RELOCATE(r, t, σ);
 7:           if (σ' ≠ σ) then
 8:             σ := σ';
 9:             shaking = true;
10:           end if
11:         end forall
12:       end forall
13:       if shaking = false then
14:         return σ;
15:       end if
16:     end repeat
17:     return σ;
End
```

Figure 4.6: A relocation of a single task

the tasks are always processed in a random order.

- ***ε-RelocateWorst***: For each resource the portion of its tasks with the maximal value of *r.gainDecommit(t)* corresponding to $\varepsilon \in \langle 0, 1\rangle$ is relocated. The decommitment gain corresponds to the resources capacity being freed by dropping these commitments or other similar metric which can be problem specific. The tasks are thus processed in decreasing order based on their respective decommitment gain estimates.

4.3.1.3 The ShakeDynamic and ShakeFinal Functions

The ShakeDynamic and ShakeFinal functions therefore adopt one of the above mentioned negotiation methods and apply it to the emerging problem solution σ. Both functions are logically identical, outlined by the Figure 4.6. We discriminate between the two to illustrate the fact, that an alternative configurations for the individual resources $r \in \sigma$ as well as for the method itself (e.g. the *loopCount* parameter discussed below) can be used for the *dynamic* and $final$ applications.

The function consists of traversing all the resources $r \in \sigma$ (line 3). For each resource a subset $R \subseteq r$ of the tasks assigned to this resource is selected according to the particular trading method specified by the m parameter (e.g. *RelocateAll*, *ε-RelocateRandom*, *ε-RelocateWorst*, including a particular setting for the parameter ε where applicable — line 4). Each such task $t \in R$ is then relocated. The process is repeated until (i) no more relocations are possible (line 13–14) or (ii) a number of iterations specified by the *loopLimit* parameter is reached.

4.3.1.4 The Push Function

The Push function is similar to the Shake function. Within the Push function, however, the neighborhood of the entering solution $\sigma, p \notin \sigma$ is traversed in order to find a solution σ_f such

```
Input: Solution σ, pushed task p, max. no of iterations loopLimit
Output: Solution σ

Procedure PUSH(σ, p)
 1:     repeat loopLimit times
 2:       shaking := false;
 3:       for r ∈ σ
 4:         forall t ∈ r
 5:           σ' := RELOCATEPUSH(r, t, p, σ);
 6:           if (p ∈ σ') then
 7:             return σ'
 8:           end if
 9:           if (σ' ≠ σ) then
10:             σ := σ';
11:             shaking = true;
12:           end if
13:         end forall
14:       end forall
15:       if shaking = false then
16:         return σ;
17:       end if
18:     end repeat
19:     return σ;
End
```

Figure 4.7: A relocation of a single task

that $Feasible(\sigma_f, p) \neq \emptyset$ — i.e. a solution in which the task p can be accommodated by one of the resources. In such a case the identified resource commits to the task p. The method is outlined within the Figure 4.7. The fundamental difference between the two methods is (i) that in case of the PUSH function for each resource all tasks $t \in r$ are processed (therefore corresponding to the *RelocateAll* trading method — line 3) and (ii) that once a solution $\sigma', p \in \sigma'$ is found, it is returned immediately (lines 6–8) and the process terminates. This corresponds to the fact that in such a case, the application of the SHAKEDYNAMIC immediately follows.

4.3.2 The BACKTRACK Function

The BACKTRACK function enables the algorithm to backtrack in a situation it is not possible to feasibly allocate the currently processed task $\bar{c}$ to the intermediate solution σ. Such a case corresponds to a situation when allocation of c by any of the resources $r \in \sigma$ would result in violating some constraint associated with the resource. In case of the VRPTW, for example, the violated constraint might be exceeding the vehicle capacity or violating a time-window constraint associated with one of the customers later in the vehicle's schedule.

The BACKTRACK function then consists of identifying the *most fitting* set of tasks C allocated to some resource $r \in \sigma$ such that if r decommits from the tasks $c_{ej_i} \in C$, it can feasibly accommodate the task $\bar{c}$. Formally, we denote any subset of tasks $C \subseteq r$ as a possible *ejection*. An ejection is *feasible* iff it has the above property — i.e. the removal of $c_{ej_i} \in C$ from r makes it possible to allocate $\bar{c}$ to r. The set of all feasible ejections for a particular resource is denoted $r.feasibleEjections(\bar{c})$. Furthermore, with each feasible ejection $C \in r.feasibleEjections(\bar{c})$ a specific *ejection cost* is associated, denoted $r.ejectionCost(\bar{c}, C)$. The ejection costs are computed locally by the resources as part of their local planning. However, as we'll show later, the ejection

```
Input: Task c̄, solution σ, max no. of ejected tasks maxSize
Output: Ejected tasks C_ej

Procedure BACKTRACK(c̄, σ)
 1:   r_b := null;
 2:   C_b := null;
 3:   minCost := ∞;
 4:   forall r ∈ σ
 5:     C_r = r.bestEjection(c̄, maxSize);
 6:     if r.ejectionCost(c̄, C_r) < minCost
 7:       r_b := r;
 8:       C_b := C;
 9:       minCost := r.ejectionCost(c̄, C_r);
10:     end if
11:   end forall
12:   forall c_ej_i ∈ C_b
13:     r_b.decommit(c_ej_i);
14:   end forall
15:   r_b.commit(c̄);
16:   return C_b;
End

Procedure r.bestEjection(c̄, maxSize)
 1:   C_r := null;
 2:   minCost := ∞;
 3:   forall C ⊆ r, |C| ≤ maxSize
 4:     if C ∈ r.feasibleEjections(c̄) and r.ejectionCost(c̄, C) < minCost then
 5:       C_r := C;
 6:       minCost := r.ejectionCost(c̄, C);
 7:     end if
 8:   end forall
 9:   return C_r
End
```

Figure 4.8: A relocation of a single task

cost computation mechanism can also make very good use of some global problem independent criteria i.e. to provide for efficient diversification of the underlying search process.

The process is outlined by Figure 4.8. The process is started with resetting the intermediate best found ejection C_b, the variable storing its cost *bestCost* and the corresponding resource r_b (lines 1,2). All the resources are traversed (line 4), in an arbitrary order. Each resource identifies the best possible ejection C_r enabling r to feasibly accommodate the task $\bar{c}$ using the *r.bestEjection* function (line 5). The *r.bestEjection* and *r.ejectionCost* functions are particular to individual resources. They are processed locally by the resources and can reflect the particular problem being solved. If the cost of the ejection C_r improves on the cost of the intermediate best found ejection *bestCost*, the variables storing the best ejection and the corresponding resource C_b and r_b are updated (lines 6–10). After all resources have been processed, the resource r_b corresponding to the identified best ejection C_b decommits from the tasks $c_{ej_i} \in C_b$ and commits instead to the task $\bar{c}$ which can now be feasibly accommodated by the resource (lines 12–15). Lastly, the ejection C_b is returned. As previously outlined in Section 4.3, within the COORDINATE function, the tasks $c_{ej_i} \in C_b$ are then added back to the list of the customers to be allocated C. The tasks are added to the start of C to be processed immediately after being ejected. Therefore,

before the next task from C is processed, the whole chain of backtracks possibly triggered by the current backtracking step has to be resolved. This is motivated by the assumption that if the intermediate problem solution σ cannot accommodate these customers at this point of the solving process, it is unlikely to accommodate them later, after other customers are added and the solution becomes more tightly constrained.

The process of identifying the best possible ejection for a particular resource is a private function of that particular resource and can be tailored to exploit the specifics of the problem being solved. On the other hand, as we mentioned at the beginning of this section, some parts of the ejection cost computation mechanism can be generalized. A generic implementation of the *r.bestEjection* function is presented within the Figure 4.8 below the BACKTRACK function. The process consists of traversing all the possible ejections with the number of ejected tasks being lower or equal to the *maxSize* parameter (line 3). An ejection C is considered only if (i) it represents a feasible ejection with respect to the task $\bar{c}$ and (ii) its cost represents an improvement over the intermediate best ejection cost *minCost* and (line 4). The particular evaluation strategy for these two conditions can be problem specific, however, some general conclusions apply as well:

- If C' is a superset of C and C is a feasible ejection, C' doesn't need to be considered. In general, any feasible ejection cost mechanism should attempt to find an ejection C_b that is *minimal* in the sense that all subsets of C_b are not feasible ejections.
- If C' is a subset of C and C is a not a feasible ejection, C' doesn't need to be considered as it is not feasible as well.
- If $r.ejectionCost(\bar{c}, C') \geq minCost$, C' doesn't need to be considered. This can be useful especially when the evaluation of the $r.feasibleEjections(\bar{c})$ function is complex, helping to avoid unnecessary invocations of the $r.feasibleEjections(\bar{c})$ function.
- Let for any task $c \in C$ the *c.fails* value indicate the number of times the BACKTRACK function was invoked with c as the first parameter — i.e. the feasible allocation of c to the intermediate solution σ failed. Then the cost estimate for an ejection C given by $r.ejectionCost(\bar{c}, C) = \sum_{c \in C} c.fails$ provides for efficient search diversification within the backtracking mechanism while being abstract, independent of the particular problem being solved.

The last mentioned ejection cost evaluation criterion $r.ejectionCost(\bar{c}, C) = \sum_{c \in C} c.fails$ thus considers the cost of the ejection C corresponding to the number of times the tasks $c \in C$ failed to be feasibly allocated. The *c.fails* can be considered as the simplest abstract indicator of the difficulty of allocating c to the solution σ. In this sense, the above cost mechanism prioritizes ejections composed of tasks, that have proved to be easier to allocate to the emerging solution σ. For example, the criterion prevents the algorithm from being trapped in an infinite cycle backtracking and reallocating over a limited subset of tasks. For such a subset the *c.fails* counters will increase and the backtracking mechanism will eventually favor ejections of tasks outside this set.

The backtracking mechanism is the last ingredient constituting the abstract algorithm. The abstract algorithm is highly configurable. It can be configured in various ways to solve different problems, with different complexity and efficiency of the resulting algorithm instances. The process of instantiating and running instances of the algorithm is discussed within the next section.

4.3.3 Instantiating the Abstract Algorithm

In previous sections we outlined abstract coordination algorithm. It provides an abstract framework that can be adapted to solve various task allocation problems. This is done by supplementing the local planning strategies to be used by the resources. We will discuss this in detail when explaining the adaptation of the algorithm to the VRPTW problem being the core research interest of this thesis. However, the abstract coordination process itself represented by the COORDINATE function can be configured as well e.g. by manipulating the *loopLimit* parameter of the SHAKEDYNAMIC, SHAKEFINAL and PUSH or the *backtrackLimit* parameter affecting the backtracking mechanism. Thus, in this section, we discuss these later settings and define some baseline configurations of the abstract algorithm.

The baseline settings with respect to the parameterization of the abstract coordination process are:

Algorithm-B setting corresponds to the following parameter values: $backtrackLimit = 0$ meaning that the backtracking mechanism is disabled and $loopLimit = 0$ for all three SHAKEDYNAMIC, SHAKEFINAL and PUSH functions. Such a setting corresponds to a simple parallel insertion procedure with no efforts being made to improve the problem solution σ throughout or at the end of the solving process.

Algorithm-F setting extends the *Algorithm-B* setting by having a $loopLimit \neq 0$ value in case of the SHAKEFINAL function. Thus, in this setting after all tasks have been allocated an effort is made to improve the solution σ in terms of social welfare maximization by performing a series advantageous trading steps.

Algorithm-D setting further extends the *Algorithm-F* setting by having a $loopLimit \neq 0$ value also in case of the SHAKEDYNAMIC and PUSH functions. Thus, in this setting an effort is made to improve the solution σ also *dynamically* throughout the solving process. The improvement can be twofold — aiming at (i) improving the social welfare by reducing the overall task allocation costs or (ii) at transforming the solution to an intermediate state that increases the chance of the success of future task allocations.

Algorithm-BT is the full-fledged unbounded setting, extending the *Algorithm-D* setting by enabling the backtracking mechanism by having $backtrackLimit \neq 0$.

For the first two *Algorithm-B* and *Algorithm-F* settings, in case a task t is encountered, such that no resource can feasibly accommodate the task (the set $Cheapest \equiv \{r \in Feasible(\sigma, t), v.costCommit(t) \text{ is minimal}\} = \emptyset$) the process fails. In the *Algorithm-D*, each task allocation is followed by the within the SHAKEDYNAMIC function being invoked in an effort aimed at increasing the chance of success of accommodating future additional tasks. Also, when a task t is encountered such that the straightforward feasible allocation is not possible an additional effort is made to reorganize the tasks in σ within the PUSH function by attempting series of trading moves in a way as the accommodate t within the solution σ. Therefore, for this setting, the chance of the solution σ being able to accommodate the task t causing the failure in the previous cases is generally higher. However, in case a task t is encountered such that t cannot be feasibly allocated to the solution σ even within the PUSH function the whole process fails as well. Such a situation is efficiently addressed within the full-fledged *Algorithm-BT* setting which allows to backtrack from such a situation by allocating t to the solution σ at the expense of ejecting some other tasks from σ.

The Example 4.3.1 illustrates the differences between the individual settings. It also illustrates how the order in which the tasks are processed within the COORDINATE function can affect the

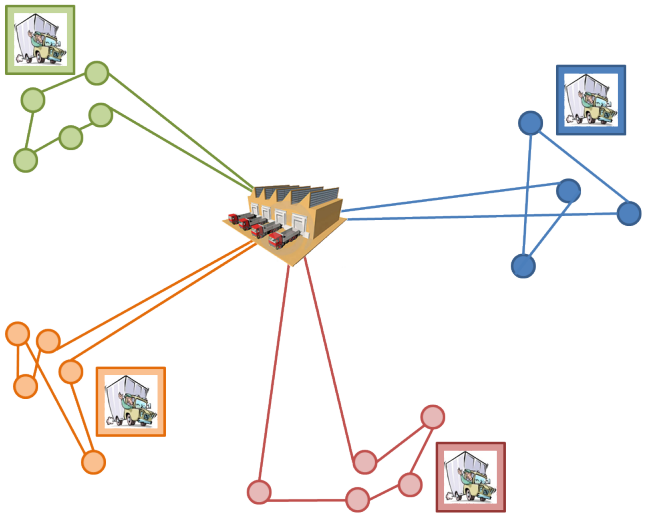


Figure 4.9: VRPTW problem instance with clustered customers — the optimal solution

success of the coordination process and the quality of the resulting solution. Therefore the choice of a particular ordering of the tasks is another important factor when instantiating the algorithm.

Example 4.3.1. Consider a specific instance of the VRPTW. a situation with tasks corresponding to customers to be served dispersed on a map and resources to the vehicles serving these customers with routes starting end ending in a single depot. Also, each customer has to be served within a specific time interval. Given a fixed-size fleet, the social welfare in this case is the total distance traveled by all the vehicles combined. Let's consider a cost estimate of a vehicle accommodating a customer within its route as the increase in the traveled distance due to the inherent detour necessary in order to serve the customer.

Imagine a problem instance with the customer locations being grouped in clusters, with distances of the customers in the same cluster being significantly shorter than distances of customers in different clusters. Now imagine that the problem has an optimal solution consisting of a single vehicle serving customers in each cluster. The situation is illustrated by Figure 4.9. Given an ordering with the customers in each cluster forming an uninterrupted segments ordered exactly in the order in which the vehicle should visit them in the optimal case, even the simplest setting can easily produce the optimal solution. However, using a different ordering, it is easily possible that a vehicle will chose to serve customers from different clusters (as this may appear beneficial at some point of the coordination process) as illustrated by Figure 4.10. In such a case, however it can easily happen that a customer is encountered such that for any of the vehicles, in order to reach the customer within the associated time-window the inherent detour would cause some other customers on the vehicle's route to be served late. In such a case, the customer cannot be feasibly allocated.

Now consider that additional trading steps are performed in between the vehicles as part of the *Algorithm-D* setting. The trading steps may enable the routes to be reorganized in a beneficial way relocating customers causing the vehicles to travel between clusters to vehicles serving some other customers in that area (thus making a shorter detour). This can improve the situation to some extent. However, imagine, that all of the routes are absolutely spot-on in terms of timing the visits and for neither of the vehicles a detour is possible in between any

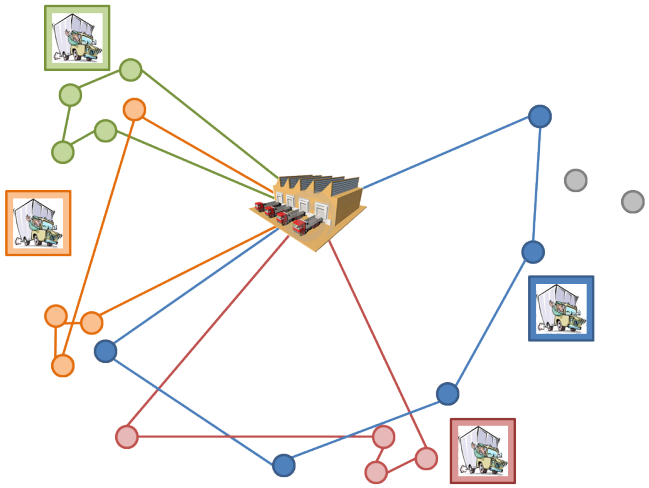


Figure 4.10: A suboptimal solution due to unfitting order of processing the customers. The gray nodes correspond to the customers that were encountered later within the solving process and couldn't be accommodated by any of the vehicles.

two customers on the vehicle's route. This can easily happen when the vehicles have to travel unnecessarily long distances. In such a case no relocation of customers will be possible — for all the vehicles, accommodating an additional customer would result in the vehicle arriving late somewhere further down the route.

The backtracking, being part of the *Algorithm-BT* enables the algorithm escape such a "deadlock" situation. It enables the algorithm to perform some destructive steps where necessary. Thus a vehicles may be forced to *eject* multiple customers from the route at once (e.g. those causing the longest detours) in order to accommodate the currently processed customer. The removed customers then reenter the allocation process. This allows the routes of the individual vehicles to be reorganized even when most local moves are rendered unfeasible due to the tightness of the underlying constraints — in this case the task incompatibilities due to the temporal aspects of the problem.

As already mentioned, a particular algorithm instance is fully specified by (i) parametrization of the of the abstract solving process, taking one of the above baseline settings, (ii) the set of tasks to be allocated and their ordering and (iii) the corresponding set of resources including the implementations of the cost evaluation functions. The set of tasks to be allocated as well as the set of resources these tasks can be allocated to are typically given as part of the definition of the particular problem instance to be solved. The choice of a particular task ordering and a particular abstract algorithm parametrization provides opportunities to customize the resulting algorithm.

Given a set of alternative orderings $\Omega = \{o_1 ... o_n\}$ and a set of parameterizations of the abstract algorithm $\Delta = \{\delta_1 .. \delta_m\}$ we refer to the set $\Omega \times \Delta$ as the *algorithm configuration space*. When solving a particular problem instance, multiple algorithm instances can be instantiated in order to solve the problem instance. For example, a set of algorithm instances can be run in parallel in an approach similar to the portfolio based planning [114] with an improving sequence of results being extracted as the individual instances finish over time. Or, when processed sequentially, it may be advantageous to modify the instances yet to be run based on the examination

of the solutions provided by the already finished instances.

Therefore, by traversing the algorithm configuration space and instantiating and executing algorithm instances in a specific way provides an interesting way of meta-optimization and parallelization when adapting the abstract algorithm mechanism to a particular problem. As part of this thesis a parallel wrapper is presented for the VRPTW case, based on traversing in a specific way the algorithm configuration space generated using specific means of *algorithm configuration diversification*. The wrapper exploits some specific attributes inherent to the VRPTW in an effort to provide for an efficient overall VRPTW algorithm with good parallelization features and parallel anytime characteristics.

4.3.4 Complexity Analysis

In this section we provide the worst case asymptotic complexity bounds for the baseline algorithm settings presented within the previous section. Let $N = |C|$ be the number of tasks to be allocated for a given problem instance and $M = |\sigma|$ be the number of resources. Within the COORDINATE function the algorithm iterates over the set C of tasks yet to be allocated. At various places of the algorithm the resources are requested to (i) examine the feasibility of accommodating some task c e.g. enumerating the *r.feasible* set (ii) estimate the cost of accommodating some task c i.e. executing the *r.costCommit*(c) function, (iii) process the actual commitment i.e. executing the *r.commit*(c), (iv) estimate the gain of dropping the commitment from some task c' i.e. executing *r.gainDeommit*(c') and (v) processing the actual decommitments i.e. executing *r.decommit*(c) for some task c. For the purposes of this analysis we consider that all resources are similar in terms of the asymptotic complexity of these functions. Let F, CC, C, CD and D denote the complexity of these functions respectively.

In case of the simplest *Algorithm-B* setting, the algorithm consists of traversing all the tasks $c \in C$. For each task (i) the most suitable resource is identified and (ii) it commits to the task. Let O_{alloc} correspond to the complexity of these two actions. The complexity O_{alloc} thus corresponds to

$$O_{alloc}(N, M) = M \times CC + C. \tag{4.1}$$

Thus, in case of the *Algorithm-B* setting, the corresponding complexity of the algorithm denoted O_B is

$$O_B(N, M) = N \times (M \times CC + C). \tag{4.2}$$

In case of the *Algorithm-F* setting, the SHAKEFINAL function is invoked at the end of the process. It consists of traversing the set of customers C *loopLimit* times, attempting to relocate each customer. The relocation of a single task c consists of (i) the resource r currently accommodating the task c decommiting from the task, (ii) identifying the most fitting resource r' that can accommodate the task c and (iii) having the resource r' to commit to the task c. The complexity of this process (given the *loopLimit* parameter is constant) is

$$O_{shake}(N, M) = N \times (D + O_{alloc}(N, M)). \tag{4.3}$$

Therefore the worst case asymptotic complexity of the *Algorithm-F* setting is bounded by

$$\begin{aligned} O_F(N, M) = N \times O_{alloc} + O_{shake} = \\ N \times (M \times CC + C) + N \times (D + M \times CC + C) = \\ O(N \times (D + M \times CC + C)) \end{aligned} \tag{4.4}$$

In case of the *Algorithm-D* setting, the SHAKEDYNAMIC function is invoked after each task has been processed as is potentially the PUSH function. The asymptotic complexity of the SHAKEDYNAMIC function is identical to the complexity of the SHAKEFINAL function. The PUSH function is very similar to the SHAKE.. functions, differing only in the RELOCATEPUSH function instead of the RELOCATE function being executed for each task within the intermediate solution σ. As opposed to the RELOCATE function, the RELOCATEPUSH function contains an additional step of identifying and traversing the set *r.feasible(t)* of possible alternative ways the resource r (the resource currently accommodating the relocated task t) can accommodate the task t in order to make it possible for p to be allocated. The *way of committing* of the resource r to the task t is loosely defined and corresponds to the particular problem being solved. In this analysis we assume the number of such ways is less or equal to the number of tasks already allocated to r. In case of the sequencing/scheduling allocations problems like the VRPTW being the primary concern of this study this corresponds to considering alternative positions in which t can be inserted within the r's schedule without reorganizing the rest of the schedule. This assumption bounds the complexity of the RELOCATEPUSH function by

$$O_{relPush}(N, M) = O_{alloc}(N, M) + N \times (D + F + C). \tag{4.5}$$

and therefore the PUSH function by

$$O_{push}(N, M) = N \times (O_{alloc}(N, M) + N \times (D + F + C)). \tag{4.6}$$

Therefore the complexity of the *Algorithm-D* setting is

$$\begin{aligned} O_D(N, M) = \\ N \times (O_{alloc}(N, M) + O_{shake}(N, M) + O_{push}(N, M) + O_{shake}) = \\ O(N^2 \times (M \times CC + N \times (D + F + C))) \end{aligned} \tag{4.7}$$

The backtracking mechanism in case of the *Algorithm-BT*, in the worst case, causes the algorithm to terminate only when the number of backtracks reaches the limit number given by the *backtrackLimit* parameter. Therefore, in this case, we consider the algorithm computationally unbounded.

4.3.4.1 Other Complexity Observations

Even for the settings we consider computationally bounded and which are polynomial (given the local planning strategy is polynomial as well), the computational complexity of the resulting algorithm depends heavily on the settings for the *loopLimit* parameters used for the SHAKE.. and PUSH functions. On the other hand, the *loopLimit* parameters are applied to trimming the number of performed *improving* relocations. Assuming reasonable implementations of the resources' local planning, the solution improving process based on task relocations will not cycle and eventually stop due to the process reaching some local optimum. Consider that this optimum is reached in some iteration of the algorithm – i.e. after allocating the i-th task t_i. After allocating the next processed task t_{i+1} to some resource r, the only new possible improving relocations are those concerning the resource r as the improving relocations between all the remaining resources have already been explored. Adding t_{i+1} to r can possibly trigger a chain of changes leading to a new local optimum. However, given the small footprint of the change corresponding to adding a single task to a single resource, the number of the resulting available improving relocations will typically be small. Therefore, even when the *loopLimit* parameter setting is high, the SHAKE.. and PUSH functions are likely to terminate early in most cases

Figure 4.11: VRPTW chain of improving relocations triggered by adding a customer

due to the solution reaching new local optimum rather than by the *loopLimit* limit number of iterations being reached. Example 4.3.1 illustrates the situation on the VRP.

Example 4.3.1. Consider the specific case of the VRP depicted on Figure 4.11. Starting from top right corner clockwise, a chain of improving relocations is illustrated resulting from adding a new customer to the solution (the gray node in the top left picture). By adding the customer to the solution the corresponding vehicle travels through the plane on a different trajectory, making it advantageous for some other task to be allocated to the vehicle as well (the second picture). The situation repeats in the fourth picture as well. In a complex scenario the number of thus triggered relocations can be substantial. On the other hand, its easy to see the limiting factors for this. Due to the locality of the change corresponding to adding a single customer, apart from very special cases, such a change is unlikely to trigger vast changes throughout the whole solution which is in local optimum with respect to improving relocations.

The monotonicity of the improvement process within the SHAKE.. functions is arguably also one of the key weaknesses of the abstract algorithm when applied to problems with complex resource constraints. Such constraints — e.g. the temporal or precedence constraints in scheduling

problems — can cause situations when in order to escape certain local optimum, the solution has to be significantly reorganized. This may not be possible using the local improvements achieved by relocations. This is where the backtracking mechanism steps in, allowing the solving process to bocktrack from such optimum.

4.3.5 Execution Strategies and Meta-optimization

As mentioned above, in order to solve a particular instance of a particular allocation problem, a number of parameters has to be provided. In particular, these parameters include:

- The initial solution σ consisting of a set of resources. Each resource has to have a particular local planning strategy in place
- The ordered list of tasks T of corresponding tasks to be allocated
- The particular coordination semantic to be used within the abstract coordination process

The first two parameters are problem specific. The number and type of resources within the initial solution σ that takes part within the coordination process is a part of the problem definition. For a particular real-world problem the initial solution σ can be composed of heterogenous resources with different capabilities and constraints. In case of the VRPTW and its extensions, this corresponds to a fleet of vehicles, that can potentially differ in the travel duration estimates for various routes, their capacities, be subjected to cargo compatibility or organization constraints, impose specific driver oriented constraints e.g. drive shift constraints etc.

Also, for some problems the order in which the tasks are processed can play an important role. For example it may be advantageous to allocate the most constraining tasks first in an approach similar to the well-known *most-constrained-first* heuristic used for example to solve the bin-packing problem — i.e. packing the heaviest items first. Thus, by understanding the particular problem to be solved, an ordering or a set of orderings that are analytically sound can be identified, based on the attributes of the set of tasks to be allocated. In case of online or dynamic allocation problems the ordering of the tasks is inherent to the particular problem instance being solved.

Lastly, the abstract global coordination algorithm can be configured to provide for a number of alternative *coordination semantics* affecting the complexity and efficiency of the resulting allocation algorithm. We recognize four baseline settings of the algorithm as introduced previously in Section 4.3.3:

- the simplest *Algorithm-B* baseline setting with neither the Push, the ShakeDynamic or the ShakeFinal functions enabled and with the backtracking mechanism being disabled as well
- the *Algorithm-F* setting incorporating a final local search based on task relocations by enabling the ShakeFinal function
- the *Algorithm-D* setting employing a similar local search improvement phase dynamically throughout the solving process by enabling the ShakeDynamic and Push functions
- and finally the *Algorithm-BT* setting, being the full-fledged unbounded setting employing the dynamic improvements as well as the backtracking mechanism

Given a particular setting the algorithm can be further refined by setting a specific value for the *backtrackLimit* and *loopLimit* parameters where applicable.

In order to solve a particular problem instance, a number of particular algorithm instances can be instantiated with different complexity, different initial solutions, featuring possibly even different implementations of the local planning strategy used by the resources. Within the presented agent architecture, each algorithm instance is represented by a single Coordination Agent and the corresponding set of Resource Agents. Based on a particular application scenario, interesting opportunities may exist for tailoring a specific execution wrapper for that scenario based on executing multiple algorithm instances in order to solve a single problem instance using meta-optimization approach based on specific way of traversing the space of alternative algorithm configurations. For example, in some cases it may be advantageous to instantiate a higher number of fast-running algorithm instances using simple coordination semantics and/or simpler approaches to local planning and run them in parallel. Or, on the other hand, in some other cases it may be better to instantiate only a limited number of algorithm instances using more complex settings. Even more interestingly, for specific problems it may be possible to combine both above mentioned approaches, for example reusing the information from the simpler algorithm instances that have already finished to modify the configuration of the running or yet to be run complex algorithm instances. Yet another possible way of tackling especially big problem instances is to use multiple algorithm instances to solve alternative partitions of the problem in parallel and then recompose the overall solution from these partial solutions. Similar or different approaches can therefore be seamlessly embedded within the presented abstract agent architecture, representing in fact alternative forms of meta-optimization.

In general, therefore, the presented agent architecture and abstract algorithm represent a flexible framework ready to accommodate various task allocation problem. Due to the flexibility of the abstract algorithm and the underlying abstract agent architecture it can be further refined to fit particular application scenarios. Later, when discussing the VRPTW case, a particular parallel execution wrapper developed specifically for the VRPTW is presented. It is based on a specific way of traversing the algorithm configuration space with both simple and complex algorithm instances being run in parallel in order to solve a single problem instance. As the individual instances finish, the resulting solutions are used to modify the configuration of the running and yet to be run instances.

4.4 The VRPTW Case

The presented VRPTW algorithm is based on the abstract coordination algorithm described above. It represents an extensions of out previous works [51, 48, 47, 49]. As discussed within the previous section, the coordination process is abstract and problem independent. Its adaptation to a particular problem is achieved by supplying a corresponding implementation of the Resource Agents providing for accurate commitment cost estimates driving the coordination process. The local planning of individual Resource Agents is therefore completely separated from the abstract coordination process. This separation in inherent to the agent-based approach adopted by the presented algorithm but also to agent-based problem solving in general [11, 60].

Within this section we discuss in detail the adaptation of the abstract algorithm for the VRPTW case. The VRPTW consists of finding a set of routes for homogeneous vehicles serving a set of geographically scattered customers. For each customer a specific demand of goods to be delivered and a specific time-window in which the service has to commence is given. The primary optimization criterium of the VRPTW is the minimization of the size of the fleet serving all the customers in a feasible fashion.

For the VRPTW, the tasks being considered correspond to the deliveries to individual customers being allocated to a set of vehicles representing the resources within the task allocation problem formulation. We denote the resource agents in this case as the Vehicle Agents[1]. Within this section we discuss the possible strategies concerning the Vehicle Agents' local planning and the way they influence the resulting VRPTW algorithm. In particular, an efficient way in which the problem specific constraints can be efficiently tackled within the vehicle's local planning is presented. Also, alternative commitment cost estimate criteria based on state-of-the-art insertion heuristics for the problem are introduced. A VRPTW specific ejection cost estimation mechanism is introduced as well. Lastly we present a specific parallel execution wrapper for the VRPTW algorithm based on the parallelization features of the presented agent architecture providing for efficient means of search diversification and intensification within the underlying search process.

4.4.1 Vehicle Agents

As mentioned, the resources in case of the VRPTW correspond to the vehicles serving individual routes as outlined by Figure 4.12. We refer to them as the Vehicle Agents. As part of the solving process the individual Vehicle Agents are required by the Coordination Agent to estimate (i) the additional cost incurred by adding an additional customer to their route and (ii) the savings resulting from removing a particular customer from the route. Given a resource and a task to be allocated (or removed) — i.e. a vehicle v being part of the emerging solution σ and a customer c to be served (or already being served by v) — these operations are denoted within the abstract coordination process as the

$$\begin{aligned} &v.costCommit(c) \\ &v.costDecommit(c) \end{aligned} \tag{4.8}$$

function. The actual commitment and decommitment are then represented by the

$$\begin{aligned} &v.commit(c) \\ &v.decommit(c) \end{aligned} \tag{4.9}$$

[1] Throughout the text we will freely interchange the terms Vehicle Agent, vehicle and route as fitting, always referring to the particular Vehicle Agent representing a single vehicle and responsible for serving the underlying route.

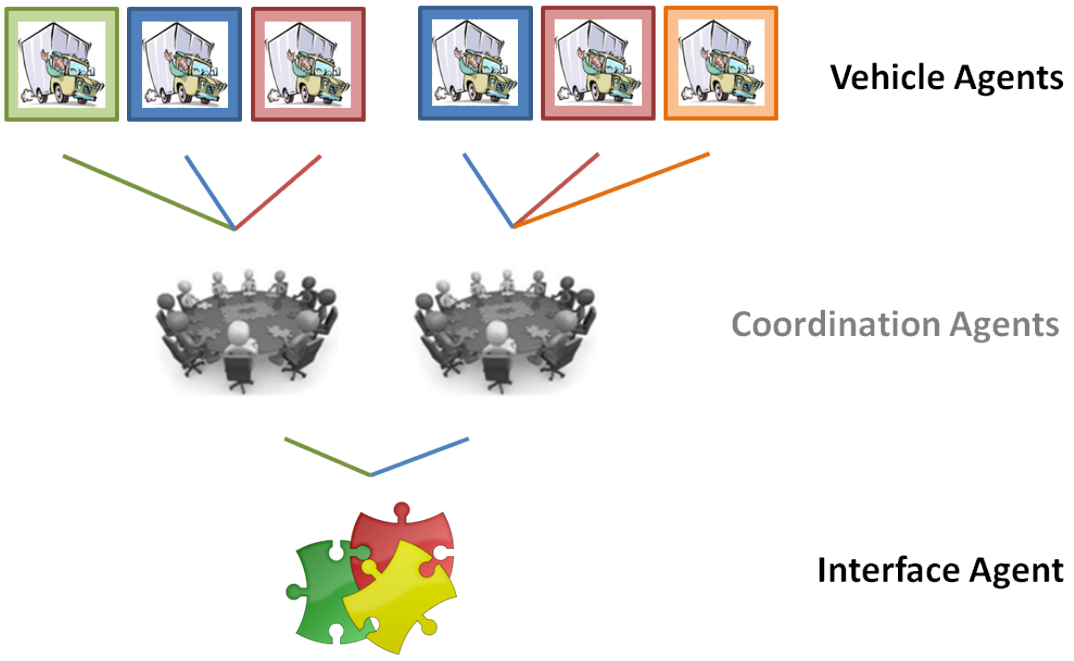


Figure 4.12: Agent Architecture for the VRPTW. The grey color corresponds to the abstract parts of the architecture. The Interface Agent corresponds to the particular application and execution scenario and is therefore considered problem specific as well.

functions.

As mentioned, each vehicle maintains its route independently taking into account the problem specific logic i.e. the efficient route construction (the underlying traveling salesman subproblem), the capacity constraints (the bin packing subproblem) and the time-window constraints. Based on the local processing, the vehicle then provides qualified estimates of the cost incurred by serving a particular customer.

Interestingly, it is not immediately clear how these estimates should be computed. Adding and removing customers to a particular vehicle's schedule affects the schedule in multiple ways. Rearranging the route in a way as to accommodate an additional customer results in an increase in the travel time of the vehicle. Therefore one of the possible aspects the insertion cost can reflect is the minimization of the corresponding increase in travel time caused by accommodating the additional customer. This corresponds to addressing the traveling salesman part of the problem. On the other hand, the cost estimate may also reflect the efficiency of the utilization of the vehicle's capacity. This corresponds to the underlying bin-packing problem. Lastly, quite complex conditions apply with respect to the way the time-window constraints affect the efficiency of the delivery process of the vehicle. For example, having the vehicle to arrive to a particular customer location significantly earlier than the service time-window for the customer begins is not efficient as it results in the vehicle having to wait idly until the beginning of the corresponding service time window before the actual service can start. Moreover it may be desirable to maintain an element of flexibility to the route schedule so that the chance of the route accommodating

future customers is increased. Consider a route with all customers having relatively wide time windows in which the customers are spaced along the route schedule in such a way that the vehicle arrives to each of the customers exactly at the moment of the beginning of the corresponding time-window. Such a route schedule is advantageous for the vehicle to make additional detours to some customers encountered later within the solving process. When a customer is encountered in spatial-temporal closeness of the route the additional time needed to serve the customer is quite likely to be found, as all of the subsequent visits can be shifted forward in the schedule. Now imagine adding a customer with a very short time-window at the end of such a route such that the vehicle can barely make it in time to serve the customer. Obviously, this reduces the flexibility within the route schedule and the temporal-spatial region which the vehicle can potentially cover decreases rapidly. Therefore, the local cost estimates may also aim at reflecting these non-trivial interactions of the individual customer services in the temporal domain.

In the following sections we discuss the possible implementations of the vehicles' local planning. In particular we present two specific local planning strategies based on the cheapest insertion principle, inspired by the known insertion heuristics for the VRPTW.

4.4.1.1 Local Planning Strategies

The local planning strategies presented within this study are all based on the cheapest insertion principle. Let c_j be the customer to be inserted and let $\langle c_0, c_1, ..c_m, c_{m+1}\rangle$ be the corresponding route served by the vehicle v, the c_0 and c_{m+1} corresponding to the depot. Let $costIns(c_j, v, i)$ represent the cost estimate of inserting the c_j between the customers c_{i-1} and c_i on the route. In case of the cheapest insertion principle, the cost estimate for the vehicle v committing to c_j is given by

$$costCommit(c_j, v) = \underset{i \in fi(1..m)}{\operatorname{argmin}} \ (costIns(c_j, v, i)) \tag{4.10}$$

where $fi(1..m)$ represents the set of all feasible insertion points on the route.

Obviously, not all possible insertions are feasible in terms of the problem specific constraints. The possible causes for an insertion of a customer c_j to the route v to be unfeasible are (i) the exceeding of the vehicle capacity, (ii) the inability of the corresponding vehicle to arrive to c_j before the end of the corresponding service time-window and (iii) the additional time needed to reach and serve the customer c_j (including the possible waiting in case the vehicle reaches the customer prior the start of the corresponding time-window) may render some of the subsequent customer visits late.

The first condition can be evaluated trivially. Given the capacity of the vehicle D, demands of the customers already on the route $d_{c_i}, i \in 1..m$ and the demand of the inserted customer d_{c_j} the capacity constraint is violated whenever $(\sum_1^m d_{c_i}) + d_{c_j} > D$. Given the current *cumulative demand* $Q(v) = \sum_1^m d_{c_i}$ of all customers served by the vehicle is stored and updated alongside each vehicle's plan, the condition $Q(v) + d_{c_j} \leq D$ can be trivially checked.

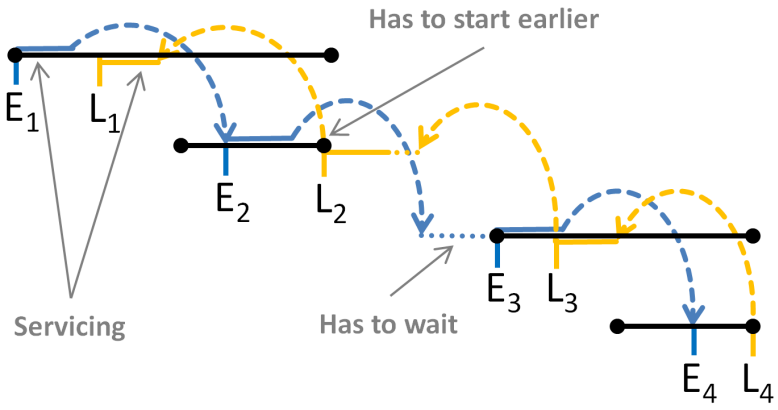


Figure 4.13: Earliest and latest service start times E_i and L_i. The black round-ended intervals specify the service time window at the four consecutively visited customers on a single route (visited in a top-down order). The E_i and L_i values are computed iteratively along the route. The blue dotted lines illustrate the forward computation of E_i values. The orange lines then the backwards computation of the L_i values. Whenever $E_k > L_k$ for some k, this means that the vehicle will arrive late to the customer k or one of the customers later within the route.

The verification of the other two conditions is a little more complicated. For each customer $c_i, i \in 1..m$ within the route $\langle c_0, c_1, ..c_m, c_{m+1}\rangle$ the (e_{c_i}, l_{c_i}) denote the service time window and s_{c_i} the service duration associated with the customer c_i. Given the known mutual travel times between the customers $t_{k,l} = t_{l,k}, k, l \in 1..N$ let (E_i, L_i) correspond to the *earliest* and *latest possible service start* at customer c_i within the considered route. These values can be computed recursively according to:

$$\begin{aligned} E_1 &= max\left(e_{c_1}, t_{c_0,c_1}\right) \\ E_i &= max\left(e_i, E_{c_{i-1}} + s_{c_{i-1}} + t_{c_{i-1},c_i}\right) \end{aligned} \tag{4.11}$$

and

$$\begin{aligned} L_{c_m} &= l_{c_m} \\ L_i &= min\left(l_{c_i}, L_{c_{i+1}} - t_{c_i,c_{i+1}} - s_{c_i}\right) \end{aligned} \tag{4.12}$$

Figure 4.13 illustrates the concept. As shown in [16], the time window constraints along the route are satisfied when $E_i \leq L_i$ for all $i \in 1..m$.

Given an insertion index i, let (E_j^i, L_j^i) represent the *earliest possible* and *latest possible service start* at c_j when inserted at index i. The E_j^i and L_j^i values can be computed according to Equations 4.11 and 4.12 as

$$E_j^i = max\left(e_{c_j}, E_{c_{i-1}} + s_{c_{i-1}} + t_{c_{i-1},c_j}\right) \tag{4.13}$$

and

$$L_j^i = min\left(l_{c_j}, L_{c_i} - t_{c_i,c_j} - s_{c_j}\right). \tag{4.14}$$

Whenever $E_j^i > L_j^i$ for the insertion index i the insertion of the task c_j at this position within the route $\langle c_0, c_1, ..c_m, c_{m+1}\rangle$ is infeasible causing a violation of the time-windows constraints due

to one of the above discussed two reasons. By storing the $E_i, L_i, i = 1..m$ along the route the temporal feasibility of a particular insertion can be efficiently evaluated in constant time. Let i be the identified best feasible insertion index in some agent v's route given by $\langle c_0, c_1, ..c_m, c_{m+1}\rangle$. The actual insertion of c_j to the route then requires the $E_k, k = i..m$ and $L_l, l = 1..i-1$ values to be updated along the route according to Equations 4.11 and 4.12 as well as the agent's cumulative demand $Q(v)$.

As mentioned above, we evaluated two respective implementations of the Vehicle Agent's interface functions based on two well known insertion heuristics for the VRPTW. The two heuristics and the corresponding local planning strategies are introduced within the next section.

4.4.1.2 Travel Time Savings Heuristic

The *travel time savings* heuristic is a well known heuristic with respect to the traveling salesman problem. Using the same example as in the previous section, the insertion cost corresponds to

$$costIns^{TT}(c_j, v, i) = t_{c_{i-1},c_j} + t_{c_j,c_i} - t_{c_{i-1},c_i}. \tag{4.15}$$

On the other hand, let c_k be a customer already within v's plan. The decommitment gain is computed accordingly as

$$gainDecommit^{TT}(c_k, v) = t_{c_{k-1},c_k} + t_{c_k,c_{k+1}} - t_{c_{k-1},c_{k+1}}. \tag{4.16}$$

The travel time savings heuristic leverages the spatial aspects of the problem, with a cost structure corresponding to the impacts of agent commitments or decommitments on the travel time of the agents. It has been shown [107, 72], however, that an insertion heuristic exploiting also the temporal relations within the route schedule given by the respective customers' time windows can yield significantly better results.

4.4.1.3 Slackness Savings Heuristic

The *slackness savings heuristic* introduces elements to the cost structure based on the interactions between time-windows of individual customers as they are spaced within the corresponding route schedule. It is an adaptation of the PDPTW heuristic presented by [72]. This heuristic represents a more elaborate an analytically sound alternative to the original Solomon's I1 heuristic, albeit at the cost of increased complexity.

Given $c_k, k \in 1..m$ corresponding to a customer on agent v's route $\langle c_0, c_1, ..c_m, c_{m+1}\rangle$ from previous examples, let $sl_k = L_k - E_k$ represent the *slack time* at customer c_k. An insertion of c_j requires the L_k and E_k values to be updated along the route possibly reducing the corresponding sl_k values. Reductions to slack times correspond to the constraining effects an insertion of a customer imposes on the rest of the agent's route. Let $sl^i_j = L^i_j - E^i_j$ represent the slack time at the inserted customer c_j after the insertion at position i. We denote $sl_j = l_j - e_j$ the slack time at c_j *prior* to the insertion. Given $sl'_k = L'_k - E'_k, k = 1..m$ being the updated slack times after the insertion of c_j to i-th position of the agent v's route, the overall reduction in route slackness is given by

$$SLR(c_j, v, i) = \left(\sum_{1}^{m} (sl'_k - sl_k)\right) + \left(sl^i_j - sl_j\right). \tag{4.17}$$

The $costIns^{SL}(c_j, v, i)$ for the slackness savings heuristic is based on both the spatial and the temporal aspects of the insertion with

$$costIns^{SL}(c_j, v, i) = \alpha * SLR(c_j, v, i) + \beta * costIns^{TT}(c_j, v, i). \tag{4.18}$$

where α and β, $\alpha+\beta = 1$ correspond to the respective weights of the two criteria being considered.

4.4.1.4 Backtracking for the VRPTW

In Section 4.3.2 we discussed the logic of the backtracking mechanism underlying the BACKTRACK function within the abstract coordination framework, featuring within the full fledged *Algorithm-BT* setting. The backtracking mechanism is an important part of the algorithm. It enables the algorithm to continue even in situation when all the remaining algorithm settings fail — e.g. when a task t is encountered within the solving process such that no resource can feasibly accommodate t, not even within the PUSH function. In such a case, the task t is allocated to σ at the expense of some other tasks being ejected from one of the resources in σ.

For solving the VRPTW efficiently the ability of the algorithm to do so is very important, especially due to the non-trivial interactions in the temporal domain between the customers being on the same route. As more customers are allocated to individual routes the schedules of these routes get denser. As this happens, it becomes increasingly difficult to modify the solution in any way. Therefore, even within the PUSH function, the chance of allocating the problematic customer is small as most task relocations are rendered infeasible due to the resulting time-window constraint violations. The only way to transform the solution in such a case, eventually permitting the allocation of the customer c would be to find a greater set of relocations to be performed simultaneously in order to move from one feasible solution to another. A similar approach is used for example in the previously outlined algorithm presented in [4] denoted as the *simulated trading*. The backtracking mechanism presents an alternative option. It allows the algorithm to transform the solution σ to a new feasible state accommodating the currently processed customer c albeit at the expense of ejecting some other customers. The ejected customers $c \in C_{ej}$ are then added to the beginning of the list of customers C in a LIFO approach. This means that prior to proceeding to the next customer $c' \in C$ in the traversed set od customers C all the ejected customers $c \in C_{ej}$ have to be feasibly allocated first. Such an approach makes sure that the chain of ejections relevant to the allocation of c is processed first prior to proceeding to the next customer $c' \in C$. This prevents the allocation of potentially problematic customers from sinking within the solving process.

Each resource thus identifies the most fitting set of customers to be ejected based on the *ejectionCost* cost estimate function, and the set C_{ej} with the globally lowest cost is selected. As already mentioned the process of identifying the particular fitting set C_{ej} can be very costly, i.e. exponentially so with respect to the *maxSize* parameter limiting the maximal size of the set C_{ej} of the ejected resources. The mechanism implemented for the VRPTW as part of the Vehicle Agents' local planning therefore uses a specific order in which individual possible ejections are enumerated as well as problem specific extensions enabling the efficient pruning of unfeasible search branches. The used ejection enumeration mechanism is an adaptation of the mechanism used in [75]. For each resource the possible ejections are traversed in a specific way corresponding to the lexicographic ordering. Let $\langle c_{e_1}, c_{e_2}, .., c_{e_k} \rangle, e_i \in 1..m$ denote an ejection of size k from a route $\{c_0, c_1, ..c_m, c_{m+1}\}$. The maximal ejection size is bounded by the *maxSize* parameter. Thus for example given $maxSize = 3$ the ejections are traversed in the following order: $\langle c_1 \rangle$, $\langle c_1, c_2 \rangle$, $\langle c_1, c_2, c_3 \rangle$, $\langle c_1, c_2, c_4 \rangle$ ... $\langle c_1, c_2, c_m \rangle$, $\langle c_1, c_3, c_4 \rangle$ ... $\langle c_1, c_{m-1}, c_m \rangle$, $\langle c_1, c_m \rangle$, $\langle c_2 \rangle$, $\langle c_2, c_3 \rangle$ etc.

The cost of an ejection C_{ej} is determined based on the number of failed allocations for individual customers as $\sum_{c \in C_{ej}} c.fails$ and the ejection with the minimal cost is chosen. This criterium was already described previously in Section 4.3.2 and is problem independent. However, for the VRPTW case, in case of equality for this criterium, the following additional criteria are used (in following hierarchical order): (i) minimization of the size of the ejection and (ii) minimization of the travel time increase for the corresponding route after the allocated task on whose behalf the ejection is being made has been inserted. Based on our computational tests

being out of scope of this study these criteria have proved to be the most efficient for the VRPTW case.

As mentioned, the number of evaluated ejections is theoretically very high. Moreover, testing the feasibility of allocating of some c to the thus modified route requires the E_i, L_i values (see Equations 4.11 and 4.12) to be recomputed along the route. Therefore in order to speed up the process several pruning strategies have been introduced enabling for avoiding the testing of feasibility for specific ejections and thus speeding up the process. Several abstract, problem independent such strategies were already discussed in detail previously in Section 4.3.2. The first strategy is trivial - by storing the contemporary best cost ejection all the ejections with costs higher than this ejection can be ignored. In case of the presented way of traversing the examined ejections, this means that whenever an ejection is found such that its cost is bigger than the contemporary best cost feasible ejection, the whole subtree corresponding to the extensions of the ejection in the traversing process can be pruned. The second already mentioned abstract strategy is based on two trivial observations: (i) if an ejection is a subset of another ejection that is unfeasible it is also unfeasible and (ii) if an ejection is a superset of a feasible ejection it is feasible and has higher cost than that ejection. In both of these cases thus the corresponding ejections can be pruned. The used traversal greatly simplifies the testing of the superset relation. When a feasible ejection is found, the whole subtree can be cut off as it cannot improve the cost. The subset relation can be efficiently tackled using hashing contributing to avoiding the actual testing of feasibility for subsets of unfeasible ejections. Lastly, after the ejection of $C_{ej} = \{c_{e_1}..c_{e_k}\}$ and the subsequent insertion of c_j the capacity constraint has to be satisfied, i.e. $\sum_{i\in 1..m}^{m} d_{c_i} - \sum_{i\in e_1..e_k} d_i + d_{c_j} \leq D$ or $Q(v) - Q(C_{ej}) + d_{c_j} \leq D$ where $Q(v)$ is the cumulative demand of all customers on the original route and $Q(C_{ej}) = \sum_{i\in e_1..e_k} d_i$ being the cumulative demand of the ejected customers. Using the above pruning rules thus significantly reduces the overhead inherent to the backtracking mechanism.

The complexity conclusions concerning the alternative vehicles' local planning implementations as well as the backtracking mechanism are introduced in the next section.

4.4.1.5 Complexity Analysis of the Local Planning

The local planning strategy of the Vehicle Agents is used (i) to maintain the vehicle's route and perform the local planning and (ii) based on this to provide cost estimates driving the global coordination process. In particular, throughout the coordination process the vehicles are requested to (i) estimate the cost of adding a customer c to the route, (ii) process the actual insertion of the customer to the route including the modifications of the relevant data structures i.e. the cumulative demand $Q(v)$ and the earliest/latest service start times E_i, L_i at individual customers along the route, (iii) estimate the gain resulting from removing some customer c already on the route and (iv) processing the removal itself correspondingly. When agent v cannot feasibly accommodate the customer c, we represent this as $v.costCommit(c) = \infty$. The above operations are represented by the following respective Vehicle Agent interface functions:

- $v.costCommit(c)$
- $v.commit(c)$
- $v.gainDeommit(c)$
- $v.decommit(c)$

Furthermore, the vehicles can be requested to identify the least costly ejection C_{ej} enabling the feasible allocation of c to the corresponding route denoted as

- *v.bestEjection*(*c*,*maxSize*)

The complexity of these functions is inherent to the particular local planning strategy used by the Vehicle Agents. Let m denote the number of customers within the agent v's route and N be the number of customers in the particular problem instance being solved.

Thus in case of the TTS strategy the complexity of the $costCommit^{TT}(c, v)$ is $O(m)$. For the SLS strategy the $costCommit^{SL}(c, v)$ requires the updated slack times to be computed for each insertion point resulting in a complexity of $O(m^2)$. As the total number of customers in agents' routes is bound by N, the worst case complexity of the *costCommit*(c) is $O(N)$ or $O(N^2)$ for the two respective local planning strategies. For the same reasons the complexity of the *gainDecommit*(c) function is O(1) for the TTS strategy and $O(N)$ for the SLS local planning strategy. The subsequent *commit*(c, v) operation of the agent v committing to the customer c requires the E_i and L_i values to be updated alongside the agent v's route for both the local planning strategies (see Equations 4.11 and 4.12), resulting in an $O(N)$ worst case complexity. For the same reasons the complexity of the *decommit*(c, v) function is $O(N)$ in the worst case as well for both the strategies. Also note that when evaluating the *costCommit*(c) for multiple agents, the overall complexity of such a process is still $O(N)$ as the number of customers in all agents' plans is still bound by N.

Considering the previously discussed complexity of the coordination mechanism (see Section 4.3.4), these conclusions provide bounds for the three basic algorithm settings. In case of the TTS strategy the basic *Algorithm-B* and *Algorithm-F* settings complexity is $O(N^2)$, while for the *Algorithm-D* setting the complexity is $O(N^3)$. In case the SLS strategy is used, the corresponding complexities increase to $O(N^3)$ and $O(N^4)$.

The complexity of identifying of the best feasible ejection within the *v.bestEjection*(*c*,*maxSize*) function potentially consists of evaluating all subsets of customers on the v's route up to a maximal size given by the *maxSize* parameter. Let the complexity of evaluating the feasibility of a particular ejection be $O(f(maxSize, N))$ for some f. The complexity is then roughly

$$O\left\{\left[\binom{N}{maxSize} + \binom{N}{maxSize - 1} + \cdots + \binom{N}{1}\right] \cdot f\right\}. \tag{4.19}$$

As

$$\binom{n}{k} \leq \left(\frac{n \cdot e}{k}\right)^k, \tag{4.20}$$

the complexity is also

$$O\left[f \cdot maxSize \cdot \left(\frac{N \cdot e}{maxSize}\right)^{maxSize}\right]. \tag{4.21}$$

being exponential in the *maxSize* parameter and polynomial with respect to N. The evaluation of the feasibility of an ejection then consists of removing the ejected customers, recomputing the E_j and L_j values along the route and invoking the *costCommit*(c) function. Based on the previous observations this can be done in

$$O\big(maxSize \cdot N + N\big) = O\big(maxSize \cdot N\big) \tag{4.22}$$

time for the TTS strategy and

$$O\big(maxSize \cdot N + N^2\big) = O\big(N^2\big) \tag{4.23}$$

time for the SLS strategy.

```
Input: Ordered list of customers C
Output: Solution σ — solution minimizing the

Procedure RouteConstruction(C)
 1:     vn := LowerBoundVN(C);
 2:     repeat
 3:       σ := fleet of vn identical empty vehicles;
 5:       σ' := Coordinate(σ, C);
 6:       if σ' is not complete
 7:         vn++;
 8:       else
 9:         return σ';
10:       end if
11:     end repeat
End
```

Figure 4.14: The abstract global coordination process

The substantial complexity of the backtracking process is however offset to some extent by limiting the size of the corresponding search space within the real search process. Firstly note, that for the experimental evaluation a $maxSize = 3$ setting is used. Also, for most ejections the feasibility of these ejections does not need to be evaluated due to the pruning ejection strategies introduced previously in Section 4.3.2 and revisited in the previous section as well.

Most importantly, as already mentioned, the relevant *Algorithm-BT* setting is only computationally bounded by imposing the *backtrackLimit* parameter within the Coordinate function. For that reason, the real world performance in terms of convergence is arguably of greater relevance than any complexity conclusions being drawn here. Actually, most of the successful traditional VRPTW algorithms are computationally unbounded as well and a similar point of view is adopted by the corresponding studies [14].

4.4.2 Fleet Minimization Objective

The algorithm instances based on the abstract coordination algorithm are instantiated over a set of resources within the initial solution σ. Within the coordination process then a particular assignment $A = \{[c_i, r_j], i = 1 \dots |C|, j \in 1 \dots |\sigma|\}$ of the tasks to be allocated $c \in C$ to the individual resources $r \in \sigma$ in the initial empty solution σ is being sought such that the social welfare is maximized. As outlined in Section 3.4, in the typical multi-agent problem definition [121] the social welfare is defined as the sum of the processing costs for the individual resource agents i.e. $Cost(A) = \sum_{r \in \sigma} cost_r(T_r)$. For the VRPTW, however, the primary optimization criterium is to find a *minimal fleet* of homogeneous vehicles such that all customers can be served in a feasible manner. Within such a definition thus the social welfare function is a constant function as any feasible solution maximizes the social welfare. The abstract coordination algorithm itself does not address such a criterium directly. Thus, within this section, we present two alternative ways in which this objective can be addressed, denoted as the *route construction mode* and the *route elimination mode*.

4.4.2.1 Route Construction Mode

A simple way of addressing the VRPTW optimization criterium is by (i) running the algorithm on a appropriately small initial solution σ and (ii) incrementing the fleet size and restarting it

```
Input: Graph G = (V, E)
Output: Size of the maximal found clique m

Procedure CliqueApproximation(G)
1:      m := 0;
2:      foreach v̄ ∈ V
3:        C := CliqueGreedy(G, v̄);
4:        m := max(m, |C|);
5:      end forach
6:      return m;
End

Procedure CliqueGreedy(G, v̄)
1:      C := {v̄};
2:      remove from V all v ∈ V such that (v, v̄) ∉ E;
3:      while C ≠ V
4:        arbitrarily select next v̄ such that v̄ = argmax_{v∈V\C} deg(v);
5:        C := C ∪ {v̄};
6:        remove from V all v ∈ V such that (v, v̄) ∉ E;
7:      end while
8:      return C;
End
```

Figure 4.15: The abstract global coordination process

until a feasible solution is found. The approach is outlined within the RouteConstruction function on Figure 4.14.

A sound setting for the initial fleet size corresponds to the lower bound number of vehicles for the problem instance being solved. As outlined in Section 2.1.2, such a number can be computed (i) based on the mutual incompatibilities of the tasks given their respective time windows and the travel times between the individual customers (the *time windows based lower bound* denoted as NV_{tw}) or alternatively (ii) based on the vehicle capacities and the demands of the individual customers (*capacity-based lower bound* denoted as NV_{cap}).

As already mentioned, the NV_{cap} can be computed as the optimal number of bins in the underlying multiple bin-packing problem. On the other hand, the identification of the NV_{tw} bound requires solving a *maxClique* problem within the graph capturing the customer mutual incompatibilities i.e. the graph $I = (C, E)$ with C being the set of all customers and $E = \{(i, j)|i$ is incompatible with $j)\}$. Obviously, the evaluation of both of these numbers is *NP-complete* [15]. Therefore, solving these sub-problems in an exact way is not a feasible approach and fitting approximations are used within the scope of this work.

For the NV_{cap} a trivial constant-time approximation given by

$$NV_{cap} = \left\lceil \sum_{i=1}^{N} d_i / D \right\rceil \tag{4.24}$$

is used. In case of the NV_{tw}, for the purposes of this study we use a $O(N^3)$ polynomial algorithm to approximate the maximal clique size inspired by [72] which is outlined by Figure 4.15 within the CliqueApproximation function. Thus for each vertex $\bar{v} \in V$ within the graph $G = (V, E)$ the algorithm finds a clique containing this particular vertex in a greedy manner (line 3, corresponding to the invocation of the CliqueGreedy function) and the size of the thus identified clique is returned. Within the CliqueGreedy function the source vertex $\bar{v}$ is added

```
Input: Set of customers C, time limit timeLimit
Output: Solution σ — solution minimizing the

Procedure RouteElimination(C)
 1:     σ := InitalSolution(C);
 2:     while timeLimit not exceeded
 3:       arbitrarily choose r ∈ σ;
 4:       σ := σ \ {r};
 5:       σ' := Coordinate(σ, r);
 6:       if σ' is not complete
 7:         restore σ to the original state;
 8:       else
 9:         σ := σ';
10:       end if
11:     end while
12:     return σ;
End
```

Figure 4.16: The abstract global coordination process

to set C representing the clique being constructed. Ale vertices not incident with $\bar{v}$ are then removed from the graph (line 2). Follows the *greedy* phase — in each iteration a next vertex $\bar{v}$ from the set V is selected such that its degree is maximal amongst all $v \in V \setminus C$ and added to the set C (lines 4 – 5). Then again all the vertices $v \in V$ not incident with $\bar{v}$ are removed from G. The process terminates when the set C covers all the remaining vertices in V. In such a case the set C represents a clique within the graph G, as for each $v \in C$, v is incident to all $w \in C, w \neq v$ which coincides with the definition of the clique. This follows from the fact that whenever a vertex is added to the set C, it satisfies this condition with respect to all the vertices already in C.

Therefore, referring back to the RouteConstruction function from Figure 4.14 the lower bound number of vehicles $VN_{LB} = \text{lowerBoundVN}(C)$ can be efficiently computed based upon the properties of the set of customers to be served C as the higher of the two above mentioned lower bound estimates. However, for a particular problem instance, the difference between the VN_{LB} and the number of vehicles within the feasible solution σ found by the RouteConstruction function (denoted $VN(\sigma)$) can be non-trivial. Firstly, the temporal and capacity constraints can interact in complex ways and can create a solution space that is fundamentally different then the space of solutions when considering these two aspects of the problem separately. Secondly, as the algorithm is only approximate, the solution being found within the corresponding run of the Coordinate function is not guaranteed to be optimal.

The difference between the lower bound number of vehicles VN_{LB} and number of vehicles $VN(\sigma)$ of the found feasible solution σ has a multiplicative effect on the resulting runtime of the RouteConstruction algorithm. For specific problem instances and especially for very large problems it can be quite substantial. Therefore, the correctness of the initial estimate of VN_{LB} is a very important factor affecting the performance of the resulting VRPTW algorithm. Within the Section 4.4.3 we present a specific parallel execution wrapper making use of the underlying agent architecture that aims at addressing this potential shortcoming.

```
Input: Set of customers C, backtrack limit backtrackLimit
Output: Solution σ — solution minimizing the

Procedure RouteElimination(C)
 1:     σ := InitalSolution(C);
 2:     for each limit ∈ 2, 4, 8 . . . backtrackLimit
 3:       for each route r ∈ σ processed in arbitrary order
 4:         σ := σ \ {r};
 5:         Shuffle(r);
 6:         σ′ := Coordinate(σ, r, limit);
 7:         if σ′ is not complete
 8:           restore σ to the original state;
 9:         else
10:           σ := σ′;
11:           restart the inner for each loop;
12:         end if
13:       end for each
14:     end for each
15:     return σ;
End
```

Figure 4.17: The abstract global coordination process

4.4.2.2 Route Elimination Mode

An alternative way of addressing the VRPTW optimization criterium is by (i) finding an arbitrary feasible complete solution to the problem and (ii) eliminating some of the routes by removing them from the solution and allocating the corresponding customers within the remaining routes using the Coordinate function. The process is outlined on figure Figure 4.16. The process thus begins with obtaining an arbitrary complete initial solution σ (line 1) corresponding to the InitialSolution function. The simplest setting for the initial solution corresponds to a baseline solution with each customer being served by a separate vehicle. An arbitrary route is then chosen to be eliminated and is removed from σ (lines 3 – 4). Then an effort is made to allocate the corresponding set of customers $c \in r$ within the remaining routes $r' \in \sigma$ (line 5). In case the process is successful i.e. the resulting solution σ' is complete, the process continues with the resulting solution σ' (line 8). In the opposite case the original solution σ is restored. The process is repeated until a given time limit is exceeded (line 2). For the sake of clarity we present a simplified version of the process in which the same route can be selected for elimination several times even when the solution σ hasn't changed in the meantime. Therefore the process can continue even when all the possible eliminations have failed. The relevant amendment of the process is, however, trivial.

The route elimination mode addresses the minimization of the size in a different way than the route construction mode. An important difference between the two is, that in case of the route elimination mode, the solution σ is feasible during the entire solving process being improved over time. On the other hand, in case of the route construction mode, the feasible solution is available only after the process has terminated.

4.4.2.3 Improved Route Elimination Mode

The route elimination algorithm outlined on Figure 4.16 is very simple. The experimental evaluation of the resulting VRPTW algorithm revealed that superior results can be achieved by

modifying it slightly. The modified improved route elimination algorithm is outlined by Figure 4.17.

In the improved version, an increasing sequence of values for the *limit* parameter is traversed (line 2). For each such value all of the routes in the intermediate solution σ are traversed in an arbitrary order (line 3). For each such route r an effort is made to eliminate the route by allocating the corresponding customers $c \in r$ to the remaining routes by invoking the COORDINATE($\sigma, r, limit$) function (line 6). The passed *limit* parameter corresponds to the *backtracksLimit* setting used by that particular invocation of the COORDINATE function. In case the elimination is successful for some route r the corresponding new solution σ' is adopted (line 10) and the inner algorithm loop is restarted (line 11). Additionally, for each invocation of the COORDINATE function the list of customers $c \in r$ corresponding to the eliminated route is shuffled thus randomizing the order in which the customers are processed within the COORDINATE function.

The outer loop thus serves to identify and eliminate first the routes that prove to be easier to eliminate i.e. that can be eliminated using a lower setting for the *limit* parameter. The experimental results prove that this is a sound heuristic with respect to the selection of the particular route to be eliminated in each step of the algorithm. Furthermore, by randomizing the order in which customers are processed for each invocation of the COORDINATE function the underlying search process is further diversified. Consider two subsequent invocations of the COORDINATE function on a single route r. The first invocation was obviously not successful, otherwise r would be eliminated and the second invocation would not occur. In between the two invocations, the route r may have changed due to some other route being successfully eliminated — resulting in the same value of the *limit* parameter being used — or the second invocation may be due to entering the next iteration corresponding to the next processed *limit* parameter value. In the latter case, considering that the rest of the solution σ was not modified as well the two subsequent invocations of the COORDINATE function would result in exactly the same search process being repeated until the number of backtracks reached the *limit* parameter value from the first run. On the other hand, by randomizing the order in which the customers are processed this redundancy is prevented and the search process is thus further diversified.

4.4.3 Execution Wrappers and Parallelization

In the previous sections we mentioned several times already that the abstract algorithm can be parameterized in multiple ways. Based on the particular parametrization, the resulting algorithm instances may differ considerably in the complexity and efficiency of the resulting algorithm. We also mentioned that the underlying agent architecture allows for executing multiple algorithm instances in parallel in an approach similar to the portfolio-based planners known from the planning literature [114]. In this section we discuss how these features of the algorithm can be exploited in order to improve the efficiency of the resulting VRPTW algorithm.

The concepts presented in this section were published as part of [46] and [48] and are based on our earlier works — [51] in particular. In this work, we analyzed the sensitivity of the *construction mode* algorithm in its various settings to the order in which the individual customers are processed within the COORDINATE function. In particular, we considered the *Algorithm-B*, *Algorithm-F* and *Algorithm-D* settings. We further considered the set of following analytically sound customer orderings:

- **Highest Demand First** (HDF): Customers are ordered decreasingly by the volume of their demands, according to the well known most-constrained-first greedy allocation principle applied to the underlying multiple bin packing problem.
- **Tightest Time Window First** (TTF): Customers are ordered increasingly by the dura-

tion of their time windows following the most-constrained-first approach, this time based on the time windows of the individual Customers.

- **Earliest Start First** (EF): Customers are ordered increasingly by the beginning time of the corresponding service time window. This setting (and the following 3 settings as well) causes naturally competing Customers to be allocated in close succession.
- **Earliest Expiry First** (EEF): Customers are ordered increasingly by the ending time of the corresponding service time window.
- **Latest Start First** (LF): Customers are ordered decreasingly by the beginning time of their time window.
- **Latest Expiry First** (LEF): Customers are ordered decreasingly by the ending time of their time window.
- **Most Distant First** (MDF): Customers are ordered decreasingly by their based on the distance of individual customers from the depot
- **Random** (RND): Baseline random ordering of customers.

The experimental assessment was based on a wide range of problem instances taken from the benchmarks known from the operations research literature [33, 107]. These benchmarks provide multiple diverse types of problem instances. Firstly, alternative ways in which the customer locations are placed on the plane are provided e.g. in clusters, in a randomized fashion or using a mix of both. Secondly, the problem instances additionally differ in the demands of individual customers and the respective time-windows constraints with some instances providing for a shorter scheduling horizons with routes of around 10 customers while other admitting a longer scheduling horizon with routes of around 50 customers per single route. The relevant experimental results are summarized later within the Chapter 5 in Section 5.2.4.

The experiments proved that in general the algorithm in all the *Algorithm-B*, *Algorithm-F* and *Algorithm-D* settings is sensitive to the used customer ordering with the sensitivity being most pronounced in the simplest *Algorithm-B* setting and gradually less so in the more complex *Algorithm-F* and *Algorithm-D* settings. The results further confirmed that the analytically sound orderings are consistently better than the random ordering. However, quite surprisingly none of the orderings proved dominant over the full set of evaluated problem instances. To the contrary each of the orderings performed well on different subset of the instances. This finding is all the more interesting as we also found that given a particular ordering, there is a correlation between the performance of the three evaluated algorithm settings i.e. the orderings providing for the best performance in the simple *Algorithm-B* and *Algorithm-F* settings for a particular problem instance typically worked well also for the *Algorithm-D* setting permitting it to yield the best results overall. These results suggest that the problem instances may differ in their nature favoring particular customer orderings. For example, given a problem instance accentuating the bin-packing subproblem i.e. with highly varied customer demands the HDF ordering may prove efficient. On the other hand, an instance where the demands of customers are close to being uniform but with highly varied time-windows for specific deliveries may respond better to the TTF ordering. Based on these observations we further developed specific ordering crossover and perturbation operators permitting the introduction of new orderings based on the analytically sound orderings mentioned above. The subsequent experiments proved that these newly introduced orderings can in some cases outperform the analytically sound orderings.

```
Input: Set of customers C, algorithm settings Δ, customer orderings Ω
Output: Solution σ

Procedure SolveParallel(C, Δ, Ω)
 1:    σ_best := null;
 2:    foreach (s ∈ Δ)
 3:      foreach parallel (o ∈ Ω)
 4:        vn := (σ_best = null ? lowerBoundVN(C) : vn(σ_best) − 1)
 5:        repeat
 6:          σ_o := fleet of vn identical empty vehicles
 7:          C_o := order C using o;
 8:          σ_o := s.Coordinate(C_o,σ_o)
 9:          if σ_o is complete then
10:            if σ_best = null or vn(σ_o) < vn(σ_best) then
11:              σ_best := σ_o;
12:              Output(σ_o);
13:            end if
14:            vn := vn(σ_best) − 1;
15:          else
16:            if σ_best = null or vn(σ_o) + 1 < vn(σ_best) then
17:              vn++;
18:            else
19:              store σ_o for orderings pruning
20:              break repeat;
21:            end if
22:          end if
23:        end repeat
24:      end foreach parallel
25:      Prune(Ω);
26:    end foreach
27:    return σ_best;
End
```

Figure 4.18: The parallel execution wrapper of the *Algorithm-PW* algorithm setting

We further developed a specific parallel execution wrapper denoted as the *Algorithm-PW* setting. The *Algorithm-PW* setting extends the route construction mode algorithm by introducing a parallel execution wrapper while at the same time addressing some of the outlined shortcomings of the route construction algorithm. Thus, initially, a set of algorithm instances using the simpler *Algorithm-B* and *Algorithm-F* settings is executed in parallel over a diversified set of orderings obtained by using the above mentioned ordering operators. Based on the results of the already finished algorithm instances the most promising orderings are identified. Follows the parallel execution of the complex *Algorithm-D* settings over the identified promising subset of the customer orderings fitting for the particular problem instance being solved. We refer to the ordering selection process as the *pruning* of the used set of orderings Ω. By using a set of diversified orderings generated using the above mentioned ordering crossover and perturbation operators the parallel wrapper provides for an efficient diversification of the underlying search process. On the other hand, the orderings pruning presents a means of search intensification, intensifying the search in the particular areas within the search space corresponding to the identified promising customer orderings. The used ordering diversification operators and the ordering pruning strategies are described in detail later in Section 4.4.3.1 and 4.4.3.2.

The parallel solving process is managed by the Interface Agent with each algorithm instance being represented by a single Coordination Agent. It is outlined within the SolveParallel function in Figure 4.18. The list of alternative algorithm settings being processed is represented as the parameter Δ and the diversified set of orderings is represented by the parameter Ω. In our case the set Δ corresponds to the set $\{$*Algorithm-B*, *Algorithm-F*, *Algorithm-D*$\}$ and the set Ω to a set of ordering initiated using the above mentioned analytically sound orderings and the two ordering diversification operators described later in Section 4.4.3.1. A single algorithm instance thus corresponds to the combination of a particular algorithm setting and a particular ordering — a tuple $[s, o], s \in \Delta, o \in \Omega$.

The process begins with resetting the temporarily best found solution σ_{best} (line 1). All the particular (increasingly complex) algorithm settings $s \in \Delta$ are processed sequentially (line 2). For each setting $s \in \Delta$ all the orderings $o \in \Omega$ are traversed (line 3) and the corresponding particular algorithm instances $[s, o_1] \dots [s, o_k], o_1 \dots o_k \in \Omega$ are executed in parallel (lines 6 – 8). The parallel wrapper uses a similar approach to addressing the VRPTW fleet minimization criterium as the route construction mode discussed above in Section 4.4.2.1. However, in an effort to decrease the number of restarts before a feasible solution is found, the number of vehicles within the initial empty solution σ_o is selected to always target a new best found solution based on the contemporary best found solution σ_{best} i.e. $vn := vn(\sigma_{best}) - 1$ (line 4). In case none of the algorithm instances has yet returned a complete solution and therefore $\sigma_{best} = null$ the lower bound fleet size is used instead with the number of vehicles computed using the LowerBoundVN function outlined previously in Section 4.4.2.1 (line 4).

The execution of a single particular algorithm instance $[s, o], s \in \Delta, o \in \Omega$ corresponding to the setting s, ordering o and the initial empty solution of vn vehicles consists of (i) instantiating the initial solution σ_o consisting of vn empty vehicles (line 6), (ii) reordering the set C using the ordering o (line 7) and (iii) running the s.Coordinate(C_o, σ_o) coordination process corresponding to the currently processed algorithm setting s (line 8). The coordination process is successful when the resulting solution σ_o is complete i.e. when all customers $c \in C_o$ have been successfully allocated to σ_o (line 9). In such a case, the solution is compared to the contemporary best known solution σ_{best} (line 10). If at that moment the σ_o represents a new best known solution i.e. $vn(\sigma_o) < vn(\sigma_{best})$ the σ_{best} solution is updated (line 11). In that case, it is also returned to the Interface Agent as the new best found solution within the Output function (line 12). This results in an improving sequence of results being returned over time. In the opposite case i.e. $vn(\sigma_o) \geq vn(\sigma_{best})$ the process is restarted with the vn parameter set to target a new best solution (line 14). Note that the Output function is just a placeholder to illustrate the way of extracting an improving sequence of results from the parallel computation and is not further specified.

On the other hand, if the solution $\sigma_o = s$.Coordinate(C_o, σ_o) is not complete, the coordination process for the current algorithm instance $[s, o]$ has failed. In such a case the process can be either restarted with an increased number of vehicles within the initial fleet (line 17) or terminated (line 20). The latter occurs whenever a complete solution σ_{best} has already been found such that $vn(\sigma_{best}) \leq vn(\sigma_o) + 1$. In such a case, increasing the fleet size and restarting the process is not feasible as it cannot improve on the already known solution σ_{best}. On the other hand, decreasing the fleet size and restarting the process is likewise non feasible, as it is unlikely that the coordination process would succeed using a smaller fleet where it has failed with the bigger fleet. The resulting incomplete solution is however stored and is analyzed when pruning the set of orderings Ω before proceeding to the next algorithm setting $s' \in \Delta$ (line 19).

After all algorithm instances $[s, o_1] \dots [s, o_k], o_1 \dots o_k \in \Omega$ for the currently processed setting $s \in \Delta$ have finished, the algorithm proceeds to the next setting $s' \in \Delta$. Prior to the execution of the corresponding algorithm instances the results of the previously finished instances are used

to select a subset $\Omega' \subset \Omega$ to participate within the next loop of the algorithm (line 25). Three alternative *ordering pruning strategies* are introduced in Section 4.4.3.2 that can be used to identify the set Ω'. After all algorithm settings are traversed and all algorithm instances have finished, the best solution σ_{best} is returned (line 26). Within the next sections we introduce the used ordering diversification operators as well as the relevant ordering pruning strategies.

4.4.3.1 Ordering Diversification

We developed two specific ordering diversification operators. As opposed to the well known *ordering crossover* and *mutation* operators used by the genetic ordering based algorithms e.g. [88], these two operators were specifically tailored to allow for traversing a neighborhood that is very close to the original analytically sound orderings and thus preserve the *nature* of these orderings. This effort was motivated by the previous findings [50] revealing that the analytically sound orderings significantly outperform randomly generated orderings.

The ordering operators are described below:

- ***k-perturb*(o)**: The *k-perturb*(o) operator is based on randomizing the order of sub-sequences of length k on the underlying set of customers ordered by the ordering $o \in \Omega$. Thus the order of the first $1 \dots k$ customers is randomized, followed by randomizing the $k+1 \dots 2k$ sequence etc. With increasing k the newly constructed ordering o' is more distant to the original ordering o. It is an equivalent of a mutation operator common in genetic algorithms [88].
- ***k-mixin*(o_1, o_2)**: The *k-mixin*(o_1, o_2) operator uses similar approach, however, instead of randomizing the order of the tasks in the selected subsequences of the original ordering $o1$, it reorders them using the ordering o_2. With increasing k thus the resulting ordering is arguably closer to the ordering o_2 and more distant to the original ordering o_1. The *k-mixin* operator represents a simple crossover operator applied on a pair of orderings.

The presented operators were introduced to provide means of *diversification* within the search process by providing for a diversified set of customer orderings Ω. Based on traversing this set and running the algorithm instances corresponding to the simpler algorithm settings in parallel and using one of the ordering pruning strategies described below, it is possible to identify a subset $\Omega' \subset \Omega$ to be used within the parallel wrapper for the next processed algorithm setting $s' \in \Delta$.

Also significantly, by executing the simple algorithm instances over a wide set of orderings enables the algorithm to find solutions of reasonable very quickly for most problems. The previous experiments [46] suggest that for majority of problem instances even the simple algorithm settings, when paired to a fitting customer ordering for the particular instance being solved, return very good results. Thus, when run in parallel such an approach provides for intriguing anytime characteristic of the algorithm. Lastly, in combination with the parallel wrapper logic this helps to reduce the complexity overhead resulting from numerous restarts affecting the previously discussed route construction algorithm by providing a better estimate for the initial fleet size to be used within the starting empty solutions.

4.4.3.2 Ordering Pruning

As mentioned above, after a particular algorithm setting $s_i \in \Delta$ has been processed, prior to the processing of the next $s_{i+1} \in \Delta$ the set of orderings $\Omega = \{o_1 \dots o_k\}$ is pruned based on the results of the previously finished algorithm instances $[s_i, o_1] \dots [s_i, o_k]$. Thus, within the next parallel wrapper loop only a selected subset $\Omega' \subseteq \Omega$ needs to be traversed using the more complex

algorithm settings. Such an approach follows our experimental evidence suggesting that different orderings may be favored by different particular problem instances.

During the orderings pruning phase thus the solutions $\sigma_1..\sigma_k$ obtained by the algorithm instances $[s_i, o_1] \dots [s_i, o_k]$ are analyzed in order to identify the good and the bad orderings. In order to evaluate the quality of the resulting solutions $\sigma_1..\sigma_k$ the following metric is used. Let σ_j be a particular solution to some VRPTW problem instance. Then the average route size of σ_j denoted $rs(\sigma_j)$ is defined as

$$rs(\sigma_j) = \frac{|\{c, c \textit{ is a customer allocated to } \sigma_j\}|}{|\{v, v \textit{ is a vehicle within } \sigma_j\}|} \tag{4.25}$$

corresponding to the average number of customers in a single route within the solution σ_j. The reason for choosing the rs metric is that it enables the means to compare even solutions that are not complete and have different fleet sizes.

Three alternative pruning strategies were introduced:

- **Basic Pruning (BP)**: the *Basic Pruning Strategy* (*BP*) is based on the assumption that the set of fitting orderings may differ significantly for each particular problem instance. Thus for a given sequence of settings to be processed within the parallel wrapper $\Delta = s_1 \dots s_k$ the numbers of orderings to keep $n_2 \dots n_k$ are given. Then the set of orderings Ω_i to be processed for the algorithm setting s_i is obtained by selecting the n_i best orderings from the set Ω_{i-1} from the previous step, using the average route size metric.

- **Minimal Covering Set Pruning (CSP)**: the *Minimal Covering Set Pruning Strategy* (*CSP*) is based on the opposite assumption — that the set of orderings and their interesting neighborhoods is similar for all the VRPTW problems in general. A weaker assumption, yet sound for many practical applications, is that for a particular real-world domain i.e. a particular distribution problem being solved the resulting problem instances will share some inherent common attributes making it possible to identify a particular set of fitting customer orderings to be used that will likely work well even for the future encountered problem instances. In order to identify such a set of orderings, a training set of problem instances is solved in an exhaustive manner by traversing the whole space of orderings and algorithm configurations $\Delta \times \Omega$. The results are evaluated and the minimal set of orderings Ω_{csp} is identified, such that for every problem instance one of the solutions corresponding to the best found solutions is obtained by an instance $[s, o]$ such that $s \in \Delta, o \in \Omega_{csp}$. For a particular real world problem, the training instances can for example be obtained from the historical data. For the purposes of this study we used a subset of benchmark problem instances (i.e. the 100 and 200 customer instances from the Solomon's and Homberger's benchmarks) in order to identify the set Ω_{csp}. The identified set was then used to solve the remaining benchmark problem instances with 400 – 1000 customers.

- **Combined Hybrid Pruning (CSP+BP)**: the *Combined Hybrid Pruning Strategy* denoted as *CSP+BP* combines the two above mentioned strategies into a single ordering pruning strategy. Thus the basic pruning strategy is used, however, only the orderings $o \notin \Omega_{csp}$ can be pruned, the orderings $o \in \Omega_{CSP}$ being excluded from the pruning process.

4.5 Overview of the Incremental Algorithm Versions

In this section we summarize the main concepts introduced in detail in this chapter and the resulting incremental algorithm versions evaluated within the next chapter. We organize this section in a similar way as the subsequent experimental evaluation.

4.5.1 Basic Algorithm

We refer as the basic algorithm to the algorithm settings employing neither the introduced parallel wrapper nor the backtracking mechanism. The three relevant basic algorithm settings differentiated by the particular coordination semantic being used are thus the (i) *Algorithm-B* setting with neither the SHAKE.. nor the PUSH functions employed, (ii) the extended *Algorithm-F* setting employing only the SHAKEFINAL function and finally (iii) the full-fledged *Algorithm-D* setting employing all the mentioned functions. For all three basic algorithm settings the simple route construction mode is considered with respect to addressing the fleet minimization criterium.

The configuration space for the basic algorithm settings is further enriched by the possibility to use one of the two introduced VRPTW local planning strategies — the *slackness savings strategy* (SLS) or the *travel-time savings strategy* (TTS).

4.5.2 Parallel Wrapper Algorithm

The introduced parallel wrapper represents a sophisticated strategy aimed at (i) providing means of search diversification within the underlying search process and (ii) addressing the inefficiencies of the simple route construction algorithm. It is based on combining the three basic algorithm settings within a parallel execution wrapper with the results of already finished algorithm instances being used to guide the search process thus in effect providing means of search intensification. The parallel wrapper algorithm is denoted as a specific *Algorithm-PW* setting. The algorithm configuration space for this setting further consists of the possibility to adopt one of the three introduced ordering pruning strategies: the *basic pruning* (BP) strategy, the *covering set pruning* (CSP) strategy and the hybrid *CSP+BP* strategy.

Initially, for the parallel wrapper algorithm, we do not consider the backtracking mechanism to be enabled. This follows also the historical reasons as this version preceded the introduction of the backtracking mechanism. On the other hand, as outlined later, the parallel wrapper was used later as well when designing the particular configuration to be used when evaluating the algorithm with backtracking.

4.5.3 Algorithm with Backtracking

The last introduced concept is the backtracking mechanism. The mechanism is abstract while a specific adaptation to the VRPTW case is presented as well applicable to a wider family of sequencing problems. The backtracking algorithm is denoted as a specific *Algorithm-BT* setting. For the backtracking algorithm, two fundamental versions were considered, corresponding to: (i) the parallel wrapper being used (which is in fact an extension of the route construction mode algorithm) and (ii) the route elimination mode being considered. These two versions are denoted as the *Algorithm-BT-RC* and the *Algorithm-BT-RE* specific settings. These two settings represent the final full-fledged settings of the algorithm combining all the previously introduced concepts.

Within the next section the incremental algorithm versions are experimentally assessed and the underlying solving processes are analyzed.

Chapter 5

Experimental Validation

Within this chapter we present the experimental assessment of the presented algorithm. The main focus is the relevant comparison of the presented algorithm to the traditional state-of-the-art centralized algorithms. A sound such comparison was missing from all but few previous similar studies leaving inadequately answered the question of the performance implications of adopting a agent-based approach to solve the VRPTW. A detailed analysis of the underlying solving process is presented as well, highlighting the outstanding attributes in a series of additional experiments.

The experimental results are presented in three separate sections dedicated to the increasingly complex configurations of the algorithm:

- In Section 5.2 the simpler *Algorithm-B, Algorithm-F* and *Algorithm-D* algorithm settings are analyzed.
- In Section 5.3 the *Algorithm-PW* algorithm setting featuring the introduced parallel execution wrapper is evaluated.
- In Section 5.4 the *Algorithm-BT-RC* and the *Algorithm-BT-RE* algorithm settings with the backtracking mechanism being enabled are analyzed.
- Lastly, in Section 5.5 a the key experimental results are summarized in a consistent way.

5.1 Experimental Settings

The experimental evaluation presented below is based on the two datasets known from operations research literature also outlined previously in Section 2.1.3. The original Solomon's benchmark set [107] has been consistently used to evaluate the performance of various VRPTW algorithms for over 25 years. It consists of 56 problems with 100 customers each. The Homberger's set of benchmark problems [39] is an extension of the Solomon's benchmark to problems with 200 – 1000 customers each, otherwise sharing the core characteristics with the Solomon's problems.

There are 6 instance types provided — the R1, R2, RC1, RC2, C1, and C2 type, each with a slightly different topology and time windows properties. For C1 and C2 types the customer locations are grouped in clusters, unlike the R1 and R2 types where the customers are randomly placed. The RC1 and RC2 instance types combine the previous two types with a mix of both random and clustered locations. The C1, R1 and RC1 also differ from C2, R2 and RC2 in terms of the scheduling horizon, the former having a shorter horizon resulting in routes of about 10 customers on the average, the latter having a longer horizon providing for routes of around 50 customers. The two benchmarks thus provide for a total of 356 problem instances. The alternative instance types provide variations in the way the customers are placed on the plane as well as the particular distribution of customer demands and time-windows for individual customers.

As outlined previously in Section 2.1.3 the quality of the solutions is evaluated using an aggregated value denoted as *cumulative number of vehicles* (CNV), given by:

$$CNV(\Sigma) = \sum_{\sigma \in \Sigma} vn(\sigma) \tag{5.1}$$

Where Σ is the set of problem solutions over the corresponding set of problem instances. The results are presented as the absolute and the relative error with respect to the state-of-the-art algorithm on Nagata [76]. This algorithm, being based on the ejection-pools principle paired with a genetic algorithm, represents the state-of-the-art with respect to the primary optimization criterium of the VRPTW. Therefore it provides a relevant benchmark for comparison in terms of efficiency of the presented agent-based algorithm. Apart from few singular cases (refer to [81] for more details) the solutions provided by this algorithm correspond to the best known solutions for the considered problems in terms of the primary VRPTW optimization criterium.

The experimental validation is based on a prototype implementation of the algorithm using the Java programming language. The prototype differs from the typically considered agent-based routing systems in that it does not consider the communication latencies associated within the agent-to-agent messaging, relying instead on a multi-threaded environment executed within a single Java Runtime Environment. While such an approach does not correspond to the distributed execution model typically attributed to the agent-based systems it allows us to concentrate on the underlying algorithm per-se and its performance. In particular, the experiments serve to outline the implications of separating the local planning of the agents from the global coordination on the performance of the resulting algorithm, being the core outstanding aspect associated to multi-agent planning.

5.2 Basic Algorithm Settings Evaluation

In this section we examine the performance of the three simpler algorithm setting — the *Algorithm-B, Algorithm-F* and *Algorithm-D* settings introduced in Section 4.3.3:

- *Algorithm-B* — the simplest baseline setting with neither the PUSH, the SHAKEDYNAMIC or the SHAKEFINAL functions enabled
- *Algorithm-F* — slightly more complex settings with the SHAKEFINAL function being enabled
- *Algorithm-D* — the most complex of the three considered settings with all the SHAKEFINAL, SHAKEDYNAMIC and PUSH functions enabled

In all three cases the backtracking mechanism is disabled.

Within the overall quality analysis we consider a simple execution wrapper that is a trivial extension of the simple route construction mode. The construction mode algorithm is run three times for three different customer orderings and the best result is considered. This approach is reflected also when evaluating the runtime and convergence attributes. The three used orderings mentioned previously in Section 4.4.3 are:

- **Highest Demand First** (HDF): Customers are ordered decreasingly by the volume of their demands.
- **Tightest Time Window First** (TTF): Customers are ordered increasingly by the duration of their time windows.
- **Earliest Start First** (EF): Customers are ordered increasingly by the beginning time of their time windows.

The overall results further consider the *RelocateAll* trading method to be considered within the SHAKEFINAL function of the *Algorithm-F* algorithm setting with the *loopLimit* parameter being set to 20. For the *Algorithm-D* setting, the same setting is considered for the SHAKEFINAL function. For the SHAKEDYNAMIC and PUSH functions the *RelocateAll* trading method is considered as well, however a $loopLimit = 5$ setting is used. When analyzing the alternative trading methods, for the ε-*RelocateWorst* and ε-*RelocateRandom* methods the $\varepsilon = 0.3$ setting is used. Lastly, where applicable, the experimental results correspond to the $\alpha = \beta = 0.5$ setting for the SLS local planning strategy. The choice of these particular values is discussed later in the text.

5.2.1 Solution Quality

The performance of the basic algorithm settings is illustrated by Table 5.1. The rows correspond to the results aggregated over alternative subsets of problem instances from the two benchmarks identified by the first column. Thus the first row identified as "All" corresponds to the results aggregated over all the 356 problem instances, the row identified as "100" then to the Solonon's 56 benchmark problems and the "200" – "1000" rows to the sets of problems of corresponding size from the Homberger's benchmark. The other six rows then correspond to the CNV aggregated over the individual instance types i.e. C1 ... RC2.

As mentioned, the results are presented as the absolute and the relative error with respect to CNV of the solutions found by the state-of-the-art algorithm of Nagata [76]. Thus within the second column the CNV for [76] over the corresponding set of problems is presented. The remaining columns then present the absolute end relative errors in terms of CNV for the three

Table 5.1: Performance of presented algorithm compared to the state-of-the-art centralized VRPTW algorithm [76]

Set Σ	CNV	Absolute (relative) errors in terms of CNV		
		Presented Algorithm		
	Nagata [76]	*Algorithm-B*	*Algorithm-F*	*Algorithm-D*
All	10695	1110, 10.4%	703, 6.6%	343, 3.2%
100	405	67, 16.5%	49, 12.1%	24, 5.9%
200	694	53, 7.6%	38, 5.5%	21, 3.0%
400	1380	120, 8.7%	73, 5.3%	38, 2.8%
600	2065	194, 9.4%	127, 6.2%	56, 2.7%
800	2734	278, 10.2%	180, 6.6%	89, 3.3%
1000	3417	398, 11.6%	236, 6.9%	115, 3.4%
C1	2914	470, 16.1%	333, 11.4%	151, 5.2%
C2	895	158, 17.7%	113, 12.6%	48, 5.4%
R1	2881	136, 5.4%	67, 2.3%	48,1.7%
R2	600	18, 3.0%	7, 1.2%	3, 0.5%
RC1	2801	218, 7.8%	120, 4.3%	65, 2.3%
RC2	603	110, 18.2%	63, 10.4%	28, 4.6%

evaluated algorithm settings. Thus, for example, the second row of the last column shows that over Solomon's benchmark the *Algorithm-D* setting was able to identify solutions that together require 24 more vehicles than the solutions identified by [76] to these problems, corresponding to a 5.9% relative error in terms of CNV.

In overall, the presented algorithm in the *Algorithm-D* algorithm setting was able to match the best known solutions for 48.6% of all the problems, requiring 343 excess vehicles across the 356 considered problem instances. This corresponds an overall relative error of 3.2%. The performance is consistent across all instance sizes. The difference in performance between the Solomon's and the extended Homberger's benchmark corresponds to the fact that some of the best results on the smaller dataset were achieved by algorithms that couldn't cope with the larger problems from the extended dataset. As will be discussed later, this is the for example the case as well with all the previously presented agent-based algorithms, none of which was assessed using the extended benchmark.

The performance of the algorithm is significantly better for the randomized R1, R2, RC1, RC2 instance types than in case of the clustered C1 and C2 types. This is arguably due to the fact that (i) for the R1 and R2 types both the compared and the agent-based algorithms are able to solve these problems very close to optimality and (ii) that the C1 and C2 instance types are particularly prone for the algorithm to get trapped in local extreme from which it cannot recover. Such an extreme corresponds to having the vehicles visit several clusters of customers which can easily happen when an unfitting ordering of the tasks is used (note that the spatial aspects of the problems are not addressed by neither of the three used orderings). This result in particular highlights the relative weakness of the basic algorithm i.e. the fact that once the solution gets dense especially in the temporal domain, the local search based on traversing the feasible relocation neighborhood can get stuck as all the possible relocations within the constrained solution are unfeasible. We will discuss this in detail in the next sections.

5.2.2 Coordination Semantics and Trading Methods

The results for the individual algorithm settings provide illustration of the significance of both the ShakeDynamic, Push and ShakeFinal functions enabled or disabled within the coordination process for the individual algorithm settings. With the *Algorithm-B* setting there is no possibility to recover from a potentially bad customer allocation taking place in the early stages of the coordination process as no task relocations are performed at all. Such a situation can for example correspond to some customers from different clusters of customers in terms of their position on the plane being assigned to a single vehicle. Such an allocation may appear as the best solution at the early stages of the coordination process. However, later in the solving process, it may turn out that an additional vehicles have to be dispatched to serve customers in the same clusters. The inefficiency stems from the fact that the concerned vehicle needs to spend excessive time traveling between the multiple clusters. As will be discussed later, this makes the *Algorithm-B* setting very sensitive to the used ordering of customers. Looking back at Table 5.1, the above mentioned situation can arguably be the reason why the error for the clustered C1, C2, RC1 and RC2 instances types is significantly greater than for the randomized R1 and R2 types for the *Algorithm-B* setting.

The *Algorithm-F* setting extends the *Algorithm-B* setting by allowing some exchanges of the tasks within and between the routes during the final stage of the coordination process. At this stage, however, as a result of previous allocations, the partial solution σ may be already tightly constrained. The schedules of the individual vehicles may be thus tight and most of the possible trading steps — relocation of customers from one vehicle to the other — can be rendered infeasible as the target vehicle is not able to feasibly accommodate the relocated customer. Thus the chance to recover from the situation like the one described above is correspondingly small.

With an average relative VN error of 3.2% the *Algorithm-D* setting significantly outperforms both the *Algorithm-B* and the *Algorithm-F* settings. This is due to the fact that the solution σ is dynamically improved throughout the coordination process. The improving relocations, being performed within the ShakeDynamic or Push functions, are performed on smaller and less tightly constrained solutions. This enables the algorithm to efficiently manipulate the solution especially in the early stages of the solving process. Therefore, situations like the previously mentioned one with a vehicle visiting multiple customer clusters can be efficiently avoided.

On the other hand, given an unfitting ordering of customers is used, the similar situation can only be discovered at the end of the solving process. For example a customer c can be encountered for which there is a vehicle v_{best} which is temporally and spatially very close to this customer and would be an ideal fit for the customer. However, due to the capacity or temporal reasons, this vehicle cannot accommodate the customer and neither do the remaining more distant vehicles $v \in \sigma, v \neq v_{best}$. In order for σ to accommodate the customer c, a *series* of moves would have to be performed. Firstly, the vehicle v_{best} has to drop some of its commitments removing specific customers from its route in order to free the required time and capacity to feasibly serve the customer c. Secondly, these removed customers have to be accommodated by the remaining vehicles $v \in \sigma$ or the vehicle v_{best} itself (by reorganizing the order in which the customers are served on its route).

The above situation cannot be addressed in neither of the three discussed algorithm settings. Within the next sections, we will discuss several alternative ways how to address this and thus improve the overall efficiency of the algorithm — the search diversification inherent to the parallel *Algorithm-PW* algorithm setting and the backtracking mechanism inherent to the *Algorithm-BT* setting. The situation is also nicely illustrated in the next section when comparing the three introduced trading methods.

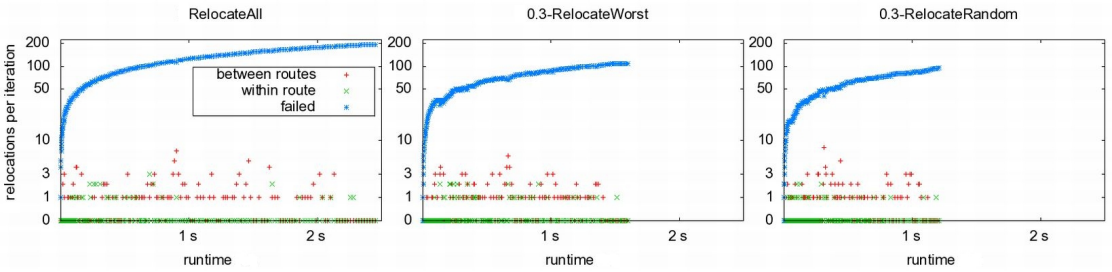


Figure 5.1: Improvement methods reallocation success

5.2.2.1 Trading Methods

In this section we analyze the respective performance of three trading methods introduced in Section 4.3.1.2 that can be used within the ShakeDynamic, Push and ShakeFinal functions. These methods are based on performing improving task relocations. The tree methods differ in the subset of customers selected in each vehicle's schedule for which the relocation is attempted. The three evaluated methods are:

- ***RelocateAll***: For each vehicle all of its customers are relocated.
- ***ε-RelocateRandom***: For each vehicle a random portion of its customers corresponding to $\varepsilon \in (0, 1)$ is relocated.
- ***ε-RelocateWorst***: For each vehicle the most costly customers (in terms of the $gainDecommit(c)$ value) are relocated, up to a $\varepsilon \in (0, 1)$ portion of its customers.

Note that for $\varepsilon = 1$ these methods therefore differ only in the order in which the customers are processed for each particular vehicle.

We evaluated the three methods when being applied for the ShakeDynamic and the Push functions, varying the ε parameter where applicable. Surprisingly, beginning with $\varepsilon = 0.3$, the quality of the resulting solution did not improve with increasing ε. On the other hand, the runtime did increase linearly. For the final improvement, we found the *ReallocateAll* method to achieve marginally better results. In overall, we found that the number of attempted reallocations does not have a strong positive influence on the quality of resulting solution.

Figure 5.1 illustrates the number of relocations performed within the individual calls of the ShakeDynamic function and the runtime in which they occurred within the progress of the coordination process. The 200 customer instances are considered with the SLS local planning strategy being used by the vehicles. Note also that the y axis is presented in logarithmic scale. The three point shapes correspond to: (i) relocations of customers between routes ($+$), (ii) relocations of customers within the routes ($\times$) and (iii) the failed relocations ($*$) i.e. when no possible improving relocation was identified for the relocated customer and thus σ was not modified. The rising curve shape for the number of failed relocations corresponds to the fact that the number of customers allocated to the solution σ increases throughout the coordination process resulting in more relocations being attempted within the ShakeDynamic function.

The results help to further highlight the inability of the algorithm in the three discussed algorithm settings to significantly transform the solution especially towards the end of the solving process. In the Figure 5.1 the number of successful relocations drops towards the end of the solving process for all three considered trading methods. Consider the two iterations of the algorithm in its *Algorithm-D* setting denoted i and $i + 1$. The iteration i concludes by the

Table 5.2: Relative error for the SLS and TTS strategy over 200 customer problem instances

	Relative error in terms of CNV	
Setting	*SLS*	*TTS*
Algorithm-B	7.6%	25.1%
Algorithm-F	5.5%	11.1%
Algorithm-D	3.0%	5.2%

invocation of the ShakeDynamic function. Let's assume this runs in an exhaustive loop until no more relocations are possible. The $i+1$ iteration begins by allocating the next processed customer $\bar{c}$ to the identified most fitting vehicle $\bar{v} \in \sigma$. Follows the invocation of the ShakeDynamic function. As the schedules of all $v \neq \bar{v}$ are unchanged, the only possible type of improving relocation is a relocation of a customer $c' \notin \bar{v}$ to $\bar{v}$. This may eventually trigger a series of other relocations, however, in general, the chance of significantly modifying the solution in the iteration $i+1$ by means of improving relocations is limited. Consider further that the iteration $i+1$ occurs later within the solving process. In such a case the schedules of most of the vehicles may by tight with limited opportunities to insert detours to accommodate additional customers. Also, for some vehicles, the capacity limit may be reached, preventing any additional customers to be added. It is easy to see, that in such a situation the chance of significantly transforming the solution by improving feasible relocations is even smaller. Given an unfitting customer ordering is used, this can easily result in sub-optimal outcome of the coordination process, corresponding to a local optimum being reached in the monotonous local search process inherent applying the improving relocation throughout the solution.

As mentioned in the previous section, we present several conceptually different ways of addressing this problem. The parallel *Algorithm-PW* algorithm setting counters this by diversifying the search by processing a number of alternative customer orderings. The underlying assumption is that using a fitting ordering the monotonous search process will converge to a good quality solution. Secondly, the backtracking mechanism inherent to the *Algorithm-BT* enables for alternative means of transforming the solution by performing customer ejections — a partially destructive transformation steps enabling the process to escape the local optima. Lastly, the SLS heuristic, by maximizing the slackness of the routes throughout the solving process, increases the chance that these routes will admit the allocation of customers encountered later within the solving process. We discuss this in detail within the next section.

5.2.3 Local Planning Strategies

As mentioned several times already, within the agent-based problem decomposition all the VRPTW problem specific logic is embedded within the local planning strategies used by the vehicles. The local planning strategies thus present a kind of a heuristic powering the abstract coordination search process. Two local planning strategies were evaluated. The TTS strategy leverages the spatial aspects of the problem by minimizing the impact of accommodating a customer has on the travel time of the vehicle serving the corresponding route. The SLS strategy further incorporates the temporal relations between the tasks by also minimizing the reductions to the slackness of target route. By favoring routes with customers with similarly long time-windows spaced along the route in a way such that the possibility to shift the schedule forward or backward is maximized, it also maximizes the possibility of future detours being accommodated by the route.

Table 5.3: Relative error for the SLS and TTS strategy and alternative customer orderings over 200 customer problem instances

	Relative error in terms of CNV			
Ordering	*Algorithm-B*		*Algorithm-D*	
	TTS	SLS	TTS	SLS
HDF	46.0%	27.9%	14.6%	9.5%
TTF	28.2%	18.5%	11.4%	8.7%
EF	25.7%	12.3%	8.5%	6.4%
EEF	27.4%	13.2%	8.9%	6.7%
LF	26.2%	13.1%	9.1%	6.5%
LEF	26.3%	13.0%	9.2%	6.6%
MDF	37.7%	19.3%	13.1%	8.9%
RAND	36.3%	23.0%	14.8%	9.7%
BEST	17.4%	7.1%	5.0%	2.9%

Table 5.2 lists relative errors of the two presented local planning strategies measured for the 200 customer benchmark set for the three discussed algorithm settings. The results correspond to $\alpha = \beta = 0.5$ parameters used for the SLS strategy, that has proved to be the most efficient in the preliminary computational tests. The results show that the SLS outperforms the traditional TTS heuristic in all three algorithm settings. The difference is most pronounced with the *Algorithm-B* setting while being less pronounced in *Algorithm-FI* and *Algorithm-DI* settings. This shows that both strategies present an efficient local planning heuristic for the VRPTW. It also served to highlight the fact that the temporal and spatial aspects of the problem coincide. Having a vehicle travel excessive distances results in less opportunities for the vehicle to accommodate additional customers within the scheduling horizon inherent to the particular problem instance being solved. The results show, that by addressing either of these aspects (or both as in the case of the SLS strategy) the solution can be efficiently improved.

5.2.4 Sensitivity to Customer Ordering

As discussed in the previous sections, the order in which customers are processed within the coordination process can significantly affect the success of the global coordination process. Table 5.3 lists the relative error of the simplest *Algorithm-B* and the most complex *Algorithm-D* settings of the algorithm for individual orderings. Apart from the three orderings mentioned in the beginning of this evaluation section the remaining orderings (as defined previously in the Section 4.4.3) are:

- **Highest Demand First** (HDF): Customers are ordered decreasingly by the volume of their demands.
- **Tightest Time Window First** (TTF): Customers are ordered increasingly by the duration of their time windows.
- **Earliest Start First** (EF): Customers are ordered increasingly by the beginning time of the corresponding service time window.
- **Earliest Expiry First** (EEF): Customers are ordered increasingly by the ending time of the corresponding service time window.

- **Latest Start First** (LF): Customers are ordered decreasingly by the beginning time of their time window.
- **Latest Expiry First** (LEF): Customers are ordered decreasingly by the ending time of their time window.
- **Most Distant First** (MDF): Customers are ordered decreasingly by their based on the distance of individual customers from the depot
- **Random** (RND): Baseline random ordering of customers.

The row denoted as the "BEST ordering" corresponds to the best results aggregated across all the orderings with the exception of the RAND ordering.

Several interesting observations emerge. Firstly, the analytically sound orderings clearly outperform the random RAND ordering in most but the HDF and MDF cases. Detailed analysis reveals, that the successful orderings address the temporal aspects of the problem. The HDF ordering addresses the underlying bin-packing problem as it corresponds to the known most-constrained-first heuristic applied to the individual demands. We argue that the relative lack of success of this ordering shows to highlight the fact that for the benchmark problems the bin-packing aspect of the VRPTW is not particularly pronounced. The vehicle capacity is 200 units in all cases with the demands of individual customers being 10, 20 or 30 units. The MDF ordering, on the other hands, leverages the spatial aspect of the problem addressing first the most distant customers. In this case, we argue that this ordering is actually not analytically very sound. The rest of the orderings address the temporal aspects of the problems. The TTF ordering corresponds to the most-constrained-first approach applied to the delivery time-window of individual customers and proves very effective. The remaining 4 orderings are very similar, being based on allocating the temporally close customers in close succession and also prove to be efficient. In general, these results thus highlight that for the evaluated benchmark, the temporal aspects of VRPTW are the dominant constraining factor.

Another interesting observation is the gap between the error listed in the BEST row and the results of any of the orderings. This results shows that none of the orderings is dominant over the whole range of problem instances. To the contrary, this result illustrates the fact that each ordering performs well on a different subset of problem instances. Surprisingly, by performing complementary experiments with ordering dominance we discovered, that given a particular problem instance and a particular ordering there is a correlation between the performance of the algorithm alternative settings. This suggests that individual problem instances differ in their nature favoring different orderings. The *Algorithm-PW* setting exploits this characteristics of the problem by introducing a specific search diversification and intensification meta-optimization strategy based on operating on a larger diversified set of customer orderings identifying the most fitting orderings for the particular problem instance being solved prior to executing the complex algorithm settings. The results presented later outline the success of such an approach.

Lastly, the results prove that the SLS local planning strategy is significantly less sensitive to the used customer ordering. In case of the baseline *Algorithm-B* setting the difference is more pronounced. This result proves once again the soundness of the SLS strategy. In the more complex *Algorithm-D* setting the difference is less marked. As mentioned also in the previous section, we argue that this is due to the fact that the spatial aspects of the problem (leveraged by the TTS strategy) and temporal aspects (leveraged by the SLS strategy) coincide and both present a sound heuristic for the local planning part of the problem. As such thus both the strategies allow the algorithm to efficiently improve the solution within ShakeDynamic and Push functions. These results also support the previous finding of [59]. Using a similar customer allocation strategy, the mentioned work shows how the improvements performed dynamically on

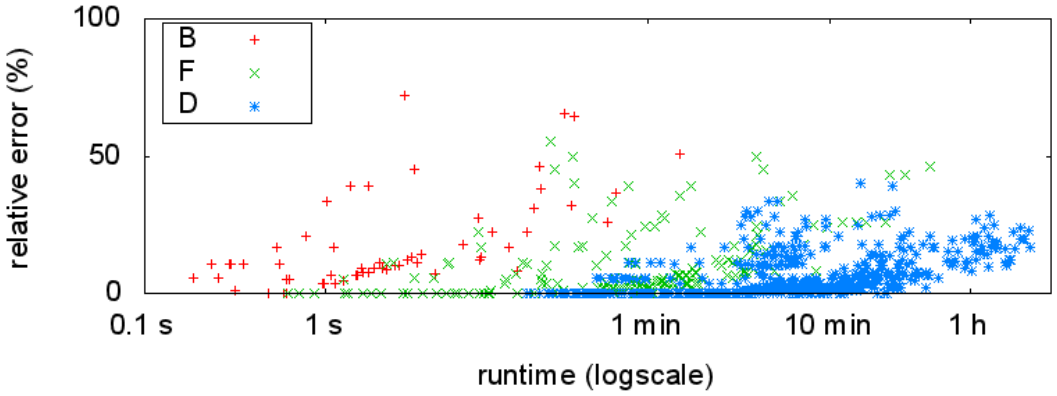


Figure 5.2: Results for individual algorithm settings for 1000 customer instances

the emerging solution can successfully offset the sensitivity of the resulting allocation algorithm to the order in which the customers are processed.

5.2.5 Runtime

Within this section we present some interesting observations concerning the runtime of the algorithm in the three discussed algorithm settings. However, we do not present a comparison to the centralized state-of-the-art algorithms here. The reason for this is that the discussed settings perform significantly worse than these algorithms. A relevant comparison is presented later for the two extended settings of the algorithm — the *Algorithm-PW* and *Algorithm-BT* which provide for a more competitive performance and where interesting conclusions apply.

The Figure 5.2 depicts the relative errors and runtimes of the individual algorithm instances executed while solving the problems from the 1000-customer subset of the Homberger's extended benchmark. There are three types of points depicted, corresponding to the three evaluated *Algorithm-B* (+), *Algorithm-F* (×) and *Algorithm-D* (∗) settings. Note that the results for all the three considered orderings (HDF, TTF, EF) are included. Note also that the x-axis uses a logarithmical scale. This chart serves to to illustrate several interesting aspects of the algorithm.

Firstly, the already discussed correlation between the increasing complexity of the three discussed algorithm settings and the increasing quality of the resulting solutions is further illustrated. The cluster of points inherent to the *Algorithm-B* is positioned highest within the chart with the relative error dropping for the clusters corresponding to the two increasingly complex algorithm settings. However, interestingly, the results also show that in many cases even the simpler settings produce good solutions matching even the best known solutions to the corresponding problems (i.e. with 0% relative error). Also interestingly, in each of the clusters, the longest running instances are also the ones returning the worst quality solutions as apparent from the *left-down* to *top-right* shape of the corresponding clusters. This is arguably due to the simple route construction (outlined in Section 4.4.2) mode being used to address the primary VRPTW optimization criterium. For this method the number of iterations within the RouteConstruction main loop is proportional to the absolute error in terms of the number of vehicles in the resulting solution σ. Thus the instances returning the worst solutions also require the highest number of iterations within the RouteConstruction function. This has a multiplicative effect

Table 5.4: Average and worst runtimes for the *Algorithm-D* setting

Size	*Algorithm-D* runtime	
	Average	Worst
200	3 s	13 s
400	22 s	3 min
600	2 min	19 min
800	6 min	47 min
1000	8 min	74 min

on the runtime of the particular algorithm instance.

This observation is further highlighted by Table 5.11 listing the average and worst runtimes for the *Algorithm-D* setting across the alternative problem sizes for the extended Homberger's benchmark. The results prove that the gap in terms of runtime between the worst and the average runtimes is dramatic. As mentioned above, the worst runtimes correspond to the algorithm instances where the number of vehicles of the found solution $vn(\sigma)$ is significantly higher than the initial fleet size estimated by the LowerBoundVN function resulting in multiple iterations within the RouteConstruction main loop. The effect is most pronounced given an unfitting ordering is used for the particular algorithm instance and is proportional also to the size of the instance.

We present these findings as they clearly illustrate the motivation behind the introduced extended *Algorithm-PW* setting. The setting profits from early success of any of the simple algorithm instances executed over a wide range of orderings with runtimes in a matter of seconds even for the most complex problem instances. The best thus found solution is returned as soon as it is found providing for an improved anytime characteristic of the algorithm. Furthermore, some of the complex instances using an unfitting ordering for the particular problem instance being solved are likely to not to be started at all as these orderings may may be pruned as part of the orderings pruning strategy being applied. Lastly, after the simple algorithm instances are processed, the more complex instances are instantiated with the size of the fleet targeting a new best known solution i.e. $vn := vn(\sigma_{best} - 1)$ in order to avoid the unnecessary invocations of the Coordinate function within the route construction mode algorithm.

In a complementary experiment we also evaluated an alternative way how to address the outlined problems. A modification to the restart strategy used within the route construction algorithm was introduced. For each iteration, instead of starting with a new empty solution σ with an incremented fleet size, the solution σ from the previous iteration is reused being extended by adding an additional empty vehicle. However, such an approach didn't prove successful in our computational testing and therefore is not presented as part of this thesis.

5.3 Parallel Wrapper Evaluation

In this section we present experimental evaluation of the extended *Algorithm-PW* setting introduced in Section 4.4.3. The *Algorithm-PW* setting builds on the experimental results for the basic *Algorithm-B*, *Algorithm-F* and *Algorithm-D* algorithm settings discussed within the previous section. Most notable, the results illustrate that:

- All the three algorithm settings are sensitive to the used customer ordering.
- Different orderings work well with different instances.
- There is a correlation between the performance of a particular ordering between the tree algorithm settings.
- There is a significant performance penalty inherent to addressing the VRPTW optimization criterium by using the simple route construction mode inherent to having to invoke the COORDINATE function several times within the simple route construction algorithm before a complete solution is found.

In order to solve a single VRPTW problem instance the *Algorithm-PW* setting is based on executing multiple algorithm instances within a specific parallel execution wrapper. The sequence of increasingly complex algorithm settings $\Delta = [s_1..s_n]$ is traversed. Thus a diversified set of customer orderings $\Omega = \{o_1..o_n\}$ is instantiated by applying the two presented ordering diversification operators to a canonical set of analytically sound orderings. Then, for the simplest s_1 setting being processed first all the algorithm instances $[s_1, o_1]..[s_1, o_n]$ are processed in parallel using the route construction mode from Section 4.4.2 to address the primary VRPTW optimization criterium. Based on the analysis of the solutions provided by these algorithm instances the set of orderings Ω is pruned prior to processing the next algorithm setting s_2. The same process is then iterated for the remaining algorithm settings $s_2..s_n$ with one significant modification — the number of vehicles in the initial empty solutions $\sigma_{[s_i,o_j]}, o_j \in \Omega$ are initialized to always target a new best known solution i.e. $vn(\sigma_{[s_i,o_j]}) = vn(\sigma_{best} - 1)$.

As mentioned, two alternative ordering diversification operators were introduced. These operators are the *k-mixin* and the *k-perturb* operators. Also, three alternative ordering pruning strategies were considered as well — the *basic* (*BP*) pruning strategy, the *minimal covering set* (*CSP*) pruning strategy and the hybrid *CSP+BP* strategy combining the two. For comparison purposes a baseline $\Delta \times \Omega$ pruning strategy was evaluated as well corresponding to traversing the whole *algorithm configuration space* $\Delta \times \Omega$ with no ordering pruning taking place.

The computational experiments were performed using the following settings. A 5 member set $\Delta = [B, F, D_1, D_2, D_3]$ was used. In this set the B corresponds to the *Algorithm-B* setting. The F corresponds to the *Algorithm-F* setting with the SHAKEFINAL function using the *RelocateAll* trading method with a *loopLimit* = 20 parameter setting. The same setting for the SHAKEFINAL function is used also by all the remaining D_1, D_2 and D_3 configurations within the set Δ. The D_1, D_2 and D_3 configurations correspond to the *Algorithm-D* setting. The D_1 configurations is using the *RelocateAll* method with *loopLimit* = 1 parameter setting for the SHAKEDYNAMIC and PUSH functions. The D_2 actually stands for three individual settings similar to $D1$ each using one of the three introduced trading methods for the SHAKEDYNAMIC and PUSH functions i.e. the *RelocateAll*, 0.3-*RelocateWorst*, 0.3-*RelocateRandom* with a *loopLimit* = 3 parameter setting. The D_3 then corresponds to a similar 3 member set as D_2, however with a *loopLimit* = 6 parameter setting for the SHAKEDYNAMIC and PUSH functions. Finally, the SLS local planning strategy is used for the vehicles in all the settings with $\alpha = \beta = 0.5$ values of the relevant parameters.

Table 5.5: Performance of the *Algorithm-PW* compared to the state-of-the-art centralized VRPTW algorithm [76]

Set	CNV	Absolute (relative) errors in terms of CNV				
			Algorithm-PW			
	Nagata [76]	*Algorithm-D*	*BP*	*CSP*	*BP+CSP*	$\Delta \times \Omega$
All	10695	343 (3.2%)	290 (2.7%)	258 (2.4%)	254 (2.4%)	–
100	405	24 (5.9%)	17 (4.2%)	16 (4.0%)	16 (4.0%)	–
200	694	21 (3.0%)	18 (2.6%)	12 (1.7%)	12 (1.7%)	–
400	1380	38 (2.8%)	35 (2.6%)	31 (2.2%)	29 (2.1%)	29 (2.1%)
600	2065	56 (2.7%)	51 (2.5%)	43 (2.0%)	43 (2.1%)	41 (2.0%)
800	2734	89 (3.3%)	76 (2.8%)	72 (2.6%)	71 (2.6%)	70 (2.6%)
1000	3417	115 (3.4%)	92 (2.7%)	84 (2.5%)	83 (2.4%)	82 (2.4%)
C1	2914	151 (5.2%)	129 (4.4%)	125 (4.3%)	123 (4.2%)	–
C2	895	48 (5.4%)	42 (4.7%)	39 (4.3%)	39 (4.3%)	–
R1	2881	48 (1.7%)	39 (1.4%)	27 (0.9%)	27 (0.9%)	–
R2	600	3 (0.5%)	1 (0.2%)	1 (0.2%)	1 (0.2%)	–
RC1	2801	65 (2.3%)	54 (1.9%)	45 (1.6%)	44 (1.6%)	–
RC2	603	28 (4.6%)	26 (4.3%)	21 (3.5%)	21 (3.5%)	–

The used set set Ω consists of the set of 5 canonical orderings i.e. {HDF, TTF, EF, LF, LEF}, their *3-perturb* and *6-perturb* mutations and by their *10-mixin* and *20-mixin* combinations, providing for a set Ω of 65 instance task orderings.

The pruning strategies were configured as follows. For the *BP* strategy with respect to the set $\Delta = [B, F, D_1, D_2, D_3]$ the numbers of orderings to keep for each configuration were set to $[65, 65, 20, 2, 2]$. The used Ω_{CSP} was computed by traversing the whole $\Delta \times \Omega$ configuration space on all of the Solomon's 100 and Homberger's 200 customer instances. Then the minimal covering set of orderings providing for all of the best found solutions was identified, resulting in the set Ω_{CSP} containing 10 orderings — the LEF canonical ordering attributing for the majority of best results, 4 orderings based on the *k-perturb* operator and 5 *k-mixin* based orderings.

5.3.1 Solution Quality

The results for the *Algorithm-PW* and its various configurations are presented by Table 5.5. The results are presented in a similar fashion as the results in the previous section dedicated to the basic algorithm settings. Thus the rows correspond to the results aggregated over alternative subsets of problem instances from the two used benchmarks, being identified by the first column. The row identified as "All" corresponds to the results aggregated over all the 356 problem instances, the row identified as "100" then to the Solomon's 56 benchmark problems and the "200" – "1000" rows to the sets of problems of corresponding size from the Homberger's benchmark. The second column shows the CNV for the given subset of problems corresponding to the best-known solutions to these problems. The rest of the columns then list the absolute and the relative error in terms of CNV for the alternative evaluated algorithm settings. The results are presented as absolute errors in terms of CNV, with the corresponding relative error being listed in brackets. For comparison, the results of the best performing *Algorithm-D* from the previous section are presented in the third column. The remaining four columns list the results for the three introduced ordering pruning strategies used within the *Algorithm-PW* algorithm setting, followed by the results for the baseline $\Delta \times \Omega$ setting listed within the last column.

The results illustrate the improvement to the solution quality achieved by the introduced search diversification and intensification strategies. In its full fledged *CSP+BP* setting the *Algorithm-PW* was able to improve on the solutions obtained by the previous *Algorithm-D* setting in 81 of the 356 problem instances (i.e. 23% of the cases). Thus, in overall, the *Algorithm-PW* achieved a CNV of 10949 over all the benchmark instances, corresponding to a 2.4% relative error in terms of the VRPTW primary optimization criterium with respect to the best known solutions to these problems. The algorithm was able to equal these best known solution in 64% of all the problem instances.

Within the next sections we analyze the improved algorithm from various points of view in an effort to provide insight into the underlying solving process, including the analysis of the relative success of the individual orderings, ordering diversification operators and the introduced pruning strategies. The implications of the introduced improvements to the runtime and convergence of the algorithm are discussed as well.

5.3.2 Orderings Diversification and Pruning

The experimental results presented within the Table 5.5 outline the respective success of the three alternative pruning strategies ant their comparison to the baseline $\Delta \times \Omega$ strategy without ordering pruning. As mentioned, the used set Ω corresponds to the 5 canonical orderings, extended by their *3-perturb* and *6-perturb* mutations and by their *10-mixin* and *20-mixin* combinations resulting in a total of 65 customer orderings. As also mentioned, the used Ω_{csp} consists of the 10 orderings identified as the minimal covering set of the winning orderings over the Solomon's 100 and Homberger's 200 customer instances, while 5 specific increasingly complex settings were used within the set Δ. The numbers of retained orderings within the *BP* pruning strategy for the 5 above mentioned settings were $[65, 65, 20, 2, 2]$.

5.3.2.1 Diversification Operators

As mentioned, the results indicate that the search diversification inherent to the *Algorithm-PW* algorithm setting is successful, enabling the algorithm to improve on the *Algorithm-D* setting in 23% of the cases. The key factor in this success is the search diversification achieved by using a wider set of customer orderings.

The most interesting result with respect to the two used ordering diversification operators is the fact, that 9 out of 10 orderings within the identified Ω_{csp} set i.e. the set of the orderings that together dominate all the remaining 55 orderings were in fact obtained by applying the introduced operators. The operators (and their used parametrization that was fine-tuned in complementary experiments) produce orderings, that are closely related to the original analytically sound orderings, always preserving the orderings of the blocks of tasks of a length k (the k prefix parameter), only changing the ordering within these blocks. Thus they operate on a relatively close neighborhoods of the original orderings, corresponding to the k parameter setting. Recall the results from ordering sensitivity analysis of the basic algorithm settings. The results showed, that the analytically sound orderings perform consistently better than the random ordering of the tasks. In the light of the previously mentioned results, this shows that the operators extend the orderings in a specific manner that has proved very efficient.

In case of the *k-mixin* operator the setting of the k parameter does not affect the success of the resulting ordering greatly, with the overall success roughly corresponding to the success of the two underlying orderings. However, with the *k-perturb* operator the lower k values dominate the higher values. These results suggest that (i) the *k-mixin* operator preserves the analytically derived soundness of the orderings irrespective of the k parameter settings providing for a flex-

ible search diversification operation while (ii) the *k-perturb* operator diverges from the feasible analytically sound orderings with increasing k.

In terms of overall success the *k-mixin* operator slightly outperforms the *k-perturb* operator, however their simultaneous appearance in the identified set Ω_{csp} mentioned above suggests that both provide for an effective alternative means of search diversification, given a fitting k is used for the *k-perturb* operator.

5.3.2.2 Pruning Strategies

The results presented in Table 5.5 outline the success of the individual pruning strategies. The results indicate several interesting facts. Consider the difference in the resulting complexity between the complete $\Delta \times \Omega$ setting and both of the presented pruning strategies. As the Ω_{csp} consists of 10 orderings, the complexity of the *CSP* and the *CSP+BP* strategies is approximately 6 times smaller. In the light of this observation both of these strategies prove to be a very efficient arriving in most cases to the exact same results but with considerable lower complexity.

For the *BP* strategy the complexity is even smaller, as the most complex settings are processed on only 2 orderings. Recall that the results presented for the previous basic *Algorithm-D* setting actually correspond to the best of three runs for the three analytically sound orderings. Thus, in fact, the used settings for the *Algorithm-PW* using the *BP* strategy present a significantly *less* complex configuration than the configuration underlying the results presented for the previous *Algorithm-D* setting with the average runtime being approximately 2 times smaller. In light of this observation, considering that the *BP* strategy actually strictly dominates the *Algorithm-D* setting, it can be considered very successful as well.

On the other hand, the significant gap between the *CSP* based strategies and the *BP* strategy shows that the process of finding the most fitting ordering for a particular problem instance is not very successful. This highlights the fact that the relationship between relative success of two different algorithm configurations using the same instance task ordering is not straightforward. Thus the effort presented within the *BP* pruning strategy to introduce a rather fine-grained meta-optimization technique is not very successful.

The very good results achieved by the *CSP* based strategies suggests that the set of dominant orderings is quite consistent over the whole benchmark set. Furthermore, these results show that the strategy used for identifying this set e.g. by using a subset of problem instances with similar attributes as the expected future data, the two ordering diversification operators and the minimal covering set approach is very successful further supporting the soundness of the *CSP* pruning strategy.

5.3.3 Runtime and Convergence

The *Algorithm-PW* setting is based on executing multiple algorithm instances in an effort to diversify the search and improve the overall solution quality. This obviously affects the overall complexity of the algorithm. On the other hand, an attempt is made to reduce the increase in the overall complexity by the introduced ordering pruning strategies. Also, the parallel wrapper enables the *Algorithm-PW* setting to address the fleet minimization optimization criterium in a more efficient way than the simple construction mode algorithm considered for the previously discussed *Algorithm-D* basic setting. Finally, due to the parallelization, the resulting algorithm boasts interesting anytime characteristics.

Recall, that in case of the simple route construction algorithm the coordination process was started with an initial solution σ with a number of vehicles corresponding to the theoretical lower bound number of vehicles computed based on the characteristics of the particular problem

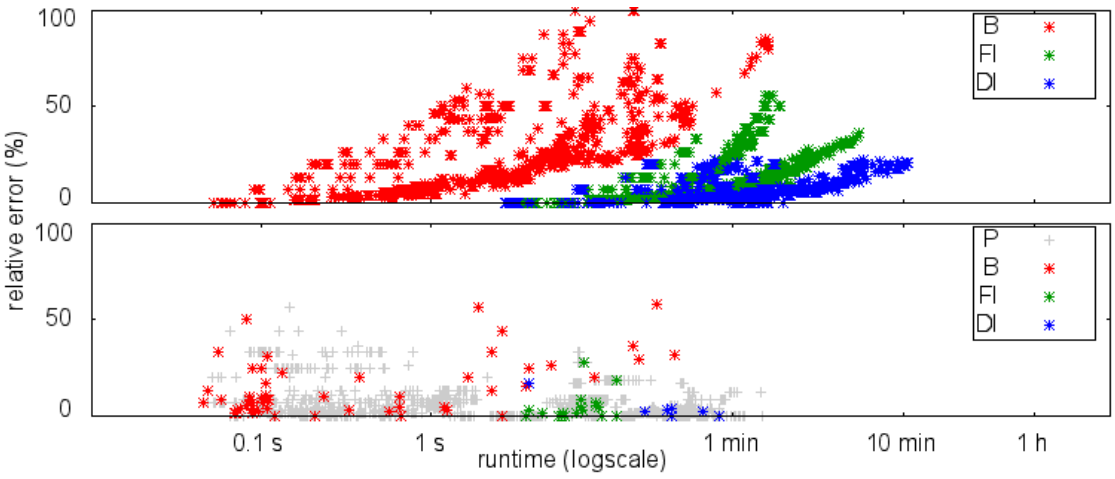


Figure 5.3: Individual algorithm instance results and runtimes with (bottom) and without (top) overall pruning strategy

Table 5.6: Average runtimes for individual instance sizes

Size	*Lim* [69]	*Prescott-Gagnon* [91]	*Nagata* [76]	*Algorithm-D*	*Algorithm-PW*	
					Single threaded	Parallel
200	93 min	265 min	20.5 min	3s	10 s	57 ms
400	296 min	445 min	81 min	22s	2 min	300 ms
600	647 min	525 min	80.4 min	2 min	8 min	2 s
800	1269 min	645 min	138 min	6 min	24 min	7 s
1000	1865 min	810 min	186 min	8 min	54 min	14 s
CPU	P4-2.8G	OPT-2.4G	OPT-2.4G	K8-2.3G	K8-2.3G	

instance being solved. Then, in case of failure, the process was restarted using an initial solution with an increased number of vehicles until eventually a complete solution was found. This, especially when an unfitting ordering was used, resulted in many redundant runs of the COORDINATE function having a multiplicative effect on the resulting algorithm's complexity. Thus the approach used by the *Algorithm-PW* algorithm settings specifically aims at avoiding these redundancies.

Figure 5.3 illustrates the resulting improvements. Individual points in the two charts correspond to the relative errors and runtimes recorded for individual algorithm instances executed while solving a set of 16 problem instances from the set of Homberger's 1000 customer problems. The top chart correspond to the algorithm instances inherent to the full $\Delta \times \Omega$ strategy (i.e. without orderings pruning). Moreover, these instances were run without the above mentioned improvements, each of them thus using the simple route construction mode approach. On the other hand, the bottom chart corresponds to the *CSP+BP* pruning strategy executed using the introduced parallel wrapper. The alternative point colors correspond to the alternative algorithm settings being used i.e. the *Algorithm-B* (B), *Algorithm-F* (F) and *Algorithm-D* (D) used by the particular algorithm instance. The instances that have been terminated without ever yielding a

complete solution — due to the previously mentioned improvements — are denoted P.

The results clearly illustrate the effect of the introduced improvements. Note that in vast majority of problem instances (as apparent from Table 5.5) both approaches produce an overall solution with the same number of vehicles. However, as apparent from the two charts, the number of the actually executed algorithm instances is significantly lower for the parallel algorithm. This illustrates the influence inherent to the orderings pruning. Also, note that the runtimes of longest running algorithm instances are significantly higher in case of the top chart then in case of the bottom chart. In the top chart, the longest running instances are the instances using the most complex *Algorithm-D* setting combined with an unfitting ordering. These instances are terminated within the parallel wrapper and thus the inherent performance penalty is avoided. Moreover, as already mentioned, due to the ordering diversification within the *Algorithm-PW* setting, in many cases the overall best found solution is actually found by one of the simpler *Algorithm-B* or *Algorithm-F* settings and is never improved upon even by the most complex algorithm instances using the *Algorithm-D* setting. This, however, means that for these algorithm instances the COORDINATE function is run exactly once, minimizing the performance overhead inherent to these most complex algorithm instances. In overall, for the above mentioned experiment, the recorded average single-threaded runtime for the improved algorithm is 41 minutes, while for the baseline experiment traversing the whole $\Delta \times \Omega$ configuration space, not using any of the introduced improvements it is 258 min — representing a 6.3 times improvement.

The comparison in terms of runtime of the introduced *Algorithm-PW* algorithm setting and the recent state-of-the-art centralized algorithms and is presented by Table 5.6. The results for the *Algorithm-PW* setting correspond to the full fledged *CSP+BP* pruning strategy being used. We denote the sum of runtimes of all executed algorithm instances as the *single-threaded* runtime of the algorithm. On the other hand, we denote the time before the best solution is found in case the parallelization potential is fully exploited as the *parallel runtime*. For comparison, the runtime of the previous *Algorithm-D* setting is presented as well. The abbreviations in the "CPU" row correspond to AMD Opteron 2.4GHz (OPT-2.4G), Pentium 4 2.8GHz (P4-2.8G) and AMD K8 2.4GHz (K8-2.3G) processors.

The outstanding observation concerning the presented results is the very good parallel runtime of the algorithm in case the parallelization properties are fully exploited. This serves in particular to highlight the success of the presented search diversification based on specific way of diversifying the set of customer orderings Ω, enabling in many cases the best results to be identified by the simpler algorithm settings. In terms of the single-threaded runtime we argue that no relevant conclusions can be easily made. The compared algorithms outperform the presented agent-based algorithm in terms of solution quality even in its extended *Algorithm-PW* setting. Moreover, the runtimes presented for the state-of-the-art algorithms consider addressing also a secondary cumulative travel duration optimization criterium which is not addressed by the presented algorithm.

5.4 Algorithm With Backtracking Evaluation

In this section we present the experimental evaluation of the full fledged *Algorithm-BT* algorithm setting. This setting extends the previous *Algorithm-D* setting by enabling the backtracking mechanism represented by the BACKTRACK function within the global coordination algorithm. The abstract backtracking mechanism is described in Section 4.3.2 and the VRPTW specific extensions are introduced in Section 4.4.1.4.

As mentioned, the backtracking mechanism is a very important part of the presented algorithm. It enables the algorithm to continue even in situation when the previous settings would fail. In case a customer c is encountered such that none of the vehicles can feasibly accommodate c, in case of the *Algorithm-D* setting, the coordination process is considered failed. On the other hand, in case of the *Algorithm-BT* setting the introduced backtracking mechanism allows c to be allocated to one of the vehicles at the expense of ejecting a some other customers from the vehicle's route. The set of ejected customers denoted C_{ej} then reenters the coordination process.

For the VRPTW the ability to do so is very important. In Section 5.2.2.1 we analyzed the three presented trading methods used within the PUSH and SHAKEDYNAMIC or SHAKEFINAL functions. The analysis revealed that towards the end of the solving process the intermediate solution σ can get tightly constrained rendering all possible improving relocations infeasible. Such a situation occurs when the spatial and temporal organization of routes within σ results in the schedules of the routes being tight. These routes then may not permit the processed customer c to be accommodated as any possible way of doing so results in an untimely arrival to some of the other customers already on the corresponding route. In order for the solving process to continue the solution σ has to be reorganized in a different and more complex way than by means of simple feasible improving relocations. The backtracking mechanism thus allows for the solution σ to be transformed in a series of allocation and backtracking steps. This makes it possible for the algorithm to escape local optima and eventually converge to a better overall solution.

The presented evaluation considers two alternative ways of addressing the VRPTW optimization criterium. Firstly, the parallel wrapper of the *Algorithm-PW* is used. The set of traversed algorithm settings $\Delta = [B, F, BT]$ where B and F are defined the same way as in the previous section and BT corresponds to the *Algorithm-BT* settings with a $maxSize = 3$ ejection size parameter value and a *RelocateAll* trading method using a $loopLimit = 3$ setting for the PUSH and SHAKEDYNAMIC functions. Various settings for the *backtrackLimit* parameter were evaluated. For the first B and F settings the SSL local planning strategy is used. For the BT setting, however, the TTS strategy is used. The used set Ω corresponds to the Ω_{csp} identified in the previous section, consisting of 10 orderings. A BP strategy is used configured to process all the 10 orderings over the first two B and F settings and than select 2 best-performing orderings to be processed by the BT setting. The main reason for executing the B and F settings prior to the full blown BT setting is provide a good initial fleet size estimate for the BT setting to start with. The orderings pruning, on the other hand, has arguably limited relevance here as the *Algorithm-BT* is significantly less sensitive to the used customer ordering.

Alternatively, we evaluated also a configuration based on the improved route elimination algorithm introduced in section Section 4.4.2.3. Thus the initial solution σ corresponds to a solution with each customer being served by a single route. In an iterative process an effort is made to eliminate individual routes while using an increasingly complex settings for the *backtrackLimit* parameter within the corresponding invocations of the COORDINATE function. Other algorithm settings are carried over from the above mentioned configuration i.e. the $maxSize = 3$ ejection size limit setting and the *RelocateAll* trading method using a $loopLimit = 3$ setting for the PUSH and SHAKEDYNAMIC functions.

Table 5.7: Results for alternative settings of the *backtrackLimit* parameter

Set	CNV	Absolute (relative) errors in terms of CNV							
		Algorithm-BT							
	Nagata [76]	Construction				Elimination			
		2,000	5,000	20,000	200×N	256	1,024	4,069	16,384
100	405	5 (1.2%)	4 (1.0%)	3 (0.7%)	3 (0.7%)	8 (2%)	7 (1.7%)	2 (0.5%)	1 (0.2%)
200	694	5 (0.7%)	3 (0.4%)	2 (0.3%)	0 (0.0%)	13 (1.9%)	5 (0.7%)	0 (0.0%)	0 (0.0%)
400	1,381	18 (1.3%)	13 (0.9%)	12 (0.9%)	4 (0.3%)	29 (2.1%)	19 (1.4%)	11 (0.8%)	4 (0.3%)
600	2,067	35 (1.7%)	23 (1.1%)	11 (0.5%)	4 (0.2%)	42 (2.0%)	30 (1.5%)	15 (0.7%)	5 (0.2%)
800	2,738	63 (2.3%)	43 (1.6%)	16 (0.6%)	8 (0.3%)	53(1.9%)	42 (1.5%)	18 (0.7%)	7 (0.3%)
1000	3,424	83 (2.4%)	49 (1.4%)	18 (0.5%)	8 (0.2%)	62 (1.8%)	48 (1.4%)	25 (0.7%)	11 (0.3%)
R1	2,881	4 (0.1%)	3 (0.1%)	2 (0.1%)	2 (0.1%)	12 (0.4%)	5 (0.2%)	1 (0.0%)	1 (0.0%)
R2	600	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
C1	2,922	125 (4.3%)	73 (2.5%)	19 (0.7%)	4 (0.1%)	121 (4.1%)	84 (2.9%)	33 (1.1%)	7 (0.2%)
C2	900	62 (6.9%)	47 (5.2%)	26 (2.9%)	16 (1.8%)	59 (6.6%)	49 (5.4%)	29 (3.3%)	15 (1.8%)
RC1	2,802	10 (0.4%)	7 (0.2%)	6 (0.2%)	3 (0.1%)	9 (0.3%)	8 (0.3%)	6 (0.2%)	3 (0.1%)
RC2	604	5 (0.8%)	4 (0.7%)	2 (0.3%)	2 (0.3%)	6 (1.0%)	5 (0.8%)	2 (0.3%)	2 (0.3%)
All	10,709	206 (1.9%)	134 (1.3%)	55 (0.5%)	27 (0.3%)	207 (1.9%)	151 (1.4%)	71 (0.7%)	28 (0.3%)

5.4.1 Solution Quality

The performance of the algorithm is illustrated by Table 5.7. The results are presented in a similar way as in case of the two previously discussed general algorithm settings. The results are listed for alternative subsets from the two considered benchmarks identified by the first column. The "100" row corresponds to the Solomon's benchmark, the "200–1000" rows denote alternative sizes within the Homberger's benchmark. The "C1–RC2" rows correspond to the alternative instance types shared across both benchmarks and the "All" row lists the overall result over both benchmarks. For individual subsets of problems the CNV corresponding to the solutions obtained by the algorithm presented by Nagata [76] representing the state-of-the-art is presented within the second column. The rest of the columns correspond to the results of the *Algorithm-BT* setting. The first four columns correspond to construction mode algorithm being used and the four alternative settings for the *backtrackLimit* parameter. The last of these column denoted $200 \times N$ corresponds to a setting with the *backtrackLimit* parameter value being proportional to the size of the instance denoted as N. The remaining three columns correspond to the route elimination algorithm and the three alternative settings for the *backtrackLimit* parameter within the RouteElimination function.

In case of the route construction mode algorithm, considering the most complex evaluated setting for the *backtrackLimit* parameter the algorithm was able to equal solutions achieved by the compared centralized state-of-the-art algorithm in 90.7% of all tested problem instances. This corresponds to an overall relative error of 0.3% in terms of CNV over the two benchmarks. The route elimination algorithm achieves nearly identical solution quality with an overall absolute CNV error of 28 vehicles corresponding to a 0.3% relative error. Interestingly, it outperformed the construction mode algorithm on the Solomon's 100 customer benchmark with two additional solutions matching the best known solutions being found.

Table 5.8: Comparison with previous agent-based VRPTW algorithms

Set	CNV	Absolute (relative) error in terms of CNV					
	Nagata	Agents			Presented Algorithm		
	[76]	*Fisher* [28]	*Leong* [67]	*Kalina* [47]	*Algorithm-D*	*Algorithm-PW*	*Algorithm-BT*
100	405	31 (7.7%)			24 (5.9%)	16 (4.0%)	1 (0.2%)
100-R1	143		30 (21.0%)		11 (7.7%)	7 (4.8%)	1 (0.7%)
200-1000	10,304			573 (5.6%)	319 (3.1%)	238 (2.3%)	24 (0.2%)
All	10,709				343 (3.2%)	254 (2.7%)	25 (0.2%)

Compared to the previously evaluated settings, this represents a significant improvement. Interesting in particular is the comparison of the construction mode algorithm with the previous *Algorithm-D* setting achieving an overall relative error of 3.2%. As discussed previously when the solution quality of the *Algorithm-D* setting was analyzed this setting cannot efficiently recover from a situation when the currently processed customer $\bar{c}$ cannot be feasibly allocated to the current intermediate solution σ. In such a case the coordination process fails. In the extended *Algorithm-PW* setting, this is partially offset by an effort to diversify the search considering a specific wide set of customer orderings. Such an approach proves highly successful in producing reasonably good solutions in a very short time given the parallelization potential is fully exploited as discussed within the previous section. However the situation discussed above still applies resulting in a 2.3% overall relative error for the *Algorithm-PW* setting. On the other hand, the *Algorithm-BT* setting is able to efficiently address such a situation using the introduced backtracking mechanism. In case the currently processed customer $\bar{c}$ cannot be feasibly allocated, instead of the process being terminated, the customer $\bar{c}$ is allocated to σ at the expense of ejecting some other customers from σ.

The backtracking mechanism uses a powerful search diversification criterium based on the values of the failure counters for individual customers $c.fails, c \in C$. These counters are incremented every time the corresponding customer fails to be feasibly allocated to the emerging solution σ. Then, when evaluating the possible ejections admitting the feasible allocations of $\bar{c}$ to σ, the ejection cost evaluation criterium corresponds to the sum of the values of the *c.fails* counters for the ejected customers. Such a criterium favors the ejection of tasks that (i) have been ejected the least times or (ii) have proved to be easy to allocate. The results support the soundness of the ejection cost evaluation criterium. Also importantly, the evaluation criterium is abstract and problem independent. As outlined previously, it can be therefore embedded within the abstract coordination mechanism and can be easily used with other task allocation problems as well. The provided experimental evidence bodes well for such possible future applications especially as the VRPTW has been extensively researched and is considered to be a very hard problem to solve.

Considering the most complex setting with the *backtrackLimit* parameter being linearly proportional to the size of the problem instance the performance of the algorithm is consistent across alternative instance sizes. This further supports the efficiency of the proposed method with respect to the existing traditional centralized algorithms. The difference in performance over the Solomon's 100 customer instances for the construction mode algorithm is arguably due to the fact that over the smaller benchmark the competing algorithms (that are typically not computationally bound) often use a non-proportionally long running times, as also illustrated by Table 5.11.

The Table 5.8 further presents the comparison of the *Algorithm-BT* setting to the previ-

Table 5.9: Comparison of local planning strategies in *Algorithm-BT* setting on Homberger's 200 customer problems.

Strategy	Relative error in terms of CNV							
	Construction				Elimination			
	512	1024	4096	8192	64	128	512	2048
TTS	1.44%	1.16%	**0.43%**	**0.29%**	2.75%	**1.59%**	**0.29%**	**0.00%**
SLS	**1.30%**	**0.57%**	0.58%	0.43%	**2.17%**	1.88%	0.87%	**0.00%**

ous comparable agent-based algorithms, including the two previous settings of the algorithm. The notation is similar as used for the previous table. The "100-R1" row corresponds to the R1-type problems from the Solomon's benchmark while the 200–1000 row denotes the complete Homberger's benchmark. The experimental results suggest that the presented algorithm significantly outperforms the previous comparable agent-based algorithms.

Note that to our knowledge these are the only comparable data as far as the agent-based VRPTW algorithms go. As discussed in detail previously as part of the review of the related work, the remaining agent-based algorithms were not evaluated in a way that enables relevant comparison to the presented algorithm or the state-of-the-art centralized algorithms. These results thus also serve to highlight the fact that agent-based algorithms for VRPTW have not been particularly well studied. Arguably this is due to the fact that historically the agent based methods haven't proved very efficient as an optimization technique.

5.4.2 Local Planning Strategies

As with the previous algorithm settings, we analyzed the relative success of the two introduced local planning strategies for the *Algorithm-BT* setting. Table 5.9 presents the experimental results. Both construction and route elimination variants relevant for this setting are considered.

The results for previous algorithm settings indicated that the SLS strategy outperforms the TTS strategy. However, surprisingly, considering the *Algorithm-BT* setting the results suggest that with increasing value of the *backtracksLimit* parameter the TTS strategy dominates the SSL strategy. The SLS strategy is based on constructing the routes in a way as to maximize the chance of these routes accommodating some other customers i.e. by maximizing the slackness of the routes. The flip-side of such an approach is, however, that the resulting routes are not as efficient in terms of minimizing the travel time. The TTS strategy, on the other hand, aims at constructing a spatially dense routes disregarding the eventual temporal tightness of the resulting route schedules. This — as we already discussed — often results in the inability of the underlying solution to accommodate customers encountered later within the solving process. However, the backtracking mechanism provides means to address such a situation while diversifying the underlying search process. Thus, arguably, the discussed results suggest that the overall solution quality coincides to a significant extent with the spatial efficiency of the underlying routes.

This result is quite surprising. The previous works discussing insertion heuristics for the VRPTW [107, 39, 72, 13] conclude that heuristics addressing also the temporal aspects of the problem are superior to the heuristics addressing only the underlying routing and bin-packing subproblems. Both presented local planning strategies are based on the cheapest insertion principle powered by the corresponding VRPTW insertion heuristics. The overall results for all the discussed algorithm settings thus suggest that while the previous findings hold for a monotonous allocation process without backtracking, with the introduction of the backtracking the mentioned

Table 5.10: Sensitivity to customer ordering of the *Algorithm-D* and *Algorithm-BT* (construction mode) settings on Solomon's 100 customer problems.

Ordering	Relative error in terms of CNV with respect to [76]				
	Algortihm-D	*Algorithm-BT*			
		128	256	512	1024
HDF	8.7%	4.4%	3.4%	1.9%	1.6%
TTF	6.7%	3.9%	3.3%	2.5%	2.0%
EF	7.1%	3.5%	3.4%	2.2%	1.8%
LEF	5.8%	3.7%	3.1%	2.1%	1.9%
RANDOM	9.7%	5.0%	3.9%	2.7%	2.2%

conclusion actually does not hold.

5.4.3 Sensitivity to Customer Ordering

The sensitivity of the *Algorithm-BT* construction mode setting to the used customer ordering is illustrated by Table 5.10. The results correspond to the relative errors in terms of CNV over the Solomon's 100 customer problems. Results are presented for four analytically sound orderings as well as the baseline random ordering of customers. Results for four alternative settings for the *backtrackLimit* are presented. For reference the results for the previous *Algorithm-D* setting are included as well.

The results show that the backtracking mechanism of the *Algorithm-BT* setting significantly offsets the sensitivity of the algorithm to the used customer ordering. In case of the basic *Algorithm-D* setting not featuring the backtracking mechanism the choice of an analytically sound ordering significantly affects the resulting solution quality. The difference between the best performing analytical ordering and the baseline RANDOM ordering is 3.9% in case of this setting. This gap drops significantly for the *Algorithm-BT* setting and with the increasing value of the *backtrackLimit* parameter. For the highest evaluated value of the *backtrackLimit* parameter the gap is reduced to 0.6%.

On the other hand, the results suggest that the choice of a fitting ordering still affects the algorithm even in the *Algorithm-BT* setting. This is particularly obvious from the fact that all the analytically sound orderings outperformed the RANDOM ordering in all the evaluated cases. This, however, is hardly surprising. As already discussed an unfitting ordering may result in encountering a customer towards the end of the solving process the accommodation of which requires a thorough reorganization of the current solution. As discussed previously, in many cases the only way in which this can be achieved is by using the backtracking mechanism. On the other hand, by using a fitting ordering, the likelihood of such a situation occurring is lower and the situation arises later within the solving process. Therefore the chance of arriving to a feasible solutions using a limited number of backtracking steps is higher.

No results are presented for the route elimination mode of the *Algorithm-BT* setting. For the route elimination algorithm, the initial solution σ already contains all the customers. As the order in which the routes are eliminated is non-deterministic the customer ordering is not a well defined term in this case. Further supporting this point of view is the fact that the elimination algorithm proved very consistent with respect to the choice of the seed used to initiate the randomization of the route-selection process.

The sensitivity of the construction mode algorithm to customer ordering can be viewed in

Table 5.11: Average runtimes for individual instance sizes

Size	Lim [69]	Prescott-Gagnon [91]	Nagata [76]	*Algorithm-BT*	
				Construction	Elimination
100	39 min	192.5 min	25 min	9 min	4 min
200	93 min	265 min	20.5 min	67 min	34 min
400	296 min	445 min	81 min	345 min	317 min
600	647 min	525 min	80.4 min	615 min	587 min
800	1269 min	645 min	138 min	1253 min	1116 min
1000	1865 min	810 min	186 min	1765 min	1578 min
CPU	P4-2.8G	OPT-2.4G	OPT-2.4G	K8-2.3G	

two ways. On one hand it makes the algorithm slightly less suited for dynamic problems where the order is given. On the other hand, using an analytically sound orderings can be considered as using an additional heuristic for the problem. A fitting customer ordering causes the construction mode *Algorithm-BT* setting to operate on a sequence of solutions that are incomplete and therefore smaller. This results in limiting the number of backtracks necessary for accommodating any single processed task especially at in the early stages of the allocation process. Using a fitting ordering thus potentially enables the algorithm to address some key aspects of the emerging solution early within the solving process reducing the size of the overall search.

Considering the above assumption a particularly interesting is the fact that the HDF ordering proves very successful for the higher *backtrackLimit* parameter settings. With the HDF ordering the customers encountered latest within the solving process are the customers with the lowest demands (the demands being 10, 20 or 30 for all the customers). This may enable the algorithm to consider some smaller ejections in the later stages of the solving process that would otherwise not be possible due to capacity constraints of the vehicles. The same effect carries through the whole phase of the algorithm before all these ejected customers (and any other customers ejected as a result of the effort to allocate these) are eventually allocated to the intermediate solution σ. In overall, this can result in the above mentioned phase requiring a smaller number of backtracks resulting in a better convergence of the overall algorithm.

5.4.4 Runtime and Convergence Analysis

The comparison in terms of runtime with the traditional VRPTW algorithms is presented by Table 5.11. Average runtimes for individual instance sizes are presented for the construction and route elimination variants of the *Algorithm-BT* setting. The abbreviations in the "CPU" row correspond to AMD Opteron 2.4GHz, Pentium 4 2.8GHz and AMD K8 2.4GHz processors. Three of the successful contemporary traditional algorithms are listed as well.

The results show that the convergence of the presented algorithm is significantly worse than in case of the best compared traditional algorithm [76]. The gap between the presented and the compared algorithm increases with increasing problem size as well. Note also, that the runtimes listed for the compared algorithms include addressing also the secondary travel time optimization criteria.

We argue that this is primarily due to the fact that the local search processes inherent to the Push and Shake.. methods are very simple compared to the heuristics used for the competing algorithms. With each feasible allocation attempt an effort is made to find a vehicle within the intermediate solution σ that can feasibly allocate the currently processed customer or, in case

such a straightforward allocation is not possible, an effort is made within the PUSH function to find a feasible customer relocation that would enable the currently processed customer $\bar{c}$ to be allocated. This corresponds to a very simple local search process. The explored neighborhood of solutions related to σ potentially permitting the accommodation of the customer $\bar{c}$ is very simple and limited, generated by the customer relocation operations. In case this effort fails and the customer $\bar{c}$ cannot be feasibly allocated to the emerging solution σ an ejection results.

To the contrary, in case of the competing algorithms, the local search inherent to accommodating $\bar{c}$ is much more complex. For example, in an effort to allocate a customer $\bar{c}$ to an intermediate solution σ the algorithm presented in [76] employs a local search mechanism denoted as the *squeeze* method. The customer $\bar{c}$ is inserted into feasible partial solution σ at the expense of violating of the capacity and/or time-window constraints. Follows an attempt to restore the feasibility by applying a specific local search procedure. The local search is based traversing the neighborhood $\mathcal{N}(\sigma)$ generated by customer relocations as well as the well-known *2-opt** [90] operator. Powering the local search is the function $f : \mathcal{N}(\sigma) \rightarrow \langle 0, 1 \rangle$ used to evaluate the measure of infeasibility of the solutions within the neighborhood of the contemporary solution σ. Only in case the local search inherent to the *squeeze* method fails to find a feasible solution the algorithm backtracks using an ejection based mechanism similar to the one used by the presented algorithm.

Apparently, this vastly more complex feasible allocation process dramatically increases the chance of a feasible allocation of the processed customer $\bar{c}$ to the intermediate solution σ. The immediate effect of this is that the algorithm doesn't need to backtrack as often. Instead, the algorithm penetrates the relevant areas of the search space by allowing for temporal solution infeasibility. We argue that similar concepts could be introduced to the presented algorithm. The coordination semantic could be enriched by evaluating more complex moves than single task relocations. Moreover, an alternative agent's local planning strategy could be developed enabling the agents to consider also the unfeasible customer allocations and trading moves, providing means of embedding more complex local search phases to the overall solving process and reducing the number of backtracking steps. Using such modifications could arguably yield significant improvements with respect to the convergence of the resulting VRPTW algorithm.

Table 5.12: Summary of the experimental evaluation

Version	CNV	Error	Runtime	Mode
Algorithm-B	11805	10,4%	7 s	RC
Algorithm-F	11398	6.6%	23 s	RC
Algorithm-D	11038	3.2%	8 min	RC
Algorithm-PW	10949	2.3%	54 min / 14 s	RC-PW
Algorithm-BT-RE	10723	0.3%	1578 min	RE
Algorithm-BT-RC	10722	0.3%	1765 min	RC-PW

5.5 Summary of Experimental Results

Within the previous sections we presented a suite of experiments aiming at assessing the efficiency of the presented algorithm in its various settings as well as illustrating the important attributes of the algorithm and the underlying solving process.

Compared to the traditional centralized algorithms represented by the state-of-the-art algorithm presented in [76] the algorithm in it's most complex *Algorithm-BT* setting was able to equal the solutions obtained by this algorithm in 90.3% of all the 356 problems taken from the two widely used benchmarks. Such a competitive performance also represents a significant improvement over all previously presented comparable agent-based algorithms. On the other hand, in terms of convergence there is still a considerable gap between the presented agent-based algorithm and the best performing compared traditional algorithms.

Three fundamental versions of the algorithm were presented. The three introduced basic settings — the *Algorithm-B*, *Algorithm-F* and the *Algorithm-D* setting — correspond to the simplest single-threaded version of the algorithm with no backtracking in place. Follows the parallel *Algorithm-PW* setting which is based on executing the three previous basic algorithm settings in a specific parallel wrapper. Lastly the *Algorithm-BT* setting presents a full-fledged algorithm by enabling the backtracking mechanism.

For the initial basic settings the VRPTW primary optimization criterium is addressed using the construction mode algorithm. Similar approach is used also in case of the *Algorithm-PW* setting, however, with some key improvements made possible by the introduced parallel wrapper. These improvements are used as well in case of the *Algorithm-BT* in the construction mode setting denoted *Algorithm-BT-RC*. An alternative setting based on the route elimination mode is considered as well denoted as the *Algorithm-BT-RC* setting.

The summary of overall experimental evaluation is presented within the Table 5.12. The summary illustrates the key differences between all the evaluated settings of the algorithm and the effect to the efficiency of the resulting algorithm. The first three rows correspond to the results for the 3 basic settings. Follows the evaluated *Algorithm-PW* setting. The last two rows then correspond to the two evaluated variants of the *Algorithm-BT* setting — the route construction and the route elimination mode algorithms. For each of the setting the essential information is summarized: (i) the overall CNV over all evaluated problem instances from the Solomon's and Homberger's benchmarks and (ii) the corresponding relative error with respect to the results achieved by [76], (iii) the average single threaded (and parallel where applicable) runtime over the largest 1000 customer problems and lastly (iv) the way in which the fleet minimization criterium is addressed (the 'Mode' column) with RC denoting the simple route construction algorithm, the RE the route elimination and RC-PW denoting process inherent to the introduced parallel wrapper.

A number of additional experiments was carried out revealing some interesting attributes of the respective algorithm versions and the underlying solving processes. We summarize the key findings here:

- **Local Planning Strategies:** The SLS planning strategy proved more efficient with the basic algorithm settings. However, with the introduction of backtracking within the *Algorithm-BT* setting, it was outperformed by the TTS strategy. We attribute this to the fact that in high quality solutions the spatial efficiency of the routes prevails over the efficiency in terms of the route slackness.

- **Sensitivity to customer ordering:** The algorithm in any of it's basic settings is sensitive to the order in which the customers are processed. However, the analytically sound orderings consistently outperform random orderings. Also, we observed that specific problem instances favor specific orderings. The parallel wrapper inherent to the *Algorithm-PW* setting is based on these findings. On the other hand, for both evaluated variants of the *Algorithm-BT* setting the sensitivity is negligible or — in case of the route elimination mode — the customers are always processed in a random order.

- **Local search of the PUSH and SHAKE.. functions:** The local search inherent to the PUSH and SHAKE.. functions is based on feasible single customer relocations. In case the underlying solution is tightly constrained, such an approach is not very efficient as most moves result in constraint violation and are therefore infeasible. The backtracking mechanism helps to escape the local extremes caused by the inability of these methods to further transform the emerging solution at the expense of ejections. By comparing the inherent local search with the state-of-the-art local search methods we argue that a more sophisticated local planning strategy and corresponding coordination semantic could be developed promising a significant boost the algorithm's convergence.

The proposed future work outlined below reflects these outstanding findings. In overall, we argue that the breadth and detail of the provided experimental evidence as well as the arguably promising very good results representing the state-of-the-art with respect to the agent-based VRPTW algorithms separate the work from the previous similar agent-based studies.

Chapter 6

Conclusion

This thesis is concerned with developing efficient agent-based algorithms for the VRPTW. An elementary overview of the VRPTW and the contemporary solving approaches is presented (Chapter 2). Follows a brief introduction to multi-agent optimization (Chapter 3). The VRPTW is reformulated as a social welfare maximization problem in a society of agents representing the individual vehicles being part of the problem. A fitting agent architecture is introduced and the underlying solving process is outlined. Alternative local planning strategies used by the agents as well as alternative interaction patterns enabling for finding efficient solutions to the joint optimization problem are analyzed. Based on these concepts, several incremental versions of the resulting VRPTW algorithm are introduced. The challenges and opportunities of adopting a multi-agent approach to VRPTW are discussed as well (Chapter 4). Follows an extensive experimental assessment of the algorithm providing a relevant comparison to the state-of-the-art traditional as well as the previous agent-based algorithms. A number of complementary experiments is presented as well providing insight into the underlying solving process (Chapter 5).

The algorithm uses a specific coordination framework empowering the agents' society with interaction primitives used in order to find an efficient joint solution to the overall problem. As all the VRPTW problem specific logic is embedded within the agents' local subproblem solving, the coordination framework remains abstract and problem independent. As such it can be reused to address a number of general task allocation problems as well. This framework, significantly extending the framework previously presented in [121] thus represents a separate authentic contribution of this thesis. Namely, as part of this framework, a specific backtracking mechanism is presented as well as a specific approach to parallelization that proved to be very efficient for the VRPTW case. The way autonomic behavioral patterns can be embedded within the agents' local sub-problem solving i.e. by means of introducing social commitments is discussed as well, potentially increasing the execution stability of the system in uncertain environments.

The experimental assessment is based on the two arguably most relevant benchmarks known from the operations research literature, providing a sound comparison with respect to the traditional methods. Such a comparison was missing from most previous comparable agent-based works. Over these benchmarks the resulting full-fledged version of the algorithm was able to equal the best-known solutions achieved by the traditional algorithms in 90.3% of all the problems resulting in a 0.3% overall relative error. This result also represents a significant improvement over previous comparable agent-based studies. The parallel version of the algorithm further provides for an exceptional anytime convergence, outperforming even the traditional algorithms in this respect. Lastly a number of interesting future research opportunities was identified. These opportunities are outlined later in Section 6.2.

6.1 Thesis Achievements

As mentioned at the beginning of this text this thesis aimed at reformulating the VRPTW as a multi-agent optimization problem and researching the alternative behavioral and interaction patterns in an effort to provide efficient multi-agent VRPTW algorithm. The fundamental goals pursued by this thesis were to achieve a competitive performance of the resulting VRPTW algorithm and to discuss its relevant features and the general implications of adopting a multi-agent approach to VRPTW. We argue that the goals of this thesis were met. The achievements of this thesis can be summarized as follows:

1. A multi-agent reformulation of the VRPTW was presented as well as a fitting agent architecture underlying the introduced agent-based algorithm. The architecture features a problem independent abstract global coordination layer and a problem specific local planning layer that were presented separately in order to stress the wider applicability of the abstract part of the architecture that can be used to solve a variety of general task allocation problems. The abstract coordination framework represents a significant improvement over the previously presented similar framework [121] namely by the introduction of the abstract backtracking mechanism which proved to be crucial with respect to the efficientcy of the algorithm for the VRPTW case.

2. Two alternative Vehicle Agents' local planning strategies were introduced based on the known insertion heuristics for the VRPTW as well as a generic configurable coordination framework providing for a number of alternative coordination semantics to be adopted by the resulting VRPTW algorithm, affecting its efficiency as well as complexity.

3. A series of incremental VRPTW algorithm versions were presented including a parallel execution wrapper with the resulting full-fledged VRPTW algorithm providing for a relative error of 0.3% over the two relevant benchmarks used widely when evaluating the performance of the competing traditional VRPTW algorithms. Such a result is competitive with respect to the traditional algorithms and to our knowledge represents a significant improvement over all previous comparable agent-based algorithm.

4. As part of the experimental evaluation an in depth analysis of the underlying solving process was provided spawning numerous observations and pointers towards the limits of the current method as well as the possible future improvements. These improvement are outlined in the next section together with other relevant topics possibly extending this work.

6.2 Future Work

A number of interesting future research opportunities was identified. Some of them have been outlined in greater detail within the text. We summarize the most relevant ones:

1. **Introduce complex trading moves inspired by the known neighborhood generation operators for the VRPTW** — In Chapter 2 we discussed the local search neighborhood generation operators used by the contemporary VRPTW algorithms. We also discussed some of the known crossover operators used in genetic such algorithms. We believe that these operators could be analyzed and corresponding trading moves involving two or more agents could be developed, potentially improving the convergence of the algorithm. Moreover, some of these advances could potentially be abstracted within the abstract coordination framework to be reused with other task allocation problems.

2. **Develop novel local planning strategies for the VRPTW** — In Chapter 2 we surveyed some of the contemporary algorithms. Central to the success of these methods is the handling of temporal infeasibility of the intermediate solutions allowing infeasible allocations to be performed, followed by a repair phase in which the feasibility is recovered. We also hinted at several places, that the local search process inherent to the PUSH function is very simple, resulting in frequent backtracking of the algorithm. Together with the previously mentioned advances, we propose a novel implementation of the PUSH function that would adopt a similar approach as outlined above based on infeasible allocation of the processed customer followed by a specific feasibility recovering procedure. Such an improvement could, in our opinion, dramatically improve the performance of the resulting VRPTW algorithm.

3. **Address additional optimization criteria for the VRPTW** — We mentioned that some of the contemporary VRPTW algorithms address also the *travel time minimization* secondary optimization criterium. Obviously, this is a very important criterium with respect to real-world applicability of the resulting solutions. We therefore propose a complementary study to be carried out aiming at developing efficient agent-based strategies for addressing this criterium.

4. **Assess the general applicability of the abstract coordination framework** — In Section 4.3 we introduced the agent-based abstract coordination framework that can be used to address a variety of general task allocation problems. The framework represents a significant extension of previous similar concepts [121], most notably by introducing the abstract backtracking mechanism. We believe that the assessment of its discussed wider applicability would be a very interesting complement to the existing evidence for the VRPTW case. Moreover, we believe that this framework could be further developed by integrating the abstract concepts of social commitments, the above mentioned alternative coordination semantics and potentially also features aimed at monitoring and controlling the system operation.

5. **Experimentally assess the applicability of commitment based plan representations to uncertain environments** — In Section 3 we introduced the concept of social commitments. Based on previous evidence [121], we discussed how this can be useful in addressing unexpected locally observable events in the environment. However, these concepts were not relevantly experimentally assessed within the presented task allocation framework. Therefore preforming such an assessment potentially extending the previous similar concepts and evidence represents another intriguing future research prospect.

6.3 Selected Publications and Response

This section summarizes the author's selected publications related to the content of the thesis:

Articles in Journals (1):

{1} **P. Kalina, J. Vokřínek, and V. Mařík. Agents towards vehicle routing problem with time windows.** *Journal of Intelligent Transportation Systems*, Accepted 2013, available at `http://dx.doi.org/10.1080/15472450.2014.889953`.

Articles in Proceedings (5):

{2} **P. Kalina, J. Vokřínek, and V. Mařík. An efficient route minimization algorithm for the vehicle routing problem with time windows based on agent negotiation.** In G. Boella, E. Elkind, B. Savarimuthu, F. Dignum, and M. Purvis, editors, *PRIMA 2013: Principles and Practice of Multi-Agent Systems*, volume 8291 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2013.

{3} **P. Kalina and J. Vokřínek. Parallel solver for vehicle routing and pickup and delivery problems with time windows based on agent negotiation.** In *2012 IEEE Conference on Systems, Man, and Cybernetics (SMC)*, pages 1558–1563, 2012.

{4} **P. Kalina, J. Vokřínek, and V. Mařík. The art of negotiation: Developing efficient agent-based algorithms for solving vehicle routing problem with time windows.** In V. Mařík, J. Lastra, and P. Skobelev, editors, *Industrial Applications of Holonic and Multi-Agent Systems*, volume 8062 of *Lecture Notes in Computer Science*, pages 187–198. Springer Berlin Heidelberg, 2013.

{5} **P. Kalina and J. Vokřínek. Improved agent based algorithm for vehicle routing problem with time windows using efficient search diversification and pruning strategy.** In *Proceedings of the 3rd Workshop on Artificial Intelligence and Logistice (AiLog) of the 2012 European Conference on Artificial Intelligence (ECAI)*, pages 13–18, 2012.

{6} **P. Kalina and J. Vokřínek. Algorithm for vehicle routing problem with time windows based on agent negotiation.** In *Proceedings of the 7th Workshop on Agents In Traffic and Transportation, AAMAS*, 2012.

Responses (4):

- {1} was cited by [115]
- {4} was cited by [43, 44]
- {6} was cited by [111]

Contribution:

The author of this thesis is also the first author of all the listed publications. The contribution ratio is 70%/30% for the publications where two authors are listed and 70%/20%/10% in case three authors are listed respectively to the order in which the authors are listed. The contribution ratio of the author of this thesis is therefore 70% for all the listed publications. Also, both the remaining authors featured in all the listed works are in fact the two author's supervisors.

Bibliography

[1] S. Aknine, S. Pinson, and M. Shakun. An extended multi-agent negotiation protocol. *Autonomous Agents and Multi-Agent Systems*, 8(1):5–45, 2004.

[2] K. J. Arrow, A. Sen, and K. Suzumura. Edited by. In A. S. Kenneth J. Arrow and K. Suzumura, editors, *Handbook of Social Choice and Welfare*, volume 2 of *Handbook of Social Choice and Welfare*. Elsevier, 2011.

[3] J. B. Atkinson. A greedy look-ahead heuristic for combinatorial Optimization: An Application to Vehicle Scheduling with Time Windows. *Journal of the Operational Research Society*, 45(6):673–684, 1994.

[4] A. Bachem, W. Hochstättler, and M. Malich. The simulated trading heuristic for solving vehicle routing problems. Technical report, DISCRETE APPLIED MATHEMATICS, 1996.

[5] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46:316–329, 1996.

[6] R. Bent and P. Van Hentenryck. A two-stage hybrid local search for the vehicle routing problem with time windows. *Transportation Science*, 38(4):515–530, Nov. 2004.

[7] J. Bíba, J. Vokřínek, and J. Hodík. Renegotiable competitive contract net protocol. Technical report, Agent Technology Group, Gerstner Laboratory, Czech Technical University in Prague, 2008.

[8] A. L. Blum and M. L. Furst. Fast planning through planning graph analysis. *ARTIFICIAL INTELLIGENCE*, 90(1):1636–1642, 1995.

[9] C. Boutilier. Planning, learning and coordination in multiagent decision processes. In *In Proceedings of the Sixth Conference on Theoretical Aspects of Rationality and Knowledge (TARK96*, pages 195–210, 1996.

[10] C. Boutilier, T. Dean, and S. Hanks. Planning under uncertainty: Structural assumptions and computational leverage. In *In Proceedings of the Second European Workshop on Planning*, pages 157–171. IOS Press, 1996.

[11] R. I. Brafman and C. Domshlak. From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 28–35, 2008.

[12] S. J. Brams and A. D. Taylor. An envy-free cake division protocol. *The American Mathematical Monthly*, 102(1):9+, Jan. 1995.

[13] O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, part I route construction and local search algorithms. *Transportation Science*, 39(1):104–118, 2005.

[14] O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, part II metaheuristics. *Transportation Science*, 39(1):119–139, 2005.

[15] D. Brélaz. New methods to color the vertices of a graph. *Commun. ACM*, 22(4):251–256, Apr. 1979.

[16] A. M. Campbell and M. Savelsbergh. Efficient insertion heuristics for vehicle routing and scheduling problems. *Transportation Science*, 38:369–378, August 2004.

[17] A. Chabrier. Vehicle routing problem with elementary shortest path based column generation. *Computers and Operations Research*, 33(10):2972 – 2990, 2006. Part Special Issue: Constraint Programming.

[18] W.-C. Chiang and R. A. Russell. Simulated annealing metaheuristics for the vehicle routing problem with time windows. *Annals of Operations Research*, 63(1):3–27, Feb. 1996.

[19] M. Cieliebak, T. Erlebach, F. Hennecke, B. Weber, and P. Widmayer. Scheduling with release times and deadlines on a minimum number of machines. In J.-J. Levy, E. Mayr, and J. Mitchell, editors, *Exploring New Frontiers of Theoretical Informatics*, volume 155 of *IFIP International Federation for Information Processing*, pages 209–222. Springer US, 2004.

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

[21] Z. Dan, L. Cai, and L. Zheng. Improved multi-agent system for the vehicle routing problem with time windows. *Tsinghua Science Technology*, 14(3):407–412, 2009.

[22] P. Davidsson, L. Henesey, L. Ramstedt, and J. T˙ An analysis of agent-based approaches to transport logistics. *Transportation Research Part C: Emerging Technologies*, 13(4):255 – 271, 2005.

[23] R. Davis and R. G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20:63–109, 1983.

[24] G. Desaulniers, F. Lessard, and A. Hadjar. Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. *Transportation Science*, 42(3):387–404, Aug. 2008.

[25] M. Desrochers. An algorithm for the shortest path problem with resource constraints. *Les cahiers du GERAD, 1988, no. G-88-27*.

[26] K. Erol, J. Hendler, and D. S. Nau. Htn planning: Complexity and expressivity. In *In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94*, pages 1123–1128. AAAI Press, 1994.

[27] FIPA. Foundation for intelligent physical agents [online]. ⟨`http://www.fipa.org`⟩, 12 2003.

[28] K. Fischer, J. P. Müller, and M. Pischel. Cooperative transportation scheduling: an application domain for dai. *Journal of Applied Artificial Intelligence*, 10:1–33, 1995.

[29] K. Fischer, J. P. Muller, and M. Pischel. Schedulling an application domain for dai. *Applied Artificial Intelligence*, 10:1–33, 1996.

[30] C. Foisy and J.-Y. Potvin. Implementing an insertion heuristic for vehicle routing on parallel hardware. *Comput. Oper. Res.*, 20(7):737–745, Sept. 1993.

[31] J. françois Cordeau, G. Laporte, École Hautes, Études Commerciales, and L. C. D. Gerad. A unified tabu search heuristic for vehicle routing problems with time windows. *JOURNAL OF THE OPERATIONAL RESEARCH SOCIETY*, 52:928–936, 2001.

[32] B.-L. Garcia, J.-Y. Potvin, and J.-M. Rousseau. A parallel implementation of the tabu search heuristic for vehicle routing problems with time window constraints. *Computers and Operations Research*, 21(9):1025 – 1033, 1994.

[33] H. Gehring and J. Homberger. A two-phase hybrid metaheuristic for the vehicle routing problem with time windows. *European Journal of Operational Research*, 162(1):220–238, 2005.

[34] M. Gendreau, H. A., and L. G. A new insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, 40:1086–1093, 1992.

[35] M. Gendreau, A. Hertz, and G. Laporte. New insertion and postoptimization procedures for the traveling salesman problem. *Oper. Res.*, 40(6):1086–1094, Nov. 1992.

[36] P. T. Giancarlo Fortino. *Internet of Things Based on Smart Objects: Technology, Middleware and Applications*. Springer International Publishing, 2014.

[37] G. Hasle and O. Kloster. Industrial vehicle routing. In G. Hasle, K.-A. Lie, and E. Quak, editors, *Geometric Modelling, Numerical Simulation, and Optimization*, pages 397–435. Springer Berlin Heidelberg, 2007.

[38] P. V. Hentenryck. Computational disaster management. In F. Rossi, editor, *IJCAI*. IJCAI/AAAI, 2013.

[39] J. Homberger, H. Gehring, F. Hagen, L. Wirtschaftsinformatik, D-Hagen, and B. Deutschland. Two evolutionary metaheuristics for the vehicle routing problem with time windows. *INFOR*, 37:297–318, 1999.

[40] R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.

[41] G. Ioannou, M. Kritikos, and P. G. A greedy look-ahead heuristic for the vehicle routing problem with time windows. 2001.

[42] S. Irnich and D. Villeneuve. The shortest-path problem with resource constraints and k-cycle elimination for k â‰¥ 3. *INFORMS Journal on Computing*, 18(3):391–406, 2006.

[43] A. Ivaschenko. Multi-agent solution for business processes management of 5pl transportation provider. In A. Nanopoulos and W. Schmidt, editors, *S-BPM ONE - Scientific Research*, volume 170 of *Lecture Notes in Business Information Processing*, pages 110–120. Springer International Publishing, 2014.

[44] A. Ivaschenko and A. Minaev. Multi-agent solution for adaptive data analysis in sensor networks at the intelligent hospital ward. In D. ŁšlČ©zak, G. Schaefer, S. Vuong, and Y.-S. Kim, editors, *Active Media Technology*, volume 8610 of *Lecture Notes in Computer Science*, pages 453–463. Springer International Publishing, 2014.

[45] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[46] P. Kalina and J. Vokřínek. Improved agent based algorithm for vehicle routing problem with time windows using efficient search diversification and pruning strategy. In *Proceedings of the 3rd Workshop on Artificial Intelligence and Logistice (AiLog) of the 2012 European Conference on Artificial Intelligence (ECAI)*, pages 13–18, 2012.

[47] P. Kalina and J. Vokřínek. Parallel solver for vehicle routing and pickup and delivery problems with time windows based on agent negotiation. In *2012 IEEE Conference on Systems, Man, and Cybernetics (SMC)*, pages 1558–1563, 2012.

[48] P. Kalina, J. Vokřínek, and V. Mařík. The art of negotiation: Developing efficient agent-based algorithms for solving vehicle routing problem with time windows. In V. Mařík, J. Lastra, and P. Skobelev, editors, *Industrial Applications of Holonic and Multi-Agent Systems*, volume 8062 of *Lecture Notes in Computer Science*, pages 187–198. Springer Berlin Heidelberg, 2013.

[49] P. Kalina, J. Vokřínek, and V. Mařík. An efficient route minimization algorithm for the vehicle routing problem with time windows based on agent negotiation. In G. Boella, E. Elkind, B. Savarimuthu, F. Dignum, and M. Purvis, editors, *PRIMA 2013: Principles and Practice of Multi-Agent Systems*, volume 8291 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2013.

[50] P. Kalina and J. Vokřínek. Algorithm for vehicle routing problem with time windows based on agent negotiation. In *Proceedings of the 7th Workshop on Agents In Traffic and Transportation, AAMAS*, 2012.

[51] P. Kalina, J. Vokřínek, and V. Mařík. Agents towards vehicle routing problem with time windows. *Journal of Intelligent Transportation Systems*, 0(ja):null, 0.

[52] B. Kallehauge. On the vehicle routing problem with time windows, a thesis. 2006.

[53] B. Kallehauge, J. Larsen, and O. B. G. Madsen. Lagrangian duality applied to the vehicle routing problem with time windows. *Comput. Oper. Res.*, 33(5):1464–1487, May 2006.

[54] M. Karlsson, F. Ygge, and A. Andersson. Market-based approaches to optimization. *Computational Intelligence*, 23(1):92–109, 2007.

[55] J. E. Kelley. The cutting plane method for solving convex programs. *Journal of the SIAM*, 8:703–712, 1960.

[56] P. Kilby, P. Prosser, and P. Shaw. A comparison of traditional and constraint-based heuristic methods on vehicle routing problems with side constraints. *Constraints*, 5(4):389–414, 2000.

[57] G. King and C. Mast. Excess travel: Causes, extent and consequences. In *Transportation Research Record 1111*, pages 126–134, 1997.

[58] N. Kohl, J. Desrosiers, O. B. G. Madsen, M. M. Solomon, and F. Soumis. 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, (1):101–116.

[59] R. Kohout and K. Erol. In-time agent-based vehicle routing with a stochastic improvement heuristic. In *11th Conference on Innovative Applications of Artificial Intelligence*. AAAI/MIT Press, 1999.

[60] A. Komenda, P. Novák, and M. Pěchouček. Domain-independent multi-agent plan repair. *Journal of Network and Computer Applications (in print)*, 2013.

[61] A. Komenda, J. VOKRINEK, M. Cap, and M. Pechoucek. Developing multiagent algorithms for tactical missions using simulation. *IEEE intelligent systems*, 28(1):42–49, 2013.

[62] A. Komenda, J. Vokřínek, and M. Pěchouček. Plan representation and execution in multi-actor scenarios by means of social commitments. *Web Intelligence and Agent Systems*, 9(2):123–133, march 2011.

[63] E. Koutsoupias and C. H. Papadimitriou. Worst-case equilibria. *Computer Science Review*, 3(2):65–69, 2009.

[64] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):pp. 497–520, 1960.

[65] A. R. Leite, F. Enembreck, and J.-P. A. BarthÃ¨s. Distributed constraint optimization problems: Review and perspectives. *Expert Systems with Applications*, 41(11):5139 – 5157, 2014.

[66] J. K. Lenstra and A. H. G. R. Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227, 1981.

[67] H. W. Leong and M. Liu. A multi-agent algorithm for vehicle routing problem with time window. In *Proceedings of the 2006 ACM symposium on Applied computing*, SAC '06, pages 106–111, New York, NY, USA, 2006. ACM.

[68] H. Li and A. Lim. Local search with annealing-like restarts to solve the vrptw. *European Journal of Operational Research*, 150(1):115 – 127, 2003. <ce:title>O.R. Applied to Health Services</ce:title>.

[69] A. Lim and X. Zhang. A two-stage heuristic with ejection pools and generalized ejection chains for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 19(3):443–457, 2007.

[70] S. Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269, 1965.

[71] F.-H. Liu and S.-Y. Shen. The fleet size and mix vehicle routing problem with time windows. *Operational Research Society*, 50:721–732, 1999.

[72] Q. Lu and M. M. Dessouky. A new insertion-based construction heuristic for solving the pickup and delivery problem with hard time windows. *European Journal of Operational Research*, 175:672–687, 2005.

[73] D. Mester. An evolutionary strategies algorithm for large scale vehicle routing problem with capacitate and time window restrictions.

[74] D. Mester and O. Bräysy. Active guided evolution strategies for large-scale vehicle routing problems with time windows. *Comput. Oper. Res.*, 32(6):1593–1614, June 2005.

[75] Y. Nagata. Edge assembly crossover for the capacitated vehicle routing problem. In *Proceedings of the 7th European conference on Evolutionary computation in combinatorial optimization*, EvoCOP'07, pages 142–153, Berlin, Heidelberg, 2007. Springer-Verlag.

[76] Y. Nagata, O. Bräysy, and W. Dullaert. A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. *Comput. Oper. Res.*, 37(4):724–737, Apr. 2010.

[77] Y. Nagata and O. Bräysy. A powerful route minimization heuristic for the vehicle routing problem with time windows. *Operations Research Letters*, 37(5):333–338, 2009.

[78] D. Nau, M. Ghallab, and P. Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[79] Y.-K. Ng. *Welfare Economics: Towards a More Complete Analysis*. Springer Berlin Heidelberg, 2004.

[80] R. Nissim and R. I. Brafman. Multi-agent a* for parallel and distributed systems. In *AAMAS*, pages 1265–1266, 2012.

[81] S. I. Norway. Transportation optimization portal — problems. [Online at http://www.sintef.no/Projectweb/TOP/Problems; accessed 10-May-2013].

[82] I. Or. *Traveling Salesman-Type Combinatorial Problems and their relation to the Logistics of Regional Blood Banking*. PhD thesis, Northwestern University, 1976.

[83] I. H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Ann. Oper. Res.*, 41(1-4):421–451, May 1993.

[84] V. Pareto. *Manual of political economy*. Kelley Publishers, New York, NJ, USA, 1971.

[85] M. Pechoucek, J. Vokrinek, and P. Becvar. Explantech: multiagent support for manufacturing decision making. *Intelligent Systems, IEEE*, 20(1):67–74, 2005.

[86] D. Perugini, S. Jarvis, S. Reschke, and D. Gossink. Distributed deliberative planning with partial observability: Heuristic approaches. In *Proceedings of the Fourth International Conference on Knowledge Systems for Coalition Operations (KSCO-2007). In Proceedings of the IEEE International Conference on Integration of Knowledge Intensive Multi-Agent Systems Modeling, Evolution and Engineering (KIMAS-2007)*, Waltham, MA, USA, May 2007.

[87] A. Petcu and B. Faltings. S-DPOP: Superstabilizing, Fault-containing Multiagent Combinatorial Optimization. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-05*, pages 449–454, Pittsburgh, Pennsylvania, USA, July 2005. AAAI.

[88] P. Poon and J. Carter. Genetic algorithm crossover operators for ordering applications. *Computers and Operations Research*, 22(1):135 – 147, 1995.

[89] J.-Y. Potvin and J.-M. Rousseau. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66(3):331–340, 1993.

[90] J.-Y. Potvin and J.-M. Rousseau. An exchange heuristic for routeing problems with time windows. *Journal of the Operational Research Society*, 46:1433–1446, 1995.

[91] E. Prescott-Gagnon, G. Desaulniers, and L.-M. Rousseau. A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. *Netw.*, 54(4):190–204, Dec. 2009.

[92] I. Rechenberg. *Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution.* Frommann-Holzboog, 1973.

[93] Y. Ren, M. Dessouky, and F. Ordóñez. The multi-shift vehicle routing problem with overtime. *Comput. Oper. Res.*, 37(11):1987–1998, Nov. 2010.

[94] F. Rossi, P. v. Beek, and T. Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence).* Elsevier Science Inc., New York, NY, USA, 2006.

[95] L.-M. Rousseau, M. Gendreau, and G. Pesant. Using constraint-based operators to solve the vehicle routing problem with time windows. *Journal of heuristics*, 8(1):43–58, 2002.

[96] L.-M. Rousseau, M. Gendreau, G. Pesant, and F. Focacci. Solving vrptws with constraint programming based column generation. *Annals OR*, 130(1-4):199–216, 2004.

[97] R. A. Russell. Hybrid heuristics for the vehicle routing problem with time windows. *Transportation Science*, 29(2):156–166, 1995.

[98] M. Savelsbergh. The vehicle routing problem with time windows: Minimizing route duration. *ORSA Journal on Computing*, 4(2):146–154, 1992.

[99] P. Scerri, A. Farinelli, S. Okamoto, and M. Tambe. Allocating tasks in extreme teams. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, pages 727–734, New York, NY, USA, 2005. ACM.

[100] G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, and G. Dueck. Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 159(2):139 – 171, 2000.

[101] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In M. Maher and J.-F. Puget, editors, *Principles and Practice of Constraint Programming*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431. Springer Berlin Heidelberg, 1998.

[102] O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1-2):165 – 200, 1998.

[103] Y. Shoham, K. Leyton-brown, and T. Jude. Multiagent systems: Algorithmic, gametheoretic, and logical foundations, 2009.

[104] M. P. Singh, A. S. Rao, and M. P. Georgeff. *Multiagent Systems A Modern Approach to Distributed Artificial Intelligence*, chapter Formal Methods in DAI: Logic Based Representation and Reasoning, pages 201–258. MIT Press, Cambridge, MA., 1999.

[105] R. G. Smith. The contract net protocol: High level communication and control in a distributed problem solver. *In IEEE Transactions on Computers*, C-29(12):1104–1113, 1980.

[106] M. M. Solomon. On the worst-case performance of some heuristics for the vehicle routing and scheduling problem with time window constraints. *Networks*, 16(2):161–174, 1986.

[107] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35:254–265, 1987.

[108] K. S'orensen, M. Sevaux, and P. Schittekat. Multiple neighbourhood search in commercial vrp packages: Evolving towards self-adaptive methods. In C. Cotta, M. Sevaux, and K. SĂ¶rensen, editors, *Adaptive and Multilevel Metaheuristics*, volume 136 of *Studies in Computational Intelligence*, pages 239–253. Springer Berlin Heidelberg, 2008.

[109] M. Stolba and A. Komenda. Fast-forward heuristic for multiagent planning. 2013.

[110] E. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J. Y. Potvin. A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows. *Transportation Science*, 31(2):170–186, 1997.

[111] J. Takahashi, R. Kanamori, and T. Ito. A preliminary study on anticipatory stigmergy for traffic management. In *Proceedings of the The 2012 IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technology - Volume 03*, WI-IAT '12, pages 399–405, Washington, DC, USA, 2012. IEEE Computer Society.

[112] S. Thangiah. Vehicle routing with time windows using genetic algorithms. Technical report, Computer Science Department, Slippery Rock University, Slippery Rock, PA, 1993.

[113] P. M. Thompson and H. N. Psaraftis. Cyclic transfer algorithms for multivehicle routing and scheduling problems. *Oper. Res.*, 41(5):935–946, Sept. 1993.

[114] M. Vallati. A guide to portfolio-based planning. In C. Sombattheera, N. Loi, R. Wankar, and T. Quan, editors, *Multi-disciplinary Trends in Artificial Intelligence*, volume 7694 of *Lecture Notes in Computer Science*, pages 57–68. Springer Berlin Heidelberg, 2012.

[115] M. Vasirani, F. KlĂĽgl, E. Camponogara, and H. Hattori. Editorial: Special issue on intelligent agents in traffic and transportation. *Journal of Intelligent Transportation Systems*, 0(ja):null, 0.

[116] J. Vokřínek, J. Bíba, J. Hodík, and J. Vybíhal. The rbvo formation protocol. *International Journal of Agent-Oriented Software Engineering*, 3(2/3):135–162, 2009. INDERSCIENCE ENTERPRISES LTD, World Trade Center BLDG.

[117] J. Vokřínek, J. Bíba, J. Hodík, J. Vybíhal, and P. Volf. Rbvo formation protocol. In *2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2007)*, pages 454–457, 2007.

[118] J. Vokřínek, A. Komenda, and M. Pěchouček. Decommitting in multi-agent execution in non-deterministic environment: experimental approach. In C. Sierra, C. Castelfranchi, K. S. Decker, and J. S. Sichman, editors, *8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), Budapest, Hungary, May 10-15, 2009, Volume 2*, pages 977–984. IFAAMAS, 2009.

[119] J. Vokřínek, A. Komenda, and M. Pěchouček. Cooperative agent navigation in partially unknown urban environments. In *Proceedings of the 3rd International Symposium on Practical Cognitive Agents and Robots*, pages 41–48. ACM, 2010.

[120] J. Vokřínek, J. Hodík, J. Bíba, J. Vybíhal, and M. Pěchouček. Competitive contract net protocol. In *LNCS 4362 – SOFSEM 2007: Theory and Practice of Computer Science*, pages 656–668, Berlin, 2007. Springer-Verlag.

[121] J. Vokřínek, A. Komenda, and M. Pěchouček. Abstract architecture for task-oriented multiagent problem solving. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 41(1):31 –40, January 2011.

[122] J. Vokřínek. *Distributed Problem Solvingby Means of Agent Negotiation.* Phd thesis, Agent Technology Center, Czech Technical University, CTU, 2005.

[123] C. Voudouris and E. Tsang. Guided local search. Technical report, European Journal of Operational Research, 1995.

[124] F. Wang, Y. Tao, and N. Shi. A survey on vehicle routing problem with loading constraints. In *Proceedings of the 2009 International Joint Conference on Computational Sciences and Optimization - Volume 02*, CSO '09, pages 602–606, Washington, DC, USA, 2009. IEEE Computer Society.

[125] G. Weiss, editor. *Multiagent systems: a modern approach to distributed artificial intelligence.* MIT Press, Cambridge, MA, USA, 1999.

[126] M. Wooldridge. *Reasoning about Rational Agents.* Intelligent robotics and autonomous agents. The MIT Press, 2000.

[127] wordle.com. Word cloud generator. [Online at http://www.wordle.com; accessed 5-July-2014].

[128] M. Y. V. Yoram Moses, Joseph Y Halpern. *Reasoning About Knowledge.* MIT Press, 2004.