Master's degree thesis

LOG950 Logistics

Solution methods for combined scheduling and transportation problems

Markus Brachner

Number of pages including this page: 97

Molde, 28.05.2013

Molde University College

Mandatory statement

Each student is responsible for complying with rules and regulations that relate to examinations and to academic work in general. The purpose of the mandatory statement is to make students aware of their responsibility and the consequences of cheating. Failure to complete the statement does not excuse students from their responsibility.

*Please complete the mandatory statement by placing a mark **in each box** for statements 1-6 below.*		
1.	**I/we herby declare that my/our paper/assignment is my/our own work, and that I/we have not used other sources or received other help than is mentioned in the paper/assignment.**	☒
2.	**I/we herby declare that this paper** 1. Has not been used in any other exam at another department/university/university college 2. Is not referring to the work of others without acknowledgement 3. Is not referring to my/our previous work without acknowledgement 4. Has acknowledged all sources of literature in the text and in the list of references 5. Is not a copy, duplicate or transcript of other work	Mark each box: 1. ☒ 2. ☒ 3. ☒ 4. ☒ 5. ☒
3.	**I am/we are aware that any breach of the above will be considered as cheating, and may result in annulment of the examinaion and exclusion from all universities and university colleges in Norway for up to one year, according to the Act relating to Norwegian Universities and University Colleges, section 4-7 and 4-8 and Examination regulations** section 14 and 15.	☒
4.	**I am/we are aware that all papers/assignments may be checked for plagiarism by a software assisted plagiarism check**	☒
5.	**I am/we are aware that Molde University college will handle all cases of suspected cheating according to prevailing guidelines.**	☒
6.	**I/we are aware of the University College`s rules and regulation for using sources**	☒

Publication agreement

ECTS credits: 30

Supervisor: Johan Oppen

Agreement on electronic publication of master thesis

Author(s) have copyright to the thesis, including the exclusive right to publish the document (The Copyright Act §2).
All theses fulfilling the requirements will be registered and published in Brage HiM, with the approval of the author(s).
Theses with a confidentiality agreement will not be published.

I/we hereby give Molde University College the right to, free of charge, make the thesis available for electronic publication: ☒**yes** ☐**no**

Is there an agreement of confidentiality? ☐**yes** ☒**no**
(A supplementary confidentiality agreement must be filled in)
- If yes: **Can the thesis be online published when the period of confidentiality is expired?** ☐**yes** ☐**no**

Date: 28.05.2013

Preface

My mother supported me in a way that is beyond words and exceeds parental care and commitment by far to make my studies here in Norway possible. There is but one way to thank her, and that is to dedicate this thesis to her – which I herewith do.

I can remember, when it still was time to decide about the topic of the thesis, Johan Oppen was giving a lecture as part of the course in Research Design. Actually, I was already pretty sure about my topic at this time, and working on the concept. And then, I can also remember quite well his last words of his talk: "If someone of you will ask me if I have a topic, I will most probably say, no, I don't. There is only one possibility, but it could be a challenging one, and it may require some coding."

What I can not remember, is the reason, why I was asking him about details after the lecture. The topic, which I had in mind at that time, was actually quite promising and could probably have opened some doors to the oil and gas industry. Maybe the only "right" reason in our meritocratic society, which conforms to a recruitment consultant's pattern of thought would be: "Because I love challenges!!" (Mind the two exclamation marks). This was not the case. I presume it was much more the pitiful way in which Johan spoke out the words. At least I felt pity, when hearing about the background of this topic: It was based on a thesis, which has been done earlier, and its author put a great deal of effort into it. A lot of preliminary work was done, and the concepts were interesting as well. However, due to time restrictions and complexity of the problem the author had to finish half way. If nobody would have continued, all the efforts of this student would have been in vain.

Now, as I am writing the last lines in this work, I can affirm, that by no means I regret the decision of having taken this topic. It proved to be a rough diamond and just needed a bit of polishing. Suddenly, a blunt problem of asphalt transport and team scheduling has become part of something which is currently a hot topic in VRP, and Johan is just on the way to Germany to present the findings of this thesis at the 1st International Workshop on Synchronisation in Transport, the SynchroTrans 2013.

I want to thank Johan Oppen for supervising me, for showing me how to become a real academic, and especially for continuing the work on this topic. Furthermore, thanks go to Anastasia Rubasheuskaya for her work, which was the basis for my thesis. Last, but not least, I want to thank Olga Sergeeva. I am in the fortunate position of having her not only as beloved girlfriend, but also as a tough and savvy sparring partner. May our future be as fruitful and exciting as it was so far.

Abstract

Weather conditions are hard and remorseless in Norway, and they take their toll on the roads. This is one of various reasons for a multitude of construction and maintenance work on the Norwegian road network. Road construction companies are challenged in the planning of their projects, as they need to coordinate teams which conduct the work on-site on one hand and trucks that provide the necessary asphalt at the locations at the right time on the other. Moreover, at the end of the working day everybody involved should be back at the depot at about the same time and as early as possible.

This thesis presents a VRP with exact temporal and spatial operation synchronization. Two classes of vehicles are synchronized, where a vehicle of one class needs to meet one of another class to proceed on its tour. This case can be found in many applications in practice. Examples include the coordination of construction teams and supplying vehicles for construction companies, the synchronization of special purpose tools with repair teams at offshore oil drilling platforms or the planning of routes for combine harvester and trucks for harvest collection.

In contrast to minimizing the sum of travel times of each vehicle, the objective is to minimize the longest of all tours. This way all tours will be fair, keeping both the differences between the tours small and the travel time as low as possible. Despite the good applicability in practice, min-max VRPs are quite rarely researched, and there has not been paid very much attention the past years to this kind of objectives.

A formulation as a linear program shows, that for practical applicability even small instances entail too long run times. As a consequence, a heuristic solution method based on a multistart greedy algorithm was developed. When it comes to synchronization, one challenge is to construct good and feasible solutions. Thus, we discuss the possibilities of efficient construction algorithms and the impact on the search process. Furthermore, particular attention is paid to the solution evaluation. The waiting times to synchronize the vehicles depend on the concrete solution and are therefore difficult to calculate. As an approach, the idea of discrete event simulation is proposed. This helps to model more complex problems while still ensuring good computational performance. An implementation of the proposed approach in C++ and an analysis of the computational results is discussed.

Contents

1. Introduction

For more than 50 years, intense research has been done in the field of vehicle route planning and scheduling, and very sophisticated methods have been developed to solve huge instances. Commercial vehicle routing software systems have become an indispensable tool for dispatchers and other decision makers (Drexl 2012a). This is also due to the paradigm shift from optimality to reality[1], which was heralded by works of Holland (1975), Christofides (1976), and others, but undoubtedly in effect at latest, when Fred Glover stated:

> *"In the face of combinatorial complexity, there must be freedom to depart from the narrow track that logic single-mindedly pursues, even on penalty of losing the guarantee that a desired destination will ultimately be reached. ('Ultimately' may be equivalent to 'a day before doomsday' from a practical standpoint.)"* (Glover 1986, p. 534)

The best solution is not a must anymore, the high aims of optimality are sacrificed to bigger size and complexity of the problems, which are much more near to reality in

[1] The term "paradigm-shift" caused considerable controversy in the proposal of this master thesis, as some found this being a too strong expression. However, the author wants to aver that it can well be named in this way. The Oxford English Dictionary defines the term "Paradigm" as "a typical example or pattern of something; a pattern or model", and mentions as an example "the society's paradigm of the 'ideal woman' ". The paradigm of an "ideal solution" in this case is exactly the analogous meaning, considering that an ideal solution does not necessarily need to be optimal.

Furthermore, Barker (1992) refers to a paradigm as "a set of rules and regulations (written or unwritten) that does two things: (1) it establishes or defines boundaries; and (2) it tells you how to behave inside the boundaries in order to be successful." This definition perfectly matches the described paradigm shift as new boundaries and rules have been established in this field, and optimality was not a criterion for success anymore. Moreover, Ferguson (1980) describes a paradigm shift as "... a distinctly new way of thinking about old problems ... it throws open doors and windows for new explorations."

The Encyclopedia of Science and Religion (Van Huyssteen 2003, s.v. "Paradigms") says the following, referring to Thomas S. Kuhn's seminal work "The Structure of Scientific Revolutions" (Kuhn 1996): "A paradigm consists of a group of fundamental assumptions forming a shared framework that provides the scholar with instruction on what to view as issues of inquiry and how to deal with these issues. Hence, a paradigm works as a criterion for choosing problems that, as long as the paradigm is taken for granted, can be assumed to have a solution."

their formulations, but non-optimal. However, while human mind is tempted to end its search as soon as a feasible solution is found, these software systems proceed and can find better, even near-to-optimal solutions.

Particularly in the field of combined problems, where one sub-problem constraints the other, only few and special cases of solvers can be found. This field has received growing attention in the past few years, which in the authors opinion is not only a consequence of better algorithms, but also more computational power, which allows the scientific community to extend present problem formulations. The intended thesis focuses on such a case and will try to contribute to this field by expanding the toolbox of solution methods. This kind of problems can be found in many different situations, where goods and services need to be provided at the customer's location and the teams in the field need to be provided with resources regularly.

This chapter will outline the general idea of the problem, introduce the terminology, and illustrate the research design. Furthermore, the current state of research in this field will be described by analyzing and referring to scientific literature which is connected to the covered topics.

1.1. Research design

This thesis tries to answer a series of questions connected to synchronized VRP, which are formulated as follows:

1. What are the specifics of synchronized VRP problems?
2. One representative real-world problem was chosen. What approaches are practical to solve the described problem?
3. Is Discrete Event Simulation a practical method to evaluate solutions?
4. How do the proposed approaches perform with different parameters and instances?

The research followed the following steps (see Figure 1.1):

- Literature research and determination of possible solution methods

  Literature research included analyzing the originating thesis of Rubasheuskaya (2012) and scanning literature, particularly scientific articles, for similar problems. Basic literature will be Pinedo (2012) and Toth and Vigo (2002).

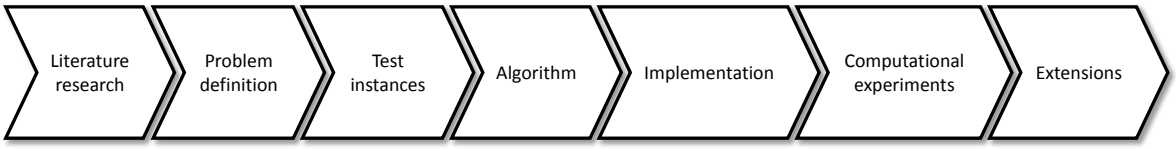

Figure 1.1.: Research process

- Problem definition

  The problem was defined, including a prosaic description and a mathematical formulation. This formulation describes the constraints of the problem and is also necessary for defining the limits of the problem, to clarify, what is not included in the model.

- Generation of test instances

  The generation of test instances needed some attention, as it should represent different aspects of real world conditions. Various types of instances regarding project size were generated, having in mind, that this property could have an influence on the computational time and solution quality. In order to simplify matters, a parametrizable script was created, which could take over this task.

- MIP implementation

  It was desirable to have optimal values for some test instances. Therefore, a first approach of solving the problem was to implement it as a MIP model and to solve it to optimality. However, even before implementing the model, literature review had already indicated, that the size of instances, which can be solved optimally, would not be very large, which proved to be true later on empirically.

- Formulation of algorithm

  As a next step, larger instances needed to be solved. Literature review showed, that this could be a kind of heuristic or meta-heuristic. Possible solution methods were determined and formulated in pseudo-code.

- Implementation

  A reference implementation in C++ was developed to show the proof of concept, to determine the algorithm, and to refine it. Furthermore, this implementation provided a basis for further analysis and experiments.

- Computational experiments

  The influence of problem size, parameters, and instance structure on the run-time, solution structure and solution quality will be evaluated. The author is aware of the concept of statistical significance and was following guidelines of Greenberg (1990), Barr et al. (1995), Coffin and Saltzman (2000), and Hall and Posner (2001) in mind, which had been found during literature review.

- Extension of the problem

  The basic problem discussed in this thesis mainly focuses on the issue of synchronization. However, in Rubasheuskaya (2012) the problem was somewhat richer. Thus, some aspects of extending the model were covered in addition.

The structure of the thesis follows a scheme, which closely relates to the conducted research process. Thus, the chapters build on each other and conclusions are drawn, which are relevant for the next step. Furthermore, the thesis does not split into theory and practice, as found in some other works, but discusses and develops theoretical and practical aspects as needed. In the author's opinion this makes the research process more transparent and reproducible.

1.2. State of research and literature review

This section serves two purposes. First, it should give an overview about the current state of research. Second, it gives ideas about different solution approaches, which can be taken into account to solve the problem at hand. Solution strategies can be found in several fields of research, depending on how the problem is constructed. Different possibilities exist, to reshape the problem formulation, in order to fit it to similar problems, where research has been done already. This formulation seems to be quite critical. As an example, Beck, Prosser, and Selensky (2003) were investigating the performance of techniques of solving a routing problem, when they applied VRP solution techniques in comparison to scheduling techniques. They concluded that, as soon as the problem becomes richer, scheduling techniques outperform the VRP techniques.

The problem can be divided into two stages with two different sub-problems of scheduling and VRP, for each of which extensive research has been done over the past decades. The scheduling of the teams can be seen as a job-shop scheduling problem. Cheung and Zhou (2001) proposed a genetic algorithm combined with heuristic rules to solve a general job-shop scheduling problem with sequence-dependent setup times. This method focuses on production planning, where the computational time is of minor importance, but still reasonable. However, it gives considerably better results compared to plain heuristic methods. Vela, Varela, and González (2010) used a genetic algorithm as well and combined it with a local search heuristic. Artigues and Feillet (2008) presented a Branch-and-Bound method.

VRP problems have been extensively researched in the past decades as well. Cordeau et al. (2002) give an overview over some of the most important classical and modern vehicle routing heuristics and assesses them in regards to accuracy, speed, simplicity and flexibility. To mention two approaches, Cordeau, Laporte, and Mercier (2001) used a tabu search heuristic for vehicle routing problems with time windows and can also solve periodic and multi-depot problems with this approach. However, Prins (2004) applied a genetic algorithm combined with local search to solve VRP problems.

There has been done some research on coordinating similar sub-problems. Liu (2011) presented an approach for sequencing jobs and delivering the completed jobs to the customer in batches. They proposed two genetic algorithms, which can find near-to-optimal solutions within an acceptable amount of computational time. Bredström and Rönnqvist (2008) studied a combined vehicle routing and scheduling problem with temporal precedence and synchronization constraints and proposed an optimization based heuristic, i.e. an approach, where significantly restricted MIP problems are solved iteratively with progressively improved solutions. Kim, Koo, and Park (2010) dealt with a combined team scheduling and vehicle routing problem. Teams and trucks should be scheduled separately, but they needed to be synchronized. A constructive heuristic was proposed to solve this problem, however a particle swarm optimization scheme was applied to find the parameters.

A promising approach is, to formulate the problem as a VRP problem with synchronization constraints. Drexl (2012b) classified synchronized VRP problems into different types and discussed the exact and heuristic approaches of solving such a class of problems. This paper is based on a much more extensive technical report (see Drexl 2011).

It is reasonable to presume, that possible solution methods can be found as well in the field of Integrated Production and Outbound Distribution Scheduling, which is a

rather recent and rapidly growing field. Chen (2010) provided a survey of such existing models. As an example of such class of problems, Cakici, Kurz, and Mason (2010) worked on a combined scheduling and transportation problem, where goods are produced and delivered by vehicles of limited capacity, making a trade-off between total weighted tardiness and transportation cost. The presented approach gives good hints how to coordinate two different sub-problems.

Durbin and Hoffman (2008) presented a solution for scheduling concrete deliveries to customers, which are of interest, as these deliveries are not only critical in terms of time windows, but also because of the perishable nature of concrete, which can be also an important factor for the problem at hand. Their approach was to use a time-space-network representation.

1.3. Problem definition

The problem originates from a past master's degree thesis. Rubasheuskaya (2012) formulated in her thesis a mathematical model of a road construction company, with the objective to provide an optimal project execution plan within a defined planning horizon. This plan needs to take two sub-problems into account.

The first problem is a scheduling problem, where a team is scheduled to execute project tasks at different locations. The setup times are sequence dependent as the teams need different times to move from one location to the next.

The second problem is dependent on the solution of the first problem. While the team is executing their tasks at the location, they need to be supplied with asphalt repeatedly in a defined interval. A visualization of this problem can be found in Figure 1.2. The dashed line represents Team 1, the dash-dotted line Team 2. Both teams are assigned a set of projects with the objective to minimize the maximum execution time for the longest tour among them. This objective will therefore lead to almost equal work plans for each team.

While the teams are working at the locations they need to be supplied regularly within this time interval (shown by a dotted line, which, however, is dependent on the current time). As an example, we could define a fixed interval of 20 minutes of supplying the teams. Consequentially, material needs to be delivered for Team 1 at location 3 at 8:00, 8:20, 8:40, ..., 10:40, at location 4 from 12:00 on and at location 5 from 15:00 on. Team 2, however, should be supplied at location 1 at 8:00, 8:20, ..., 13:40, and at location 2

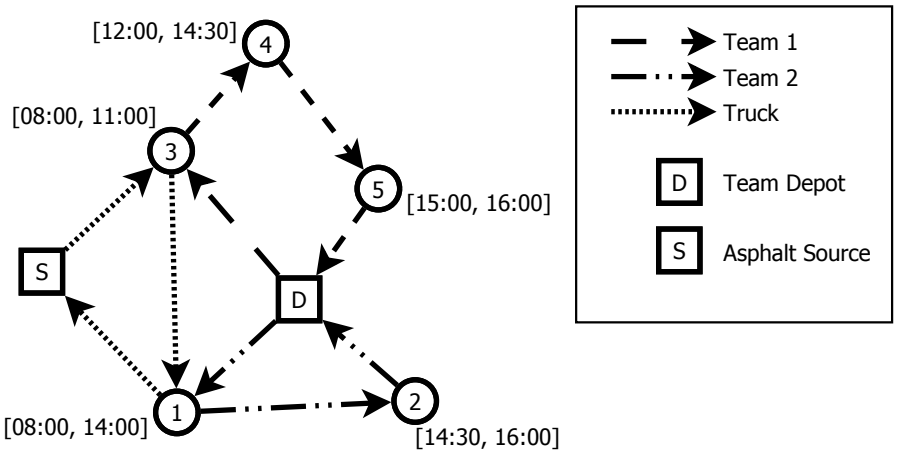

Figure 1.2.: Problem visualization. The route of the truck changes over time.

from 14:30 on. As the supplying trucks have limited capacity, the tours need to include a source for the material, where they will go, when they are empty.

1.4. Assumptions

This problem contains many different aspects, and as the time for this thesis was a hard constraint, the only possibility to fit into the timeline was to clearly define and limit the scope to a manageable amount. The part, which was most interesting from the author's point of view was the aspect of synchronization, which is not well researched and thus has potential to provide new findings. Therefore the original problem was seen more as an inspiration for a new class of vehicle routing problems than a real world problem that needs to be solved. A multitude of assumptions facilitated to concentrate on the synchronization aspect.

As a first assumption the asphalt trucks will have enough capacity to fulfill all demands on their tour. In other words, there is no capacity restriction on the trucks. For teams this restriction would not apply anyway. This assumption is maybe the most problematic one, however it can lead to unsolvable instances, because the number of vehicles is given and multiple use of one vehicle is not intended to be built into this problem.

Furthermore, we will assume that the speed of teams and trucks to move from one node to another is constant and independent from any factors. In a real world problem, the load of a truck would influence the speed, as a fully loaded truck will be slower than an empty truck. Above all, it is assumed that teams and trucks move with the same speed. We also assume that all teams work with the same speed, and there is

no dependency on quantity or quality of workers. There is also no service time for the trucks to unload the asphalt.

Moreover, when it comes to distances, we will assume an 2-dimensional Euclidean space. The distance d between two points x and y is defined by the Euclidean metric on R^3

$$d(x, y) = ||x - y|| = \sqrt{\sum_{i=1}^{2}(x_i - y_i)^2}$$

As a consequence, the distance matrix between any point x and y will be symmetrical

$$d(x, y) = d(y, x)$$

This implies also that the triangle inequality holds, i.e. any distance from a point x to a point z will be smaller than the sum of distances from a node x to any other node y plus the distance from y to z:

$$d(x, z) < d(x, y) + d(y, z)$$

Another assumption is that projects only last less than a day and each project gets only one delivery of asphalt. This is true for small projects, but many projects can get quite big and even last several days, weeks or months. Clearly, in this case the projects would need to be supplied by a lot of asphalt transports.

Finally, we do not consider a planning horizon. The planning will be done for only one day. It is assumed that there is no maximum working time on this day and the work is done without any time-dependent break (only breaks which are induced by waiting for a asphalt truck or working team). There is, for example, no lunch break taken into account. If all teams and trucks did the lunch break at the same time, then this has no consequences on the solution (only on the objective value). However, the breaks could be planned individually for every team/truck within a given time-frame such that they are placed optimally in order to affect the objective value least, which is not the case.

1.5. Conventional VRP problems - definition and basic notation

This problem is accountable to the class of vehicle routing problems (VRPs). This section gives a short overview of the basic problem, terms and definitions, and some possible variations of the problem.

The basic objective of a VRP involves a *fleet* of *vehicles*, which can have the same, but also distinct characteristics. These vehicles are initially located at one starting point, called *depot*. Each node should be visited exactly one, and finally the vehicle should end up again in the depot. A solution of a VRP will give a set of routes, which will be executed by one vehicle each. Among the variants, which have received the greatest attention are the following (Toth and Vigo 2002, pp. 5):

- **Capacitated VRP (CVRP):** All the nodes represent deliveries with deterministic demands known in advance. The demand may not be split. This problem assumes identical vehicles with a defined capacity. The sum of demands on each tour must not exceed this capacity.

- **Distance-Constrained VRP(DVRP):** This is a problem similar to the CVRP, but the capacity-constraint is replaced by a constraint which limits each tour to a maximum route length.

- **VRP with Time Windows(VRPTW):** For each customer a time window and a service time is given. Along with the time instant, when the vehicle leaves the depot and the travel time to get to the customer, the service at the customer must start within the given time window. The vehicle is usually allowed to wait at the node in case of an early arrival.

- **VRP with Backhauls(VRPB):** Here some customers are marked as *linehaul customers*, where a certain demand has to be delivered, and *backhaul customers*, who need to be visited to pick up a defined amount of inbound products. In addition, if there are both types of customers on one tour, all linehaul customers need to be served before the first backhaul customer may be visited.

- **VRP with Pickup and Delivery(VRPPD):** Each customer is associated with a quantity for pickup and for delivery. Furthermore, for each customer the origin of the delivery and the destination of the pickup is given. The tours need to be constructed in a way, that

- each tour contains the depot
- each customer is visited exactly once
- the load along the tour must be nonnegative and may never exceed the vehicle capacity
- each customer must be visited after the origin of its associated delivery
- each customer must be visited before the destination of its associated pickup

It also should be mentioned, that the VRP with unlimited capacity corresponds to the multiple Traveling Salesman Problem (m-TSP). In this thesis, we will always refer to this problem as VRP and will consider the m-TSP as the aforementioned special case.

1.6. VRP with multiple synchronization constraints

While the classical VRPs have been extensively researched, the synchronized VRP constitutes an emerging field, and receives increasing attention. This section is basically a summary of Drexl (2012b), who has been very active in this field lately.

The presented problem of road construction and repair could be viewed as VRP with multiple synchronization constraints (VRPMS), because the classic VRP contains already synchronization constraints in respect to which vehicle visits which customer. Beyond that, Drexl defines the VRPMS as *"... a vehicle routing problem in which more than one vehicle may or must be used to fulfill a task"*. This implies also, that a change in one route may have effects on other routes. This issue is called the *interdependence problem*.

The following types of synchronization are named in the described paper:

1. **Task synchronization:** A task is a duty, which must be fulfilled at each location. This can be delivering or collecting supply, or as in our problem, fulfilling a service, i.e. accomplishing the street construction or repair. Task synchronization is the typical case of standard VRPs.

2. **Operation synchronization:** This type describes something, that must be performed by one class of vehicles in order to allow the execution of a task. In our case we need supply of asphalt at one location in order to fulfill the task of repairing the street. We can subdivide this type of synchronization in respect to the consideration of the temporal aspect into

a) *Pure spatial operation synchronization:* This case does not consider any temporal aspect. The only condition is therefore, that one location needs to be visited by both types of vehicles on their route.

b) *Operation synchronization with precedences:* One type of vehicle needs to be at the location always earlier or always later than the other one, optionally with some offset. This could be considered for the asphalt delivery, the truck could just go to the location before the team arrives, unload its supply, and proceed. However, the asphalt must be kept hot until it is used, and must be loaded into a spreading machine, which is a tool owned by the team.

c) *Exact operation:* Both vehicles need to be at the location at the same time in order to start fulfilling the task. This is the only way to handle the synchronization in our problem, as a synchronization with precedence would lead to the described issue.

3. **Movement synchronization:** In this case there is a non-autonomous vehicle, which can be only moved with an autonomous one. This would be applicable for a synchronization of trucks and trailers.

4. **Resource synchronization:** This type describes the fact, that different vehicles compete for common, scarce resources. Only one truck can load the supply into the spreading machine of one certain team at a certain location at a certain point in time.

Following this classification, we can therefore deal with the problem at hand as a VRPMS with task, resource, and exact operation synchronization.

1.7. Discrete optimization

This problem is attributable to the class of discrete optimization problems. This term is often used to describe optimization problems, which can be solved by a direct or indirect search through a finite set of variants (Leont'ev 2007). Well-known classes of problems in the field of discrete optimization are:

- Discrete programming
- Integer linear programming
- Packing and covering problems

- Combinatorial optimization
- Discrete geometry
- Network problems
- Path-finding problems
- Location problems

In theory an exhaustive search could solve any of these problems. However, such algorithms are very inefficient and search time grows very fast with increasing problem size. Leont'ev (2007) describes different approaches how to solve discrete optimization problems:

- **exact:** Algorithms, which return the optimal value of the problem. Approaches for algorithms are search algorithms, dynamic programming, matroid optimization, and linearization.
- **approximate:** For these class of algorithms some information about the objective value being optimized in respect to the optimal value is available.
- **heuristic:** Heuristics do not guarantee the optimal solution. They can even return a arbitrarily bad solution. However, many applications of heuristics have proven to be successful.

This master thesis deals with exact algorithms and heuristics.

1.8. Test data

In order to evaluate different approaches of solving this class of problems, data is needed, which describes concrete problems in numbers. Strong evidence can be found in literature, that the data can have considerable influence on the solution time and quality[2]. Thus the issue of data always needs to be considered in interconnection to the problem.

Following common terminology, a set of data, which describes one concrete problem, will be named *instance*. Furthermore, we will refer to *properties of an instance*, which define the general character. To name one example of a property, the *instance size*

[2]França et al. (1995), for example, found that their algorithm for a m-TSP worked much better for structured data compared to unstructured ones.

describes how many nodes the problem contains, which in the case at hand would reflect the number of road building projects.

There are different ways to get these test instances, and all of them do have their advantages and disadvantages. Furthermore, it is absolutely legitimate to mix these approaches in the research process, as long as this is reasonable. The proper approach depends on *availability* and *accuracy* of existing data, and the *universality* of the problem.

The availability of the data describes, how difficult it is to get the data. Many companies are reluctant to provide data, as there is fear that this information could fall into wrong hands. Furthermore, availability of data also concerns the case of data that is not present, inasmuch as it was not of interest so far. To name one example, the fuel consumption of each truck in a fleet was maybe not available so far. However, in order to minimize the costs of a heterogeneous fleet it can be necessary to determine this data.

Finally, the data could be available, but connected to a greater or lesser extent of data processing in order to convert it into a format, which is suitable for the solver in use. The generation of the distance matrix of a real-world VRP provides a perfect example for this. In a vast amount of cases, all the data needed is present. The addresses of customers to be served is available in an ERP database, and the distances can be obtained from maps or even routing software. However, it is quite a challenge, to convert these addresses into geographic coordinates[3], since inaccurate address data in the customer database may not match the data in the geocoding software, or the geocoding database may not contain up-to-date data. Moreover, a distance matrix needs to be generated based on the coordinates. As the distances need to be calculated for every possible connection, this can demand a great deal of effort. To solve a VRP with one depot and n customers, the number of distances to be calculated is

$$\frac{(n+1)^2 - (n+1)}{2}$$

Even with 20 projects on one day, which is a realistic order of magnitude in the case of the presented asphalt work company, and in the simpler symmetrical case, where the distance between two arbitrary points does not depend on the direction, this will be 210 different distances to be determined[4].

[3]This process is called *geocoding*. See Goldberg (2008) for further reading and best practices.

[4]One example of an asymmetrical case, where the distance between two points can be different, is a VRP in an urban area with one-way streets.

One noteworthy aspect to mention is the accuracy of data. It is very important to bear in mind, that data in ERP systems is only connected to reality by its semantics. However, this semantics very often is not the one, which is expected or needed. To formulate it a bit ironically, the final goal of an ERP is still to issue a right invoice, and this sometimes makes it very difficult, to get the data which should represent what is expected. Thus, it should not be a surprise, when the data contains negative working times for adjusting entries or two building projects at the same place and time, because costs are shared by two customers.

One last aspect to consider is the universality of the problem. The question appears, if it is reasonable to try to fully reflect all aspects of one company in the data, or if it is better to see the problem on a more abstract and generic level. The risk is prevalent to tailor the algorithm too much to one special case. However, research should never end up in consulting work for one business. Especially in a state, where a problem is not well researched, it could be a good idea to leave the specialization for later and explore the problem in a top-down approach, by beginning on the general problem and going deeper in subsequent steps.

In many cases, the problem originates from a real-world issue, which is based on a running business or an introduced process. Hence, a straight forward approach would be to use real-world data from historic records, for example extracts from ERP systems. This approach can ensure, that the actual problem is solved and the algorithm is suitable to solve similar problems. Furthermore, this data gives the opportunity to compare the actual outcomes to the theoretical solutions of the optimization. For example, in case of the road construction company it would be possible to extract one week of projects and compare the actual costs to the theoretical costs of a solution gained by applying the optimization algorithm.

However, even though this approach seems quite reasonable, it has some serious drawbacks, when it comes to the described issues of availability and accuracy. Moreover, to gain statistically significant results, it is not sufficient to work with just one instance. Several instances, which should be quite similar to each other in their properties, should be tested to gain valid results, however, there should be also instances which differ in their properties to see the behavior of the algorithm in different circumstances.

Hence, yet another approach is to generate data by a randomized procedure, by applying building rules, that reflect the circumstances and special properties of the problem to be solved. This is a much easier approach, as it allows to vary the different building parameters as needed, and to build an arbitrary number of instances without any need

to request, process or clean data. Special attention should be paid to generate representative instances, i.e. test instances, that contain reasonable data and have properties, that could be an instance of real-world data. A computer program can easily execute this procedure on demand.

Especially in the early stage of algorithm-development it is very reasonable to make the instances "as small as possible, but no smaller". As shown later, the solution time for a MIP implementation increases very fast with instance size. Waiting even one minute or more will impede developing and testing the implementation and is not reasonable, if it can be done with equal results using a small instance of 4 nodes within a split second. The same applies for a heuristic solver, which could initially provide solutions that are very far from the optimality. On the other hand, the proof of concept can be made already with very small instance sizes. However, after being finished with developing and testing the model, the subsequent discussion should provide information about the influence of the instance size on solution time and quality. To have instances of different sizes available for testing is therefore inevitable throughout the development process.

For the problem at hand, data was generated by a program instead of using real-world data. There were several factors supporting this decision: First, as already indicated, the synchronized VRP is not well researched and the issue of the contemplated company was just an inspiration for the problem. It was not an immediate goal to provide the company with a ready-made planning tool. However, it was much more of interest, how instances with different properties will influence the solution quality and time, which can be much easier achieved by generating instances while varying these properties. Furthermore, as pointed out, it is advisable to have small instances in an initial stage, which cannot be real-world data anyway. Moreover, concerns regarding accessibility and accuracy of company data and the related effort of coordination did not justify the use of real-world data, as this turned out to be disproportionate to the gained benefit.

The test instances contain the following data:

- **Node identifier:** An unique identifier for referencing the node.
- **Distance matrix:** This matrix represents the number of time-units to get from one node to another.
- **Project time:** This data describes, how many time-units the team will work at one node until it proceeds to the next node

Instances are considered to be euclidean, which makes it easier to present and analyze the solutions. The nodes are represented in an 2-dimensional Cartesian coordinate system, and the distance matrix is calculated before the optimization procedure starts.

To regulate the instance size, a slightly different approach to common practice was chosen. Instead of generating various instance sizes all instances were generated with the same size (100 nodes). However, the number of projects to be planned could be determined by a parameter in the solver. This made it much easier to discuss the influence of the instance size on the run-time, and gave the opportunity to be much more flexible in computational experiments.

Figure 1.3 shows one possible test instance. 0 marks the team-depot and 1 denotes the truck-depot. The numbers represent the projects to be done, with the duration of the project as a subscript.

ID	X	Y	Duration
0	315	113	–
1	241	67	–
2	144	85	71
3	291	11	70
4	220	128	306
5	381	114	213
6	297	165	454
7	78	55	143
8	55	107	446
9	171	199	331
10	163	173	245

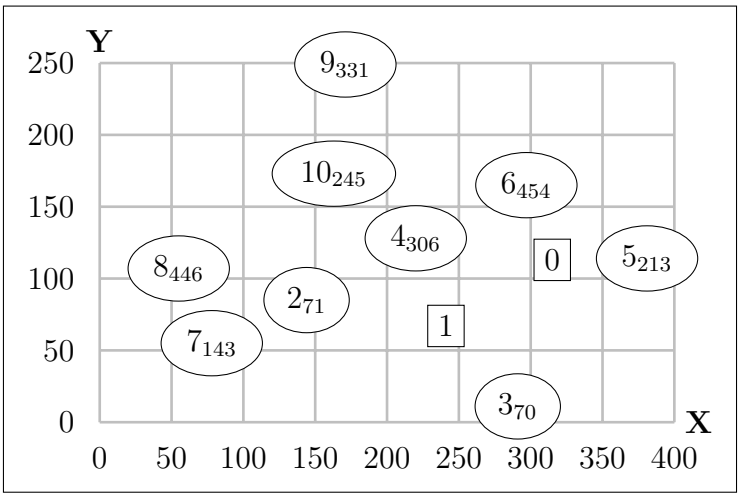

Figure 1.3.: A test instance in its numerical and graphical representation

2. Solving to optimality - a classical approach

To solve this problem, means of classical optimization methods suggest themselves. The model is easy to formulate, and the solution can be left to widely proven methods, which are implemented in a black-box style. The inputs, outputs and requirements are clearly defined, but the internal mechanics are not of particular interest for the modeler.
In this chapter, a Mixed Integer Programming model is formulated, which subsequently is implemented in a modeling language, where it can be sent to a solver. These are the prerequisites to discuss this approach and to point out the benefits, but also limits of this method.

2.1. The min-max objective as an alternative to min-sum

Before presenting the model formulation, one important issue of this problem should be discussed, to wit, the objective. Against common practice, a min-max objective was chosen in favor to min-sum in order to keep the tours as short as possible, but balanced. While the min-sum problem minimizes the sum of all tour costs (or time, or distance), the min-max objective minimizes the longest of all tours, as explicated later in more detail. The min-max VRP has received considerable less attention in comparison to the min-sum problem. This may be not very surprising due to the objective's nature, because at first sight it seems always more attractive to go for minimal costs instead. The sum of the costs of tours of a min-max problem, however, can never be less than the corresponding min-sum problem. However, based on his experience from practice, the author wants to take up the cudgels on behalf of the min-max objective for several reasons. This chapter will introduce the reader to the motivation to choose min-max objectives, and expand on how to solve these kind of problems.

In a lot of cases, the min-sum problem is acting on wrong assumptions and ignores important facts. First, we have to face the fact, that it is very unlikely, that an optimal solution will be exactly executed as it was calculated. In the field of optimization the base is always a model, and as the statistician George Box said, *"...all models are wrong, but some are useful"* (Box and Draper 1987, p. 424). As a result, there will never be the best solution for the given problem, but only an optimal solution of its formulated model. Too many factors can not be taken into account and changes need to be bargained for. As outlined by Drexl (2012a), commercial vehicle routing systems will continue to act as decision *support* systems[1] in the foreseeable future, as there are in most of the cases side constraints, that can not be dealt with automatically. In a min-sum solution there can be quite unbalanced, very short or very long tours, and it is always a matter of luck, if these side constraints can be handled without considerable complications. A good customer, who is calling for an emergency delivery after the planning is already finished, could be located near a short tour, which has still quite some room for further customers, or in a worse case near a tour, which cannot serve this customer anymore because he would exceed his limit of working hours. Balanced tours are usually much easier to handle, when changes occur.

Second, the human factor has to be considered. The planner has usually quite some need for explanation, if one tour gets just a few customers and is finished before lunch, while there is another tour, that can hardly finish before the end of the working day. Another important issue is the trade union, which can be a significant stakeholder and to a great extend co-determine the formulation of objectives and constraints, especially if these lead to inequity in the assignment of work. Furthermore, particularly for drivers the issue of exceeding the driving time is quite crucial, as a violation can be penalized by a considerable fine. A min-max objective would by itself balance the tours, such that either everybody will be within the allowed working time or everybody will exceed the time limit, which should lead to subsequent actions as e.g. adding teams and/or trucks, or postponing projects.

Furthermore, there are many cases, where it is necessary to have balanced routes, e.g. when it is essential, to serve all customers in reasonable time. One example can be routing for relief efforts in case of disasters (Ann, Vandenbussche, and Hermann 2008), where it is extremely important to be at the sites as fast as possible, even at the last locations of a tour.

[1] in contrast to decision *making* systems

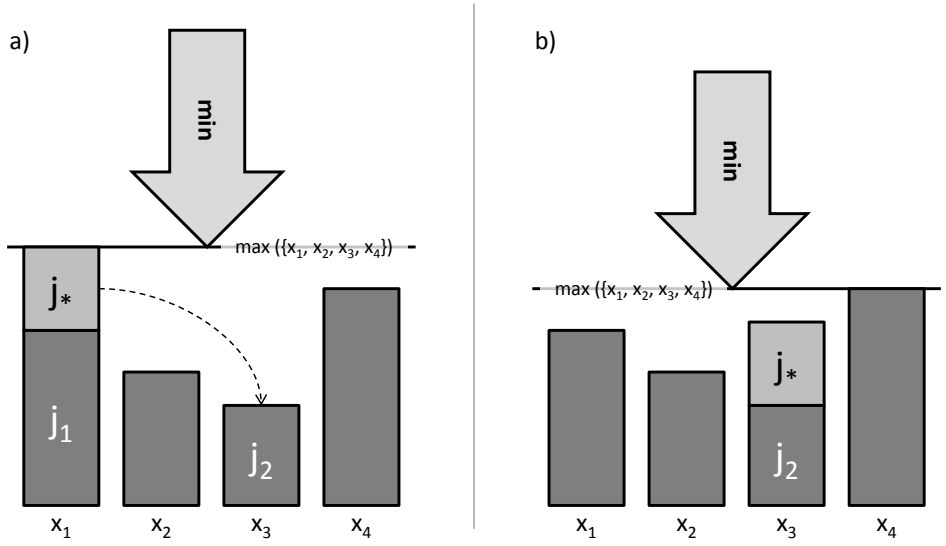

Figure 2.1.: min-max objective

Figure 2.1 shows the effects of applying a min-max objective. The boxes could, for example, be understood as jobs on machines $x_1, \ldots, x_4$. The height indicates the length of one job and each stack represents the sum of all jobs, visualizing the total work length per machine. In order to gain a lower objective value, job $j_\star$, visualized by the light-gray box, needs to be rescheduled to machine x_3. An optimal solution will have quite equal individual machine working times, as the jobs of the machines with the maximum working times will be spread to the other machines until this is not possible anymore.

Moreover, vehicle routing problems have the additional property of sequence dependence. Amongst others, this can also be found in machine scheduling problems with sequence dependent setup times. Figure 2.2 illustrates this characteristic. If we consider picture a) in this figure as starting solution, where machine x_1 is running longest, and shift the job $j_\star$ depicted by the gray box to machine x_3, as shown in picture b), machine x_3 will run longest. The length of the $j_\star$ changes now, as setup time increases (for example, because the change of tools from its preceding job j_2 from the job j_1 on machine x_1). However, if the sequence on machine x_3 is changed, as depicted in picture c), such that $j_\star$ precedes j_2, the setup for j_2 only increases a bit, but for $j_\star$ it decreases considerably. The sum of jobs decreases, and finally x^4 runs longest.

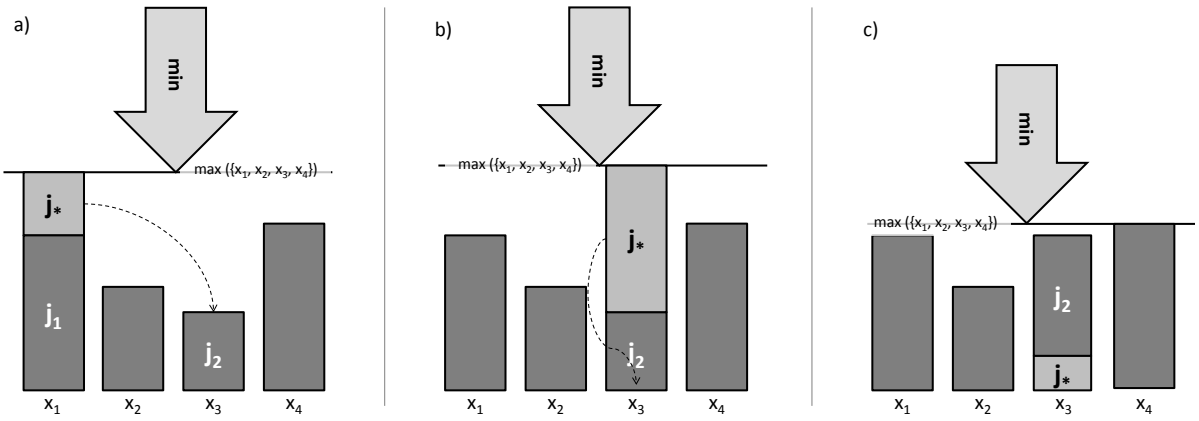

Figure 2.2.: min-max objective with sequence dependence

2.2. A Mixed Integer Programming formulation

As a base for a subsequent implementation, the model is formulated as the following mixed integer program:

Sets

$\{0\}$	team depot
$\{1\}$	truck depot
$\mathcal{N}$	set of projects
$\mathcal{T}$	set of teams

Parameters

U	maximum working day duration
M	number of teams
P	number of trucks
S_{ij}	traveling time between locations i and j
Z_i	duration of job i

Variables

$\beta_{ij} = \begin{cases} 1 & \text{if arc between location } i \text{ and } j \text{ is traversed} \\ 0 & \text{otherwise} \end{cases}$

$\lambda_{ij} = \begin{cases} 1 & \text{if a team fulfills job } j \text{ after job } i \\ 0 & \text{otherwise} \end{cases}$

t_i arrival time at location i

u latest return time of all trucks. The return time is defined as the time, when the truck arrives at node 1.

x_i starting time of job i fulfillment

y latest return time of all teams. The return time is defined as the time, when the team arrives at node 0.

w latest time

$$\min w \tag{2.1}$$

subject to

$$w \geq u \tag{2.2}$$

$$w \geq y \tag{2.3}$$

$$\sum_{j \in \mathcal{N}: i \neq j} \lambda_{ij} = 1 \qquad \forall i \in \mathcal{N} \tag{2.4}$$

$$\sum_{i \in \{0\} \cup \mathcal{N}} \lambda_{ij} = \sum_{k \in \{0\} \cup \mathcal{N}} \lambda_{jk} \qquad \forall j \in \{0\} \cup \mathcal{N} \tag{2.5}$$

$$\sum_{j \in \mathcal{N}} \lambda_{0j} \leq M \tag{2.6}$$

$$U(1 - \lambda_{ij}) + x_j \geq x_i + Z_i + S_{ij} \qquad \forall i \in \{0\} \cup \mathcal{N}, j \in \mathcal{N} \tag{2.7}$$

$$U(1 - \lambda_{i0}) + y \geq x_i + Z_i + S_{i0} \qquad \forall i \in \mathcal{N} \tag{2.8}$$

$$\sum_{j \in \mathcal{N}: i \neq j} \beta_{ij} = 1 \qquad \forall i \in \mathcal{N} \tag{2.9}$$

$$\sum_{j \in \{1\} \cup \mathcal{N}} \beta_{ij} = \sum_{k \in \{1\} \cup \mathcal{N}} \beta_{jk} \qquad \forall j \in \{1\} \cup \mathcal{N} \tag{2.10}$$

$$\sum_{j \in \mathcal{N}} \beta_{1j} \leq P \tag{2.11}$$

$$U(1 - \beta_{ij}) + t_j \geq t_i + S_{ij} \qquad \forall i \in \{1\} \cup \mathcal{N}, j \in \mathcal{N} \tag{2.12}$$

$$U(1-\beta_{i1})+u \geq t_i + S_{i0} \qquad \forall i \in \mathcal{N} \tag{2.13}$$

$$t_i = x_i \qquad \forall i \in \mathcal{N} \tag{2.14}$$

The objective function (2.1) minimizes the maximum return time for teams and trucks. This objective, together with the constraints (2.3), (2.2), (2.8) and (2.13) is a reformulation of the min-max objective (see Fourer, Gay, and Kernighan 2003, pp. 380-382).

Because of constraints (2.2) and (2.3), the maximum return time always has to be greater or equal than the return time of all teams and all trucks. The family of constraints (2.4) mean, that each project needs to be visited exactly once by a team. Constraints (2.5) indicate, that if a node is visited by a team, it should be left as well. Constraints (2.6) limit the number of teams to a maximum of M outgoing teams from node 0, which is the team depot.

The following two families of constraints (2.7) and (2.8) define the working time. If project j does not follow project i, then $\lambda_{ij} = 0$ and the term $U(1-\lambda_{ij})$ will be a number, which will always be bigger than the right hand side, consequently x_j will not be forced by this constraint to take a nonzero value. However, if a project j is adjacent to project i, then $\lambda_{ij} = 1$. Therefore the term $U(1-\lambda_{ij})$ will be zero, and x_j needs to take a value, which is at least the project start time of x_i, plus the time to work on project i, Z_i, plus the time to go from project i to project j, S_{ij}.

One interesting detail here is, that this constraint provides at the same time a means of sub-cycle elimination, as the constraint says, that time in one tour has to increase monotonically. In a sub-cycle this could not hold, as the same element on the beginning would be assigned another time than at the end of the sub-cycle. This principle follows closely the MTZ formulation of the TSP in Miller, Tucker, and Zemlin (1960).

The constraints (2.9) guarantee, that every project is visited exactly once by a truck. (2.10) is a set of balance constraints to ensure, that all trucks going to a project are leaving it as well. The number of trucks which is allowed to leave the truck depot 1 is limited by constraints (2.11). The constraints (2.12) and (2.13) work in the same manner as (2.7) and (2.8). In this case they determine the working time of the trucks. (2.14) are the synchronization constraints. They ensure, that for each project on each day the arrival time of the truck will be the time, when the team starts working on the project.

2.3. Implementation of the model

The model was implemented in Pyomo. This is a relatively new Algebraic Modeling Language (AML), fully integrated in the high-level programming environment *Python*[2]. A description and overview of Pyomo can be found in Hart, Watson, and Woodruff (2011), and a more detailed insight is presented in Hart et al. (2012). Pyomo was preferred to other AMLs, as it adds a high degree of flexibility and allows a much better flow control of the solution process. It was, for example, very easy, to generate graphical visualizations of the solutions using the Python language and LaTeX.

However, Pyomo is just a modeling language and hence needs a solver for the computational part. For this, *Gurobi*[3] was be used. While initially CPLEX was considered, recent developments and licensing issues for the academic environment lead to this solver as it is much more open to academic licensing, quite fast, and perfectly integrated with Pyomo.

It is noteworthy to mention, that in the author's opinion it is advisable to use a two-step approach, where development and prototyping of the model is done in another language, in the concrete case AMPL. Pyomo uses a mighty language, but this leads to a somewhat confusing notation, which makes it difficult to read. However, as soon as the model is working, it can be easily re-implemented in Pyomo, where all the power of the underlying Python-language can be unleashed for further processing. As an example, from the numerical solutions the graphs shown in this thesis were generated using the PGF/Ti*k*Z language[4]. One more possibility would be to generate the solution as an animation, for example using the SVG format[5].

During computational experiments one issue popped up, which was hard to track down, as this happened only in rare cases and not connected to the model, but the solver: Even if variables are defined as integer or binary, Gurobi has some tolerance. If variables are within this tolerance, they are considered as integer. By default, this tolerance is 1e-05. However, with this value objective values are found, which are actually not feasible.

This issue was discovered, because the heuristic solver, which is presented later on, did not reach this minimum objective value when comparing run-times to the MIP implementation. It continued running, as reaching the objective value produced by the MIP

[2]see `http://www.python.org/` (accessed November 22, 2012)
[3]see `http://www.gurobi.com/` (accessed November 22, 2012)
[4]see http://sourceforge.net/projects/pgf/
[5]see http://www.w3.org/Graphics/SVG/

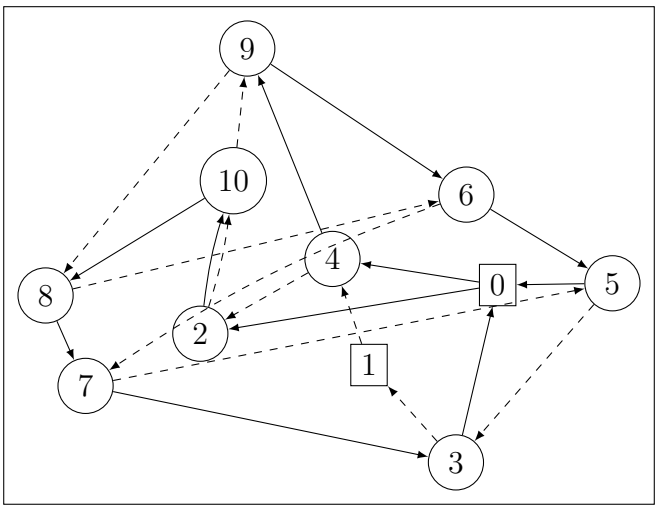

Figure 2.3.: Example solution for instance with 2 teams and 1 truck.

solver was a stopping criterion. However, by sending the parameter *IntFeasTol=1e-09* to the solver, this tolerance can be narrowed down. The author is aware, that this is more of an academic problem, as feasible solutions still may be found in shorter time by relaxing this tolerance, but on the other hand the requirements when comparing two approaches should be the same. Thus, the decision was, to set this tolerance as tight as possible.

Figure 2.3 shows the solution for instance 005 with 2 teams and 1 truck. When it comes to the run-time, Figure 2.4 shows the big drawback of this approach. With instances getting bigger, the run-times get disproportionately longer. Even if an instance with 9 nodes (2 depot nodes + 7 projects) can be still solved within less than 4 seconds, the calculation run for 12 nodes was aborted after 4 hours. For an application like the contemplated problem of planning for road construction and maintenance, the solvable problem sizes are therefore not large enough.

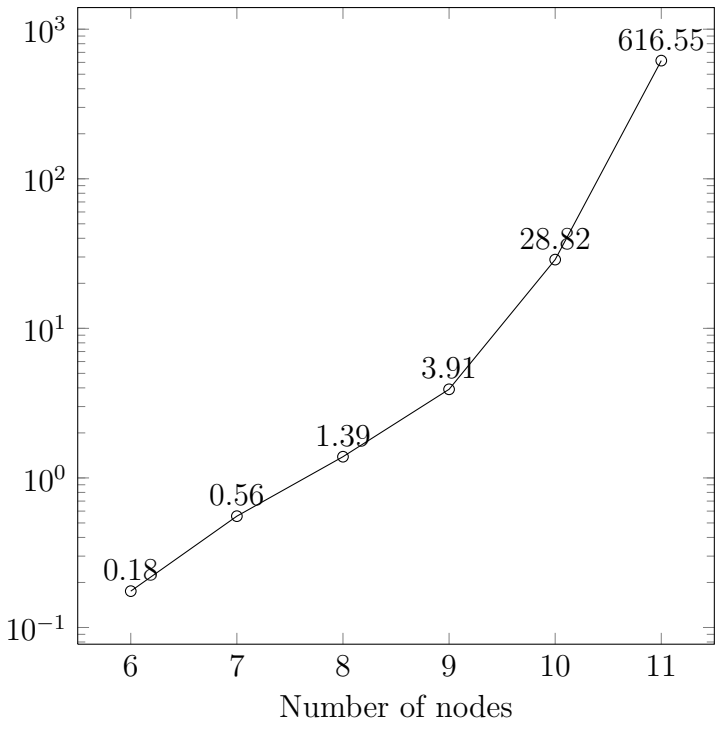

Figure 2.4.: Run-times for instance 005 with 3 teams and 2 trucks, dependent on the number of projects

2.4. Advanced methods of solving a min-max VRP to optimality

As the run-times of the MIP model show, solving min-max VRPs by classical means, i.e. solving it by branch-and-bound or branch-and-cut algorithms, shows a quite poor performance. The reason is, that solutions at intermediate nodes of the search tree usually do not contain sufficient information on the value of the objective to allow fathoming (see França et al. 1995).

For the min-max VRP without synchronization constraints some, but not much work is already done to solve bigger instances to optimality. Applegate et al. (2002) could prove, that a solution for one specific instance with 120 nodes and 4 vehicles, which was earlier found by a simulated annealing algorithm, was optimal. Before this, França et al. (1995) solved the min-max m-TSP up to 50 nodes optimally, which is already an adequate dimension for our real world problem. This approach is interesting, because the min-max m-TSP was reformulated as distance-constrained asymmetrical VRP and solved iteratively by a branch-and-bound algorithm described in Laporte, Nobert, and

Taillefer (1987), decreasing the maximum allowed distance until the problem gets infeasible. The last feasible run is finally the optimal solution. As an example, the average computational time for 10 comparable structured instances with 20 nodes and 3 vehicles were approximately 84 seconds. Considering the computational power of the time when this paper was published, it should be fairly easy to calculate this solution with up to date hardware. Furthermore, Almoustafa, Hanafi, and Mladenović (2013) improved Laporte's exact algorithm for the asymmetrical DVRP, which could be again utilized to solve the min-max m-TSP instead of the one used by Laporte.

3. Solving bigger instances with heuristics

As shown in the previous chapter, it will not be possible to solve bigger instances to optimality within reasonable time. As already noted, Leont'ev (2007) pointed out heuristics as an alternative to solve discrete optimization problems. This chapter shows an approach of building a heuristic solver. Furthermore, the connected topic of evaluating candidate solutions will be covered.

3.1. Heuristics

Generally heuristics can be divided into two different classes: *classical heuristics* and *metaheuristics* (Toth and Vigo 2002, p. 109). Classical heuristics were most actively developed between 1960 and 1990. They explore the search space quite sketchy and provide therefore solutions, which are quite good in reasonable time. Metaheuristics, on the other hand, explore the promising regions of the solution space much more in depth, typically using neighborhood search rules, memory structures, and recombination of solution. Most popular metaheuristics for VRP are Ant Colony Optimization, Genetic Algorithms, Greedy Randomized Adaptive Search Procedure, Simulated Annealing, Tabu Search and Variable Neighborhood Search (see Gendreau et al. 2008, pp. 143–147).

3.2. Evaluation of a candidate solution

In order to determine the quality of a solution, it needs to be evaluated. Figure 3.1 shows different approaches, how to deal with this issue. In the standard min-max VRP problem the solution can be evaluated analytically, in a very easy and straightforward

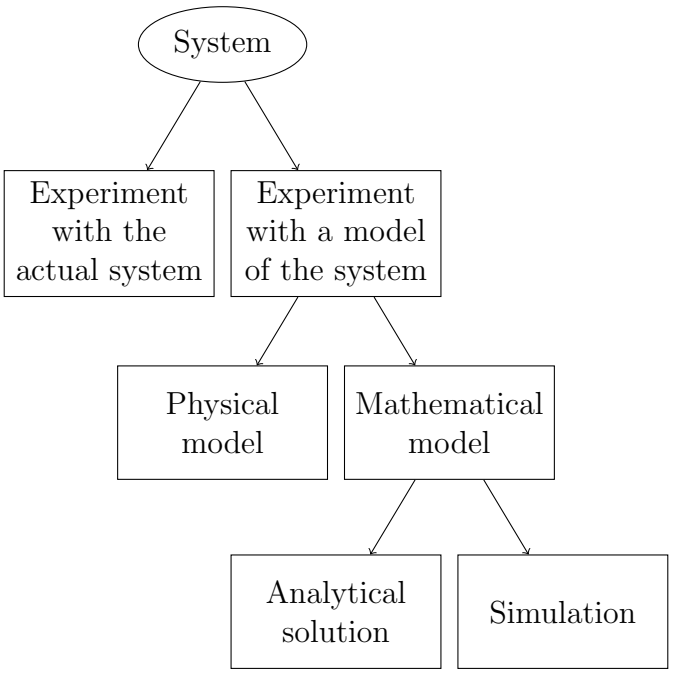

Figure 3.1.: Ways to study a system (Source: Law and Kelton 2000, p. 4)

manner. Given a distance matrix

$$D = \begin{pmatrix} 0 & d_{01} & \dots & d_{0j} \\ d_{10} & 0 & \dots & d_{1j} \\ \dots & \dots & 0 & \dots \\ d_{i0} & d_{i0} & \dots & 0 \end{pmatrix}$$

where d_{ij} denotes the distance between node i and node j, and a solution matrix for each vehicle k in a set of vehicles $\mathcal{K}$

$$S_k = \begin{pmatrix} 0 & s_{01k} & \dots & s_{0jk} \\ s_{10k} & 0 & \dots & s_{1jk} \\ \dots & \dots & 0 & \dots \\ s_{i0k} & d_{i0k} & \dots & 0 \end{pmatrix}$$

where $s_{ijk} \in \{0,1\} = \begin{cases} 1 & \text{if arc between location } i \text{ and } j \text{ is traversed by vehicle } k \\ 0 & \text{otherwise.} \end{cases}$

Then the objective value of this solution will be

$$z = \max_{k \in \mathcal{K}} \{D^\top S_k\}$$

However, in a system, where the set of vehicles $\mathcal{K}$ is synchronized with vehicles of another class $\mathcal{L}$, waiting times will be involved, where one vehicle is staying at the node, until the other party arrives.

$$z = \max_{k \in \mathcal{K}, l \in \mathcal{L}} \{D^\top S_k^{\mathcal{K}} + w_k^{\mathcal{K}}(S_m^{\mathcal{K}}, S_n^{\mathcal{L}} | \forall m \in \mathcal{K}, n \in \mathcal{L}), \\ D^\top S_l^{\mathcal{L}} + w_l^{\mathcal{L}}(S_m^{\mathcal{K}}, S_n^{\mathcal{L}} | \forall m \in \mathcal{K}, n \in \mathcal{L})\}$$

Thus, the individual waiting times $w_k^{\mathcal{K}}$ and $w_l^{\mathcal{L}}$ are not only dependent on their own solution matrix $S_k^{\mathcal{K}}$ and $S_l^{\mathcal{L}}$, but on also on all other solutions of the own and of the other vehicle class, $\mathcal{K}$ and $\mathcal{L}$. These interdependence makes it much more difficult to determine the objective value analytically.

Usually analytical methods are still preferable, when they have a solution, because they are computationally fast. However, as soon as systems get more complex, it is very difficult or even impossible to find an analytical model (Altiok and Melamed 2007, p. 3). Furthermore, any change of the model will result in the need to derive a new analytical model. Therefore, the use of simulation seems reasonable, as models can be implemented straightforward and changes can be conducted much easier. This idea is not completely new. Almeder, Preusser, and Hartl (2009), for example, proposed to connect a nonlinear and stochastic simulation model and a simplified linear program to support operational decisions for a supply chain network.

3.3. A different view on the problem - the synchronized VRP as a system

So far we have learned the following features about this problem:

- The applied classical methods were able to solve some small instances, but were not suitable for bigger ones.
- Therefore we try to apply heuristic approaches.
- For heuristic approaches we need to evaluate a possible solution.

- The evaluation is not straightforward because we need to consider the inter-dependencies between the two vehicle classes of teams and trucks, that need to wait for each other.

In order to tackle the evaluation issue, and to understand how to build an evaluation function, which can provide the objective value of a candidate solution, we will look at this problem by considering it as a system (see Cassandras and Lafortune 2008, pp. 2). The IEEE standard dictionary of electrical and electronics terms[1] defines a system as *"a combination of components that act together to perform a function not possible with any of the individual parts"*. Webster's Dictionary[2] describes this term as *"a regularly interacting or interdependent group of items forming a unified whole"*. Finally, the Oxford Dictionaries[3] defines a system as *"a set of things working together as parts of a mechanism or an interconnecting network; a complex whole"*.

What all of these definitions have in common is the aspect of *components*, which perform a defined *function*. Furthermore, these components show inter-dependency. This inter-dependency is a necessity to fulfill the purpose of this system. While it can be left to discussion, if a standard VRP can already be considered as a system, the VRPMS is clearly describable by this term.

As a part of this thesis, and as a consequence of what we have learned until now from the problem, the question arises, how we can deal with such a system, and how we can explore its behavior under given circumstances by providing it with input (in our case the candidate solution) and measuring certain indicators as shown in Figure 3.2. This should be performed in the best possible way, i.e. the chosen approach should be

- *useful:* It should be possible to obtain the necessary and relevant information about the system as an output. This will be for sure the objective value, the length of the longest tour, but it could also be necessary to retrieve the length of the other tours or the waiting times. Also feasibility can be an information to be obtained.

- *efficient:* The approach for studying the system should require the least possible effort (e.g. computational cost). Features, that are not of interest, can be disregarded in order to enhance performance. For example, as we will see later, it can

[1] Radatz, Jane. *The IEEE standard dictionary of electrical and electronics terms*. 6th ed. s.v. "System." New York: IEEE, 1996.

[2] *Merriam-Webster's Collegiate Dictionary*. 11th ed. s.v. "System." Springfield, MA: Merriam-Webster, 2003. Accessed April 16, 2013. `http://www.merriam-webster.com/`.

[3] *Oxford Dictionaries*. April 2010. s.v. "System." Oxford University Press. Accessed April 16, 2013. `http://oxforddictionaries.com/definition/english/system`.

be of advantage to discretize time, or even ignore big parts of the timeline and concentrate only on the "interesting" points in time.

- *flexible:* It should be easy to change the behavior of the system. Additional measurements should be possible without reconsidering the approach. For improving the metaheuristic it could be necessary to determine additional indicators, and this should to be possible without much effort. Furthermore, extending or altering the problem should not lead to issues in evaluating the solution.

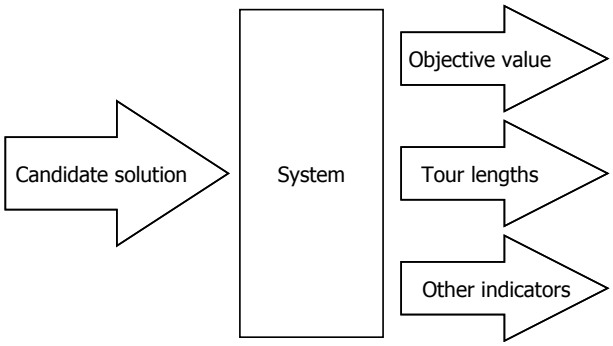

Figure 3.2.: Obtaining indicators from the system after providing a candidate solution. Source: adapted from Gill 1962, p. 2.

To understand better, how the synchronized VRP system can be handled, we will discuss it by focusing on some major classification criteria of systems named by Cassandras and Lafortune (2008) (pp. 44):

- *Static and Dynamic Systems:* While in static systems, the output is always independent from past inputs in history, dynamic systems will have different outputs when provided with other inputs in history. To describe such systems, differential or difference equations are usually used. The considered synchronized VRP system can clearly be classified as **dynamic system**. Using the next node to go with a vehicle as an input, it will make a difference, where this vehicle is departing from, as the distances are most probably different and the sequence of nodes to be visited will be a different one.

- *Time-varying and Time-invariant Systems:* Time-varying systems will behave different at various points of time, whereas time-invariant do not. In our system the behavior will not change over time, it is therefore **time-invariant**.

- *Continuous-state and discrete-state systems:* When examining the possible states of a system, there are two possibilities: If the system variables can take on any real value, then this is a continuous-state system. Otherwise, if the state of the system can be described by a finite set of states, we deal with a discrete-state system. In principle, we could define the synchronized VRP system as continuous-state system. This would, for example, be necessary if we would like to know the exact position of the vehicles at any time t. However, considering it as a **discrete-state** system will make the evaluation much easier, as shown in section 3.4.

- *Continuous-time and discrete-time systems:* Additionally to the state of the system, the time needs to be considered. In many classical systems, as for example in the field of physics, which often works with differential equations, time is considered to be continuous, that is, t is a real value. However, it can be handy or even necessary to discretize time for several reasons. One of these reasons could be, that systems (like most of modern computer architectures) are already designed as discrete-time systems, and driven by a clock generator, which provides the timing for processing a list of commands. Another reason could be to approximate a continuous-time system by discretizing time in order to facilitate system evaluation. In our system we will consider time as **discrete**, as only certain points in time are of interest.

- *Time-driven and Event-driven Systems:* In time-driven systems the state of the system continuously changes over time. In event-driven systems state transitions are triggered by asynchronously generated events. Between events the system state does not change. Our system is considered as an event-driven system. To name one example, the arrival of an asphalt truck at a certain node is considered as an event, and will trigger a waiting working team to begin its work at this node. The behavior of our system can be fully described by **events**.

- *Deterministic and Stochastic Systems:* If it is possible to predict the future behavior of a system at the time $t = t_0$, assuming knowledge of the input, the system is considered to be deterministic. However, if the system has unpredictable elements in it, it is stochastic. The synchronized VRP under examination does not have any stochastic elements in it. However, as a possible extension of the problem, uncertainty could be included by adding, for example, stochasticity in working or driving times. For now we will consider it as **deterministic**.

Cassandras and Lafortune (2008) (p. 32) define a discrete event system as *"... a discrete-state, event-driven system, that is, its state evolution depends entirely on the occurrence of asynchronous discrete events over time."* Our system features these properties and can therefore be classified as a discrete event system. In order to implement a discrete event simulation, we first need appropriate tools for describing the model. While the MIP formulation is perfect for solving the model by classical optimization, for simulation it may be better to view the model as system of various model elements that act dependent on the system state. The discrete behavior of these entities can be described through finite-state transition systems, by so-called Finite State Machines.

The idea behind this is an abstract machine which can be in a defined, finite number of states. It is only possible to be in one state at a time, however the machine changes between states through triggering events or conditions (see Gill 1962).

Figure 3.3 shows the behavior of the two vehicle classes, the teams and the trucks, as Finite State Machines. This follows the specification of the Object Management Group (2011).

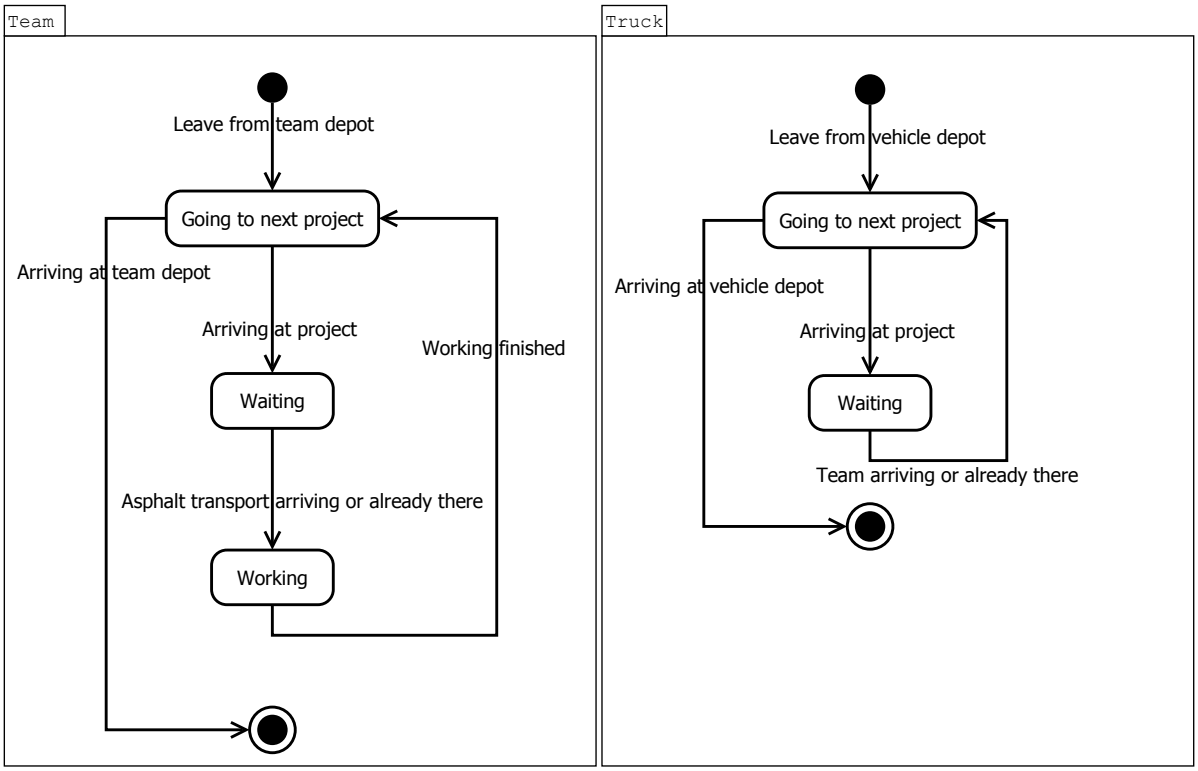

Figure 3.3.: State machines describing working teams and asphalt trucks

3.4. Evaluation using discrete event simulation

As already pointed out, in discrete event systems state variables change instantaneously at separable points in time (Law and Kelton 2000, p. 6). This differs from a continuous system, where the state variables change steadily over time.

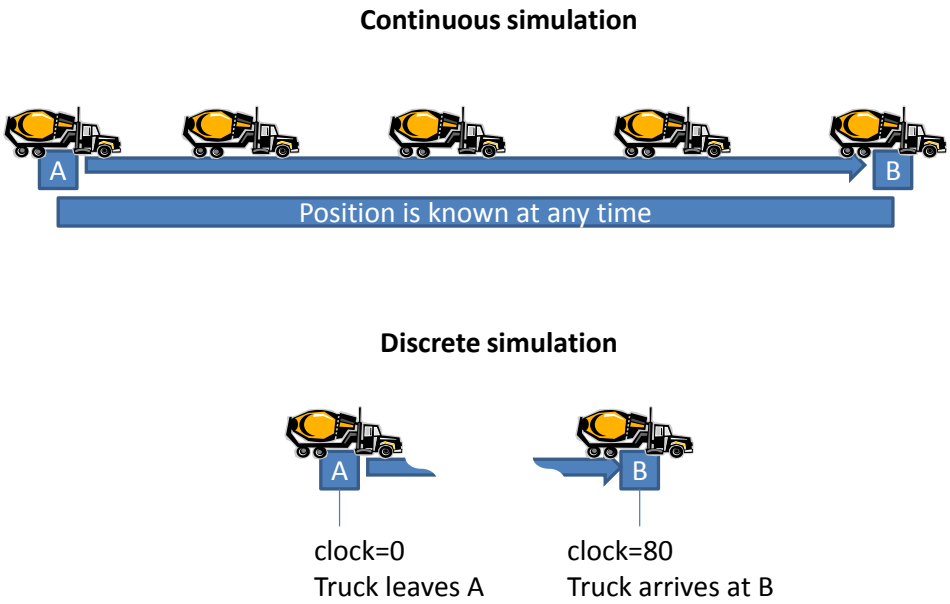

Figure 3.4.: An asphalt truck going from A to B in a continuous and discrete simulation model

Figure 3.4 visualizes the difference between these two approaches. In the case of continuous simulation, the asphalt truck leaves from A at $clock = 0$, and over time, the position will always be known. However, in the discrete simulation case, after the truck leaves A at $clock = 0$, the system will stay in this state until it approaches B at $clock = 80$, and until then there is no new information about the truck, except that it is on its way. For evaluating the VRPMS system, it is not of interest what happens on the arcs. The synchronization of the vehicles should be the matter of case, which can be expressed by discrete events.

There are several software products on the market, which allow to design, run and evaluate discrete event systems using different approaches. The simulation package *Arena*[4] allows to design the models graphically and provides a rich set of tools to evaluate the solutions statistically. Furthermore, there are libraries, that integrate into a high

[4]see `http://www.arenasimulation.com/` (accessed March 17, 2013)

level language to provide tools for simulation. SystemC[5], for example, provides a library for C++. Parsec[6] is a C-based simulation language. JiST[7] is a discrete event simulation engine that runs over a standard Java virtual machine. SimPy[8] is an object-oriented, process-based discrete-event simulation language for Python.

However, each of these products has some drawbacks, that do not qualify them for using them for the intended purposes. The main argument for that is performance, as the search process will involve a high amount of evaluations. Even though only little advice can be found in scientific literature, which options to choose, it is reasonable to presume, that none of these aforementioned approaches will reach the performance of a self-made implementation.

First, some of the packages are just not able to provide the highest possible performance by design. To name an example, some simulators interpret the model at run-time, like SimPy. JiST, on the other hand, runs the simulation as compiled Java-code and therefore is able to outperforms SimPy by far, as Weingartner, Lehn, and Wehrle (2009) show in a network simulation model.

Second, these packages usually include big overhead, as they are written for a broader field of applications. They contain tools for statistics collection, random number generation and functionality, that allows the implementation of models which go far beyond what is needed for the synchronized VRP.

These arguments lead to the decision to implement a tailor-made greenfield implementation of a basic discrete-event simulator in form of a library, which can be utilized and tightly integrated with the simulation model. The model can, as well as the simulator, run as compiled code. C++ was chosen as implementation language, because this is considered as one of the high-level programming languages, which generates very efficient code. In order to implement a discrete-event simulator for the concrete VRPMS system at hand, the following central elements needed to be programmed (see Law and Kelton 2000, pp. 9):

- *System state:* Set of variables representing the state of the system at a particular time. These variables are changing, when specific events occur.
- *Simulation clock:* A variable denoting the point of time in the course of simulation.

[5]see `http://www.accellera.org/downloads/standards/systemc/about\_systemc/` (accessed March 18, 2013)
[6]see `http://pcl.cs.ucla.edu/projects/parsec/` (accessed March 18, 2013)
[7]see `http://jist.ece.cornell.edu/` (accessed March 18, 2013)
[8]see `http://simpy.sourceforge.net/` (accessed March 17, 2013)

- *Event list:* A list containing the events, which will be the trigger for transitions from one system state to another. This list will be processed one by one.
- *Initialization routine:* A routine which sets the state of the system to a defined initial point.
- *Timing routine:* Two different mechanisms are suggested historically to advance the clock, the *next-event time advance* and the *fixed-increment time advance*. The great majority of implementations, including the one for the VRPMS simulation, uses the next-event time advance mechanism, which will be presented subsequently.
- *Event routine:* A routine which processes each event. Usually the state variables are changed based on the type of event.
- *Main program:* The program responsible for coordinating the different components, initializing the state, invoking the timing routine to get the next event and processing the event through the event routine.

As mentioned above, the simulator uses a next-event time advance mechanism. Figure 3.5 visualizes this concept using the example of the events occurring on a synchronized tour of one team and one truck. In this example the event list is shown as time bar on the x-axis, with the team (T1) and the truck (V1) as two separate lanes. Furthermore, the triangle shows the position of the simulation clock.

The list is initialized with two events: The event of T1 arriving at node 1 at time 20, and V1 arriving at the same node 1 at time 50. These are the travel times from the team / truck depots to this node and can be easily obtained through the distance matrix. The simulation clock is set to 0.

In the first step the simulation clock is set to 20, which is the first occurring event after its present position, this is the arrival event at node 1 (A1) of team T1. In this step, not very much happens, as this team needs an asphalt delivery from a truck to commence work. Only the system state changes, as the truck is moved from the truck depot node to node 1.

The simulation proceeds in step 2 to the next event in the event list, the arrival of truck V1 at node 1 (A1), which happens at time 50. The position of the truck is moved to node 1 as well, and because team 1 is already at node 1, the state of team 1 changes to "working". Furthermore, now we can insert two more events: Team 1 will stay in the working state until an event work done at node 1 (W1). The duration of the work can be again just looked up from the provided data, in this case team 1 works for 20 time

units. Meanwhile truck 1 will proceed to node 2 for 30 time units, arriving at the time when event A2 occurs at time 80.

Step 3 marks the end of the work of team 1 at node 1. The team departs from the node, and one more event is created at the point of time, where the team is arriving at node 2 (A2). At this time the truck is already traveling to node 2, but has not yet arrived.

In the next step, the system clock is set to the arriving event for truck 1. Besides of setting the position of the truck in the system state nothing more happens, as it is still waiting for the team to arrive.

Finally, the team arrives in step 5, and can instantly begin to work, while the truck can proceed to node 3. Again, 2 more events are created, the event to mark the end of the work of the team, and the event of truck 1 arriving at node 3.

In order to implement this mechanism into a solver, we need to consider how we can represent it in terms of data which can be processed by a computer. The event list can be described as a list of entries, which is monotonically increasing by system time and of the following structure:

1. Point in time: When does a specific event happen?
2. Object: Which object is this event connected to?
3. Event type: Which kind of event is it? This determines how the simulation engine reacts on the event, e.g. how the system state will be changed.

To exemplify this, the event list will look like this in step 5, with the system clock set to 100:

Clock	**Event**	**Arg 1**	**Arg 2**
20	A	T1	Node 1
50	A	V1	Node 1
70	W	T1	
80	A	V1	Node 2
100	A	T1	Node 2
130	W	T1	
140	A	V1	Node 3

As events are not needed anymore, after they are processed, we can design this list as a queue, which contains the elements in increasing order by system time, and always

points to the current event as the first in list. If the event is processed, then this element is discarded and the following event will be the first in list. In computer science this concept is called *priority queue* (see Cormen et al. 2001, pp. 138–142).

There are efficient implementations of priority queues in most of the common high level languages, including C++, which perform much better compared to naive implementations with respect to inserts and removals of elements. Moreover, using this mechanism facilitates the execution of the simulation and keeps memory usage low.

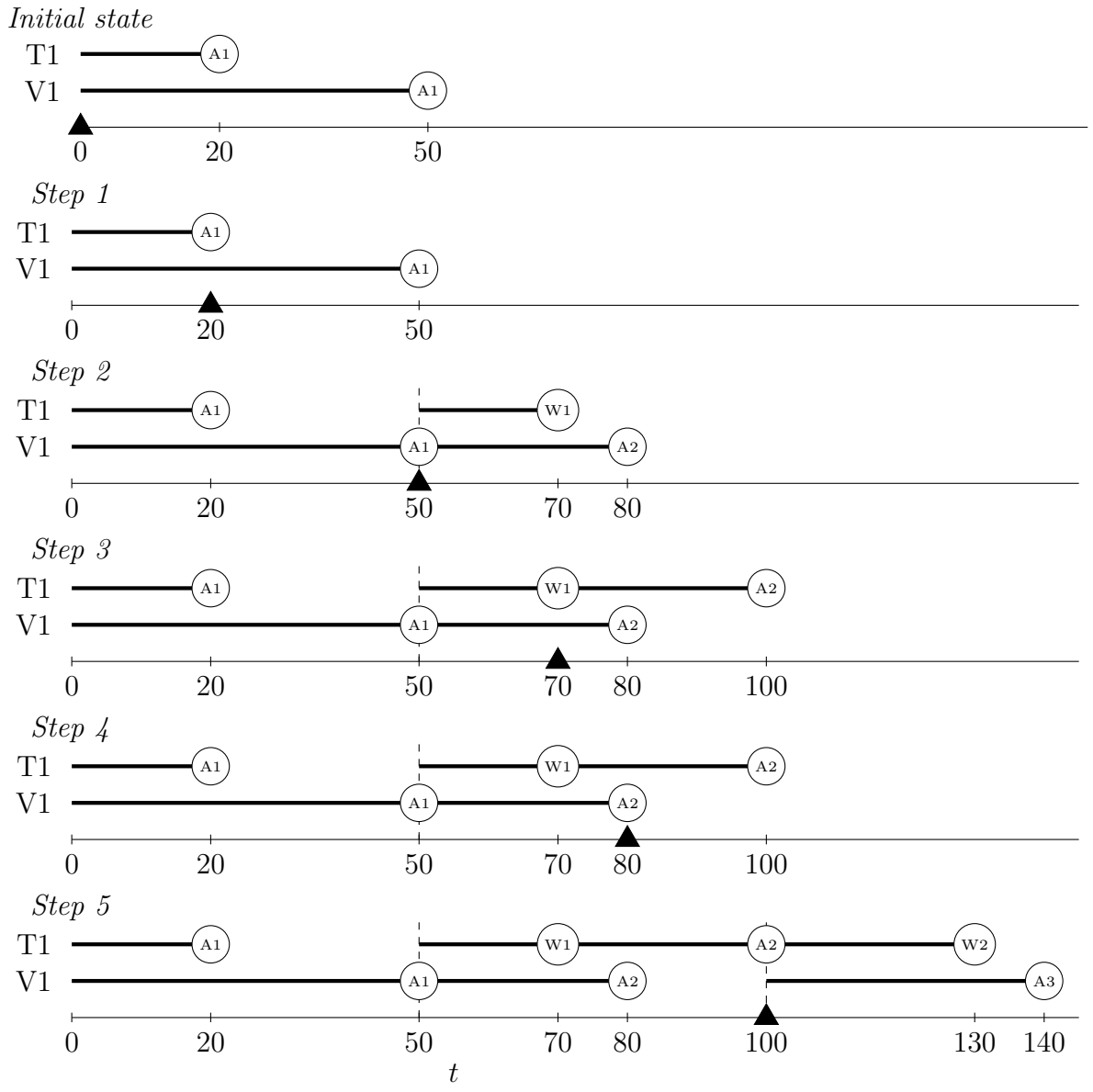

Figure 3.5.: Execution of the event list

3.5. Experiments with a ruin and recreate algorithm

One metaheuristic, which has proven to be successful, is Ruin and Recreate, first proposed by Schrimpf et al. (2000). The reason to choose this algorithm lies in the nature of the VRPMS: Improving the solution of one vehicle class can affect the tours of the other synchronized class. Even a very simple 2-opt move can make the solution infeasible, which results in the need for rearranging the tours of the synchronized class. However, this rearrangement can again cause the need for rearranging the first class to remain a feasible solution. This seems one of the big issues in exact operation synchronization, as Drexl (2012b) states in his survey as well.

In the ruin and recreate algorithm however, nodes are taken out of the solution in both classes and the tours still stay feasible. Constructing a candidate solution by reinserting the nodes one by one is easier (however, still not straightforward, as we will see in section 3.6).

The idea of the algorithm is to start with an initial solution, to iteratively destroy large parts of it, and to rebuild it as good as possible. A simple min-max VRP (without any synchronization of vehicles) was implemented following algorithm 1 in order to test the approach.

```
Data: pool = A set of nodes in randomly generated sequence, numDestroy =
      Number of nodes to be ruined after one iteration
Result: A tour for each vehicle
while stopping criterion not met do
    while pool not empty do
        newNode ← next element from pool;
        insert newNode at the tour and position which increases the maximum
        travel cost over all tours the least;
    end
    for numDestroy times do
        Take an arbitrary node of any tour and put it into the pool;
    end
end
```

Algorithm 1: Ruin-and-Recreate Algorithm for min-max VRP

Figure 3.6 shows the objective value trajectory during an optimization run with 1,300 iterations. The upper trajectory visualizes a run with 20 ruined nodes. After the creation

of an initial solution in iteration 0 the objective value escapes to a much higher level than the initial objective value and is not able to return, even if there are phases of improvement. It turns out, that increasing the number of ruined nodes improves the algorithm notably. When ruining 90 nodes, some better solutions than the initial solution are found. However, going to the extreme and deleting all nodes seems to be the best possibility.

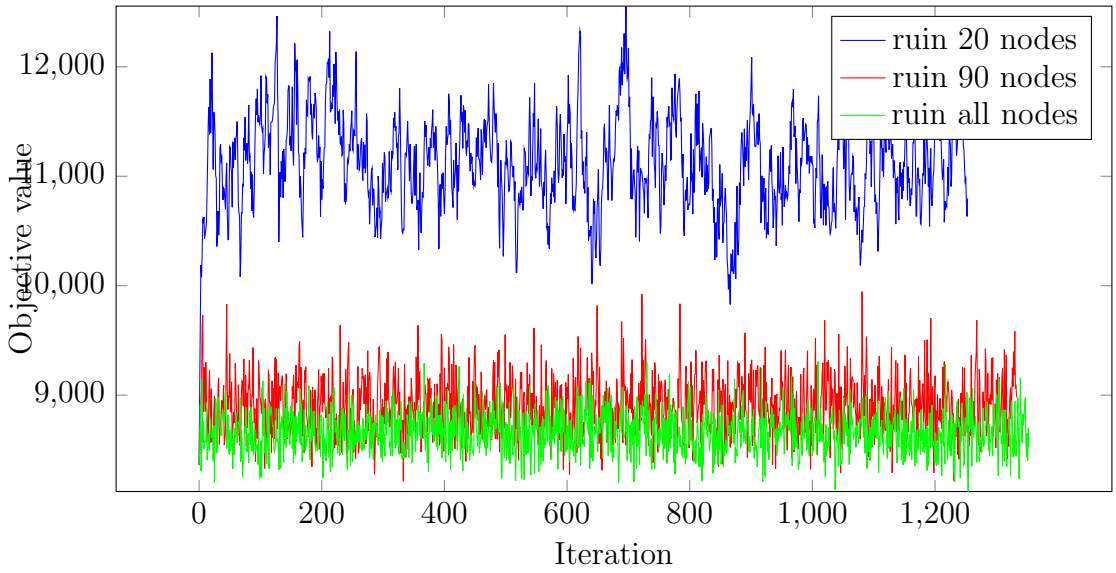

Figure 3.6.: Objective value trajectory of optimizing a 100-nodes instance with an ruin-and-recreate algorithm

The reason, why there are still different objective values when applying the algorithm, is the fact that the nodes in the pool are shuffled in each iteration. Another sequence, in which nodes are inserted into the solution, gives different solutions, even if the same algorithm is applied. By storing the solution with the best objective value and comparing it to subsequent iterations with different starting sequences, good solutions could be achieved.

3.6. Description of a construction algorithm for the VRPMS

Because this approach showed useful for a simple min-max VRP, the algorithm was extended to construct both team and truck tours as described in Algorithm 2. It was decided, not to try to find other, maybe more reasonable, insertion procedures for the

ruin-and-recreate approach, but to accept, that it works quite well to construct all tours from scratch instead. By removing all nodes this cannot be called an improvement algorithm anymore, because it does not search in neighborhoods to some current solution, but just creates a completely new one from scratch. This method can therefore be described much better as a multistart greedy construction algorithm. It does not even fulfill the properties of a GRASP as described e.g. in Pitsoulis and G.C. (2001), but as the computational results show, this still proves to be sufficient.

```
Data: pool = A set of nodes in randomly generated sequence
Result: A tour for each team and truck
while stopping criterion not met do
    while pool not empty do
        newNode ← next element from pool;
        for each team position in each team tour do
            temporarily insert newNode at team position;
            for each truck position in each truck tour do
                temporarily insert newNode at truck position;
                objectiveValue ← evaluate (solution);
                if (solution feasible) and (objectiveValue < tempOptimalValue) then
                    tempOptimalValue ← objectiveValue ;
                    optimal team position ← team position;
                    optimal truck position ← truck position;
                end
                remove newNode from truck position;
            end
            remove newNode from team position;
        end
        insert newNode permanently in team tour at optimal team position;
        insert newNode permanently in truck tour at optimal truck position;
    end
    if (tempOptimalValue < optimalValue) then
        optimalValue ← tempOptimalValue ;
        optimal team tour ← team tour;
        optimal truck tour ← optimal truck tour;
    end
    put back all nodes in pool and shuffle;
end
```

Algorithm 2: Multistart greedy construction algorithm for VRPMS

It is important to note, that the insertion of a vehicle can generate an infeasible solution. Considering following temporary solution while constructing the tours:

Team 1:	0-2-3-4-0
Truck 1:	1-2-4-1
Truck 2:	1-3-1

We want to insert node 5, for example in the tour of team 1, between 3 and 4.

Team 1:	0-2-3-5-4-0
Truck 1:	1-2-4-1
Truck 2:	1-3-1

This position within the tour of team 1 restricts feasible positions for the insertion in the tours of truck 1 or 2. One position, which could not work in this case would be for truck 1 right after leaving the depot:

Team 1:	0-2-3-5-4-0
Truck 1:	1-5-2-4-1
Truck 2:	1-3-1

In this case, team 1 goes from the depot to node 2 and would wait for a truck. However, truck 1 leaves the depot proceeding to node 5 and would wait for a team. This leads to a deadlock situation, where each vehicle waits for one of the other class, which makes the solution is infeasible. By positioning node 5 after node 2 in the tour of truck 1 or after node 3 in the tour of truck 2 gives a feasible solution, because these nodes will have been visited by the team at the time of arriving node 5. The solver is checking infeasibility in the evaluation procedure, and the described situation of infeasibility was already prevented in the solver in the construction algorithm in an effort to optimize it. However, there can be still infeasibilities when it comes to interdepencies with additional teams. Preventing more of these situations already in the construction phase could help making the solver more effective.

3.7. Solver description

Figure 3.7 describes the basic architecture of the solver. The solver class is responsible for the overall coordination and solution process. In the beginning, this class creates a new instance, based on the data given in the external data file. This instance contains

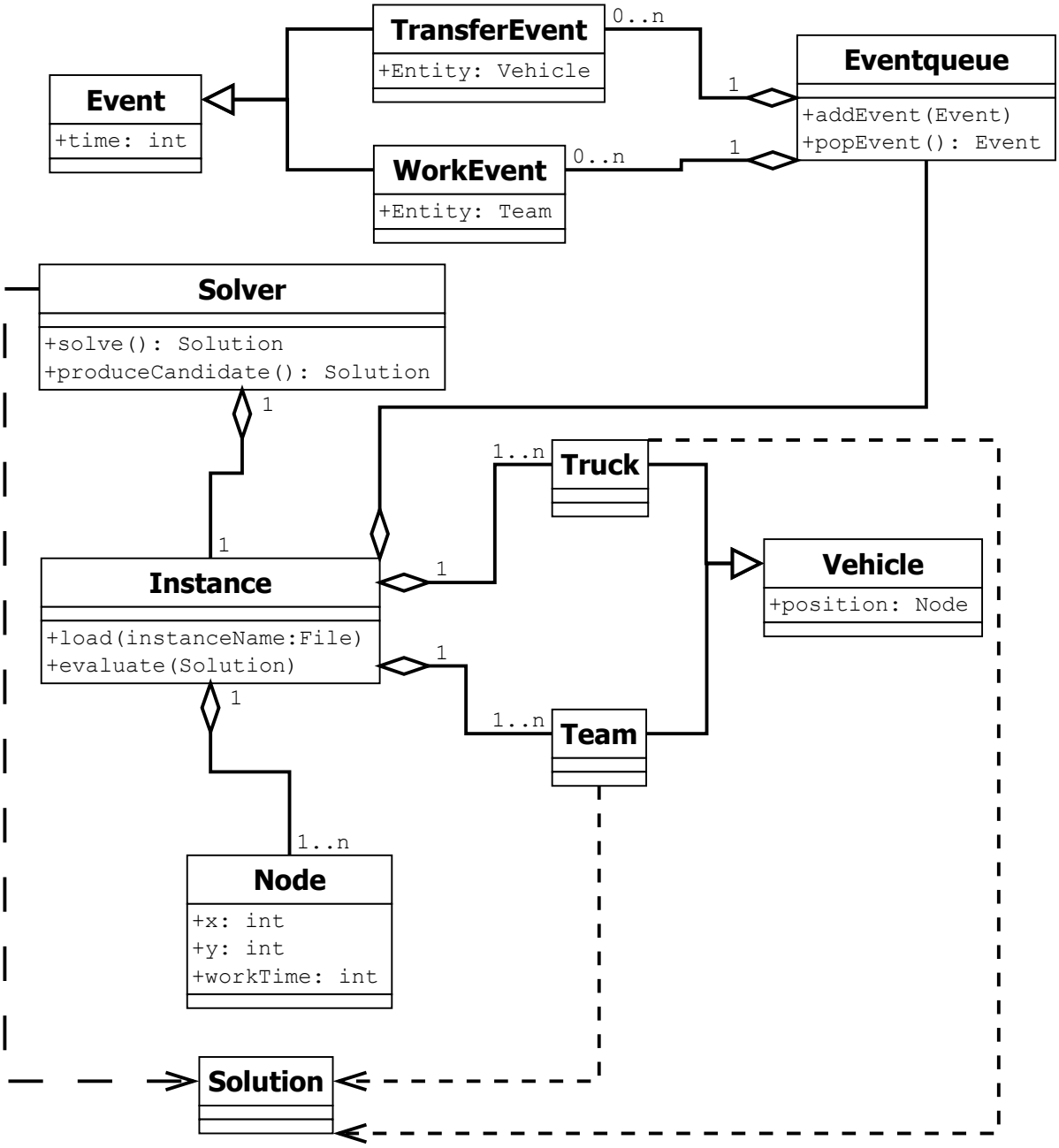

Figure 3.7.: UML diagram describing the solver

the defined nodes, for which one Node-class is created each. Furthermore, the teams and trucks are instantiated[9] as given by a command-line parameter, and placed on the team- and truck-depots.

The solver generates candidate solutions as explained in section 3.5 and passes them to the instance for evaluation. When starting the evaluation, the instance takes control of the discrete event simulation and generates TransferEvents and WorkEvents and places them in the Eventlist. The process works exactly as explained in section 3.4. After the solution is simulated, the obtained objective value or infeasibility of the solution is passed back to the solver.

3.8. Lower bounds

Heuristics do usually not produce optimal results, and moreover, the gap to optimality is unknown. As shown above, a plain approach with a MIP implementation allows to find optimal solutions only for very small instances. On the other hand, in order to evaluate the quality of a heuristic procedure, this information would be essential, particularly for bigger instances. In order to assess the solution quality, it is therefore advisable to reflect about alternatives for finding a value with less computational effort, which is guaranteed to be less or equal, but as close as possible to the optimal solution.

As a principle, an objective value will always be better or equal, if constraints are removed[10]. Thus, it would be interesting to decouple the two vehicle classes and see, what solutions can be found without the synchronization constraint.

If $z_T^\star$ represents the minimum value for the team min-max VRP, and $z_A^\star$ the analogous value for the trucks, we could define a lower bound of the synchronized problem as

$$\underline{z} = \max(z_T^\star, z_A^\star)$$

Table B.5 in the appendix shows the optimal objective values for both unsynchronized cases of the test instances, including run-times, and the solutions to the synchronized problems. Indeed, the run-times for solving the unsynchronized MIP models are generally less than solving the synchronized model. However, the gain is not big, and what is even more unfavorable, the sum of the run-times for both unsynchronized problems

[9] This is a term of object oriented programming and describes the creation of an object which is described by its class definition.

[10] In our minimization problem this means it goes down.

sometimes exceeds the run-time of the original problem. Still, this is already progress for two main reasons.

First, to determine if a heuristic solution is optimal, it is not obligatory to solve both unsynchronized models. If the heuristic solution equals the solution of the unsychronized integer program which is solved first, then this will be the indisputable optimum solution of the synchronized problem. However, if this is not the case, the second problem has to be solved as well, and the higher value will be the lower bound.

Second, as already mentioned in section 2.4, some work has been done on solving min-max problems to optimality. This algorithms could be adapted to produce good lower bounds within a practicable period of time.

4. Discussion of results and computational experiments

In this chapter we will discuss the proposed heuristic algorithm and amongst others try to answer the questions stated by Barr et al. (1995):

1. What is the quality of the best solution found?
2. How long does it take to determine the best solution?
3. How quickly does the algorithm find good solutions?
4. How robust is the method?
5. How "far" is the solution from those more easily found?
6. What is the trade-off between feasibility and solution quality?

On one hand, the properties of the algorithm will be evaluated by theoretical analysis, on the other hand the solver will be considered as black box, on which computational experiments are conducted in an empirical analysis. Hall and Posner (2001) noted, that there is wide concern about many computational experiments being inadequate. Amongst other errors, test data may be biased, or comparison of algorithms lack statistical significance. The design and analysis of the experiments was done with guidelines of Greenberg (1990), Barr et al. (1995), Coffin and Saltzman (2000), and Hall and Posner (2001) in mind. The given suggestions were followed as long as they showed reasonable, practicable and realizable.

4.1. Computational environment

The computational tests were conducted in the following environment:

Model:	Acer Aspire TimelineX 5820TG
Processor:	Intel(R) Core(TM) i5 CPU M 480 @ 2.67GHz
Memory:	8 GB RAM
Operating System:	Windows 7 (64 Bit)

4.2. Evaluation of the proposed approach

Cordeau et al. (2002) assessed VRP heuristics with respect to 4 different criteria. This section evaluates the proposed heuristic approach in respect to these attributes.

Accuracy

This property focuses on several aspects of accuracy. First, it describes the gap of an heuristic solution from an optimal value. The presented heuristic solver was mostly able to find the optimal value, where it was possible to determine an exact solution within reasonable time with the MIP model. Only in 2 cases the optimal value could not be found by the heuristic[1]. However, if the optimal value is not known, it is not possible to determine, if the obtained heuristic value is optimal.

One other aspect regarding the accuracy of the heuristic is consistency. This means, that an accurate heuristic should perform well most of the time. There should be no sporadic situations, where it performs poorly. The presented algorithm worked well for all test-instances. However, it must be again pointed out, that in some very rare cases, which are also dependent on number of teams and trucks, optimality cannot be reached, but still good solutions were obtained.

The last aspect is the fact, that users prefer algorithms, that give quite good solution at an early stage. Also in this respect the presented algorithm works quite well. As shown in the exemplary solver run in Figure 4.1, the algorithm improves the best solution quickly in the beginning, giving a solution after 9 seconds or 68 iterations, which is only 6.15% away from the best known solution and was found after 6076 seconds or 45.000 iterations. One interesting property, which is striking here, is the fact, that the minimum objective values over time can be closely approximated by a logarithmic regression. This follows the ideas of Oppen and Woodruff (2009) and gives interesting opportunities of predicting search behavior.

[1]These two cases were found when performing tests for Table B.2 in the appendix.

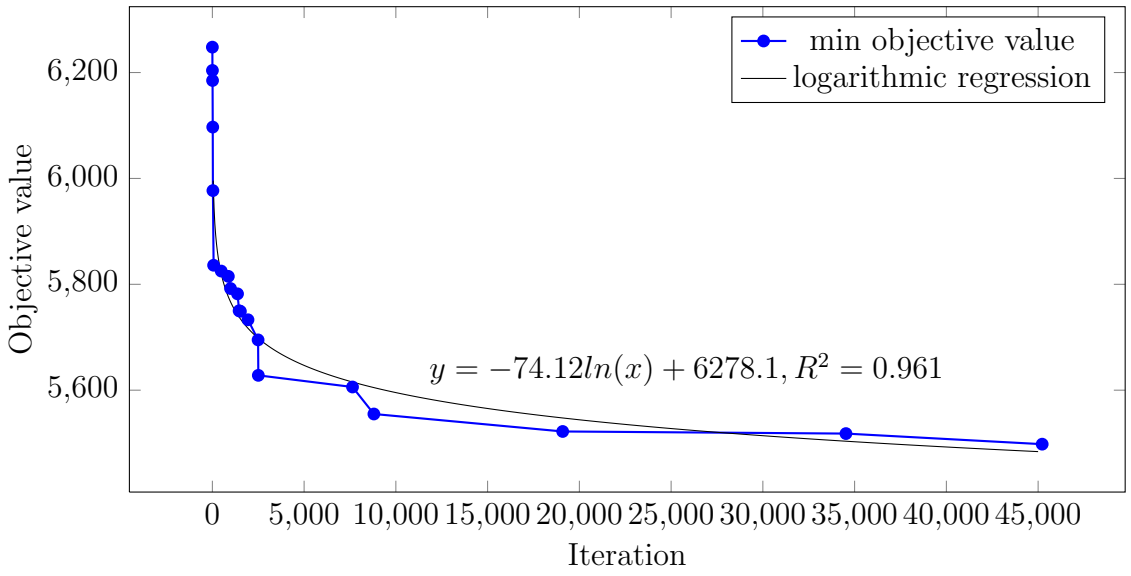

Figure 4.1.: Minimum objective value while optimizing 40-nodes in instance 000, 3 teams and 2 trucks

To provide a numerical indication how fast this algorithm can find good solutions, the quality-effort relationship $r_{0.05}$ was determined. This measure follows a suggestion of Barr et al. (1995) and is basically an indicator, how fast a good solution can be found. If we define $t_{0.05}$ as the time to find a solution which is within 5% of the best known value, and $\hat{t}$ the time to find the best known solution, then this indicator is defined as

$$r_{0.05} = \frac{t_{0.05}}{\hat{t}}$$

The results can be found in appendix B.2. The very small values of $r_{0.05}$ indicate a fast convergence of the solution value.

Speed

The importance of the speed of the solver is very much dependent on its application and the planning level. While there are applications, where an instant answer is required because the customer is waiting on the phone-line, tactical planning allows considerably more calculation time. Describing the speed of an algorithm can be quite difficult, especially if it should be compared to other algorithms in scientific literature. Runtimes depend considerably on the used hardware. Even if another algorithm is run on the same machine, it still depends on the quality of the implementation and the degree of optimization in the program code.

In order to measure the performance of the heuristic solver, the MIP implementation was taken as a reference. Figure 4.2 shows the run-time of both MIP solver and heuristic solver in comparison. It is worth to mention, that the y-axis is logarithmic.

In order to determine run-time behavior, the instance was solved in multiple runs for the first 6 to 11 nodes. The MIP solver was used to obtain the optimal value, and this value used as a stop-criterion for the heuristic solver in the second step. Both run-times were recorded and visualized in the chart. The test was done with other instances as well and showed the same behavior. The results of all instances are shown in Table B.3 in the appendix.

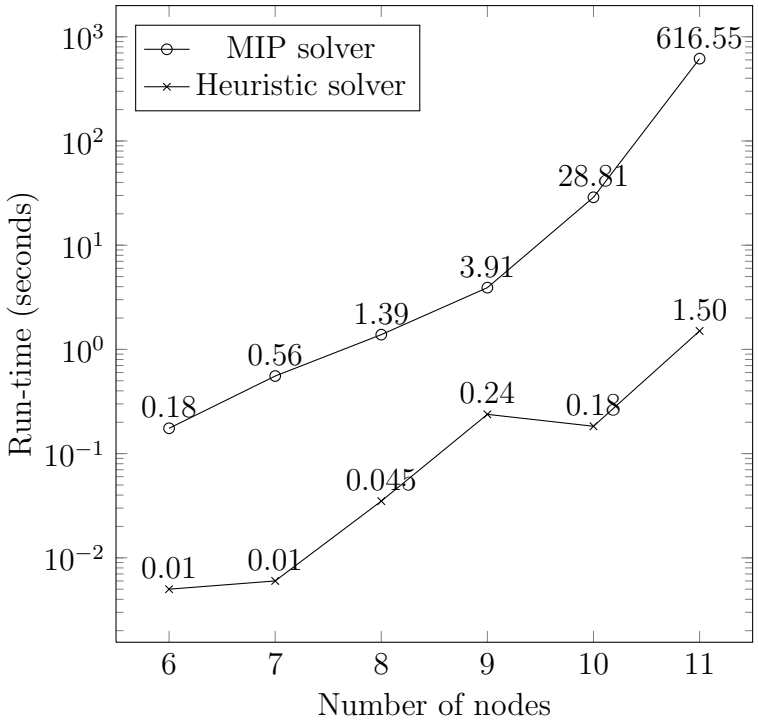

Figure 4.2.: Comparison of MIP and metaheuristic solver. Run-times for instance 005, with 3 teams and 2 trucks, dependent on the number of projects.

Especially two things are noticeable. First, the heuristic solver is much faster than the MIP implementation. But what is even more important, is the fact, that the heuristic solver shows a much flatter slope with increasing number of projects. This provides evidence, that the solver will be able to handle also bigger instances with a good solution quality.

Simplicity

Even if some heuristics show superior performance to other ones, the effort of implementation needs to be considered as well. The savings heuristic of Clarke and Wright (see Clarke and Wright 1964), for example, is easy to understand and to implement, and many times serves its purpose. It is therefore still taught at universities and popular among practicioners.

Furthermore, particularly in the field of metaheuristics, one issue regarding simplicity is the use of parameters. The number of parameters of some metaheuristics are legion. Xu and Kelly (1996), for example, use 32 different parameters in their network flow-based tabu search heuristic. Articles like Coy et al. (2000) just deal with how to find effective settings of parameters by statistical design of experiments and gradient descent, and there are even metaheuristics like the one of Hepdogan et al. (2005), which optimize the parameter settings of metaheuristics. It is discussible, to what extent a metaheuristic which contains a multitude of parameters, that are tailored and tuned to perform well on the test instances, can be ranked superior to competitors.

The proposed algorithm is extremely easy to understand and requires little implementation effort. The most complex part is actually the evaluation routine. Moreover, this solver does not need any parameters which need to be set and tuned.

Flexibility

Extending a heuristic to solve a VRP with additional constraints can be a critical issue. Especially if real world problems need to be solved, this can be a deal-breaker. Coming back to the aforementioned Clark and Wright heuristic, it is to note that for this very fast and simple algorithm it is difficult to include additional constraints. The difficulty lies not only in how to include this constraint, but also in the fact, that solution quality can deteriorate sharply.

For the presented algorithm additional constraints can be added easily, and the behavior of the system can be changed without effort, as the concept of discrete event simulation is very flexible. However, it is to suspect that this will strongly impair the solution quality, as in its character it is a greedy algorithm. There are no mechanisms to correct "youthful follies" in the construction process, that is, earlier insertions will stay at their place. Yet the algorithm can be improved by adding such a mechanism, which is subject to further research.

4.3. Processor utilization

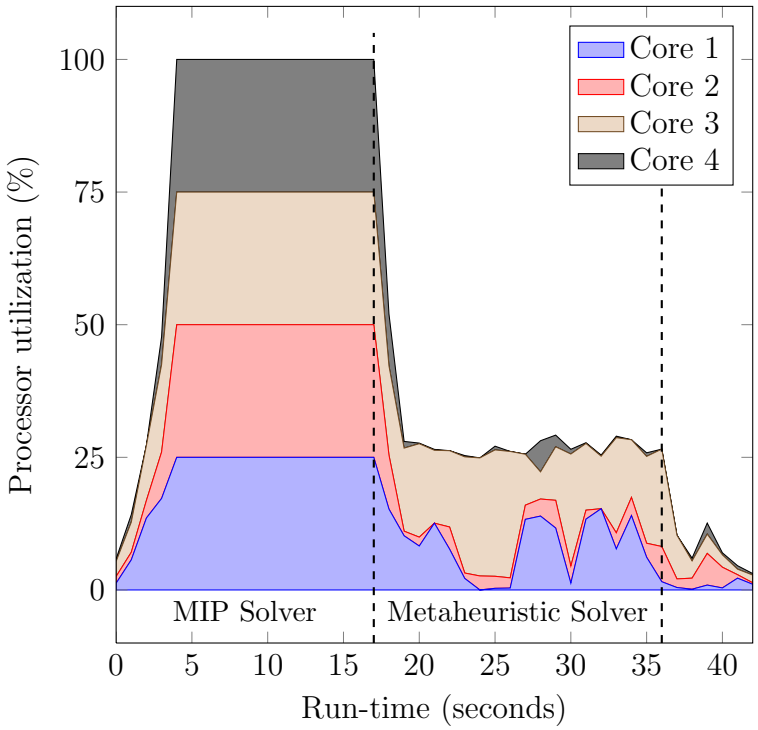

Figure 4.3.: Processor utilization while running the solvers.

Figure 4.3 shows the processor utilization on a quad-core CPU, running the MIP-solver the first 17 seconds and immediately after that running the Metaheuristic solver from second 17 to second 36. This diagram is not intended to provide any information about the run-times of these two solvers, but it points out something, that should be taken into consideration when comparing run-times: It is important to set the stated run-times in relation to processor utilization. Modern processors have multiple cores, that can run in parallel.

However, software needs to be adapted to fully utilize the possibilities of parallelization in order to speed-up run-times. While the MIP solver utilizes all cores, the heuristic solver only uses one. The program is not always run on the same core, which results in times of utilization on all cores, but it can be clearly seen, that the total processor utilization is at around 25%. The presented metaheuristic solver can easily be parallelized, as there would be no need of coordinating the parallel threads among each other, which would require more advanced mechanisms as researched in Jin (2013). One can assume that this will lead to a significant speed gain, however this topic is not within the scope

of the thesis and is subject to further research, where Crainic (2008) or Crainic and Toulouse (2010) could be a good starting point.

The stated run-times of the MIP solver are under assumption of using 4 cores, the heuristic solver uses just 1 core. This inequity in terms of prerequisites always should always be kept in mind when comparing the run-times.

4.4. Code profiling

In order to find potentials for improvement, the solver code can be examined by profiler software, as it was done in this case with the open source profiler "Very Sleepy"[2]. This software is able to evaluate, how much time is spent in various functions. Figure 4.4 shows the different function calls, and how much time is spent. The "Exclusive" column shows the time spent barely in the named function, without considering calls of other functions. The "Inclusive" column considers the time of other function calls as well.

The profiling shows, that considerable time, namely 87.27% of the time is spent with evaluating the solutions. Within the 96 seconds of profiling the evaluation was called 3.700.000 times. This is already the result of some tuning which was conducted within the evaluation function. However, it would be still desirable to improve it.

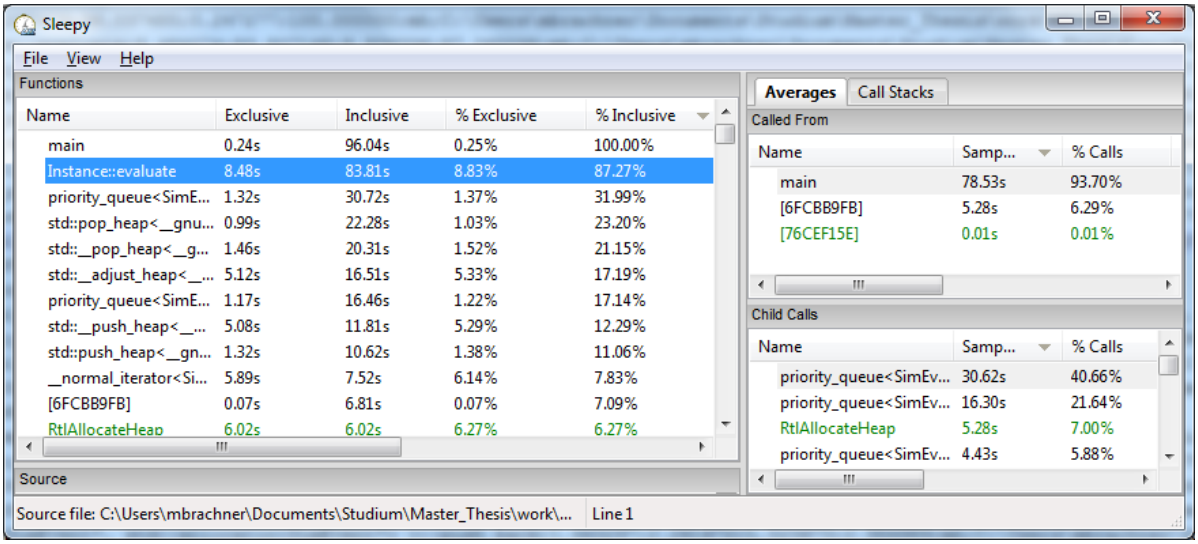

Figure 4.4.: Profiling the code reveals potentials for improvement

Two approaches of improving the existing algorithm could be identified:

[2]see `http://www.codersnotes.com/sleepy` (accessed 23.04.2013)

- Improve the speed of the evaluation: Any improvement that can be achieved within the evaluation algorithm would have a considerable impact on the overall performance: A reduction of 10% just in the evaluation function would lead to a theoretical reduction of the solver runtime by 10% x 87.27% = 8.73%. An interesting approach could be to store the system state at some point for reuse. When inserting a node into a solution, all the previous part would still be feasible and there is no need to simulate the model again up to this point.

- Limit the need of evaluating the solution: The algorithm uses the evaluation function not only to determine the objective value, but also to check feasibility. The detection of infeasibility may be achieved by approaches that are cheaper in terms of computational time. Furthermore, if it would be possible to create only feasible solutions, then a feasibility check would not be needed. However, in the concrete case only 170.000 of the 3.700.000 were infeasible. If it would be possible to eliminate these infeasibilities, this would mean a theoretical performance gain of around 4% for the solver.
  Another possibility to decrease the number of evaluations needed would be to detect earlier in the course of constructing the tours, that the solution so far will not lead to an improved solution. The present version inserts nodes from the pool one by one until the pool is empty and the solution is complete. The insertion of one node into a solution with n nodes leads to nearly $(n+1)^2$ evaluations[3]. Especially if insertions into nearly complete solutions could be avoided, one can assume that this will lead to a performance gain.

4.5. Possible applications

With the heuristic solver as a tool to find good solutions for generated instances, it is interesting to perform some computational experiments to analyze the behavior of solutions for synchronized problems and to reflect on different applications of the solver. It should again be pointed out very clearly, that without capacity constraints the algorithm will not be ready for the real world problem which served as a starting point. However, in this thesis the focus is on the synchronization aspect, and this section deals with peculiarities and consequences of working with a synchronized system.

[3]it is less than $(n+1)^2$ because it was possible to avoid some clearly infeasible solutions

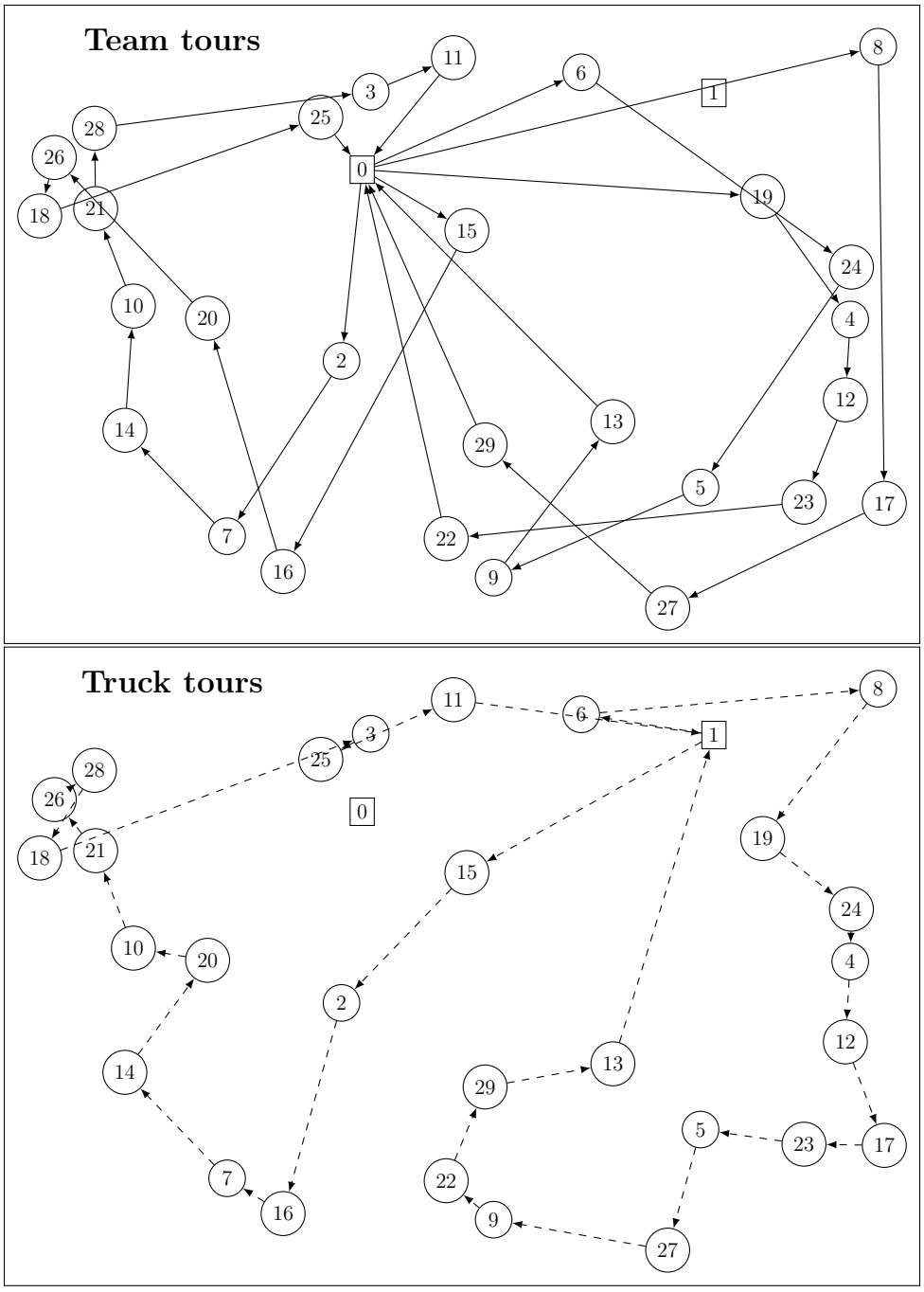

Figure 4.5.: Instance 041 with 5 teams and 2 trucks

The most obvious application is the operative tour-planning for teams and trucks. Figure 4.5 shows an example of planned tours. It is striking that in this solution the team tours do not follow the cloverleaf-pattern which is typical for a VRP. In fact, the tours are placed rather in parallel, in a way, that it is easy for the trucks to be at another node quickly after having supplied one node. On the other hand, the truck tours try to follow the main bundle of these parallel team tours in order to minimize the tour distance.

Even without capacity constraints for the trucks these solutions could be practicable. When following the solution for the team tours as suggested, the projects are scheduled to be close in respect to time and space, therefore trucks can easily supply the projects in one area without moving too long distances between them.

Talking about synchronization, we have a mutual dependency of resources. Even if the number of teams is increased, there may be not enough trucks, and they will spend a disproportional part of their working time waiting to be supplied with asphalt. On the other hand, too many trucks will be of no use for the same reasons. Therefore, it is reasonable to plan the fleet sizes of these two different classes, obtaining a balance between them. For this purpose, the solver could serve as a support decision tool for fleet sizing.

Figure 4.6 shows the minimum objective value as a function of the number of teams and shows to alternatives, one with 1 truck, the other with 2 trucks. The objective value was determined by running the solver for 1 minute for each combination of teams and trucks.

It is quite obvious, that a higher number of teams improves the objective value considerably, but the improvement flattens out with an increasing number of teams. However, using 2 trucks instead of 1 has no big impact. The same calculations were done on another instance, which is shown in Figure 4.7.

In this case increasing the number of teams has very little effect when using just 1 truck. However, adding 1 truck more causes a significant improvement by itself, and increasing the number of teams gives again a better objective value.

In general the two instances were generated with the same parameters, however the vital difference is the project sizes. In the first case the sizes are much wider spread between 5 and 3000 time units, while the second instance contains only small projects between 5 and 200 time units. This indicates, that the structure of projects to be planned influences the requirements of an ideal fleet composition.

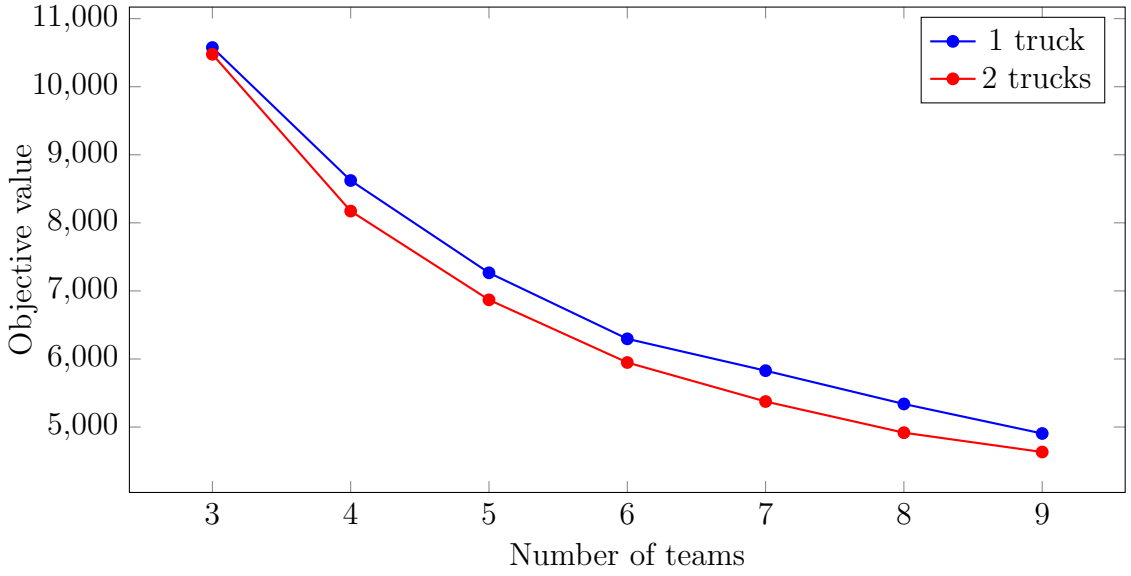

Figure 4.6.: Instance 025 with 18 projects between 5 and 3000 time units

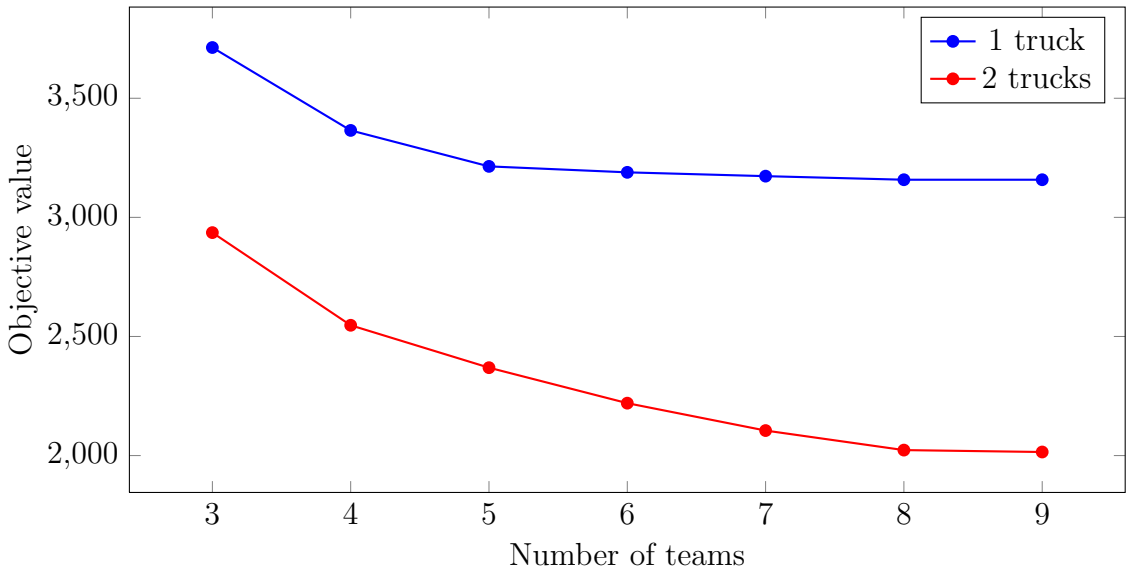

Figure 4.7.: Instance 000 with 18 projects between 5 and 500 time units

5. Extending the model

Until now the problem focused mainly on the synchronization part of the problem, which is also the main purpose of the master thesis. Time is very limited for a master thesis, whereas possibilities are manifold. However, the author wants to unlock some doors on the way to real world problems, particularly the road construction problem, which was the initial motivation and inspiration. In this chapter we will look at the main extensions of the basic problem, that will enable us to have a much more realistic model, which already could be considered useful for supporting real-world applications.

5.1. An extended MIP model

The model so far has four serious drawbacks, that makes this version prohibitive for use in the presented problem:

1. The asphalt trucks are assumed to have unlimited capacity. Thus the tours can get too long, and the truck can get empty when fulfilling its tour.

2. If there would be capacity restrictions, we would need a possibility to reload the trucks with asphalt at specialized nodes within its tour in order to fulfill its deliveries.

3. In this model it is assumed that the project gets only one delivery of asphalt and it lasts less than one day.

4. The planning is only done for one day. No multiple-days planning-horizon is implemented.

Regarding the first point, we have to add capacity constraints. There will still be assumptions, but they are much more realistic: We will assume, that each project will get the same amount of asphalt per delivery. This is quite realistic, inasmuch as the asphalt supply is usually loaded onto an asphalt-spreading machine, which typically has

a capacity of 10 tons. We define the amount for one filling of this asphalt-spreading machine as *one load unit*. By assuming this, we can easily see, that the capacity of the truck equals at the same time the number of projects, which can be supplied at most by this vehicle. Thus, if the trucks have the capacity of three load units each, at most three projects can be supplied by them. We will also assume, that the fleet is homogeneous, having the same capacity restriction of C load units each.

Having this capacity restriction in mind, the process of the model must be extended, such that a truck would be able to load asphalt at *loading depots*. This kind of nodes are not obligatory to visit, however, by visiting one of them the truck will be able to continue its tour, even if it has reached the maximum projects to be served before. Furthermore, the truck needs to visit one of these depots after leaving the truck depot in order to begin its supply tour. A typical tour of such an asphalt truck with capacity $C = 2$ would look like this: [Truck depot]–[Loading depot]–[Project 1]–[Project 2]–[Loading depot]–[Project 3]–[Project 4]–[Truck depot].

These loading depots usually can be visited multiple times in reality. In the model this will be approached by duplicating the loading depot, similar to what was proposed by Rubasheuskaya (2012). This approach gives not only the opportunity to base it on the original formulation, but also adds the flexibility to place alternatively located loading depots, such that the nearest location can be chosen in the optimization run.

To tackle issue number three, the project nodes can be duplicated for each needed delivery. Thus, a project that needs 60 time units, but delivery is needed every 20 time units, could be splitted into 3 projects with 20 time units each. Typically these splitted projects would be in a row, but this is not obligatory. An adjacency constraint as proposed by Rubasheuskaya (2012), which allows these projects only to be executed in a row could be considered, however it would exclude the possibility to work on projects with more than one team and other cases, where the optimal solution would propose to split one big project into several units of construction. Other constraints to represent real-world conditions as, e.g., set-up times when going to another project location, would in this case perhaps be more reasonable, but are not within the scope of this thesis.

Finally, in order to plan for several days, the dimension of the decision variables need to be extended and constraints need to be added to ensure integrity. Not to forget, teams and trucks need to be scheduled for the same day, this means if a team visits a project node, a truck needs to do the same and vice versa.

Finally, we have the following MIP model:

Sets

$\{0\}$ team depot

$\{1\}$ truck depot

$\mathcal{N}$ set of projects

$\mathcal{T}$ set of teams

$\mathcal{L}$ set of loading depots

$\mathcal{D}$ set of time periods within a planning horizon (number of working days)

Parameters

U Maximum working day duration

M Number of teams

P Number of trucks

C Capacity of trucks in load units

S_{ij} traveling time between locations i and j

Z_i Duration of job i

Variables

$\beta_{ij}^d = \begin{cases} 1 & \text{if arc between location } i \text{ and } j \text{ is traversed on day } d \\ 0 & \text{otherwise} \end{cases}$

$\lambda_{ij}^d = \begin{cases} 1 & \text{if a team fulfills job } j \text{ after job } i \text{ on day } d \\ 0 & \text{otherwise} \end{cases}$

e_i^d number of arcs to node i from last depot on day d

f^d biggest number of arcs from last depot on day d

t_i^d arrival time from location i at day d

u^d latest return time of all trucks on day d. The return time is defined as the time, when the truck arrives at node 1.

x_i^d starting time of job i fulfillment at day d

y^d latest return time of all teams on day d. The return time is defined as the time, when the team arrives at node 0.

w latest time

$$\min w \tag{5.1}$$

subject to

$$w \geq u^d \qquad \forall d \in \mathcal{D} \tag{5.2}$$

$$w \geq y^d \qquad \forall d \in \mathcal{D} \tag{5.3}$$

$$\sum_{d\in\mathcal{D}} \sum_{j\in\mathcal{N}:i\neq j} \lambda_{ij}^d = 1 \qquad \forall i \in \mathcal{N} \tag{5.4}$$

$$\sum_{i\in\{0\}\cup\mathcal{N}} \lambda_{ij}^d = \sum_{k\in\{0\}\cup\mathcal{N}} \lambda_{jk}^d \qquad \forall j \in \{0\} \cup \mathcal{N}, d \in \mathcal{D} \tag{5.5}$$

$$\sum_{j\in\mathcal{N},d\in\mathcal{D}} \lambda_{0j}^d \leq M \qquad \forall d \in \mathcal{D} \tag{5.6}$$

$$U(1-\lambda_{ij}^d) + x_j^d \geq x_i^d + Z_i + S_{ij} \qquad \forall i \in \{0\} \cup \mathcal{N}, j \in \mathcal{N}, d \in \mathcal{D} \tag{5.7}$$

$$U(1-\lambda_{i0}^d) + y^d \geq x_i^d + Z_i + S_{i0} \qquad \forall i \in \mathcal{N}, d \in \mathcal{D} \tag{5.8}$$

$$\sum_{d\in\mathcal{D}} \sum_{j\in\mathcal{N}\cup\mathcal{L}:i\neq j} \beta_{ij}^d = 1 \qquad \forall i \in \mathcal{N}, d \in \mathcal{D} \tag{5.9}$$

$$\sum_{j\in\{1\}\cup\mathcal{N}\cup\mathcal{L}} \beta_{ij}^d = \sum_{k\in\{1\}\cup\mathcal{N}\cup\mathcal{L}} \beta_{jk}^d \qquad \forall j \in \{1\} \cup \mathcal{N} \cup \mathcal{L}, d \in \mathcal{D} \tag{5.10}$$

$$\sum_{j\in\mathcal{N}\cup\mathcal{L}} \beta_{1j}^d \leq P \qquad \forall d \in \mathcal{D} \tag{5.11}$$

$$U(1-\beta_{ij}^d) + t_j^d \geq t_i^d + S_{ij} \qquad \forall i \in \{1\} \cup \mathcal{N} \cup \mathcal{L}, j \in \mathcal{N} \cup \mathcal{L}, d \in \mathcal{D} \tag{5.12}$$

$$U(1-\beta_{i1}^d) + u^d \geq t_i^d + S_{i0} \qquad \forall i \in \mathcal{N} \cup \mathcal{L}, d \in \mathcal{D} \tag{5.13}$$

$$t_i^d = x_i^d \qquad \forall i \in \mathcal{N}, d \in \mathcal{D} \tag{5.14}$$

$$\sum_{j\in\{0\}\cup\mathcal{N}} \lambda_{ij}^d = \sum_{j\in\{1\}\cup\mathcal{N}\cup L} \beta_{ij}^d \qquad \forall i \in \mathcal{N}, d \in \mathcal{D} \tag{5.15}$$

$$\sum_{d\in\mathcal{D}} \sum_{j\in\{1\}\cup\mathcal{N}\cup\mathcal{L}:i\neq j} \beta_{ij}^d \leq 1 \qquad \forall i \in \mathcal{L} \tag{5.16}$$

$$C(1-\beta_{ij}^d) + e_j^d \geq e_i^d + 1 \qquad \forall i \in \{1\} \cup \mathcal{N} \cup \mathcal{L}, j \in \{1\} \cup \mathcal{N}, d \in \mathcal{D} \tag{5.17}$$

$$C(1-\beta_{i1}^d) + f^d \geq e_i^d + 1 \qquad \forall i \in \mathcal{N} \cup \mathcal{L}, j \in \{1\} \cup \mathcal{L}, d \in \mathcal{D} \tag{5.18}$$

$$f^d \leq C + 1 \qquad \forall d \in \mathcal{D} \tag{5.19}$$

$$\sum_{j\in\mathcal{L}} \beta_{1j}^d = P \qquad \forall d \in \mathcal{D} \tag{5.20}$$

This model is an extension of the basic model and contains the following additions: First, most of the variables have been extended by the dimension of the planning horizon, i.e. an index d was added. This index describes, on which day within the planning horizon this variable is taking the value. All existing constraints have been changed to incorporate this new dimension. It was also necessary, to add the new family of

constraints (5.15), which causes, that a project is visited by both the team and the truck on the same day.

The constraints (5.16) allow maximum one truck to visit a loading depot within the planning horizon. However, it is not obligatory to visit all loading depots. Even if this approach allows one to include more loading depots than needed into the instance data, it is advisable to keep the number of loading depots as small as possible in order to keep the problem small. This applies especially when duplicating a depot to imitate multiple supplies from the same location. However, this approach can be also used to select the best locations, which makes it necessary to include all options.

Constraints (5.17) and (5.18) determine the number of arcs from the last loading depot. In principle they work similar to the constraints (5.7 and 5.8), (5.12 and 5.13) for determining the time[1]. However they are formulated in a way, such that the variable which determines the number of arcs can take on the value zero after a loading depot without violating the constraints. The constraints (5.19) restrict the maximum distance between two loading depots or a loading depot and the truck depot to the truck capacity plus one. This is because we do not count the nodes between the depots, but the connections. If, for example, 2 nodes are placed between the depots, 3 arcs connect them.

Finally, constraints (5.20) state, that the sum of all outgoing connections from the asphalt depot to a loading depot should be the number of trucks used. Moreover, together with constraint (5.11) this means, that it is not allowed to go from the truck depot to other nodes but loading depots.

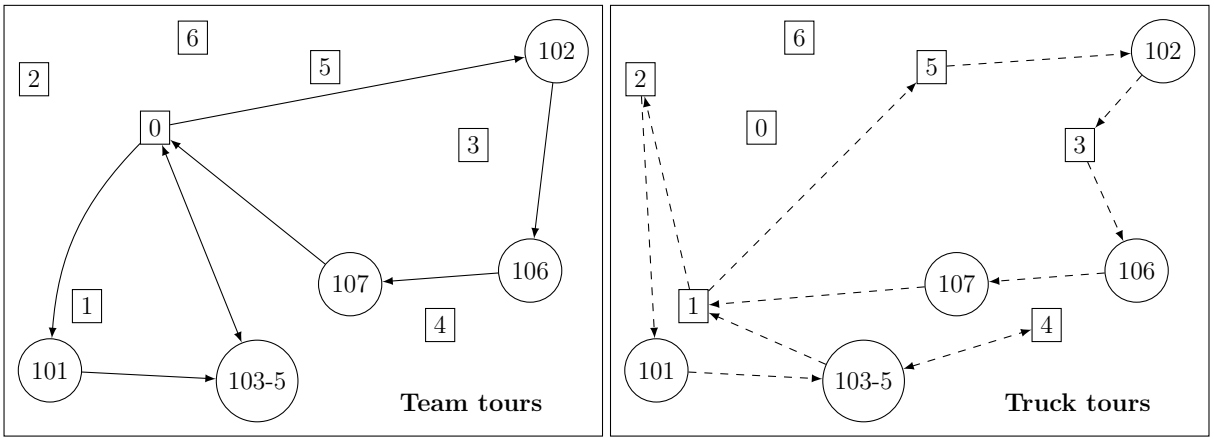

Figure 5.1.: Solution for instance 040, 3 teams and 2 trucks, planning horizon 1 day.

[1]See the constraints (2.7) and (2.8) in the basic model in section 2.2 for an explanation of this mechanism.

Figure 5.1 shows a solution for instance 040. We route 3 teams and 2 trucks, and assume the capacity C of the trucks to be 2 load units. In table form the tours of the teams and trucks are as follows:

Team 1	0-101-103-0
Team 2	0-102-106-107-0
Team 3	0-103-103-0
Truck 1	1-2-101-103-4-103-103-1
Truck 2	1-5-102-3-106-107-1

Still, node 0 is the team depot, and node 1 is the truck depot. Additionally, there are now depots 2 to 6, which are depots for loading asphalt. The projects are numbered from 101 upward to distinguish them better from the infrastructural facilities. One important fact is also node 103. This is a node, which is duplicated, as it is a bigger project and needs 3 times longer than the others ones. This means, in the instance data node 104 and 105 are placed on the same location as node 103.

Team 3 is busy exclusively with project 103. However, after team 1 is finished with project 101, it will also join to help team 3 with finishing the big project. Meanwhile, team 2 works on the rest of the projects. Truck 1 goes first to the asphalt depot 2 and continues further to 101. Then it can supply also project 103 the first time, before it gets empty and needs to go to asphalt depot 4 to fill up again. Afterwards, the truck can go back to project 103 to supply it again. It will just stay there and wait until the second load is demanded for this project. After this, it will go back again to the truck depot. Truck 2 will supply the other projects meanwhile. It is to mind, that it seems more advantageous to only load supply for project 102 at asphalt depot 5, and to take 2 loads on depot 3 for supplying project 106 and 107. Depot 6 is not taken into consideration.

Figure 5.2 shows the - admittedly rather trivial - solution for the other extension, the planning horizon, again in tabular form as well:

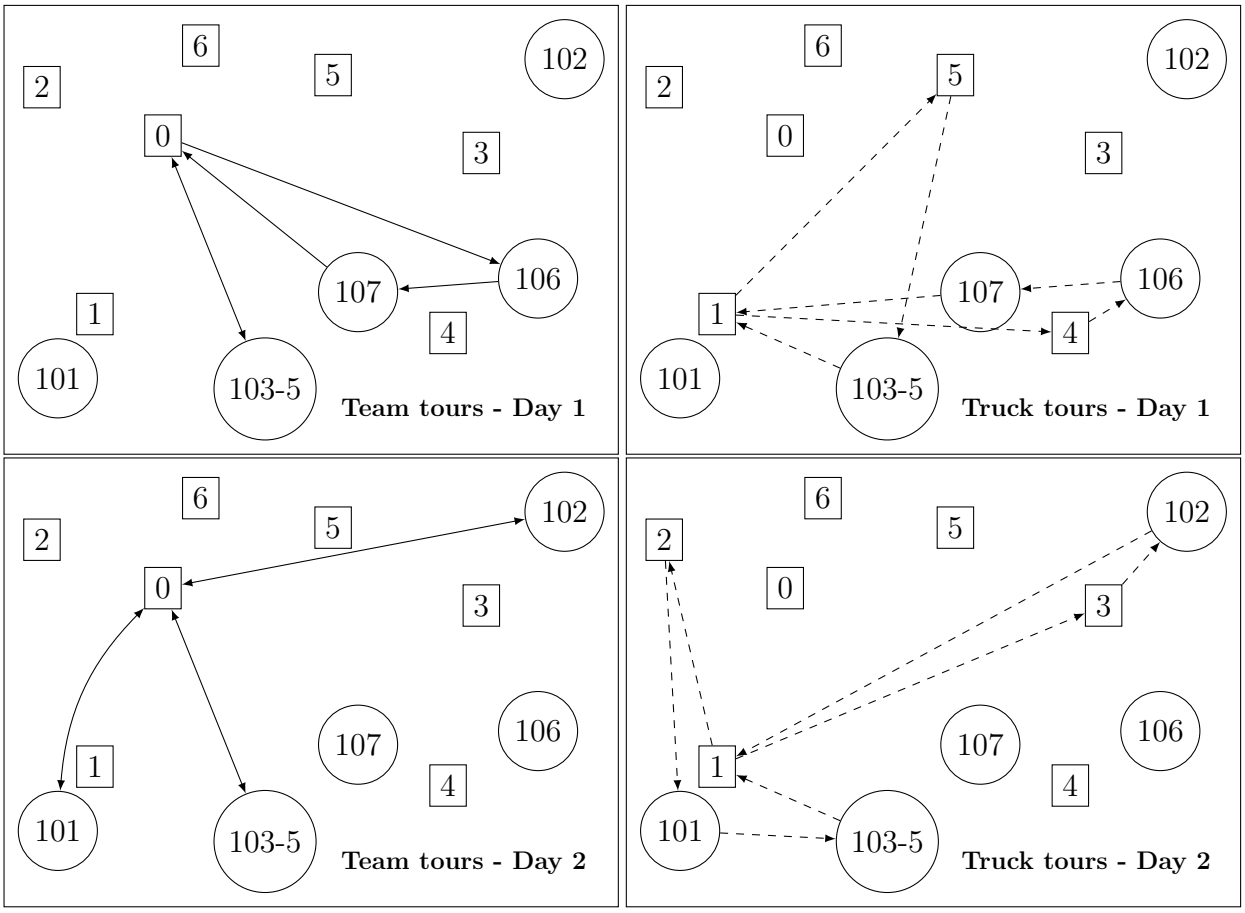

Figure 5.2.: Solution for instance 040 (extended version), 3 teams and 2 trucks, planning horizon 2 days.

	Day 1	**Day 2**
Team 1	0-103-0	0-101-0
Team 2	0-103-0	0-102-0
Team 3	0-106-107-0	0-103-0
Truck 1	1-4-106-107-1	1-2-101-103-1
Truck 2	1-5-103-103-1	1-3-102-1

Project 103 is spread over 2 days, which seems reasonable. This solution is a proof-of-concept, however, run-times make it impossible to work on bigger problem sizes.

5.2. Run-times of the MIP solver with extended model

Even if the basic model still showed reasonable run-times up to 10 projects, the extended model proves to be prohibitive for instances of this size. Besides the additional constraints, decision variables for the loading depots need to be added, which increases the problem size considerably.

Figure 5.3 shows a comparison of the 2 models on basically the same data (however, the extended instance uses additional loading depots). It is quite visible, that the run-time behavior of the extended model is by far worse than the basic model.

What is very striking, is the considerable sensitivity to the number of teams and trucks compared to the basic model. While one instance with 8 nodes, 3 teams, 2 trucks, and a planning horizon of 2 days still could be solved in 7 minutes, the same instance with 2 teams and 1 truck needed more than 6 hours.

5.3. Extending the heuristic solver

Even if a ready-made implementation of the extensions in the heuristic solver is not in the scope of this thesis, some propositions are given, how this implementation could look like.

The extension of the discrete event simulator for evaluation of the solution is fairly easy and straight-forward. Particular attention should be paid to the fact, that the simulation has to consider load, and should indicate infeasibility in case the truck is empty when trying to supply a project.

The main challenge in extending the heuristic solver is beyond doubt the adaption of the algorithm to construct a candidate solution. While it was already difficult to

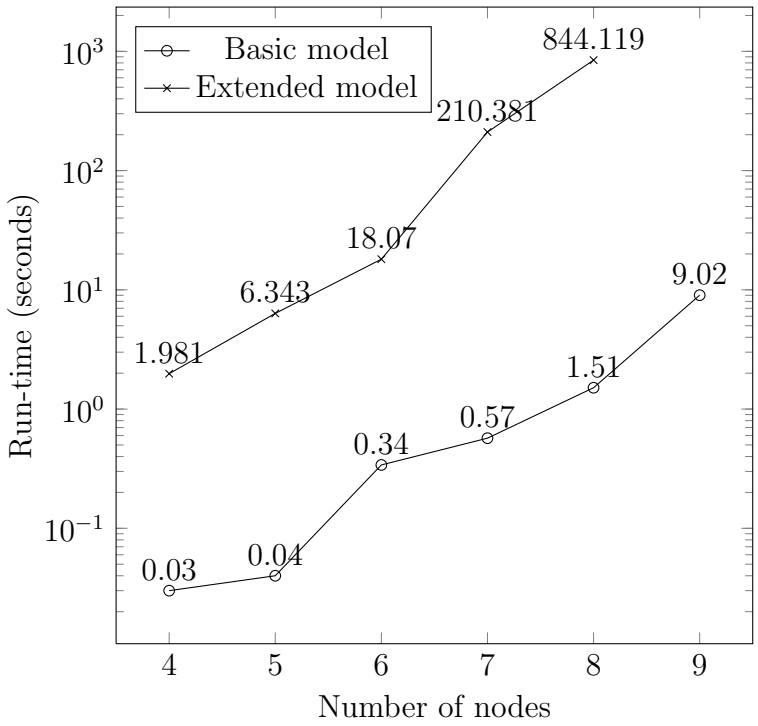

Figure 5.3.: Comparison of basic and extended MIP model. Run-times for instance 040, with 3 teams, 2 trucks, planning horizon 1 day, dependent on the number of projects.

maintain feasibility for the basic model, the new process of loading together with the capacity constraint makes it not easy to construct a solution, which is still feasible.

One approach would be to construct the synchronized tours first without considering capacity, and then place the visits to the loading depots within the tour until the solution gets feasible. Having fixed the sequence of projects which should be visited by the teams and trucks, the placement of visits to the loading depots cannot make the tours infeasible, they can only turn into feasible solutions by respecting the capacity restrictions. It is suspected, that good solutions could be achieved by inserting them in a reasonable manner.

Finally, the solver should add the possibility of distributing the projects over several days within a defined planning horizon in order to achieve an optimal objective value. This is not so difficult in respect to the feasibility of the solution, because the only condition should be, that projects should be scheduled for teams and trucks on the same day. However, this extension increases the search space considerably, which will make it even more difficult to find not only a feasible, but also good solution.

It is quite possible and even to suspect, that a pure greedy construction algorithm, as it was used for the basic problem, will not work anymore for this quite rich problem. A guided search could make sense, and to develop the destroy-and-recreate approach further and to refine it would be a reasonable try.

6. Conclusions and further research

In this thesis a way to approach the VRPMS with min-max objective was presented. While there has been sparse research done about both main features, VRP with min-max, and VRPMS, no work is known to the author, which connects these two type of problems.

It turned out, that the main specific of vehicle routing problems with multiple synchronization constraints is the interdepency between vehicles, which results in difficulties of keeping feasibility in any heuristic or metaheuristic solution method on the one hand, and in evaluating candidate solutions on the other hand.

A multistart constructive greedy heuristic was proposed. A series of computational experiments showed superior performance over the MIP implementation for larger instances from 10 nodes on. For most of the test-instances which could be solved to optimality in a MIP model, the heuristic solver found the optimal value as well in considerable less time, and it was possible to obtain solutions for larger instances as well, however, the quality of the solution could not be evaluated, as there are no competing approaches. An approach for finding good lower bounds was presented, but the implementation was not in the scope of the thesis due to time restrictions. While the search part of the heuristic is very simple and performs well on this problem, is to suspect, that performance will deteriorate with increasing problem complexity.

However, the core innovative feature of the solver is the evaluation of candidate solutions, which utilizes discrete event simulation. This turned out to be beneficial because of its ease of implementation and flexibility.

There is great potential to develop the proposed ideas further in various directions.

- As already mentioned, the original problem was mostly reduced to the aspect of the synchronization of different classes of vehicles. To make it usable for real world problems, the existing model needs to be extended by adding constraints regarding vehicle capacity, precedence, multiple use of vehicles and many more. For this, the greedy algorithm needs to be improved in order to correct earlier misleading insertions.

- At the moment the model plans just for one day. The model could be extended to assign projects to a day within a defined planning horizon.
- The solver can be extended to run multi-threaded in order to utilize multiple cores. This is believed to improve run-time considerably, as there is no interdependence among the threads and no synchronization is needed.
- Construction algorithms can be improved to avoid invalid candidate solution.
- The search can be improved by scanning the solution space in a smarter way.
- Evaluation should be invoked less to improve run-times.
- An interesting direction could also be to add stochasticity to the model. Particularly interesting would be, how stochastic driving times between the nodes would affect solutions, as synchronization among vehicles is required.
- There are only few discrete event simulation libraries available, and none of them proved to be helpful for the presented purpose. The implemented discrete event simulator could be generalized and provided as a library, which would be preferably Open Source Software for use in the academic environment.
- The algorithm performed very well even in the min-max VRP without any synchronization. It would be very interesting to compare the performance of this algorithm to other approaches.
- A combined approach of heuristic and MIP solver as presented in França et al. (1995) would be an interesting possibility to solve this problem to optimality. The heuristic solver can give upper bounds which can be used as constraints to limit the search space for solving a distance constrained VRPMS.

Bibliography

Almeder, Christian, Margaretha Preusser, and Richard F Hartl. 2009. "Simulation and optimization of supply chains: alternative or complementary approaches?" *OR Spectrum* (Heidelberg, Netherlands, Heidelberg) 31 (1): 95–119.

Almoustafa, Samira, Said Hanafi, and Nenad Mladenović. 2013. "New exact method for large asymmetric distance-constrained vehicle routing problem." *European Journal of Operational Research* 226 (3): 386–394.

Altiok, Tayfur, and Benjamin Melamed. 2007. *Simulation modeling and analysis with Arena.* xxi, 440. Amsterdam ; Boston: Academic Press.

Ann, Melissa C., Dieter Vandenbussche, and William Hermann. 2008. "Routing for Relief Efforts." *Transportation Science* 42 (2): 127–145.

Applegate, David, William Cook, Sanjeeb Dash, and Andre Rohe. 2002. "Solution of a min-max vehicle routing problem" [in English]. *INFORMS Journal on Computing* 14 (2): 132–132.

Artigues, Christian, and Dominique Feillet. 2008. "A branch and bound method for the job-shop problem with sequence-dependent setup times." *Annals of Operations Research* 159 (1): 135–159.

Barker, Joel A. 1992. *Future Edge: Discovering the New Paradigms of Success.* 32. New York, N.Y.: William Morrow & Company, Inc.

Barr, Richard S, Bruce L Golden, James P Kelly, Mauricio G C Resende, and William R Stewart. 1995. "Designing and reporting on computational experiments with heuristic methods" [in English]. *Journal of Heuristics* 1 (1): 9–32.

Beck, J. Christopher, Patrick Prosser, and Evgeny Selensky. 2003. "Vehicle Routing and Job Shop Scheduling: What's the difference?" In *Proc. of the 13th International Conference on Automated Planning and Scheduling,* 267–276.

Box, George E.P., and Norman R. Draper. 1987. *Empirical model-building and response surfaces.* xiv, 669. New York: Wiley.

Bredström, David, and Mikael Rönnqvist. 2008. "Combined vehicle routing and scheduling with temporal precedence and synchronization constraints." *European Journal of Operational Research* 191 (1): 19–31.

Cakici, Eray, Mary E. Kurz, and Scott J. Mason. 2010. "Integrated Production and Distribution Scheduling with Multiple Objectives." *IIE Annual Conference. Proceedings:*1–6.

Cassandras, Christos G, and Stéphane Lafortune. 2008. *Introduction to Discrete Event Systems.* 2nd ed. xxiv, 776. Boston, MA: Springer US.

Chen, Zhi-Long. 2010. "Integrated Production and Outbound Distribution Scheduling: Review and Extensions." *Operations Research* 58 (1): 130–148, 252.

Cheung, Waiman, and Hong Zhou. 2001. "Using genetic algorithms and heuristics for job shop scheduling with sequence-dependent setup times." *Annals of Operations Research* 107 (1): 65–81.

Christofides, Nicos. 1976. *Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem.* Report 388. Pittsburgh, Pennsylvania: Graduate School of Industrial Administration, CMU.

Clarke, G, and J W Wright. 1964. "Scheduling of vehicles from a central depot to a number of delivery points." *Operations Research* 12 (12): 568–581.

Coffin, Marie, and Matthew J Saltzman. 2000. "Statistical Analysis of Computational Tests of Algorithms and Heuristics." *INFORMS Journal on Computing* (Linthicum, United States, Linthicum) 12 (1): 24–44.

Cordeau, J. F., M. Gendreau, G. Laporte, J. Y. Potvin, and F. Semet. 2002. "A guide to vehicle routing heuristics." *The Journal of the Operational Research Society* 53 (5): 512–522.

Cordeau, J. F., G. Laporte, and A. Mercier. 2001. "A unified tabu search heuristic for vehicle routing problems with time windows." *The Journal of the Operational Research Society* 52 (8): 928–936.

Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2001. *Introduction to algorithms.* 2nd ed. xxi, 1180. 2nd ed. Cambridge, Mass.: MIT Press.

Coy, Steven P., Bruce L. Golden, George C. Runger, and Edward A. Wasil. 2000. "Using Experimental Design to Find Effective Parameter Settings for Heuristics." 7:77–97.

Crainic, T. G. 2008. "Parallel solution methods for vehicle routing problems." In *The Vehicle routing problem: latest advances and new challenges,* edited by Bruce L. Golden, Edward A. Wasil, and S. Raghavan, IX, 589. New York: Springer.

Crainic, T. G., and M. Toulouse. 2010. "Parallel metaheuristics." In *Handbook of Metaheuristics,* edited by M. Gendreau and J.-Y. Potvin.

Drexl, Michael. 2011. *Synchronization in Vehicle Routing-A Survey of VRPs with Multiple Synchronization Constraints.* 1103. Technical report.

———. 2012a. "Rich vehicle routing in theory and practice." *Logistics Research* (Heidelberg, Netherlands, Heidelberg) 5 (1-2): 47–63.

———. 2012b. "Synchronization in Vehicle Routing-A Survey of VRPs with Multiple Synchronization Constraints." *Transportation Science* 46 (3): 297–316.

Durbin, Martin, and Karla Hoffman. 2008. "The Dance of the Thirty-Ton Trucks: Dispatching and Scheduling in a Dynamic Environment." *Operations Research* 56 (1): 3–19.

Ferguson, Margaret. 1980. *The Aquarian Conspiracy: personal and social transformation in the 1980s.* Los Angeles: J.P. Tarcher.

Fourer, Robert, David M. Gay, and Brian W. Kernighan. 2003. *AMPL: a modeling language for mathematical programming.* xxi, 517. 2nd ed. Pacific Grove, Calif.: Thomson/Brooks/Cole.

França, Paulo M., Michel Gendreau, Gilbert Laporte, and Felipe M. Müller. 1995. "The m-Traveling Salesman Problem with Minmax Objective." *Transportation Science* 29 (3): 267–275.

Gendreau, M., J. Y. Potvin, Bräysy O., G. Hasle, and A. Løkketangen. 2008. "Metaheuristics for the Vehicle Routing Problem and Its Extensions: A Categorized Bibliography." In *The Vehicle routing problem: latest advances and new challenges,* edited by Bruce L. Golden, Edward A. Wasil, and S. Raghavan, IX, 589. New York: Springer.

Gill, Arthur. 1962. *Introduction to the theory of finite-state machines.* xi, 207. New York: McGraw-Hill.

Glover, Fred. 1986. "Future paths for integer programming and links to artificial intelligence." *Computers & Operations Research* 13 (5): 533–549.

Goldberg, Daniel W. 2008. *A Geocoding Best Practices Guide.* University of Southern California - GIS Research Laboratory. Accessed May 2, 2013. `http://www.naaccr.org/LinkClick.aspx?fileticket=ZKekM8k_IQ0%3D&tabid=239&mid=699`.

Greenberg, Harvey J. 1990. "Computational Testing: Why, How and How Much." *INFORMS Journal on Computing* 2 (1): 94–97.

Hall, Nicholas G, and Marc E Posner. 2001. "Generating experimental data for computational testing with machine scheduling applications." *Operations Research* (Linthicum, United States, Linthicum) 49 (6): 854–865.

Hart, William E., Carl Laird, Jean-Paul Watson, and David L. Woodruff. 2012. *Pyomo - Optimization Modeling in Python.* New York: Springer.

Hart, William E., Jean-Paul Watson, and David L. Woodruff. 2011. "Pyomo: modeling and solving mathematical programs in Python." *Mathematical Programming Computation* 3:219–260.

Hepdogan, Seyhun, Reinaldo Moraga, Gail DePuy, and Gary Whitehouse. 2005. "Non-parametric Comparison of Two Dynamic Parameter Setting Methods in a Meta-Heuristic Approach." *Journal of Systemics, Cybernetics and Informatics* 5 (5).

Holland, John H. 1975. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* viii, 183. Ann Arbor, Mich.: University of Michigan Press.

Jin, Jianyong. 2013. "Cooperative Parallel Metaheuristics for Large Scale Vehicle Routing Problems." PhD diss., Molde University College.

Kim, Byung-In, Jeongin Koo, and Junhyuk Park. 2010. "The combined manpower-vehicle routing problem for multi-staged services." *Expert Systems with Applications* 37 (12): 8424–8431.

Kuhn, Thomas S. 1996. *The Structure of Scientific Revolutions.* 3rd ed. Chicago: University of Chicago Press.

Laporte, G., Y. Nobert, and S. Taillefer. 1987. "A branch-and-bound algorithm for the asymmetrical distance-constrained vehicle routing problem." *Mathematical Modelling* 9 (12): 857–868.

Law, Averill M., and W. David Kelton. 2000. *Simulation modeling and analysis.* xxi, 760. 3rd ed. Boston: McGraw-Hill.

Leont'ev, V K. 2007. "Discrete optimization." *Computational Mathematics and Mathematical Physics* (Oxford, Netherlands, Oxford) 47 (2): 328–340.

Liu, Cheng-Hsiang. 2011. "Using genetic algorithms for the coordinated scheduling problem of a batching machine and two-stage transportation." *Applied Mathematics and Computation* 217 (24): 10095–10104.

Miller, C. E., A. W. Tucker, and R. A. Zemlin. 1960. "Integer Programming Formulation of Traveling Salesman Problems." *J. ACM* (New York, NY, USA) 7, no. 4 (October): 326–329.

Object Management Group. 2011. "OMG Unified Modeling Language^{TM}(OMG UML), Superstructure Version 2.4.1." August. Accessed April 16, 2013. `http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/`.

Oppen, Johan, and David L. Woodruff. 2009. "Parametric models of local search progression." *International Transactions in Operational Research* 16:627–640.

Pinedo, Michael. 2012. *Scheduling.* 4th ed. xx, 673. New York: Springer.

Pitsoulis, Leonidas S., and Resende Mauricio G.C. 2001. *Greedy Randomized Adaptive Search Proce.* Technical report. AT&T Labs.

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

Rubasheuskaya, Anastasia. 2012. "Combined scheduling-transportation model. Veidekke Industri AS case study." Master Thesis, Molde University College - Specialized University in Logistics.

Schrimpf, Gerhard, Johannes Schneider, Hermann Stamm-Wilbrandt, and Gunter Dueck. 2000. "Record Breaking Optimization Results Using the Ruin and Recreate Principle." *Journal of Computational Physics* 159 (2): 139–171.

Toth, Paolo, and Daniele Vigo. 2002. *The Vehicle routing problem.* xviii, 367. Philadelphia, Pa.: Society for Industrial / Applied Mathematics.

Van Huyssteen, J. Wentzel Vrede. 2003. *Encyclopedia of science and religion.*

Vela, Camino R., Ramiro Varela, and Miguel A. González. 2010. "Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times." *Journal of Heuristics* 16 (2): 139–165.

Weingartner, E., H. vom Lehn, and K. Wehrle. 2009. "A Performance Comparison of Recent Network Simulators." In *Communications, 2009. ICC '09. IEEE International Conference on,* 1–5. IEEE, June.

Xu, J., and J. Kelly. 1996. "A Network Flow-based Tabu Search Heuristic for the Vehicle Routing Problem." *Transportation Science* 30:379–393.

Appendices

A. Description of test data

The test data was generated by a script. Algorithm 3 shows the pseudocode of the generator.

Input: numNodes = Number of nodes to be created for the instance, x_max = maximum x coordinate, y_max = maximum y coordinate, p_min and p_max = bounds for p (project size), d_min = minimum euclidean distance from any other node
Result: A test instance
for *(x ← 0; x < numNodes; x++)* **do**
  repeat
    nodeValid ← TRUE;
    x ← Uniform random between 0 and $x_{\max}$;
    y ← Uniform random between 0 and $y_{\max}$;
    if $x \leq 1$ **then**
      p ← Uniform random between $p_{\min}$ and $p_{\max}$
    else
      p ← 0
    end
    newNode ← new Node(position at cooridnates x and y with project size p);
    for *each insNode in instance* **do**
      d = Euclidean distance from insNode to newNode;
      if $d < d_{\min}$ **then**
        nodeValid ← FALSE
      end
    end
  until *nodeValid*;
  Add newNode to instance;
end

Algorithm 3: Algorithm for generating test data

The test instances are named to reflect the parameters:

[Instance number]_[numNodes]_[$x_{\max}$]x[$y_{\max}$]_[$p_{\min}$]-[$p_{\max}$]_[$d_{\min}$]

Following parameters were used to create the instances:

Instance number from	to	$x_{\max}$	$y_{\max}$	$p_{\min}$	$p_{\max}$	$d_{\min}$
000	009	1000	500	5	500	40
010	019	1000	500	500	3000	40
020	029	1000	500	5	3000	40
030	039	1000	500	0	0	40
040	Adapted for extended model					
041	Adapted for visualization purposes					

Instance 040 and 041 are special purpose instances and not included in the computational tests. The whole package of test instances can be requested by e-mail: m.bracher@gmail.com

B. Output of computational experiments

B.1. Speed comparison MIP solver / heuristic solver

Run with 10 nodes (8+2), 3 teams and 2 vehicles. In each group g the instance, where the metaheuristic performed worst is marked with W, the one which performed average is marked with M, and finally the best performing instance is marked with B.

- g Group number based on project sizes
- $t^\star$ Time do find optimal value
- $t^H_{\min}$ Minimum time in seconds from 11 runs with different random seeds to obtain optimum value with heuristic solver
- $t^H_{\max}$ Minimum time in seconds from 11 runs with different random seeds to obtain optimum value with heuristic solver
- $\hat{t}^H$ Median time in seconds from 11 runs with different random seeds to obtain optimum value with heuristic solver
- s Measure for speed improvement $t^\star/\hat{t}^H$

Table B.1.: Speed comparison

Instance name	g	$z^\star$	$t^\star$	$t^H_{\min}$	$t^H_{\max}$	$\hat{t}^H$	s
000_100_1000x500_5-500_40	1	1793	65.433	0.04	0.326	0.125	523
001_100_1000x500_5-500_40	1	2558	22.033	0.013	0.092	0.034	648
002_100_1000x500_5-500_40	1	2748	49.034	0.021	0.565	0.215	228
003_100_1000x500_5-500_40	1	1912	70.177	0.008	0.451	0.106	662
004_100_1000x500_5-500_40	1	1861	38.459	0.03	1.435	0.177	217
005_100_1000x500_5-500_40^W	1	2644	27.623	0.062	0.314	0.195	142
006_100_1000x500_5-500_40^B	1	2324	115.765	0.024	0.109	0.047	2463

Table B.1 – *Continued from previous page*

Instance name	g	$z^\star$	$t^\star$	$t^H_{\min}$	$t^H_{\max}$	$\hat{t}$	s
007_100_1000x500_5-500_40	1	1897	73.138	0.034	1.32	0.388	189
008_100_1000x500_5-500_40	1	2148	58.173	0.031	0.097	0.049	1187
009_100_1000x500_5-500_40^M	1	1830	34.741	0.05	1.166	0.119	292
010_100_1000x500_500-3000_40	2	6716	161.92	0.011	0.134	0.04	4048
011_100_1000x500_500-3000_40^W	2	5450	79.085	0.182	2.065	0.635	125
012_100_1000x500_500-3000_40^M	2	5867	148.985	0.095	2.119	0.368	405
013_100_1000x500_500-3000_40	2	6483	119.105	0.014	0.368	0.048	2481
014_100_1000x500_500-3000_40	2	5725	44.883	0.022	0.818	0.155	290
015_100_1000x500_500-3000_40	2	7507	36.581	0.029	0.502	0.118	310
016_100_1000x500_500-3000_40	2	6906	334.582	0.012	0.567	0.143	2340
017_100_1000x500_500-3000_40	2	6949	178.968	0.025	0.316	0.102	1755
018_100_1000x500_500-3000_40^B	2	7088	3411.75	0.034	1.91	0.624	5468
019_100_1000x500_500-3000_40	2	6633	33.655	0.024	0.593	0.209	161
020_100_1000x500_5-3000_40	3	5571	32.285	0.018	0.975	0.219	147
021_100_1000x500_5-3000_40	3	5759	33.089	0.014	0.93	0.084	394
022_100_1000x500_5-3000_40	3	5351	32.982	0.061	0.299	0.14	236
023_100_1000x500_5-3000_40	3	6434	33.932	0.101	0.759	0.178	191
024_100_1000x500_5-3000_40	3	5698	115.226	0.04	0.498	0.183	630
025_100_1000x500_5-3000_40	3	5767	35.083	0.044	3.856	0.399	88
026_100_1000x500_5-3000_40^M	3	5903	42.065	0.022	0.904	0.11	382
027_100_1000x500_5-3000_40^B	3	4701	154.963	0.014	0.866	0.125	1240
028_100_1000x500_5-3000_40^W	3	6312	41.138	0.518	20.269	7.805	5
029_100_1000x500_5-3000_40	3	6198	137.73	0.049	0.601	0.328	420
030_100_1000x500_0-0_40^M	4	1362	143.196	0.01	0.041	0.02	7160
031_100_1000x500_0-0_40	4	1192	236.101	0.01	0.054	0.018	13117
032_100_1000x500_0-0_40	4	1553	326.247	0.019	0.061	0.038	8585
033_100_1000x500_0-0_40	4	1455	122.785	0.017	0.428	0.077	1595
034_100_1000x500_0-0_40^W	4	1349	40.475	0.016	0.086	0.035	1156
035_100_1000x500_0-0_40	4	1822	90.98	0.009	0.032	0.019	4788
036_100_1000x500_0-0_40	4	1893	157.843	0.009	0.053	0.026	6071
037_100_1000x500_0-0_40	4	1684	161.002	0.01	0.035	0.016	10063
038_100_1000x500_0-0_40^B	4	1936	404.262	0.011	0.052	0.027	14973

Table B.1 – *Continued from previous page*

Instance name	g	$z^\star$	$t^\star$	$t^H_{\min}$	$t^H_{\max}$	$\hat{t}$	s
039_100_1000x500_0-0_40	4	1528	218.5	0.011	0.036	0.017	12853

All instances marked with W, M or B were picked in order to conduct further tests by varying teams and trucks, one at each time. In 2 out of 589 cases the instance was not solved to optimality (marked in the table by X):

- 011_100_1000x500_500-3000_40 with 9 nodes, 4 teams and 1 truck, objective value 4377 instead of 4354

- 022_100_1000x500_5-3000_40 with 9 nodes, 4 teams and 1 truck, objective value 3920 instead of 3824

The value in the cells of table B.2 denotes the measure for the speed improvement $s = t^\star/\hat{t}^H$. This is the time for a run with the MIP solver divided by the median time of 11 runs with different random seeds with the heuristic solver.

Table B.2.: Speed comparison dependent on number of teams and trucks

	Teams	Trucks →						
Instance name	↓	1	2	3	4	5	6	7
005_100_1000x500_5-500_40	1	694	2248	695	904	995	700	1275
	2	126	155	116	207	419	1529	120
	3	467	15	201	51	40	29	58
	4	146	54	154	891	449	431	849
	5	360	39	614	817	9128	4490	504
	6	398	107	1259	332	739	499	403
	7	244	132	529	805	641	282	4856
006_100_1000x500_5-500_40	1	1326	555	538	446	398	619	1177
	2	4381	535	509	1977	901	853	737
	3	3195	242	109	200	139	280	277
	4	4489	443	396	968	413	792	558

Table B.2 – *Continued from previous page*

Instance name	Teams ↓	Trucks → 1	2	3	4	5	6	7
	5	8211	493	135	732	248	171	635
	6	2559	3071	337	168	264	257	250
	7	3218	558	546	762	889	872	1234
009_100_1000x500_5-500_40	1	1245	781	1314	864	822	759	925
	2	1111	194	132	134	97	82	153
	3	345	490	109	84	89	92	60
	4	467	12	141	276	159	128	683
	5	1247	7	171	74	116	118	72
	6	764	119	85	801	423	311	505
	7	2062	952	330	253	283	297	358
011_100_1000x500_500-3000_40	1	407	355	297	328	320	471	289
	2	126	396	175	262	601	210	294
	3	51	56	15	22	13	12	13
	4	X	15	38	49	91	51	56
	5	51	43	111	182	233	157	145
	6	754	635	471	431	291	382	459
	7	558	366	414	1094	1199	905	274
012_100_1000x500_500-3000_40	1	619	638	567	358	415	504	372
	2	187	118	140	108	84	74	76
	3	48	55	72	58	41	34	33
	4	106	18	26	25	28	22	17
	5	92	32	17	32	49	69	21
	6	177	428	742	749	590	659	261
	7	203	486	763	303	352	343	400
018_100_1000x500_500-3000_40	1	991	745	769	814	756	534	580
	2	140	28	21	19	23	17	11
	3	58	77	140	70	80	67	53
	4	17	14	46	43	33	31	28
	5	71	34	139	97	98	78	92
	6	608	94	139	76	91	87	84
	7	535	419	696	593	301	367	253

Table B.2 – *Continued from previous page*

Instance name	Teams ↓	Trucks → 1	2	3	4	5	6	7
022_100_1000x500_5-3000_40	1	745	887	874	555	546	556	360
	2	203	70	31	30	32	25	20
	3	110	39	68	47	66	64	33
	4	X	30	23	23	25	24	18
	5	482	318	255	396	294	265	201
	6	517	531	686	365	458	457	542
	7	602	1216	650	501	416	368	520
027_100_1000x500_5-3000_40	1	914	791	1035	845	580	682	811
	2	40	73	154	52	50	46	45
	3	21	73	49	47	49	36	27
	4	161	403	817	483	229	208	270
	5	641	658	627	440	517	486	771
	6	460	603	484	516	479	437	585
	7	511	603	3353	540	383	408	1131
028_100_1000x500_5-3000_40	1	688	507	428	1431	475	410	291
	2	106	77	47	38	128	37	23
	3	26	104	55	49	50	44	31
	4	10	3	6	6	6	5	5
	5	153	248	418	454	303	459	436
	6	192	607	455	473	361	342	342
	7	958	434	354	446	373	341	318
030_100_1000x500_0-0_40	1	4662	2112	1494	13687	12699	2280	3274
	2	2881	5998	1007	2020	765	603	1239
	3	3688	13164	234	340	1051	782	133
	4	11842	5246	426	3030	1021	1160	1966
	5	3265	924	2516	497	3070	2167	1668
	6	12884	3837	327	3768	3206	1218	806
	7	1058	587	1114	3474	1121	997	2534
034_100_1000x500_0-0_40	1	3312	6644	2681	6279	20197	22083	2958
	2	4857	1043	581	367	1040	615	792
	3	1987	1568	7810	5186	4412	1063	8538

Table B.2 – *Continued from previous page*

Instance name	Teams ↓	Trucks → 1	2	3	4	5	6	7
	4	7355	1528	1211	2884	8544	7659	3570
	5	9911	1073	2212	10967	1971	3916	1431
	6	5431	1467	5139	3888	1813	3088	2159
	7	15528	1079	4722	1477	5110	2844	1888
038_100_1000x500_0-0_40	1	1103	5351	3605	2003	757	12227	796
	2	6564	896	3320	501	275	526	533
	3	14338	1899	2286	1283	1021	1160	337
	4	14767	1615	1206	2153	2228	1864	1717
	5	1841	767	1163	870	1878	4159	2931
	6	2370	1498	2266	1668	8638	1797	4093
	7	17774	854	3306	1239	1394	1378	1617

Table B.3 provides data about run-times of both solvers based on the size of the instance.

- n number of nodes in the instance (including team and truck depot)
- $t^\star$ Time do find optimal value
- $t^H_{\min}$ Minimum time in seconds from 11 runs with different random seeds to obtain optimum value with heuristic solver
- $t^H_{\max}$ Minimum time in seconds from 11 runs with different random seeds to obtain optimum value with heuristic solver
- $\hat{t}^H$ Median time in seconds from 11 runs with different random seeds to obtain optimum value with heuristic solver
- s Measure for speed improvement $t^\star/\hat{t}^H$

Table B.3.: Speed comparison dependent on instance size

Instance name	n	$z^\star$	$t^\star$	$t^H_{\min}$	$t^H_{\max}$	$\hat{t}^H$	s
005_100_1000x500_5-500_40	6	2227	0.175	0.004	0.018	0.005	35
005_100_1000x500_5-500_40	7	2259	0.555	0.005	0.007	0.006	92.5
005_100_1000x500_5-500_40	8	2441	1.386	0.016	0.047	0.035	39.6
005_100_1000x500_5-500_40	9	2486	3.913	0.054	0.369	0.238	16.44

Table B.3 – *Continued from previous page*

Instance name	g	$z^\star$	$t^\star$	$t^H_{\min}$	$t^H_{\max}$	$\hat{t}$	s
005_100_1000x500_5-500_40	10	2644	28.818	0.057	0.328	0.183	157.48
005_100_1000x500_5-500_40	11	2669	616.553	0.387	3.349	1.503	410.21
009_100_1000x500_5-500_40	6	1521	0.177	0.008	0.014	0.01	17.7
009_100_1000x500_5-500_40	7	1521	0.385	0.007	0.023	0.012	32.08
009_100_1000x500_5-500_40	8	1599	1.373	0.016	0.061	0.027	50.85
009_100_1000x500_5-500_40	9	1772	21.178	0.009	0.212	0.037	572.38
009_100_1000x500_5-500_40	10	1830	24.526	0.056	1.164	0.122	201.03
009_100_1000x500_5-500_40	11	1968	1882.584	1.228	28.759	12.327	152.72
006_100_1000x500_5-500_40	6	1908	0.209	0.003	0.005	0.004	52.25
006_100_1000x500_5-500_40	7	1990	0.708	0.007	0.04	0.016	44.25
006_100_1000x500_5-500_40	8	1990	1.677	0.011	0.038	0.018	93.17
006_100_1000x500_5-500_40	9	2246	7.877	0.016	0.215	0.043	183.19
006_100_1000x500_5-500_40	10	2324	118.421	0.023	0.094	0.054	2,192.98
011_100_1000x500_500-3000_40	6	3595	0.189	0.004	0.013	0.005	37.8
011_100_1000x500_500-3000_40	7	4094	0.341	0.008	0.031	0.013	26.23
011_100_1000x500_500-3000_40	8	4357	0.818	0.009	0.094	0.029	28.21
011_100_1000x500_500-3000_40	9	4843	5.604	0.04	0.917	0.099	56.61
011_100_1000x500_500-3000_40	10	5450	84.433	0.169	2.099	0.63	134.02
012_100_1000x500_500-3000_40	6	3173	0.161	0.006	0.013	0.008	20.13
012_100_1000x500_500-3000_40	7	3781	0.408	0.008	0.033	0.017	24
012_100_1000x500_500-3000_40	8	4158	1.037	0.018	0.279	0.053	19.57
012_100_1000x500_500-3000_40	9	4963	3.657	0.046	0.249	0.075	48.76
012_100_1000x500_500-3000_40	10	5867	135.093	0.12	2.138	0.398	339.43
018_100_1000x500_500-3000_40	6	4626	0.177	0.004	0.008	0.005	35.4
018_100_1000x500_500-3000_40	7	5387	0.266	0.012	0.025	0.019	14
018_100_1000x500_500-3000_40	8	5599	0.94	0.011	0.077	0.028	33.57
018_100_1000x500_500-3000_40	9	6784	4.487	0.01	0.089	0.045	99.71
018_100_1000x500_500-3000_40	10	7088	3477.953	0.03	1.813	0.622	5,591.56
028_100_1000x500_5-3000_40	6	3924	0.172	0.005	0.007	0.006	28.67
028_100_1000x500_5-3000_40	7	4801	0.249	0.014	0.023	0.016	15.56
028_100_1000x500_5-3000_40	8	4801	0.766	0.012	0.052	0.039	19.64
028_100_1000x500_5-3000_40	9	5294	3.912	0.007	0.092	0.042	93.14

Table B.3 – *Continued from previous page*

Instance name	g	$z^\star$	$t^\star$	$t^H_{\min}$	$t^H_{\max}$	$\hat{t}$	s
028_100_1000x500_5-3000_40	10	6312	39.906	0.533	19.766	7.565	5.28
022_100_1000x500_5-3000_40	6	3471	0.185	0.008	0.016	0.01	18.5
022_100_1000x500_5-3000_40	7	3471	0.359	0.008	0.017	0.013	27.62
022_100_1000x500_5-3000_40	8	4198	0.872	0.018	0.047	0.035	24.91
022_100_1000x500_5-3000_40	9	4590	3.05	0.019	0.307	0.078	39.1
022_100_1000x500_5-3000_40	10	5351	32.347	0.057	0.282	0.125	258.78
027_100_1000x500_5-3000_40	6	4023	0.15	0.004	0.006	0.005	30
027_100_1000x500_5-3000_40	7	4023	0.393	0.004	0.007	0.006	65.5
027_100_1000x500_5-3000_40	8	4023	0.771	0.01	0.022	0.015	51.4
027_100_1000x500_5-3000_40	9	4363	2.859	0.019	0.206	0.049	58.35
027_100_1000x500_5-3000_40	10	4701	151.825	0.015	0.812	0.127	1,195.47
034_100_1000x500_0-0_40	6	950	0.197	0.007	0.011	0.009	21.89
034_100_1000x500_0-0_40	7	1094	0.586	0.01	0.029	0.014	41.86
034_100_1000x500_0-0_40	8	1094	4.114	0.009	0.036	0.021	195.9
034_100_1000x500_0-0_40	9	1305	54.709	0.013	0.055	0.032	1,709.66
034_100_1000x500_0-0_40	10	1349	39.718	0.011	0.1	0.036	1,103.28
030_100_1000x500_0-0_40	6	822	0.228	0.006	0.016	0.008	28.5
030_100_1000x500_0-0_40	7	829	0.629	0.007	0.021	0.014	44.93
030_100_1000x500_0-0_40	8	1007	2.35	0.009	0.02	0.011	213.64
030_100_1000x500_0-0_40	9	1007	98.391	0.01	0.017	0.013	7,568.54
030_100_1000x500_0-0_40	10	1362	143.609	0.014	0.05	0.024	5,983.71
038_100_1000x500_0-0_40	6	1807	0.185	0.008	0.018	0.011	16.82
038_100_1000x500_0-0_40	7	1862	0.611	0.007	0.017	0.009	67.89
038_100_1000x500_0-0_40	8	1862	2.621	0.008	0.033	0.018	145.61
038_100_1000x500_0-0_40	9	1883	33.577	0.013	0.041	0.022	1,526.23
038_100_1000x500_0-0_40	10	1936	401.612	0.013	0.065	0.033	12,170.06

B.2. Best known values and quality-effort relationship

Table B.4 shows the output of solution runs for instances with 20 nodes (2 depots + 18 customers), 3 teams and 2 trucks.

z^H Best known value from heuristic solver

t^H Time in seconds to obtain best known value from heuristic solver

$t_{0.05}^{\min}$ Minimum time in seconds from 11 runs with different random seeds to obtain solution within 5 % of best known value

$t_{0.05}^{\max}$ Maximum time in seconds from 11 runs with different random seeds to obtain solution within 5 % of best known value

$\hat{t}_{0.05}$ Median time in seconds from 11 runs with different random seeds to obtain solution within 5 % of best known value

$r_{0.05}$ Measure for quality-effort relationship $\hat{t}_{0.05}/\bar{t}$ (see Barr et al. 1995). A low value indicates fast convergence towards the best known value.

Table B.4.: Best known value and quality-effort relationship

Instance name	z^H	t^H	$t_{0.05}^{\min}$	$t_{0.05}^{\max}$	$\hat{t}_{0.05}$	$r_{0.05}$
000_100_1000x500_5-500_40	2860	77.619	3.562	52.606	24.026	0.30954
001_100_1000x500_5-500_40	3390	110.282	0.202	8.419	2.269	0.02057
002_100_1000x500_5-500_40	3558	172.847	0.335	19.815	6.056	0.03504
003_100_1000x500_5-500_40	3492	326.248	0.129	4.047	1.66	0.00509
004_100_1000x500_5-500_40	3236	163.062	0.313	4.723	2.775	0.01702
005_100_1000x500_5-500_40	3751	168.476	0.105	6.635	1.268	0.00753
006_100_1000x500_5-500_40	3331	215.97	0.144	7.471	0.974	0.00451
007_100_1000x500_5-500_40	2893	332.829	0.493	10.96	2.01	0.00604
008_100_1000x500_5-500_40	3010	7.799	0.155	3.557	1.449	0.18579
009_100_1000x500_5-500_40	3482	33.816	0.127	4.089	1.029	0.03043
010_100_1000x500_500-3000_40	12183	63.717	0.052	1.55	0.36	0.00565
011_100_1000x500_500-3000_40	11416	120.37	0.045	0.824	0.18	0.0015
012_100_1000x500_500-3000_40	12149	215.541	0.154	2.238	0.428	0.00199
013_100_1000x500_500-3000_40	12391	112.175	0.042	1.237	0.219	0.00195
014_100_1000x500_500-3000_40	12790	72.432	0.074	0.804	0.215	0.00297
015_100_1000x500_500-3000_40	15609	222.455	0.03	0.792	0.112	0.0005
016_100_1000x500_500-3000_40	11465	152.736	0.044	0.514	0.148	0.00097
017_100_1000x500_500-3000_40	14556	242.638	0.027	0.348	0.13	0.00054
018_100_1000x500_500-3000_40	12261	175.84	0.039	1.593	0.291	0.00165
019_100_1000x500_500-3000_40	11897	110.011	0.052	0.724	0.28	0.00255

Table B.4 – *Continued from previous page*

Instance name	z^H	t^H	$t^{\min}_{0.05}$	$t^{\max}_{0.05}$	$\hat{t}_{0.05}$	$r_{0.05}$
020_100_1000x500_5-3000_40	11376	165.848	0.101	1.143	0.197	0.00119
021_100_1000x500_5-3000_40	10775	213.565	0.037	0.426	0.179	0.00084
022_100_1000x500_5-3000_40	11457	2.391	0.033	0.894	0.254	0.10623
023_100_1000x500_5-3000_40	11825	170.419	0.073	1.744	0.403	0.00236
024_100_1000x500_5-3000_40	10112	237.095	0.047	1.513	0.503	0.00212
025_100_1000x500_5-3000_40	10382	262.174	0.105	1.215	0.386	0.00147
026_100_1000x500_5-3000_40	12237	257.404	0.079	2.324	0.67	0.0026
027_100_1000x500_5-3000_40	10787	231.015	0.037	2.485	0.264	0.00114
028_100_1000x500_5-3000_40	12208	62.324	0.034	0.808	0.337	0.00541
029_100_1000x500_5-3000_40	12847	202.697	0.065	0.984	0.222	0.0011
030_100_1000x500_0-0_40	1901	37.908	0.043	0.631	0.144	0.0038
031_100_1000x500_0-0_40	1551	11.198	0.042	0.866	0.191	0.01706
032_100_1000x500_0-0_40	1823	2.592	0.049	0.786	0.323	0.12461
033_100_1000x500_0-0_40	2143	55.267	0.067	0.356	0.192	0.00347
034_100_1000x500_0-0_40	1747	4.708	0.049	1.464	0.39	0.08284
035_100_1000x500_0-0_40	2158	0.242	0.05	0.776	0.266	1.09917
036_100_1000x500_0-0_40	2176	1.595	0.045	0.619	0.079	0.04953
037_100_1000x500_0-0_40	1814	2.284	0.057	0.575	0.32	0.14011
038_100_1000x500_0-0_40	2130	5.242	0.094	0.967	0.238	0.0454
039_100_1000x500_0-0_40	1825	0.229	0.076	0.952	0.246	1.07424

B.3. Lower Bounds

Table B.5 shows the calculated lower bounds for the 40 main test instances for 1 nodes (team depot + truck depot + 8 customers). The headings in this table are defined as follows:

t Solution time for synchronized problem in seconds

t_T Solution time for teams without waiting for trucks in seconds

t_A Solution time for trucks without waiting for teams in seconds

$z^\star$ Optimal solution value for synchronized problem

$z_T^\star$ Optimal solution value for teams without waiting for trucks

$z_A^\star$ Optimal solution value for trucks without waiting for teams

$\underline{z}$ Lower bound, $\max(z_T^\star, z_A^\star)$

$\underline{z}/z^\star$ Measure for gap to optimality

Table B.5.: Lower bounds and optimality gap

Instance name	t	t_T	t_A	$z^\star$	$z_T^\star$	$z_A^\star$	$\underline{z}$	$\underline{z}/z^\star$
000_100_1000x500_5-500_40	65.433	2.465	5.399	1793	1701	1371	1701	0.95
001_100_1000x500_5-500_40	22.033	2.555	7.274	2558	2557	1873	2557	1
002_100_1000x500_5-500_40	49.034	2.041	3.866	2748	2716	1276	2716	0.99
003_100_1000x500_5-500_40	70.177	2.891	12.492	1912	1750	1531	1750	0.92
004_100_1000x500_5-500_40	38.459	2.465	6.9	1861	1715	1419	1715	0.92
005_100_1000x500_5-500_40	27.623	3.093	4.048	2644	2488	1214	2488	0.94
006_100_1000x500_5-500_40	115.765	2.272	5.415	2324	2038	1872	2038	0.88
007_100_1000x500_5-500_40	73.138	1.739	5.832	1897	1897	1162	1897	1
008_100_1000x500_5-500_40	58.173	3.253	6.512	2148	1873	1890	1890	0.88
009_100_1000x500_5-500_40	34.741	3.038	6.251	1830	1680	1225	1680	0.92
010_100_1000x500_500-3000_40	161.92	3.177	6.781	6716	6602	1335	6602	0.98
011_100_1000x500_500-3000_40	79.085	2.651	6.087	5450	5241	1594	5241	0.96
012_100_1000x500_500-3000_40	148.985	3.235	5.943	5867	5867	1386	5867	1
013_100_1000x500_500-3000_40	119.105	2.978	10.838	6483	6483	1469	6483	1
014_100_1000x500_500-3000_40	44.883	3.611	13.323	5725	5483	1871	5483	0.96
015_100_1000x500_500-3000_40	36.581	3.502	11.772	7507	7507	2300	7507	1
016_100_1000x500_500-3000_40	334.582	3.121	11.704	6906	6906	1350	6906	1
017_100_1000x500_500-3000_40	178.968	3.629	3.319	6949	6949	1302	6949	1
018_100_1000x500_500-3000_40	3411.75	3.325	4.487	7088	6975	1217	6975	0.98
019_100_1000x500_500-3000_40	33.655	3.701	4.853	6633	6633	1046	6633	1
020_100_1000x500_5-3000_40	32.285	2.926	5.715	5571	5411	1294	5411	0.97
021_100_1000x500_5-3000_40	33.089	2.777	7.235	5759	5745	1687	5745	1

Table B.5 – *Continued from previous page*

Instance name	t	t_T	t_A	$z^\star$	$z_T^\star$	$z_A^\star$	$\underline{z}$	$\underline{z}/z^\star$
022_100_1000x500_5-3000_40	32.982	2.798	4.899	5351	5245	1328	5245	0.98
023_100_1000x500_5-3000_40	33.932	3.153	4.707	6434	6434	1368	6434	1
024_100_1000x500_5-3000_40	115.226	3.265	10.624	5698	5698	1254	5698	1
025_100_1000x500_5-3000_40	35.083	3.458	8.1	5767	5761	1412	5761	1
026_100_1000x500_5-3000_40	42.065	2.668	7.747	5903	5903	2005	5903	1
027_100_1000x500_5-3000_40	154.963	2.925	4.864	4701	4701	1017	4701	1
028_100_1000x500_5-3000_40	41.138	2.85	6.456	6312	6312	1564	6312	1
029_100_1000x500_5-3000_40	137.73	3.086	4.115	6198	6198	1395	6198	1
030_100_1000x500_0-0_40	143.196	2.279	4.933	1362	1092	1362	1362	1
031_100_1000x500_0-0_40	236.101	2.255	6.052	1192	1132	1190	1190	1
032_100_1000x500_0-0_40	326.247	2.502	8.078	1553	1214	1502	1502	0.97
033_100_1000x500_0-0_40	122.785	2.148	6.373	1455	1264	888	1264	0.87
034_100_1000x500_0-0_40	40.475	2.777	6.103	1349	1001	1305	1305	0.97
035_100_1000x500_0-0_40	90.98	2.321	5.675	1822	1366	1814	1814	1
036_100_1000x500_0-0_40	157.843	2.556	8.367	1893	957	1880	1880	0.99
037_100_1000x500_0-0_40	161.002	2.136	9.66	1684	1391	1452	1452	0.86
038_100_1000x500_0-0_40	404.262	2.234	5.588	1936	1738	1889	1889	0.98
039_100_1000x500_0-0_40	218.5	2.476	4.859	1528	1290	1332	1332	0.87