Erasmus University Rotterdam [1]

Erasmus School of Economics

Master Thesis Operations Research and Quantitative Logistics

A hybrid quantum-classical approach for multimodal container planning

Author:
S.L. Huizinga (Simone) (Student ID number: 446850)

Supervisor Erasmus University Rotterdam:
dr. R. Spliet (Remy)

Second assessor Erasmus University Rotterdam:
prof.dr.ir. R. Dekker (Rommert)

Supervisors TNO:
dr. F. Phillipson (Frank)
I. Chiscop (Irina)

Thesis Coordinator:
dr. W. van den Heuvel (Wilco)

Abstract
In the years 2011-2020 the commercial quantum annealer of D-Wave Systems grew from 128 to over 5000 qubits. This rapid growth creates new opportunities within the quantum computing area. In this thesis, the quantum annealer is used within a genetic algorithm to solve the multimodal container planning problem. The multimodal container planning problem determines which route every container takes to arrive at its destination before its due date. For every container there is an option to be transported from its origin to its destination by truck. The quantum annealer can only solve Quadratic Unconstrained Binary Optimization problems. Such a formulation is known for the multimodal container planning problem. This formulation only considers two transportation options per container. The first option is the transportation by truck. The second option is a single given multimodal route. The quantum annealer determines for every container whether the first or the second option is used. Which multimodal route is used as option still needs to be determined. This thesis focuses on determining the given multimodal route option for every container. A genetic algorithm is used as the solution approach to determine the given multimodal route for every container.

April 28, 2021

[1]The content of this thesis is the sole responsibility of the author and does not reflect the view of the supervisor, second assessor, Erasmus School of Economics or Erasmus University.

Contents

1 Introduction

Nowadays there is a lot of import and export all over the world. Containers are more often used to transport these packages from their origin to their destination. Reasons for this containerization are the safety of the cargo, standardization and accessibility to multiple modes of transportation, as stated in Crainic and Kim (2007) and SteadieSeifi et al. (2014). To accommodate the increasing freight demand, it is important to transport the containers from their origin to their destination in a very efficient way.

There are different definitions within container transportation. Relevant examples are unimodal transportation, multimodal transportation, intermodal and synchromodal transportation and the Physical Internet (Mankabady (1983), Ballot et al. (2014) and SteadieSeifi et al. (2014)). Different transportation modes are available for container shipments, examples are trucks and barges. When only a single transportation mode is being used it is called unimodal transport. The transport is usually done by truck if unimodal transportation is used to transport goods on land, because it is the most efficient way to move products from door to door. Multimodal transport means that multiple transportation modes are used to transport the cargo. An example is when goods are transported by barge to an intermediate port, where they are transferred to a truck in which they will travel to their final destination. Intermodal transportation is a type of multimodal transportation where the freight is moved within one containment unit, for example a container. This means that the goods themselves are not being handled, which leads to more security and less damage costs. Then, there is the term synchromodal transport. This is a form of intermodal transport where the stakeholders of the transport chain interact within a cooperative network and real-time data can be used to switch between available resources, as stated in Pfoser et al. (2016). A concept where this is emphasized even more is the Physical Internet (Ballot et al. (2014)). The idea is to create a collaborative and robust network, such that all logistic networks are connected and synchronized. This network should be capable of continually optimizing the shipment of goods of many types and sizes by means of optimizing both the operators and the customers economic models. The advantage of the Physical Internet is that the cost efficiency is increased, while the environmental impact is decreased.

There are multiple decisions that need to be made when scheduling containers in an intermodal environment. These decisions are divided into three planning levels according to Crainic and Laporte (1997), the strategic, tactical and operational planning levels. Strategic planning problems consider long term decisions and require large investments, such as the decision where to build a new terminal. The tactical planning level considers the medium horizon planning, which concerns the itineraries and frequencies of barges for example. Operational planning problems are the short term decisions. It concerns the day-to-day choices, such as determining which truck takes which container at what location. An operational planning decision often has to be made a lot of times within a small time period. That is why it is important that these decisions can be made quickly. Operational problems can be categorized into three problems as stated in Zweers et al. (2019). The first class of problems consists of the problems in which containers are assigned to existing barge services. To the second category belong the problems where the routes of barges are determined for a given demand. The last category contains the problems where both the assignment of containers to barges and the route of barges are decided upon.

In the synchromodal environment and for the operational planning decisions, there is a lot of uncertainty. It is often required that a planning problem is solved within a short computation time. This way, real-time changes can be added to the model. The container assignment problem is such an operational problem and, as proven by Choi et al. (2012), it is a NP-hard problem. For these reasons, a lot of algorithms have been created to solve the container assignment problems within a reasonable time, Ziliaskopoulos and Wardell (2000), Erera et al. (2005) and Bell et al. (2011) contain examples. Next to algorithms, also a technique has been developed to reduce the number of variables in the problem (Kalicharan et al. (2019)). Not only algorithms and reduction techniques can help speed up the computation of these kind of problems, also hardware could help to solve a problem faster. Quantum computing is an example of a potential hardware improvement.

Quantum computing can be used to decrease the complexity of a certain problem. A popular example is the integer factorization problem. The most efficient known classical algorithm to solve this problem, runs in sub-exponential time. For the quantum computer the polynomial-time algorithm of Shor (1999) was shown to solve the integer factorization problem faster than classical computers. This algorithm was implemented on a quantum computer by Vandersypen et al. (2001). Here, the number 25 was factored. The paper of Gyongyosi and Imre (2019) declares that quantum computing techniques create new possibilities in computing, networking and communications. However, the quantum computers are currently in their early stages of development. They can only manage small instances at this time, especially compared to classical computers with their advanced solvers and algorithms. This is why solving large instances on the current quantum computers is a very challenging task (Ajagekar et al. (2020), Phillipson and Chiscop (2021)). For this reason, the developed quantum algorithms so far are usually not made for the short term solutions, but hold great promise for the future.

Another way to solve large scale problems using quantum algorithms is by creating a hybrid quantum-classical algorithm. In hybrid algorithms, the quantum computer is used to solve a part of the problem and a classical algorithm is used to solve another part. Hybrid algorithms can be used to enlarge the class of problems solvable with the quantum computer, as stated in the paper of Ding et al. (2020).

In quantum computing, qubits are used instead of classical bits. A qubit can be in state zero, state one or a linear combination of both states. When it is a linear combination of both states it is said that the qubit is in both states at the same time. This is called the superposition of the qubit. However, if you check which value the qubit has, it will result in either zero or one. So, during the computation every qubit can be in its superposition, but if the qubit is measured, it will be in either the zero or one state with some probability. Doing this multiple times will give an indication of the true value.

An example of a quantum algorithm is quantum annealing. For this algorithm a hardware implementation has been made by D-Wave Systems. Some optimization problems can be modeled as energy minimization problems. Quantum physics can be used to solve such optimization problems. A fundamental part of physics is that everything is always trying to find its minimal energy state, like objects that slide downhill or hot materials that cool down. This principle is leveraged to solve energy minimization problems in quantum annealing.

Quantum annealing can be explained using simulated annealing. In each iteration of simulated annealing a solution is considered. Then solutions are considered that are "close" to the solution in the current iteration. The solutions that are close to the current solution can be obtained by slightly altering the current solution, for example by changing the route of one container. So, the values of the variables are used to determine which solutions are close, not the solution value. This new solution is called the neighbour of the solution you started out with. If the neighbour under consideration has a better or the same objective value it is selected as the starting solution for the next iteration. If the neighbour has a worse objective value it could still be selected for the next iteration with a certain probability.

Simulated annealing bases the probability of selecting a worse solution on two things. The first influence factor is the "temperature". If the temperature is high, the probability of selecting a worse solution is also, relatively, high. The second component is the solution value of the worse solution under consideration. If there is a large difference between the current solution and the neighbour with a worse solution value, the probability of selecting it is lower than if there is only a small difference.

The temperature decreases as the algorithm continues, such that the probability of accepting a worse solution also decreases. By selecting a solution with a worse objective value the algorithm tries to escape a local optimum. If the problem is formulated as an energy minimization problem, a bad solution could be seen as a barrier where the simulated annealer tries to "jump" over. Such a jump can be seen as a thermal jump.

Quantum annealing does not accept worse solutions, but looks beyond these worse solutions. How far the quantum annealer looks is based on the tunneling strength, the counterpart of the temperature in simulated annealing. This is shown in Figure 1. Just as simulated annealing, quantum annealing does not guarantee that the optimal solution has been found. Quantum annealing has an advantage over simulated annealing when the energy barriers are tall, but thin (Heim et al. (2015)). A more detailed explanation of quantum annealing is given in Appendix A.

D-Wave Systems created the first commercial quantum annealer. They started with 128 qubits and continue to expand the number of qubits. Very recently, D-Wave released another system with over 5000 qubits[3]. With more qubits and more connections between these qubits, the quantum annealer will be able to handle larger problems. The quantum annealer has already been used to solve multiple combinatorial problems (Hauke et al. (2020)).

The success of using D-Wave Systems to solve complex problems is expressed by Ghosh and Mukherjee (2013). They review the possibility to use the quantum annealer to solve NP-hard problems. Furthermore, multiple references are given to papers that successfully use the quantum annealer in multiple applications. The statement is made that these applications will keep on growing. Neukart et al. (2017) state that it is likely to assume that the expansion of the hardware of the quantum annealer will continue. This makes it promising for the future to already formulate problems for this annealer, such that large problems can

[2]Source: `https://arxiv.org/pdf/0801.2193.pdf`

[3]`https://www.dwavesys.com/press-releases/d-wave-announces-general-availability-first-quantum-computer-built-business?`

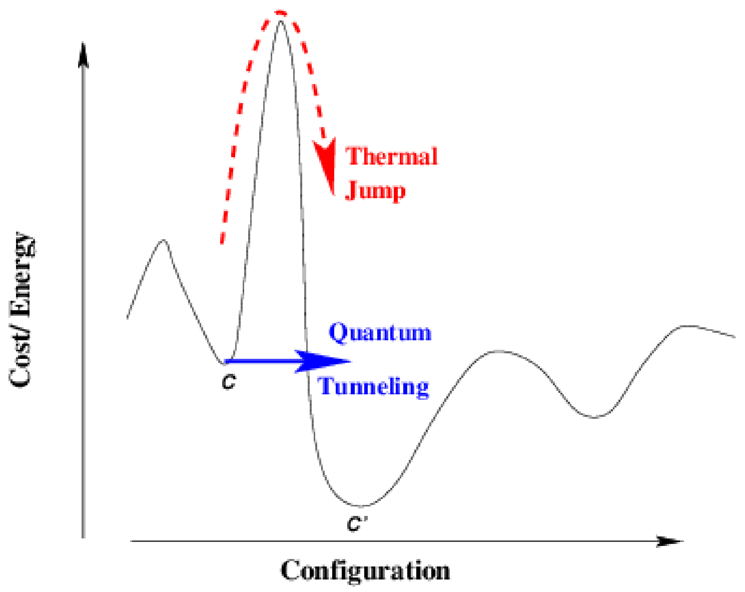


Figure 1: Energy landscape[2]

be solved once the expanded hardware is available. It is especially promising to create algorithms that can solve large problems, while only implementing a relatively small problem onto the quantum annealer. This way large problems can be solved sooner.

To implement combinatorial problems on the quantum annealer, the problem has to be in the Quadratic Unconstrained Binary Optimization (QUBO) formulation. The reason for this is related to the hardware architecture, and is explained in detail in Appendix A. The formulation has a quadratic function as the objective value, binary decision variables and no constraints. With x as binary decision variables, the formulation can be written as the optimization problem described in Equation 1.

$$\min / \max \; x^T Q x \tag{1}$$

The QUBO formulation is already known for a lot of problems (Kochenberger et al. (2014), Glover et al. (2019)). Most combinatorial problems have constraints. To reformulate a problem with constraints to a QUBO problem, the constraints can be moved to the objective function by means of a penalty term.

Currently, the quantum annealer can still only solve small problems with not too many variables. For classical computers it also applies that if there are too many variables, it becomes computationally intractable to solve the problem. An exact approach to solve this issue is column generation (Lübbecke and Desrosiers (2005)). In column generation not all the variables are included in the problem. In each iteration, new variables are generated. These variables are based on the solution to the problem with the variables found so far. Column generation can be applied to linear programming problems for which the found solution to the problem, with just some of the variables, is guaranteed to be optimal. This means that the algorithm is not suitable in combination with the quantum annealer. However, the idea of solving the problem using only a subset of the variables and use the information about that solution to construct new variables can be used. The problem that then arises is how to find variables that can be added to the problem, such that the optimal solution might be found.

The aim of this thesis is to efficiently solve the container assignment problem, while using a quantum annealer. The quantum annealer can be used to solve the container assignment problem when not all variables are included. As a consequence, it needs to be determined which variables are included in this problem, such that the optimal solution can be found. So, this paper attempts to find the optimal variables in the problem that is solved with the quantum annealer. To determine these variables a genetic algorithm is proposed.

A genetic algorithm is a metaheurstic inspired by the mechanics of natural evolution and the survival of the fittest strategy found in biological organisms (Whitley (1994)). In a genetic algorithm, you keep track of a set of solutions, called a population. In each iteration some of these solutions are selected to be modified, such that they hopefully provide a better solution or escape a local optimum. The population in each iteration is called a generation. The selection is based on the quality of the solution, which is determined using a fitness function. A fitness function usually takes the objective value and the violation of constraints into account. The selected solutions are combined to create new solutions. This is called a cross-over operation. To find a more diverse solution, a mutation is applied to some of the solutions. A mutation means that some of the newly found solutions are randomly adjusted. The new solutions are added to the population. Then the whole process is repeated until a stopping criterion is met. A more elaborate explanation of a genetic algorithm can be found in the paper of Whitley (1994) or the paper of Sastry et al. (2005).

The choice of using a genetic algorithm to determine the variables that are used in the problem that is solved with the quantum annealer is based on the evolution towards better solutions. The quantum annealer can be used to determine the quality of the variables that were given. This information can be used within a genetic algorithm to construct new variables, such that the quality of the variables gradually improves. Furthermore, within a genetic algorithm it is possible to keep track of different combinations of variables, such that multiple possible solutions are monitored.

In this thesis, a hybrid quantum-classical algorithm is created to solve the container assignment problem. The quantum annealer solves a part of this problem, where not all the variables are included. The proposed algorithm determines which variables are included. A genetic algorithm is used as the solution approach for this problem.

First, the problem is described in Section 2. Then, the literature is reviewed in Section 3. In Section 4, the methodology is described. Next, the used data is explained in Section 5. Afterwards, an evaluation of the different methods described in the methodology is shown in Section 6. How the value of the parameters is determined, is also explained in this section. Then, the results are shown in Section 7. To determine the quality of the quantum annealer and to evaluate the performance of the genetic algorithm without the quantum annealer, two alternatives have been used instead of the quantum annealer. The first alternative is the use of the simulated annealer also provided by D-Wave Systems instead of the quantum annealer. The second alternative is the use of the optimal solution to the problem with only a few variables instead of the quantum annealer. The results of these alternatives are also given in Section 7. After the results are shown, a discussion is given in Section 8. Finally, a conclusion is stated in Section 9.

2 Problem description

The problem that will be the focus of this thesis is the container assignment problem. For each container, it needs to be determined which route it will take to let it arrive at its destination on time. The container assignment problem has some similarities to the multicommodity flow problem. In the most basic version of the multicommodity flow problem there are multiple commodities that need to travel from their origin to their destination through a directed network. The flow on the arcs needs to be determined, such that all the demand is satisfied. Often, the flow can be split among multiple paths, but this is not necessarily the case. For the container assignment problem, a commodity can be seen as a single container that needs to travel through a network from its origin to its destination.

There are also some differences to the multicommodity flow problem. One difference is that, for the container assignment problem, there are different transportation modes available. This means that multiple arcs might be connecting the same two nodes. Another difference is that it is important that time is incorporated in the model. Containers often have to be at their destination before a certain due date. Also, they have the possibility to be stored at one of the nodes for a period of time. In the latter case there are usually extra costs incurred.

Both of these differences can have an impact on the graph on which the problem is defined. By having multiple transportation modes there are multiple arcs connecting the same nodes. There are different ways to incorporate time. For example, the graph could be a time-space network. In a time-space network, the nodes are cloned such that the same node exists in every time period. The arcs can show the travel times between the nodes. A small example of a time-space network is shown in Figure 2. Another way to include time, is to keep track of the arrival (departure) times of a container at (from) a node. So, there are extra variables included. The parameters, the capacity for example, cannot differ per time period in this case.

The problem in this thesis is defined on a directed (time-space) graph $G = (N, A)$ with $N = \{1, ..., n\}$ the set of nodes and A the set of arcs. Let $K = \{1, ..., n_{cont}\}$ be the set representing the containers. Each container $k \in K$ has a release and due date, rel_k and due_k respectively. The set N consists of origins,

[4]Source: `https://www.researchgate.net/publication/305751679_Stochastic_scheduled_service_network_design_in_the_presence_of_a_spot_market_for_excess_capacity`

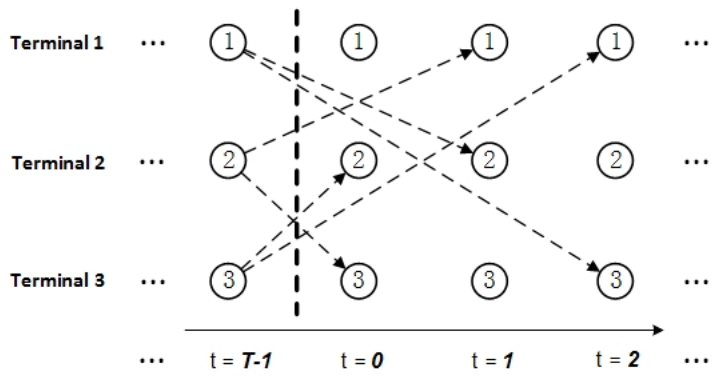


Figure 2: Time-space network[4]

destinations and intermediate nodes. Let O_k and D_k correspond to the origin and destination of container $k \in K$. A route consists of a selection of arcs. The arcs represent existing shipping services. The shipping services might consist of multiple transportation modes, making it a multimodal transportation problem. Each arc connects two nodes, $i \in N$ and $j \in N$, and has a certain capacity v_{ij}, travel time t_{ij} and costs c_{ij}. Note that the arc parameters do not have to be defined per time unit, because the problem is defined on a time-space network. So, each arc already corresponds to a certain time period. Assume that the travel times are deterministic. Let the binary decision variable x_{kij} contain the value one if container $k \in K$ uses arc $(i, j) \in A$.

The objective is to minimize the costs, such that the constraints are not violated. The first constraints are to ensure that each container leaves its origin and arrives at its destination, while only passing intermediate nodes. The second constraints are to ensure that each container arrives at its destination before its due date. The last constraints are to make sure that the capacities of the arcs are not exceeded. The decisions that need to be made to achieve this to define the routes that the containers will take.

3 Literature review

First, the literature about the container assignment problem is reviewed. Here, there is a difference between papers that explain and review transportation concepts and papers that have solved the container assignment problem. Secondly, problems that have been solved by a quantum annealer are discussed. Then, some multicommodity flow problems are reviewed. Within this subsection, the standard multicommodity flow problem is briefly reviewed, followed by a review of the multicommodity multimodal transportation problem. Finally, some genetic algorithm applications are described, with a part dedicated to the combination of genetic algorithms and quantum computing.

3.1 Container transportation

Container transportation has been studied extensively in the literature. Literature reviews are given by the papers of Caris et al. (2008), SteadieSeifi et al. (2014) and van Riessen et al. (2015). The paper of Caris et al. (2008) provides a literature review on intermodal transportation problems. In the article of SteadieSeifi et al. (2014), a literature overview is given based on the strategical, tactical and operational planning levels. The paper of van Riessen et al. (2015) contains an overview of recent developments on the topic of operational implementation of synchromodal transportation for the case of the hinterland network of European Gateway Services, which is a subsidiary of the Rotterdam container terminal operator ECT.

Transportation concepts have been elaborated a few times in the literature. The article of Crainic and Kim (2007) explains intermodal transportation in detail and also contains some models for intermodal terminal decisions, container fleet management decisions and other relevant intermodal decisions. In the paper of Behdani et al. (2014) a detailed description of the synchromodal transportation concept is given. They also create a mathematical model for the operational resource scheduling problem. The tactical service design is assumed to be given, which means that the routes are given. So, it only needs to be determined which container is being shipped and when the shipment takes place. In the article of De Juncker et al. (2017) different kinds of synchromodal transportation problems are described. The paper of Ortega del Vecchyo et al. (2018) gives linear formulations to incorporate different attributes into a synchromodal transportation network. Examples of such attributes are robustness and customer flexibility.

The problem where the route has to be determined for containers in a multimodal network has been solved quite often. In the papers of Ziliaskopoulos and Wardell (2000), Bock (2010) and Bell et al. (2011) delays of shipments are incorporated in the proposed models, sometimes by making travel times stochastic. The paper of Ziliaskopoulos and Wardell (2000) creates an algorithm that computes intermodal least time paths, while incorporating time-dependent arc travel times and switching delays. Bock (2010) creates and solves a model where multimodal transport chains and multiple transshipments are integrated to enable continuous plan adaptations. In the article of Bell et al. (2011) containers are assigned to ships with a given frequency. The departure times of the ships are independent random variables. If the capacity of a ship is exceeded, the dual variable of that constraint is used to find an alternative route for one of those containers.

Other papers focus just on finding the optimal routes for the containers, while the travel times are assumed to be known. It is still required that containers arrive before their due dates. This is used in the papers of Erera et al. (2005), Rivera and Mes (2016), Altendorfer and Minner (2017) and Zweers et al. (2019). In the article of Erera et al. (2005) a model is created where containers are assigned to existing shipping services. The model is a large-scale multicommodity flow problem on a time-space network. They integrate loaded and empty container routing. The problem is solved with integer programming software. The paper of Rivera and Mes (2016) assumes known transportation characteristics, like schedules and capacity, to determine where freight goes each day. Every day, it is decided for each container if it stays at the same place, goes to its destination or goes to an intermediate terminal. Altendorfer and Minner (2017) consider only two options for container transportation, by truck or by using intermodal transportation. Trucks are often more expensive, but intermodal transport might be subject to delays. They found that delay and transportation costs have to be balanced, such that it is not always optimal to travel by intermodal means. The paper of Zweers et al. (2019) assigns containers to barges and determines which terminals each barge visits. They have created an integer linear program to model the problem and propose a two-stage heuristic to solve it. They use two types of binary variables, the first shows if a barge visits a certain terminal and the second represents if a container is transferred by a certain barge. In the first stage, the problem is solved with relaxed container assignment constraints, meaning that the routes for the barges are being determined. Secondly, the routes are fixed and the problem is solved again only now with the binary constraints reinstated.

3.2 Quantum annealer

Only Quadratic Unconstrained Binary Optimization problem formulations can be used as input for the quantum annealer. Such a formulation is of the form $\min / \max\ x^tQx$ (Glover et al. (2019)) and belongs to the class of NP-hard problems. The x's are binary decision variables and Q is a nxn coefficient matrix. There are no constraints in a QUBO problem. However, if there are constraints to consider, then they can be moved to the objective function by means of a penalty. This way the objective value will deteriorate when the constraints are violated. An equivalent formulation to the QUBO problem that is often used is $\min / \max\ H = x^tq + x^tQx$ or a combination of these terms as in $\min / \max\ W_A \times H_A + W_B \times H_B + ...$, where $W_A, W_B, ...$ are the tuning weights. These weights represent the relative weight of the objective function and the possible constraints.

The paper that solves the container assignment problem on a quantum annealer is the paper of Phillipson and Chiscop (2021). They construct a QUBO formulation for the container assignment problem. The QUBO created is relevant for this thesis, so this formulation is explained in detail in Section 3.2.1.

The paper of Phillipson and Chiscop (2021) has some similarities to the paper of Neukart et al. (2017). In this paper a QUBO formulation is made to minimize traffic congestion. For each car, it is determined which route it should take to minimize the congestion. Each car has three maximal dissimilar route options to pick from. This is similar to only having two or four options for the container assignment problem, as is done by the paper of Phillipson and Chiscop (2021). The minimization of traffic congestion problem is solved with the D-Wave quantum annealer.

There are a lot of other problems that have been formulated as a QUBO problem, such that they fit on the quantum annealer of D-Wave Systems. The papers of Kochenberger et al. (2014) and Glover et al. (2019) contain extensive lists of QUBO formulations for all kind of combinatorial problems. Hauke et al. (2020) also state a list with applications for the quantum annealer and summarize possible ways to solve some questions in the field of quantum annealing.

Other examples of problems that have been solved using the quantum annealer are the papers of

Neukart et al. (2018a), Neukart et al. (2018b) and Neumann et al. (2020). Neukart et al. (2018a) create a QUBO formulation for the clustering problem and solve it with the D-Wave quantum annealer. The article of Neukart et al. (2018b) creates a QUBO formulation for Monte Carlo policy iteration. The paper of Neumann et al. (2020) creates a reinforcement learning algorithm for grid-traversal using multiple agents in a way such that it can be solved using a quantum annealer.

There is an equivalent formulation to a QUBO model that also fits on the quantum annealer, the Ising model. This thesis will focus on the QUBO formulation, so the Ising model will not be explained in detail. The equivalence between the models is explained in the paper of van Vreumingen et al. (2019). Furthermore, they create an algorithm, that fits on the D-Wave quantum annealer, which optimizes a three-dimensional shape under given physical criteria. Lucas (2014) explains the Ising model in more detail and gives multiple formulations for combinatorial optimization problems.

Another branch in which research is conducted is how the quantum annealer could be used in combination with a classical heuristic. This is called a hybrid quantum-classical algorithm. The paper of Tran et al. (2016) describes a more general hybrid algorithm. They use a branching tree to determine the solution to the problem. At each node the quantum annealer is used to find solutions that do not violate the constraints, but the problem solved with the quantum annealer does not incorporate the objective function. A node in the tree is pruned when the quantum annealer does not find configurations for which the problem is feasible. The paper of Ushijima-Mwesigwa et al. (2019) is another example of a hybrid algorithm. Here the graph partitioning problem and the community detection problem are being solved. Ajagekar et al. (2020) create four of such algorithms to solve the molecular conformation problem, the job-shop scheduling problem, the manufacturing cell formulation problem and the vehicle routing problem.

The papers of Feld et al. (2019) and Ding et al. (2020) have created hybrid algorithms to solve logistical problems. In the article of Feld et al. (2019) a hybrid classical-quantum algorithm is developed to solve the capacitated vehicle routing problem. They divide the problem into two phases, a clustering phase and a routing phase. The clustering phase is modeled as a knapsack problem and the routing phase as a traveling salesman problem. The clustering phase is solved classically and the routing phase is solved using the quantum annealer. Ding et al. (2020) solve the customer-facility problem by using a hybrid algorithm. Using the quantum annealer proves difficult, because information about whether or not the facility is open is needed to determine the feasibility. This information is not available during the optimization, because the values of qubits cannot be checked at intermediate points within the anneal. They construct two phases to solve this issue. In the first phase it is determined which facilities are build. Secondly, the customers are assigned to the open facilities. The second phase is implemented on the quantum annealer. They use a simulated annealing algorithm to find the optimal solution. First, neighbours of the current solution are determined, which are different facilities that might be opened or closed. This solution is then used in the quantum annealer. If this solution has a worse objective value, it might still be selected for the next iteration with some probability.

3.2.1 QUBO formulation container assignment problem

Each container has two possibilities for shipment. The first option is by only using a truck for shipment. The other possibility is to use a multimodal route through the network. Let the set $K = \{1, ..., n_{cont}\}$ represent the containers and the set $M = \{1, ..., m\}$ the tracks. The tracks could be compared to arcs within a graph. Each track $j \in M$ has a capacity V_j. The parameters c_k^b and c_k^t are the costs for container $k \in K$ using the multimodal route and the truck respectively. Let r_k be the multimodal route for container k, which can be represented as a binary string. This string has a length equal to the number of tracks. Each bit contains the value one if the corresponding track is a part of the route. The binary parameter r_{kj} equals one if the route for container k contains track j. The decision variables x_k are also binary and contain the value one if container k is transported by truck and zero if the container is transported by barge route r_k. Assume that all routes in the model are feasible with regard to due dates et cetera. This leads to the binary integer linear program:

$$\min \sum_{k=1}^{n_{cont}} c_k^b + (c_k^t - c_k^b)x_k, \tag{2}$$

$$\text{such that} \sum_{k=1}^{n_{cont}} (1 - x_k)r_{kj} \leq V_j \qquad \forall j \in M, \tag{3}$$

$$x_k \in \{0, 1\}. \tag{4}$$

Only equality constraints can be added to the QUBO. This means that if a track is not used to its full capacity, some additional slack variables have to be added to create equality constraints for Constraint (3). Let y_{jl} be additional binary slack variables and let the extra parameter L contain the value $\max_j(log_2(V_j))$. The inequality constraints can be transferred to equality constraints as shown in Equation 7. The variables have to be binary, because otherwise they cannot be put into the quantum annealer. The parameter L has the assigned value based on the binary numeral system, because the slack variables have to be binary. For example, if a track is not used in any route and the capacity of that track is five, the parameter L will be two. The variable y_{j0} needs to have the value zero and the variables y_{j1} and y_{j2} have to contain the value one to create equality constraints by $0 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 = 5$. Let W_A and W_B be weights assigned to the objective value and the violation of constraints respectively. The QUBO can be formulated as follows:

$$\min W_A \cdot H_A + W_B \cdot H_B, \tag{5}$$

$$\text{with } H_A = \sum_{k=1}^{n_{cont}} c_k^b + (c_k^t - c_k^b)x_k, \tag{6}$$

$$H_B = \sum_{j=1}^{m}(\sum_{k=1}^{n_{cont}} (1 - x_k)r_{kj} + \sum_{l=0}^{L} 2^l y_{jl} - V_j)^2. \tag{7}$$

As can be seen, this model takes only one multimodal route into consideration, while there are a lot more options. One important note is that binary variables are the same as their quadratic counterpart, so $x_k = x_k^2$. This means that Equation 6 can still be formulated as a quadratic function.

The model could be reformulated to consider three different multimodal routes, which is shown in Phillipson and Chiscop (2021). Only considering a subset of all the possible routes means that if the combination of optimal routes is not being taken into consideration the optimal solution will not be found.

In this thesis the model formulation with only one multimodal option is used. The genetic algorithm created in this thesis determines the multimodal route that is given as option for every container. The focus of the paper of Phillipson and Chiscop (2021) was the performance of the quantum annealer when only one multimodal route is used as option. These given multimodal routes were considered as the optimal variables. In these thesis, the challenge lies in finding which multimodal route is the optimal option for every container.

3.3 Multicommodity flow problem

The multicommodity flow problem is a well known problem. Some common formulations and applications are given by Ahuja et al. (1988), Barnhart et al. (2001) and Wang (2018a). Different solution methods have been introduced to solve this problem. A survey of these methods can be found in the paper of Wang (2018b).

The multicommodity, multimodal routing problem has been solved a few times in the literature. Different solution methods have been created. Haghani and Oh (1996) provide two solution methods to solve the problem. The first one uses a decomposition technique. The second fixes some of the integer variables and relaxes the others. They find that the second solution method gives better results. The heuristic proposed in the paper of Chang (2008) uses a decomposition method. The Lagrange relaxation is used to relax the linking constraints. This creates multiple shortest path problems. Moccia et al. (2011) use column generation to solve the linear programming relaxation of the problem. A heuristic is used to create integer values. The paper of Ayar and Yaman (2012) proposes two formulations to the problem. The first formulation contains valid inequalities and the second formulation uses the Lagrange relaxation. In the article of Xiong and Wang (2014) an algorithm is created that consists of two parts. The first part solves the k-shortest path problem to create feasible routes. The second part uses a genetic algorithm to assign transportation modes to the feasible routes. The paper of Sun and Lang (2015) linearizes the problem, such that it can be solved with mathematical programming solvers. This model is extended, in the paper of Sun et al. (2016), by adding fuzzy time windows and fuzzy demand.

3.4 Genetic algorithms

Genetic algorithms have been extensively studied in literature. The first genetic algorithm was introduced in the book of Holland et al. (1992). Note that this is a more recent version of the book, the first introduction was in 1975. The paper of Ribeiro Filho et al. (1994) locations the genetic algorithm in context with other search techniques. The article of Inagaki et al. (1999) uses a genetic algorithm to construct multiple (semi-)short routes. The shortest path problem is related to the container assigned problem, because the best routes for the containers have to be determined. A genetic algorithm has been used to solve the shortest path problem in the paper of Ahn and Ramakrishna (2002). In the paper of Ahn and Ramakrishna (2002), partial routes are exchanged in the cross-over phase. It is required that both of the parents, between which the routes are exchanged, visit the same two nodes. A repair function is used to make infeasible routes feasible. The mutation operator randomly selects a node and changes the consecutive node, then a new route is constructed from that new node to the destination node.

Some other applications, where a genetic algorithm is used, are a clustering technique (Maulik and Bandyopadhyay (2000)), vehicle routing problems (Baker and Ayechew (2003),Tikani and Setak (2019)), image processing (Paulinas and Ušinskas (2007)), the optimization of truss structures (Toğan and Daloğlu (2008)), workflow schedule optimization (Ahmad et al. (2016)) and face recognition (Zhi and Liu (2019)). What all of these applications have in common is that there are a lot of different solutions possible, just as in the container assignment problem.

Genetic algorithms have been mentioned a few times in combination with quantum computing. An example where quantum annealing is used within a genetic algorithm is the paper of King et al. (2019). They use (reverse) quantum annealing as the mutation step within a genetic algorithm for randomly generated Ising problems.

Most papers that mention quantum computing in combination with genetic algorithms have created a genetic algorithm based on quantum computing principles to solve multiple problems. Examples such papers are the knapsack problem (Han and Kim (2002)), the job shop scheduling problem (Gu et al. (2010)), the flow shop scheduling problem (Li and Wang (2007),Gu et al. (2008a)), the machine scheduling problem (Gu et al. (2008b)), the economic dispatching problem (Lee et al. (2011)), the traveling salesman problem (Talbi et al. (2004)) and the numerical optimization problem (Wang et al. (2005)). The algorithms in these papers are known as quantum-inspired genetic algorithms. They use the fact that a qubit can be in a superposition. The chromosomes of a genetic algorithm are described using qubits instead of classical bits. Practically, this means that every entry of a single chromosome is described using a probability instead of a zero or one value. It starts with a chromosome where every qubit has the same probability of having a zero or one value. Actual solutions are made by generating random numbers and comparing this random number with the probability value of an entry of a chromosome. For example, if an entry of a chromosome is 0.2 it means that the actual solution will have a value 1 with a probability of 0.2. Based on the actual solutions the probability is adjusted such that the solutions converge to fitter states. A more detailed explanation of this concept can be found in the paper of Han et al. (2001).

In this thesis, the quantum annealer is used to solve the container assignment problem. The container assignment problem has been solved with the quantum annealer in the paper of Phillipson and Chiscop (2021). To use the quantum annealer, the problem has to be in a QUBO formulation. This formulation was shown in Section 3.2.1. Furthermore, a construction is used where the quantum annealer determines for each container if it travels by a multimodal route or by truck. This multimodal route has been predetermined and the choice that the quantum annealer makes is whether a container uses this multimodal route or not. To show that the container assignment problem remains NP-complete in this form, a proof has been constructed in Appendix B. Since only one possible multimodal route is considered for each container, it is important to carefully choose which route this is per container. If the final optimal multimodal route for every container is not chosen, the optimum will never be found. There are a lot of different combinations possible when considering all the containers. For example, consider a problem with two containers. A possible multimodal route input is that container 1 travels through nodes A, B, C and container 2 through the nodes B, C, D. A different input is if the multimodal routes for container 1 and 2 use the nodes A, D, C and B, A, D respectively. This thesis solves the problem of determining which routes should be included in the problem that is solved with the quantum annealer. A genetic algorithm is used as the solution approach for this problem. To my knowledge this has not been done before in the literature.

4 Methodology

A genetic algorithm consists of multiple parts. First, it needs to be shown how the population is represented. The representation used in this thesis, is explained in Section 4.1. Then, the genetic algorithm has to be initialized. The initialization part is described in Section 4.2. After the initialization, the iterations start. There are five components within the iterations. The first component is a fitness function. This function is defined in Section 4.3. Based on this fitness function, a selection procedure is determined. The selection procedure for this thesis is described in Section 4.4. Then, a cross-over operation is defined and a mutation operator is constructed. These parts are described in Sections 4.5 and 4.6 respectively. A mutation operator changes a solution randomly. When this change is not based on randomness, it is called a learning operator. Finally, a removal procedure is used to control the size of the generation. How the size is controlled in this thesis is explained in Section 4.7.

In this thesis, multiple solution methods have been implemented for different parts of the genetic algorithm. There are four different initialization methods and three different selection, cross-over and mutation methods. Two of these mutation operators are learning operators.

4.1 Representation

A genetic algorithm is used to keep track of some of the different possible combination of routes. This is done by using multiple populations. Every population corresponds to a combination of routes, with only one multimodal route per container. Let the set containing all the possible populations be denoted as $\mathcal{P}$.

For every combination of routes still needs to be determined which container uses this route and which container travels by truck. This problem is solved with the quantum annealer. So, a combination of routes is used as input for the quantum annealer. This means that a single population contains the routes r_k for every container $k \in K$ from the formulation that is described in Section 3.2.1. Let these r_k values be stored in the vector $\mathbf{r}$.

An individual population keeps track of a single problem that can be solved with the quantum annealer. Hence, the solution of the quantum annealer can be assigned to the corresponding population. A population $p \in \mathcal{P}$ can be described by Equation 8, where the vector $\mathbf{r}$ represents one possible combination of routes and $QA(\mathbf{r})$ the corresponding solution returned by the quantum annealer. The solution $QA(\mathbf{r})$ contains which container travels by multimodal route and which container travels by truck.

$$Population_p = \{\mathbf{r},\ QA(\mathbf{r})\} \qquad \forall p \in \mathcal{P} \tag{8}$$

The quantum annealer often returns multiple solutions. The number of solutions stored in $QA(\mathbf{r})$ from Equation 8 is dependent on the mutation operator that is used. One of the learning operators proposed in this thesis, uses multiple solutions returned by the quantum annealer. When this operator is used, an individual population will contain the best n_{mut} solutions returned by the quantum annealer. When this operator is not used, it is only necessary to store the fittest solution. This means that it is only necessary to store the solution with the best fitness value.

Let the number of populations be denoted by n_{pop}, the number of pairs by n_{pairs} and the number of iterations by n_{iter}. The mutation rate is the probability of whether the mutation or learning operator is applied. Let this value be denoted as *mutRate*. The set containing the populations in iteration θ is described by P_θ, where $\theta = 1, ..., n_{iter}$ is the iteration under consideration. An element of this set is described as $P_\theta^{(e)}$, where e is the index of the element under consideration. Let the set S_θ contain the pairs that are made in iteration θ. This set can be described as $S_\theta = \{(P_{\theta-1}^{(s11)}, P_{\theta-1}^{(s12)}), (P_{\theta-1}^{(s21)}, P_{\theta-1}^{(s22)}), ..., (P_{\theta-1}^{(sn_{pairs}1)}, P_{\theta-1}^{(sn_{pairs}2)})\}$, $\theta = 1, ..., n_{iter}$, where $(sd1)$ is the index of the element that is used as the first parent in the d'th duo. Every pair is subject to a cross-over operation. This operation creates two new populations from the duos that are used as input. Let $(O_\theta^{(d1)}, O_\theta^{(d2)})$ be the populations that are created in the cross-over part from the duo $(P_\theta^{(sd1)}, P_\theta^{(sd2)})$, $d = 1, ..., n_{pairs}$.

An outline of the genetic algorithm is given in Algorithm 1. First, the populations are initialized in step 1. Secondly, the iterations start. Every iteration begins by creating pairs of populations based on the fitness function. This are the populations of the previous iteration. The n_{pairs} pairs are made in step 4. Each pair is combined in a cross-over operation to create new combinations of routes, which means new populations. The cross-over is shown in step 6. The newly found combination is evaluated by solving it with the quantum annealer. This is a part of the cross-over operation in step 6. In step 7, you let the set containing all the populations in this iteration be equal to the populations from the previous iteration. The newly found populations can simply be added to this set. The offspring that is created by the cross-over operation might be subject to changes based on a mutation or learning mechanism. In step 8 and 12, it is determined whether this change will be applied to either one of the offspring created in the cross-over operation. Uniform(0,1) means that a random number between 0 and 1 is generated. If the change is applied, the problem has to be solved with the quantum annealer again. This is displayed in step 9 and 13. Finally, the offspring is added to the population in step 15. At the end, a part of the populations is removed such that the number of populations remains the same at the end of every iteration. This is shown in step 17.

Algorithm 1 Genetic algorithm

```
 1: $P_0$ = initialize($n_{pop}$)
 2: $\theta = 1$
 3: while $\theta < n_{iter}$ do
 4:     $S_\theta$ = select($P_{\theta-1}$,$n_{pairs}$)
 5:     for ($P_\theta^{(sd1)}$, $P_\theta^{(sd2)}$) $\in S_\theta$ do
 6:         ($O_\theta^{(d1)}$, $O_\theta^{(d2)}$) = cross-over(($P_\theta^{(sd1)}$, $P_\theta^{(sd2)}$))
 7:         $P_\theta = P_{\theta-1}$
 8:         if Uniform(0,1)<$mutRate$ then
 9:             $O_\theta^{(d1)}$ = mutation($O_\theta^{(d1)}$)
10:         end if
11:         $P_\theta$.add($O_\theta^{(d1)}$)
12:         if Uniform(0,1)<$mutRate$ then
13:             $O_\theta^{(d2)}$ = mutation($O_\theta^{(d2)}$)
14:         end if
15:         $P_\theta$.add($O_\theta^{(d2)}$)
16:     end for
17:     $P_\theta$ = removePart($P_\theta$)
18:     $\theta$+=1
19: end while
```

4.2 Initialization

The initialization part determines the multimodal routes for the first populations and lets the quantum annealer determine which container uses this multimodal route and which container travels by truck. The problem that is solved with the quantum annealer contains only one variable per container. This variable represents whether a truck or the given multimodal route is used to transport the container. This multimodal route is assumed to be feasible. A route is feasible if the container travels from its origin to its destination and arrives there before its due date. So, in the initialization part a feasible route has to be found for every container. If no feasible multimodal route exists at all, the container will travel by truck and can be removed from the problem.

At the beginning of the algorithm, every container needs one initial route. To find this route, the Dijkstra shortest path algorithm is used (Dijkstra et al. (1959)). This algorithm minimizes the total weight to travel from a certain node to another node. The Dijkstra algorithm is applied to find the shortest path for every container. The weights that are used are the travel costs.

The shortest path goes from a single origin node to a single destination node, but because the problem is defined on a time-space network there are multiple nodes corresponding to the origin and destination locations. To ensure that the container arrives before its due date, a dummy node is created for each container. This dummy node is connected to all the destination nodes in the time periods before the due date of this container. The dummy node is set as the destination for the container. An equivalent approach is used for the origin node. So, to ensure that the container departs from its origin after the release date, another dummy node is created. This dummy node is connected to all the origin nodes after the release date. The shortest path has to be found from the dummy origin node to the dummy destination node.

An example of a time-space network with dummy nodes for the origin and destination, can be found in Figure 3. This network considers three physical locations, A, B and C, and four time periods. Let location A in time period 1 be denoted by A1, location B in time period 1 by B1 and so on. When the origin is location A and the destination is location C, the dummy origin node O and the dummy destination node D can be made. Assume the container is released in the second time period and is due in time period 4. This means that the dummy origin node O is connected to location A in time periods 2, 3 and 4. The dummy destination node D is connected to location C in all time periods. Let a transportation service depart from location A to location B in every time period. This service has a travel time of 1 time unit. Also, in every time period a transportation service departs from location B to location C. This service has a travel time of 2 time units. There is also a transportation service that departs from location B to location C in time period 3. This service has a travel time of 1 time unit. If the cost of traveling 1 time unit is 1, minimizing the costs is equivalent to minimizing the travel time. The Dijkstra algorithm is applied to find the shortest path from O to D. The shortest path in this case is from A2 to C4.

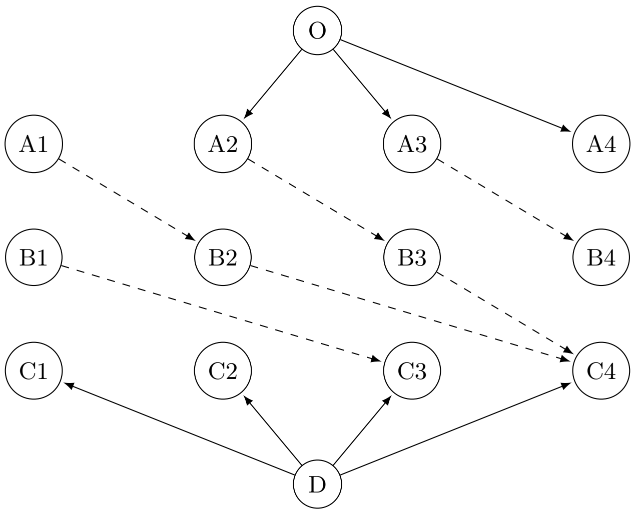


Figure 3: Time-space network with dummy nodes

Since multiple populations are used, multiple populations have to be initialized. This means that different routes have to be found per container. Dijkstra's algorithm can only be applied to the same graph once, because there might be only one shortest path for each container. Therefore, the shortest paths only determine the first population in the set of initial populations.

The shortest routes are chosen, because if the capacity allows every container to travel by the multimodal route, this is found by the algorithm. Furthermore, if the optimal solution contains a lot of the shortest paths, it is important that a most of them are found in the genetic algorithm, because that influences the solution value quite a bit.

In this thesis, two slightly different ways to determine this first population have been created. After the first population has been constructed, two different methods have been implemented to determine the remaining populations. First, the different ways to construct the first population are discussed. Then, the initialization methods used to construct the rest of the initial populations are explained.

4.2.1 The first population

The shortest paths are used to determine the first population. It might occur that multiple shortest paths exist for a single container. For example, when a shortest route departs from the origin in the first time period and this same route exists only now it departs from the origin in the second time period.

The first method to create the first population chooses the first shortest path that is found as multimodal route. This means that if there are multiple containers with the same origin and the same destination locations, those will all have the same multimodal route. After the multimodal route has been determined for every container, it is determined which container travels by this route and which container is transported by truck. This problem is solved with the quantum annealer.

The set containing all the initial populations is denoted as P_0. The first element of this set, so the first population, is described as P_0^0. The route of container $k \in K$ in population P_0^0 is stored in $P_0^0(r_k)$. The first method to create the first population is described in Algorithm 2. So in step 3, the firstly found shortest route is added to the set of routes of the first population for every container. Then, the quantum annealer determines which containers travel by this multimodal route and which by truck. Finally, the first population is added to the set containing all the initial populations in step 6.

The second method to create the first population determines all the shortest paths for every container. The path that is used as multimodal route for a container is chosen randomly between all the found shortest paths for this container. The second method to create the first population is shown in Algorithm 3. Here, all the shortest paths for a container are determined in step 3. Then a random one is chosen in step 4. When one route has been chosen for every container, the quantum annealer determines which

Algorithm 2 Genetic algorithm first population method 1

```
1: $P_0 = \emptyset$
2: for $k \in K$ do
3:     $P_0^0(r_k)$ = firstDijkstra($k$,totalGraph)
4: end for
5: $P_0^0(QA(\mathbf{r}))$ = quantumAnnealer($P_0^0(\mathbf{r})$)
6: $P_0$.add($P_0^0$)
```

container uses this route and which travels by truck in step 6. Finally, the first population is added to the set with all the initial populations in step 7.

Algorithm 3 Genetic algorithm first population method 2

```
1: $P_0 = \emptyset$
2: for $k \in K$ do
3:     allShortestRoutes$_k$ = Dijkstra($k$,totalGraph)
4:     $P_0^0(r_k)$ = pickRandom(allShortestRoutes$_k$)
5: end for
6: $P_0^0(QA(\mathbf{r}))$ = quantumAnnealer($P_0^0(\mathbf{r})$)
7: $P_0$.add($P_0^0$)
```

4.2.2 First initialization method

The first initialization method applies a random mutation to every route in the first population to construct the second population. This random mutation is then applied to the routes of the second population to construct the third population and so on.

The mutation is to randomly remove one of the selected nodes in the previous route from the available nodes for this container. Then, Dijkstra's algorithm is applied again to find the new shortest path without this node. For example, if the previous route for a container travels from A to B to C, it might be possible that node B is removed. Then, another route has to be found to travel from A to C. The mutation is applied per container, so if a node is removed for one container it does not mean that this node is also removed for another container.

The problem is defined on a time-space graph. So, there are multiple nodes corresponding to the same location, but in different time periods. The removed node is the node corresponding to one time period. This means that it might be possible that the mutation leads to the same route, but in a different time period.

Not just the randomly selected node from the previous route of a container is removed from the available nodes for the new route for this container. All the nodes that have been removed in previously applied mutations are removed from the available nodes for this container. So, all the nodes that have been randomly chosen are added to a kind of tabu-list and a new route has to be found without using any of these nodes. It only considers the nodes that have been removed for this container. So, if a node is removed for container 1 it does not mean that this node is also removed for container 2. As long as another route can be found, the nodes that have been selected are added to this tabu-list. When too many nodes have been removed, it might be possible that no route can be found anymore. Then, the shortest path of the first population is used in the new population and the tabu-list is emptied for the next population. Every container has its own tabu-list.

For example, if the shortest path used in the first population is A, B, C, D for a container, any of these nodes can be chosen randomly. If node B is randomly selected, the route for this container in the second population cannot use node B. The shortest route without using node B might be A, E, C, D for this container. This route is then used as multimodal option for this container in the second population. For the third population a random node is chosen from A, E, C, D and assume that now node C has been chosen. The route for this container in the third population cannot contain the nodes B or C. This means that the tabu-list for this container consists of the nodes B and C. This process is repeated and more and more nodes will be added to the tabu-list. When too many nodes have been added to the tabu-list, it might occur that no new route can be found without using any of the nodes in the tabu-list. Then, for that population, the shortest path from the first population is used as route for this container. So, in the example, it means that for that population the container will have the route A, B, C, D as multimodal route option, because that route was used in the first population. Then, the tabu-list is emptied and again a random node from the previously found route is added to this list.

Let the tabu-list of container k be denoted as Ω_k, $\forall k \in K$. The first added element is the second population. This element contains an index of 1, as the first population was denoted as P_0^0. The first initialization method is shown in Algorithm 4. The tabu-list starts empty, as is described in step 2. Then, new populations are added until the required number of populations is attained. The creation of a new population starts by picking one random node of the previous route for every container, which is shown in step 7. This node is added to the tabu-list of this container in step 8. A new route is tried to be found without any of the nodes in the tabu-list in step 11. If no new route can be found, the shortest path is used, as can be seen in step 13. The tabu-list is then emptied in step 14. When every container has a multimodal route to choose from, the problem is sent to the quantum annealer in step 17. Finally, the created population is added to the set of all initial populations in step 18.

Algorithm 4 Genetic algorithm initialization method 1
```
1: for $k \in K$ do
2:     $\Omega_k = \emptyset$
3: end for
4: $e = 1$
5: while $e < n_{pop}$ do
6:     for $k \in K$ do
7:         randomNode = pickRandom($P_0^{e-1}(r_k)$)
8:         $\Omega_k$.add(randomNode)
9:         smallGraph = totalGraph.remove($\Omega_k$)
10:        if A new route can be found then
11:            $P_0^e(r_k)$ = Dijkstra(smallGraph)
12:        else
13:            $P_0^e(r_k) = P_0^0(r_k)$
14:            $\Omega_k = \emptyset$
15:        end if
16:    end for
17:    $P_0^e(QA(\mathbf{r}))$ = quantumAnnealer($P_0^e(\mathbf{r})$)
18:    $P_0$.add($P_0^e$)
19:    $e$ += 1
20: end while
```

4.2.3 Second initialization method

The second initialization method is based on randomness. For every container a random node is selected that needs to be in the route for this container. This random node must lie within the release and due period of the container.

The first population is determined using the shortest paths as explained before. Using the shortest path algorithm for the first population ensures that a feasible route exists. If only one feasible route for a container, this route is found by Dijkstra's algorithm and used in the first population.

The second population is determined by randomly selecting a node that needs to be in the route. A new random node is chosen for each container. It might happen that no feasible route exists when a route must contain a certain node. These nodes are added to a tabu-list and cannot be picked again for this container. The nodes that lie before the release date or after the due date of a container are also added to the tabu-list of this container.

So, for the population in the current iteration, a new random node selected to be in the route for this container. This random node cannot be a node from the tabu-list. This process is repeated until a feasible route has been found for this container. When a node from the shortest path is chosen as random the route for this container in this population is the shortest path. The release and due date of container $k \in K$ are described by rel_k and due_k respectively. Let the tabu-list be denoted as Ω_k, $\forall k \in K$. The second initialization method can be found in Algorithm 5. First, the tabu-list is constructed per container in step 2 till step 6. For every container, it is determined which nodes lie before the release date of that container in step 3. Those are added to the tabu-list in step 4. The same is done for the nodes that lie after the due date for this container in step 5 and 6. Then, populations are added until there are n_{pop} populations in P_0, which is shown by step 9. A route is tried to be found per container. A random node is chosen in step 12. This node cannot be in the tabu-list and must be in the route for this container in this population. If this route exists, it is added to population P_0^e n step 14. If such route does not exist, this node is added to the tabu-list in step 16 and a new random node is chosen. When a multimodal route has been determined for every container, it must be determined which container uses this route and which travels by truck. This problem is solved with the quantum annealer, as is shown in step 20. Finally, the newly created population is added to the set containing all the initial populations in step 21.

Algorithm 5 Genetic algorithm initialization method 2

```
1: for $k \in K$ do
2:     $\Omega_k = \emptyset$
3:     nodesBefore = allNodes(:, $rel_k$)
4:     $\Omega_k$.add(nodesBefore)
5:     nodesAfter = allNodes($due_k$, :)
6:     $\Omega_k$.add(nodesAfter)
7: end for
8: $e = 1$
9: while $e<n_{pop}$ do
10:    for $k \in K$ do
11:        while No route is found do
12:            randomNode = pickRandom(allNodes,$\Omega_k$)
13:            if A new route can be found using this node then
14:                $P_0^e(r_k)$ = Dijkstra(totalGraph,randomNode)
15:            else
16:                $\Omega_k$.add(randomNode)
17:            end if
18:        end while
19:    end for
20:    $P_0^e(QA(\mathbf{r}))$ = quantumAnnealer($P_0^e(\mathbf{r})$)
21:    $P_0$.add($P_0^e$)
22:    $e$ += 1
23: end while
```

4.3 Fitness function

There are two fitness functions within this genetic algorithm. One that measures the quality of an individual solution within the $QA(\mathbf{r})$ part of a population and one that measures the quality of a population. An individual solution within a population represents which containers travel by truck and which travel by the given multimodal routes. A population represents a given combination of multimodal routes and contains one or more solutions to the problem that specifies which containers use the multimodal route and which use the truck. How many solutions are included in the population is dependent on the mutation operator that is used.

Both fitness functions take the solution value and the violation of constraints into account. An individual solution represents which containers travel by multimodal route and which by truck. If a constraint has been violated, a penalty value will be added to the fitness value. This penalty value is equal to the sum of the costs for traveling by truck with every container. Assume that there are n_{cont} containers. Let c_k^b and c_k^t, $k = 1, ..., n_{cont}$, be the costs for container k to travel by multimodal route or truck respectively. Let f contain the value 1 if a constraint has been violated and the value 0 otherwise. If x_k contains the value 1 when container k travels by truck, the fitness function for an individual solution looks like the function described in Equation 9.

$$Fitness(x) = \sum_{k=1}^{n_{cont}} c_k^b + \sum_{k=1}^{n_{cont}} (c_k^t - c_k^b)x_k + f \sum_{k=1}^{n_{cont}} c_k^t \tag{9}$$

So, for an individual solution, the fitness function is the solution value plus a penalty value if this solution violates a constraint. For a whole population, the best found solution in terms of the objective value and the violation of constraints is used to determine the fitness value of this whole population.

4.4 Selection procedure

Three different selection procedures have been implemented. The selection mechanism chooses populations not individual solutions. A selection procedure consists of determining which populations can be picked for a cross-over operation and to make pairs from these selected populations. First, a common selection procedure called elitism is explained. Then, a selection method based on the combination of good methods with bad methods is described. Finally, the selection procedure known as the binary tournament is explained.

4.4.1 Elitism

A common selection mechanism is elitism. In the elitism selection mechanism the individuals with the highest fitness value are selected to create new solutions. In this genetic algorithm that means that the populations with the highest fitness value are selected. First a part of the population is selected, then pairs are created from this selected group. The first part picks the best populations in terms of fitness function. From the selected populations, the pairs are made randomly. The elitism procedure is described in Algorithm 6. At the beginning, the populations of the previous iteration are sorted in step 2. Then, only the best $n_{pairs} \times 2$ populations are obtained in step 3. Then, the pairs are made in steps 5 till 12. In step 6, a random population is chosen from the set $P_{\theta-1}$. This population is removed from $P_{\theta-1}$ in step 7. The second part of the duo is randomly chosen in step 8 and this population is removed in step 9. The created pair is added to the set containing all pairs in step 10.

Algorithm 6 Elitism

```
1: $S_\theta = \emptyset$
2: $P_{\theta-1} = \text{sort}(P_{\theta-1})$
3: $P_{\theta-1} = P_{\theta-1}(:, n_{pairs} \times 2)$
4: $d = 1$
5: **while** $d < n_{pairs}$ **do**
6:     $P_{\theta-1}^{(sd1)} = \text{pickRandom}(P_{\theta-1})$
7:     $P_{\theta-1}.\text{remove}(P_{\theta-1}^{(sd1)})$
8:     $P_{\theta-1}^{(sd2)} = \text{pickRandom}(P_{\theta-1})$
9:     $P_{\theta-1}.\text{remove}(P_{\theta-1}^{(sd2)})$
10:    $S_\theta.\text{add}((P_{\theta-1}^{(sd1)}, P_{\theta-1}^{(sd2)}))$
11:    $d \mathrel{+}= 1$
12: **end while**
```

4.4.2 Best and worst populations

Another way the parents can be paired is to combine the best and the worst population. To do so, first the number of available populations is reduced. The populations are sorted based on their fitness function. Then, the list with populations is reduced to the number of parents that need to be selected. So, the best populations remain available for the cross-over operation. This is similar to the start of the elitism selection mechanism. From this reduced list, the best population is paired with the worst population. Then, the second best population is combined with the second worst population and so on. Let the last element of the population set of iteration θ be denoted by P_θ^{-1}. This selection mechanism is described in Algorithm 7. Again, the procedure starts by sorting and retrieving the best $n_{pairs} \times 2$ populations from the previous iterations in step 2 and 3. The best population is selected in step 6. This population is then removed from the available populations in step 7. Step 8 picks the worst population and step 9 removes this population. The chosen populations are combined in a pair and added to the set containing all pairs in step 10.

4.4.3 Binary tournament

Another selection mechanism is that of a binary tournament. Here, the parents are chosen separately. First, two populations are chosen randomly. The first parent is the population with the best fitness value from these two randomly chosen populations. Then, again two random populations are chosen and the best of these two populations is chosen as the second parent. This selection procedure is described in

Algorithm 7 Best and worst

```
1: $S_\theta = \emptyset$
2: $P_{\theta-1}$ = sort($P_{\theta-1}$)
3: $P_{\theta-1} = P_{\theta-1}(:, n_{pairs} \times 2)$
4: $d = 1$
5: while $d < n_{pairs}$ do
6:     $P^{(sd1)}_{\theta-1} = P^{0}_{\theta-1}$
7:     $P_{\theta-1}$.remove($P^{0}_{\theta-1}$)
8:     $P^{(sd2)}_{\theta-1} = P^{-1}_{\theta-1}$
9:     $P_{\theta-1}$.remove($P^{-1}_{\theta-1}$)
10:    $S_\theta$.add(($P^{(sd1)}_{\theta-1}$, $P^{(sd2)}_{\theta-1}$))
11:    $d$ += 1
12: end while
```

Algorithm 8. This procedure starts by picking two different random populations, as is shown in step 4 and 5. The exclamation mark means that this value cannot be picked. In step 6, it is determined which of these populations has the best fitness value. The population that has the best fitness value is chosen as the first part of the pair and removed from the population, which is shown in step 7 and 8 or step 10 and 11 depending on which population is the best. This process is repeated to find the second part of the pair in steps 13 till 21. Finally, the chosen populations are combined in a pair and added to the set containing all the pairs in step 22.

Algorithm 8 Binary tournament

```
1: $S_\theta = \emptyset$
2: $d = 1$
3: while $d < n_{pairs}$ do
4:     random1 = pickRandom($P_{\theta-1}$)
5:     random2 = pickRandom($P_{\theta-1}$, !random1)
6:     if random1.fitness<=random2.fitness then
7:         $P^{(sd1)}_{\theta-1}$ = random1
8:         $P_{\theta-1}$.remove(random1)
9:     else
10:        $P^{(sd1)}_{\theta-1}$ = random2
11:        $P_{\theta-1}$.remove(random2)
12:    end if
13:    random3 = pickRandom($P_{\theta-1}$)
14:    random4 = pickRandom($P_{\theta-1}$, !random3)
15:    if random3.fitness<=random4.fitness then
16:        $P^{(sd1)}_{\theta-1}$ = random3
17:        $P_{\theta-1}$.remove(random3)
18:    else
19:        $P^{(sd1)}_{\theta-1}$ = random4
20:        $P_{\theta-1}$.remove(random4)
21:    end if
22:    $S_\theta$.add(($P^{(sd1)}_{\theta-1}$, $P^{(sd2)}_{\theta-1}$))
23:    $d$ += 1
24: end while
```

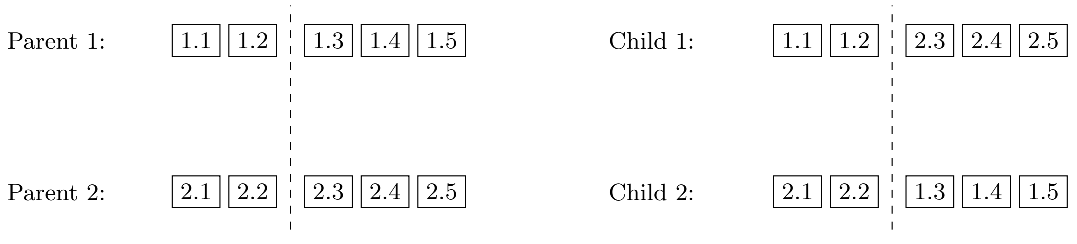
4.5 Cross-over operator

Three cross-over operators have been implemented. A one-point, two-point and partial cross-over operator. All these operators create two new populations from two given populations. In the proposed genetic algorithm a population corresponds to a combination of routes. The newly found combinations are solved with the quantum annealer to determine the quality of this new combination of routes. First, the one-point cross-over operator is explained. Then, the two-point cross-over operator is described. Finally, a cross-over operator that uses route segments is introduced.

4.5.1 One-point cross-over operator

The one-point cross-over operator does not change individual routes, but switches total routes for some containers. In the one-point cross-over operator a random container is chosen. The first child contains the routes of the first parent up till the randomly chosen container. The rest of the containers have the routes from the second parent as multimodal option. For the second child it means that the routes up till the randomly chosen container are the same as the routes of the second parent, while the rest of the routes are the same as the routes of the first parent.

For example, consider two parental populations and 5 containers. Denote the multimodal route for container 1 in parent 1 as 1.1 and the route for container 1 in parent 2 as 2.1. So, the route for container 2 in parent 2 is shown as 1.2 and the route for container 2 in parent 2 as 2.2 and so on. Thus, the first parent has the routes 1.1, 1.2, 1.3, 1.4 and 1.5 as options and the second parent has 2.1, 2.2, 2.3, 2.4 and 2.5 as multimodal routes for the containers. Assume that the randomly chosen container is container 3. Then, the first child has the routes 1.1, 1.2, 2.3, 2.4 and 2.5. The second child has the routes 2.1,2.2,1.3,1.4 and 1.5. This cross-over operator is shown in Figure 4.



(a) Parents (b) Offspring

Figure 4: One-point cross-over operator

The random point is chosen between 2 and n_{cont}, because otherwise the populations would just be exchanged. The one point cross-over operation is described in Algorithm 9. So, the cross-over operation is applied to every created pair, as indicated in step 1. A random point is chosen between 2 and the number of containers in step 2. It has to be from 2, otherwise no containers would be exchanged. The containers until the randomly chosen point are assigned to the corresponding offspring. So, for all the containers under the randomly chosen point, the routes of the first population of the pair are assigned to the first offspring in step 4. In step 5, the routes of the containers until the randomly chosen point of the second population are given to the second offspring. For the other containers, are the routes of the second population of the pair given to the first offspring, as was shown in step 8. The routes of the containers that are after the randomly chosen points of the first population of the pair are assigned to the second offspring, as is indicated in step 9. When all containers have been assigned a route for both offspring, the problem is solved with the quantum annealer in steps 11 and 12.

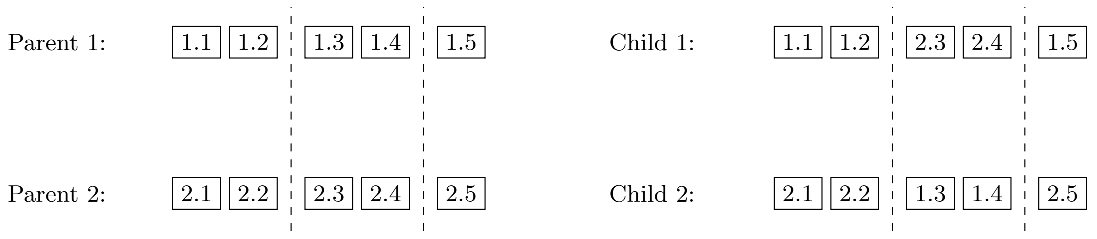
Algorithm 9 One point cross-over

```
1: for ($P^{sd1}_{\theta-1}$,$P^{sd2}_{\theta-1}$)$\in S_\theta$ do
2:     randomPoint = randint(2,$n_{cont}$)
3:     for $k$<randomPoint do
4:         $O^{d1}_\theta(r_k) = P^{sd1}_{\theta-1}(r_k)$
5:         $O^{d2}_\theta(r_k) = P^{sd2}_{\theta-1}(r_k)$
6:     end for
7:     for $k \geq$randomPoint do
8:         $O^{d1}_\theta(r_k) = P^{sd2}_{\theta-1}(r_k)$
9:         $O^{d2}_\theta(r_k) = P^{sd1}_{\theta-1}(r_k)$
10:    end for
11:    $O^{d1}_\theta(QA(\mathbf{r}))$ = quantumAnnealer($O^{d1}_\theta(r_k)$)
12:    $O^{d2}_\theta(QA(\mathbf{r}))$ = quantumAnnealer($O^{d2}_\theta(r_k)$)
13: end for
```

4.5.2 Two-point cross-over operator

The two-point cross-over operator is similar to the one-point cross-over operator. Now, there are two points, so containers, chosen randomly. Between these two points the routes for the containers are exchanged.

Consider the same example as before, so parent 1 contains the multimodal routes 1.1, 1.2, 1.3, 1.4 and 1.5 and parent 2 contains the routes 2.1, 2.2, 2.3, 2.4 and 2.5. Now, let the first randomly chosen point be at container 3 and the second point at container 5. So, the first child now contains 1.1, 1.2, 2.3, 2.4 and 1.5 as multimodal routes and the second child has 2.1, 2.2, 1.3, 1.4 and 2.5 as routes. This is shown in Figure 5.



(a) Parents (b) Offspring

Figure 5: Two-point cross-over operator

The first point is chosen between 2 and n_{cont}-1. The second point between the first randomPoint+1 and n_{cont}. The two point cross-over operation is described in Algorithm 10. For every pair, it starts by picking two random points. The first point must be between 2 and $n_{cont} - 1$, because otherwise there is no two point exchange. The second point must be from the first random point until the number of containers. This is shown in step 2 and 3. Then, the first offspring gets assigned the routes of the first population of the pair, of the containers that are before the first randomly chosen point or after the second randomly chosen point, as is shown in step 4 and 5. Steps 4 and 6 show that this also applies for the second offspring. For the containers between the first randomly chosen point and second randomly chosen point applies that, the first offspring gets the routes of the second population of the pair and the second offspring gets the routes of the first population of the pair. This is indicated by steps 8 till 10. When every container has been assigned a route, the problem is sent to the quantum annealer, as is shown in steps 12 and 13.

Algorithm 10 Two point cross-over

```
1: for ($P^{sd1}_{\theta-1}$,$P^{sd2}_{\theta-1}$)$\in S_\theta$ do
2:     randomPoint1 = randint(2,$n_{cont}$-1)
3:     randomPoint2 = randint(randomPoint1+1,$n_{cont}$)
4:     for $k$<randomPoint1 and $k$>=randomPoint2 do
5:         $O^{d1}_\theta(r_k) = P^{sd1}_{\theta-1}(r_k)$
6:         $O^{d2}_\theta(r_k) = P^{sd2}_{\theta-1}(r_k)$
7:     end for
8:     for randomPoint1 $\leq k<$ randomPoint2 do
9:         $O^{d1}_\theta(r_k) = P^{sd2}_{\theta-1}(r_k)$
10:        $O^{d2}_\theta(r_k) = P^{sd1}_{\theta-1}(r_k)$
11:    end for
12:    $O^{d1}_\theta(QA(\mathbf{r}))$ = quantumAnnealer($O^{d1}_\theta(r_k)$)
13:    $O^{d2}_\theta(QA(\mathbf{r}))$ = quantumAnnealer($O^{d2}_\theta(r_k)$)
14: end for
```

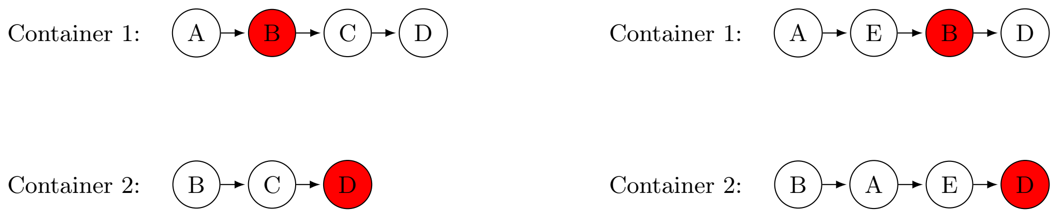
4.5.3 Partial cross-over operator

For the cross-over operator that uses route segments, two populations are again selected to create two new populations. This cross-over operator creates new populations by combining the routes of the parent populations. A part of the route for a single container of parent 1 is exchanged with a part of the route for the same container of parent 2. To create a route segment each route is split into two pieces. The first part of the route of the first parent is matched with the second part of the route of the second parent. This is done for every container and creates new combinations of routes.

So, for every container, a route segment is exchanged between every route of the two parent populations to create the offspring. To exchange a part of a route for a single container, it is required that both routes visit the same node. From all the nodes that both routes visit, a random node is chosen. This random node determines how the partial routes are made. The first child contains the route segment of the first parent that goes from the origin to the randomly chosen node and the route segment of the second parent that goes from the randomly chosen node to the destination. The second child travels from the origin to the randomly chosen by the same route segment as the second parent. Then, it travels from the randomly chosen node to the destination by using the same route segment that the first parent used to travel from the randomly chosen node to its destination.

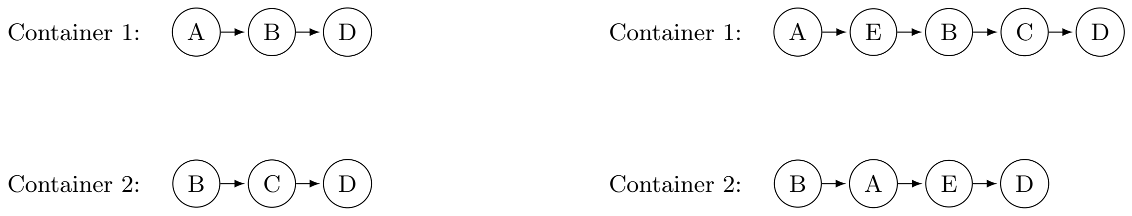
It is possible that the only nodes that are visited by both routes are the origin and destination nodes. If the origin node is randomly chosen, then the entire route is exchanged between the populations. So, the first child inherits the route of the second parent and the second child gets the route of the first parent for this particular container. If the destination node is selected the route is not exchanged. So, now the first child inherits the route of the first parent and the second child inherits the route of the second parent. The origin and destination nodes are also an option when there are multiple nodes visited by both routes. This cross-over operator is applied to every container.

An example of the cross-over operation can be seen in Figures 6 and 7. The example considers a problem with two containers and two populations that have been chosen by the selection procedure. Figure 6 displays the routes for the two populations that are combined. The first container has the route A, B, C, D in population 1 and A, E, B, D in population 2. The nodes that both these routes visit are A, B, D. Assume that node B is chosen randomly. The second part of the routes are swapped, leading to the routes for container 1 displayed in Figure 7. For the second container the origin and destination are the only nodes that are visited by both routes. Assume that the destination node is chosen randomly, meaning that nothing will be exchanged.



(a) Population 1 (b) Population 2

Figure 6: Before the cross-over operation



(a) Population 1 (b) Population 2

Figure 7: After the cross-over operation

Let O_k and D_k correspond to the origin and destination of container $k \in K$. The partial cross-over method is described in Algorithm 11. For every container, it is first determined which nodes are both visited by the routes of the populations in the pair, as is shown in step 3. Then a random node is chosen from these nodes in step 4. If the randomly chosen node is the origin of the container under consideration, the first offspring gets the route of the second population, as is displayed in step 6. The second offspring gets the route of the first population, as is shown in step 7. If the randomly chosen node is the destination of this container, the routes are not exchanged at all. So, as step 9 implies, the first offspring gets the route of the first population for this container. The same applies for the second container, as shown in step 10. When the randomly chosen node is not the origin or destination, the steps 12 till 19 are executed. The route of the first population of the container under consideration, is split into two parts. The first part contains the path from the origin to the random node, as is shown in step 12. The second part contains the route from the random node until the destination, which is demonstrated in step 13. The same is done for the route of the second population of the pair, as is displayed in steps 14 and 15. The new route of the first offspring is constructed by using the part of the origin to the random node of the first population and the part of the random node to the destination of the second population. This is shown in step 16. In step 17, the path from the origin to the random node of the second population is combined with the path from the random node till the destination of the first population. When both offspring have been assigned a multimodal route for every container, the problem is sent to the quantum annealer in steps 20 and 21.

Algorithm 11 Partial cross-over

```
1: for ($P^{sd1}_{\theta-1}$,$P^{sd2}_{\theta-1}$)$\in S_\theta$ do
2:     for $k \in K$ do
3:         commonNodes = common($P^{sd1}_{\theta-1}(r_k)$,$P^{sd2}_{\theta-1}(r_k)$)
4:         randomNode = pickRandom(commonNodes)
5:         if randomNode = $O_k$ then
6:             $O^{d1}_\theta(r_k) = P^{sd2}_{\theta-1}(r_k)$
7:             $O^{d2}_\theta(r_k) = P^{sd1}_{\theta-1}(r_k)$
8:         else if randomNode = $D_k$ then
9:             $O^{d1}_\theta(r_k) = P^{sd1}_{\theta-1}(r_k)$
10:            $O^{d2}_\theta(r_k) = P^{sd2}_{\theta-1}(r_k)$
11:        else
12:            firstPartRoute1 = $P^{sd1}_{\theta-1}(r_k)$(:, randomNode)
13:            secondPartRoute1 = $P^{sd1}_{\theta-1}(r_k)$(randomNode, :)
14:            firstPartRoute2 = $P^{sd2}_{\theta-1}(r_k)$(:, randomNode)
15:            secondPartRoute2 = $P^{sd2}_{\theta-1}(r_k)$(randomNode, :)
16:            $O^{d1}_\theta(r_k)$ = combine(firstPartRoute1,secondPartRoute2)
17:            $O^{d2}_\theta(r_k)$ = combine(firstPartRoute2,secondPartRoute1)
18:        end if
19:    end for
20:    $O^{d1}_\theta(QA(\mathbf{r}))$ = quantumAnnealer($O^{d1}_\theta(r_k)$)
21:    $O^{d2}_\theta(QA(\mathbf{r}))$ = quantumAnnealer($O^{d2}_\theta(r_k)$)
22: end for
```

4.6 Mutation and learning operators

Three methods have been implemented as mutation or learning operator. In each iteration the population might be subject to some change. The probability of a population being subject to change is called the mutation rate. This value will be determined by parameter tuning. When the change is based on randomness it is called a mutation operator. If the change is not based on randomness it is called a learning mechanism. The first described method is based on randomness, so this is a mutation operator. The other two methods are learning operators.

For the mutation and the first learning operator the change is applied per container. This means that for every container it is determined whether it is subject to change. It might happen that the first container is subject to change, but the second is not. For the second learning operator applies that the whole population is changed or not. So, the route is changed for every container if this operator is applied.

4.6.1 Mutation operator

The mutation operator that has been implemented changes the route for every container with a certain probability. This probability is based on the mutation rate. The rate is denoted as *mutRate*. When it is decided to change the route of a container, a random node of this route is chosen. This node cannot be in the next route for this container. Dijkstra's algorithm is used to find a new route for this container, without using the randomly chosen node. If no such route exists, another random node is chosen. This is repeated until a new route has been found or every node has been tried. This process is shown in Algorithm 12. To initialize, the new population is set to the population that is used as input, which is shown in step 2. If the random number that is generated from the uniform distribution between 0 and 1 is smaller than the mutation rate, the change is applied. This is indicated in step 3. The nodes used in the current path are determined in step 4. From these nodes, a random node is chosen in step 6. A new route is tried to be found without using this node in step 7 and 8. If no route can be found, this node is removed from the list in step 11. When every container has been assigned a route, the problem is sent to the quantum annealer in step 16. Finally, the newly constructed population is added to set of populations within that iteration in step 17.

Algorithm 12 Mutation operator

```
1: for $k \in K$ do
2:   $P_\theta^{input+1}(r_k) = P_\theta^{input}(r_k)$
3:   if Uniform(0,1)<$mutRate$ then
4:     nodeList = currentNodes($P_\theta^{input}(r_k)$)
5:     while nodeList != $\emptyset$ do
6:       randomNode = pickRandom(nodeList)
7:       if A new route can be found then
8:         $P_\theta^{input+1}(r_k)$ = Dijkstra($k$,!randomNode)
9:         nodeList = $\emptyset$
10:      else
11:        nodeList.remove(randomNode)
12:      end if
13:    end while
14:  end if
15: end for
16: $P_\theta^{input+1}(QA(\mathbf{r}))$ = quantumAnnealer($P_\theta^{input+1}(\mathbf{r})$)
17: $P_\theta$.add($P_\theta^{input+1}$)
```

4.6.2 Learning operator 1

In the first implemented learning operator, the solutions with the best fitness values are used to determine how a route is adjusted. As stated before, the quantum annealer returns multiple solutions. The best n_{mut} solutions within an individual population are checked. The solutions are the best in terms of objective value and the violation of constraints. If less than n_{mut} solutions are returned by the annealer, the number of solutions that have been returned are used. An individual solution represents which containers travel by a multimodal route and which by truck. A container might travel by truck in one solution and by multimodal route in another solution. When the best n_{mut} solutions are compared, the number of times that a container travels by multimodal route within these solutions can be determined. If a container travels by multimodal route more than a threshold value, then this route is said to be selected often. Thus, the set with the routes that are selected often is a subset of the $\mathbf{r}$ of a population. The threshold value is determined by parameter tuning. From the arcs of the often selected routes, the arc that is used the most is determined.

Every container can be subject to the learning operator with a certain probability. This probability is based on the mutation rate, which is denoted by $mutRate$. A lower probability is assigned to containers that are often selected to travel by multimodal route. A higher probability is assigned to containers that are not often selected to travel by multimodal route. If a container is subject to the learning operator, consider the route for this container. If the previously determined most used arc is not used in the route of this container, create a route for this container that uses this arc. If the arc is used in the route, then construct a route that does not use this arc.

It might happen that a route is not feasible when a certain arc must or must not be used in a route. If no route can be found, the previously found route is used for the corresponding container. This method is described in Algorithm 14. First, it is determined which routes are selected often in step 1. Then, the most used arc in these routes is gained in step 2. Then, for every container, start by initializing the current route as the route of the input population in step 4. Determine whether the learning operator is applied by generating a random number from the uniform distribution in step 5. In step 6, it is checked whether the most used arc is a part of the route for the current container of the input population. If the arc has been used, try to find a route without this arc in step 7. If this arc has not been used, a route with this arc is tried to be found in step 9. When every container has been assigned a route, the problem is sent to the quantum annealer in step 13. Finally, the newly created population is added to the set of populations in the current iteration in step 14.

Algorithm 13 Learning operator method 1

```
 1: oftenRoutes = selectedOften($P_\theta^{input}(\mathbf{r})$)
 2: mostUsedArc = mostUsed(oftenRoutes)
 3: for $k \in K$ do
 4:     $P_\theta^{input+1}(r_k) = P_\theta^{input}(r_k)$
 5:     if Uniform(0,1)<$mutRate$ then
 6:         if mostUsedArc $\in P_\theta^{input}(r_k)$ then
 7:             $P_\theta^{input+1}(r_k)$ = Dijkstra(!mostUsedArc)
 8:         else
 9:             $P_\theta^{input+1}(r_k)$ = Dijkstra(mostUsedArc)
10:         end if
11:     end if
12: end for
13: $P_\theta^{input+1}(QA(\mathbf{r}))$ = quantumAnnealer($P_\theta^{input+1}(\mathbf{r})$)
14: $P_\theta$.add($P_\theta^{input+1}$)
```

4.6.3 Learning operator 2

The second learning operator that has been implemented leverages the capacity of the arcs more. As stated in the Section 3.1, it is reasonable to assume that most multimodal routes are cheaper than the transportation by truck.

In this operator, the arcs of the multimodal routes in the given population are accumulated. So, this accumulation considers not just the arcs that are selected in the solution returned by the annealer, but all the arcs that correspond to the routes within this population. It is checked for each arc if the accumulation of this arc exceeds the capacity. Then, it is determined which containers have a route as multimodal option that uses an arc for which the capacity is exceeded by the accumulative arcs. Pick a random container that uses such an arc and create a new multimodal route without using this arc for this container. For example, if the capacity of each arc is 1 and there are two containers that both use the arc from A to B in their multimodal route option, then the arc from A to B is exceeded by 1. One of these two containers is selected randomly and for this container a new route needs to be determined without the arc from A to B. This process is repeated until the arcs that were originally exceeded are not anymore.

This learning operator is shown in Algorithm 14. Start by initializing. For every container, the route is set to the route of the input population in step 4. Then, for every arc, it is determined how often this arc is used in the routes of the input population in steps 7 till 12. Step 13 checks whether this is more than the capacity of this arc. If that is the case, this arc is added to the exceededArcs list in step 14. The steps 15 until 21 determine a list of containers that use this arc. Then, the new routes are constructed in the steps 24 till 47. In step 25, a random arc is chosen from the list of arcs that are exceeded. Next, a random container is chosen from the list of containers that uses this arc in step 26. The goal is to find a new route for this container, without using the chosen arc, as is indicated by step 27. If a new route can be found for the randomly chosen container, the number of times the arc has been used is decreased by one and this container is removed from every list, as is shown in steps 29 till 32. So, the container is also removed from the list of containers corresponding to the other arcs, which corresponds to step 30. If the randomly chosen arc no longer exceeds the capacity, the arc is removed from the list of arcs with exceeding capacity, which is demonstrated by steps 33 till 35. Since a new route has been found for this container, the arcs used in the previous path are not used anymore. So, every arc that was used in the input path for the random container, is used one less time. This is shown in step 36 till 38. If with this decrease the capacity is not exceeded anymore, the arc can be removed from the exceededArcs list, as is shown in steps 39 till 41. If no route can be found, this container is removed from the list of this specific arc, but remains on the container list for the other arcs. This is displayed in step 45. As long as there exists an arc that was exceeded in the population that was subject to the learning operator, new routes have to be found. When a route has been found for every container, the problem is sent to the quantum annealer in step 48 and added to the set containing all populations of this iteration in step 49.

Algorithm 14 Learning operator method 2

```
1: exceededArcs = $\emptyset$
2: containersPerArc = $\emptyset$
3: for $k \in K$ do
4:     $P_\theta^{input+1}(r_k) = P_\theta^{input}(r_k)$
5: end for
6: for $a \in A$ do
7:     sumArc[$a$] = 0
8:     for $k \in K$ do
9:         if $a \in P_\theta^{input}(r_k)$ then
10:            sumArc[$a$] += 1
11:        end if
12:    end for
13:    if sumArc[$a$] > capacity[$a$] then
14:        exceededArcs.add($a$)
15:        useArc[$a$] = $\emptyset$
16:        for $k \in K$ do
17:            if $a \in P_\theta^{input}(r_k)$ then
18:                useArc[$a$].add($k$)
19:            end if
20:        end for
21:        containersPerArc[$a$].add(useArc[$a$])
22:    end if
23: end for
24: while exceededArcs != $\emptyset$ do
25:    randomArc = pickRandom(exceededArcs)
26:    randomContainer = pickRandom(containersPerArc[randomArc])
27:    $P_\theta^{input+1}(r_{randomContainer})$ = Dijkstra(randomContainer, !randomArc)
28:    if A route can be found then
29:        sumArc[randomArc] -= 1
30:        for $a \in$ exceededArcs do
31:            containersPerArc[$a$].remove(randomContainer)
32:        end for
33:        if sumArc[randomArc] $\leq$ capacity[randomArc] then
34:            exceededArcs.remove(randomArc)
35:        end if
36:        for $a \in P_\theta^{input}(r_{randomContainer})$ do
37:            if $a \in$ exceededArcs then
38:                sumArc[$a$] -= 1
39:                if sumArc[$a$] $\leq$ capacity[$a$] then
40:                    exceededArcs.remove($a$)
41:                end if
42:            end if
43:        end for
44:    else
45:        containersPerArc[$a$].remove(randomContainer)
46:    end if
47: end while
48: $P_\theta^{input+1}(QA(\mathbf{r}))$ = quantumAnnealer($P_\theta^{input+1}(\mathbf{r})$)
49: $P_\theta$.add($P_\theta^{input+1}$)
```

4.7 Removal procedure

The number of populations is fixed and remains the same throughout the entire iterative process. This means that when new populations have been found, a part has to be removed. An evident option is to remove the worst populations. However, this can decrease the diversity of the populations. To avoid this, only a part of the best solutions are going to the next generation. At the end of an iteration, the populations are sorted based on their fitness functions. Let n_{pop} be the number of final populations. The first half of the final populations are the best populations. The other half is determined by choosing randomly between the remaining populations.

5 Data

A problem instance consists of a graph and a container set. The graph is a time-space network. The container set contains an origin, destination, release date, due date and the cost of traveling by truck for every container. Six different networks have been created. Multiple networks have been created to evaluate the performance of the genetic algorithm for different kind of networks. The networks differ in number of nodes and graph structure. The expectation is that for a network with fewer nodes, the optimal solution can be found more easily. However, this expectation is also based on the number of available routes between the origin and destination of every container. This is also depending on the chosen origin and destination. For every network, three different container sets have been created. First the created networks are introduced. Then, it is explained how the container sets have been constructed.

5.1 Networks

Multiple networks have been created. Every network is a time-space network. So, different nodes in the time-space network correspond to the same location, but in a different time period. Every edge in a network is directed and represents an individual transportation service with a certain cost and capacity. Also, it is possible that a container is stored in the same location for a time period. The cost of the edge between the same location in different time periods is 1 and the capacity is 10 for all the created networks.

All the generated networks are weakly connected graphs. This means that if all the directed edges are replaced with undirected edges, then a path exists between every pair of vertices. However, if the edges stay directed, then there exists a pair of vertices for which no path exists. This follows from the fact that a directed edge cannot exist from a node in a later time period to a node in a previous time period.

5.1.1 Random networks

Three networks have been generated randomly. These networks contain 15, 20 or 25 locations and are defined for 4 time periods. Random transportation services have been created between these locations. To do so, two random locations are chosen. A connection is created from the first location to the second location and another connection is created from the second location to the first location. Meaning that, the randomly chosen locations are connected to and from each other. The costs and capacities of these two connections are generated randomly and might differ from each other. So, in total two costs and two capacities are generated per two randomly chosen locations. The cost and capacity both have a value between 1 and 3, every value with an equal probability. The travel time is set to 1 time unit for every connection. The connections exist in every time period.

For example, if location A and location B have been chosen randomly, a connection is made from A to B and from B to A. The cost and capacity are randomly generated and can differ for the two generated connections. This means that the cost and capacity of the service from node A to node B might be different than the cost and capacity of the service from node B to node A. For example, the cost to travel from A to B is 1 and the capacity of this connection is 2, while the cost to travel from B to A is 3 and the capacity of this connection is 1. The connection from A to B is transformed to a transportation service by having an edge from location A in time period 1 to location B in time period 2 with a cost of 1 and a capacity of 2. Since the connection exists in every time period, also edges are created between A in time period 2 and B in time period 3 and A in time period 3 and B in time period 4, all of these edges have a cost of 1 and a capacity of 2. The same is done for the connection from B to A. This example is shown in Figure 8. The cost and capacity of a transportation service are shown next to the corresponding edge and are denoted as cost, capacity.

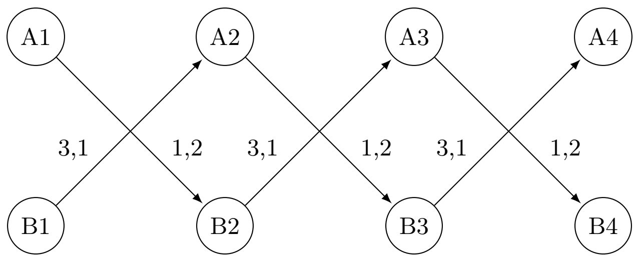


Figure 8: Example connection random graph

By letting the connections depart every time period, multiple transportation services have been created from a single generated connection. This is repeated until the required number of connections have been made.

For the network with 15 locations it was required that 80 connections were made. For this network 45 edges have been added to connect the nodes representing the same location in different time periods to allow the storage of a container at the same location for a time period. Next to these edges, 240 transportation services have been added from the 80 connections. Thus, the network with 15 locations has a time-space network with 60 nodes and 285 edges. From the nodes and the edges, the density can be determined. The density describes the portion of actual edges to the potential edges. The potential edges contain the edges that connect every node to every other node in later time periods. This means that the total number of possible edges departing in the first time period is 3×15×15. In the second time period it is only possible to have a travel time of 2 time units. So, now there are only 2×11×11 potential edges. In the third time period only transportation services with a travel time of 1 time unit are possible. For the network with 15 locations this means that the number of potential edges is 1350. Actually having 285 edges leads to a density of 21%. The maximum degree of a node in this network is 10. The indegree of a node is equal to its outdegree, because every connection goes back and forth. So, a maximum degree of 10 means that there exists a node where 10 edges depart from and 10 edges arrive there. The minimum indegree is 0, because there are no arcs into the nodes in the first time period. The same can be said about the outdegree, because the nodes in the last time period do not have arcs towards later time periods.

The network with 20 locations had 140 as the required number of connections. It follows that 420 transportation services have been added from the 140 connections. Now, 60 edges were added to allow the storage at a location for a time period. So, the network with 20 locations has a time-space network with 80 nodes and 480 edges. The number of potential edges for this network is 2400. This leads to a density of 20%. This network has a maximum degree of 12.

For the network with 25 locations 180 connections were needed. For this network, 540 edges have been added for these connections and 75 edges were added to connect the nodes representing the same location in different time periods. This leads to a time-space network with 100 nodes and a total of 615 edges. There are 3750 possible edges for this network, leading to a density of 16% for this network. The maximum degree is 8 for this network.

All the edge characteristics of the randomly created graphs are shown in Appendix C.1.

5.1.2 Dutch network

Another network that has been created is based on the Dutch railway network, which is displayed in Figure 9. The network has been simplified to 11 locations and is defined for 6 time periods. The only locations that have been used are the ones corresponding to Alkmaar, Amsterdam, Utrecht, Groningen, Zwolle, Arnhem, Hengelo, Rotterdam, Breda, Eindhoven and Maastricht. The displayed network is a planar graph, meaning that it can be drawn on a plane in such a way that its edges intersect only at their endpoints.

A higher cost and capacity are assigned to edges between bigger cities, Amsterdam, Utrecht and Rotterdam. The travel times are chosen between 1 and 3 and are based on the travel times between the cities. The used edge characteristics can be found in Appendix C.2.

The time-space network has 66 nodes and 191 edges. The total number of potential edges is 1815. So, the density for this network is 11%. The maximum degree for this network is 8. The time-space network is not planar anymore.

[5]Source: `https://www.eltis.org/discover/news/netherlands-develop-metro-urban-rail-system-handle-40-`

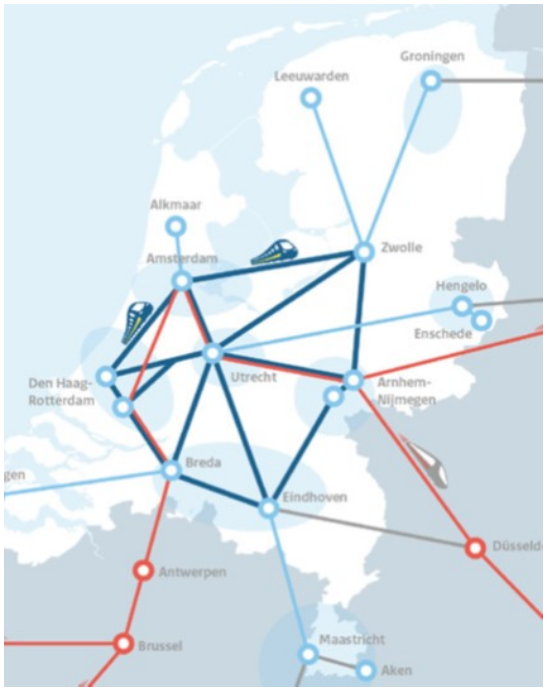


Figure 9: Dutch network[5]

5.1.3 Graph 17 and graph 30

Two other networks, graph 17 and graph 30, are based on the network described in the paper Miao and Ni (2019). This network is displayed in Figure 10. Graph 30 includes all the displayed locations in its network, while graph 17 only uses the first 17 locations. From Figure 10 can be seen that these networks are planar graphs.

The travel time of each arc is set to 1. Every transportation service departs in every time period. The red edges have been assigned a capacity of 2 and a cost of 1, while the other edges have been assigned a capacity of 1 and a cost of 2. These choices are made, because the red lines in the network correspond to train services and the black lines to transportation by truck. Multiple containers fit on a train, so the capacity is chosen relatively high. Also, the train is usually cheaper than the transportation by truck. Graph 17 is defined for 7 time periods, graph 30 for 9 time periods.

The time-space network for the graph with 17 locations has 119 nodes and 546 edges. The number of possible edges is 6069. This means that this network has a density of 9%. The maximum degree for this network is 7.

The graph with 30 locations has a time-space network of 270 nodes and 1408 edges. There are 32400 potential edges for this network. So, the density of this network is 4%. This network has a maximum degree of 9.

5.2 Container sets

For each instance, the shortest paths for each container can be determined. Consider the edges used in all the shortest paths. There is a capacity for each of these edges. A rate at which the capacity is exceeded by the shortest paths can be determined. This is done by calculating the excess of the edges used by the shortest paths divided by the total use of the edges within the shortest paths. For example, consider two containers with the shortest path for container 1 A, B, C and the shortest path for container 2 A, B, D and let the capacity for each edge be 1. The edge from A to B has an excess of 1 and the total use of edges is 4. The rate is then 25%. Let this rate be called the overlap. The three different container sets per network differ in overlap. The used container sets with their overlap can be seen in Appendix D.

The characteristics of the created networks are summarized in Table 1. For every network three different instances have been created. For every instance the optimal value has been determined with the PuLP modeler (version 2.4) and the default CBC solver (version 2.9.0). The optimal solution and corresponding solution value for each network can be found in Appendix E.

`percent-growth-public-transport`

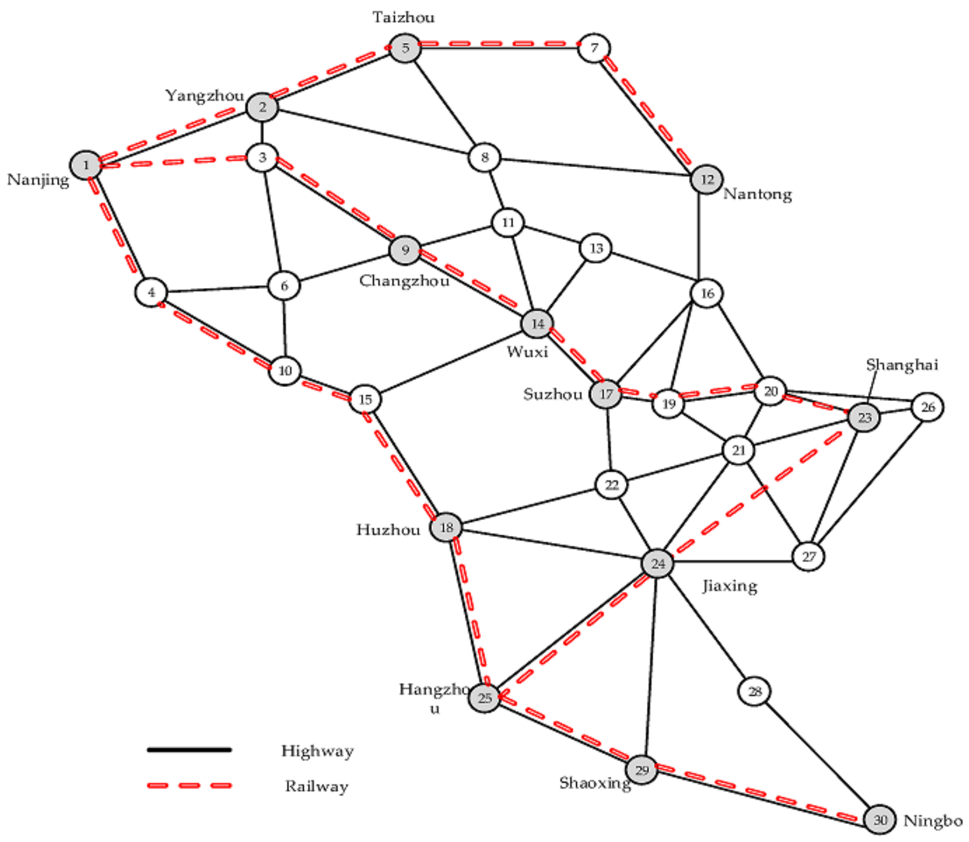


Figure 10: Transportation network 30 nodes

Table 1: Characteristics per network

Network	Nodes	Edges	Density	Maximum Degree
Random 15	60	285	21%	10
Random 20	80	480	20%	12
Random 25	100	615	16%	8
Dutch	66	191	11%	8
Graph 17	119	546	9%	7
Graph 30	270	1408	4%	9

6 Tuning

In Section 4, many different choices for the genetic algorithm steps have been presented. There are four different initialization methods, three different selection methods, three different cross-over methods and three different mutation methods. This results in a total of 108 different possible combinations, and thus 108 different settings of the genetic algorithm.

Thus, there are four different initialization methods. The first two initialization methods create the first population by choosing the first shortest path that can be found for every container. The last two methods choose randomly between all the found shortest paths per container. The first and third method continue according to the initialization method described in Section 4.2.2. So, a random node from the previous route is removed from the options for the next route. Method 2 and 4 use the initialization method that was explained in Section 4.2.3. Here, a random node is chosen that needs to be in the next route.

Furthermore, there are three different selection mechanisms. Three different cross-over methods have been introduced. Finally, there are three different mutation methods. An overview of where each method is described is given in Table 2. The sections corresponding to the initialization methods consider the initialization part after the first population has been created.

Table 2: Overview of the described methods

Method	Section
Initialization method 1	4.2.2
2	4.2.3
3	4.2.2
4	4.2.3
Selection method 1	4.4.1
2	4.4.2
3	4.4.3
Cross-over method 1	4.5.1
2	4.5.2
3	4.5.3
Mutation method 1	4.6.1
2	4.6.2
3	4.6.3

Next to multiple methods, also multiple parameters are used in the genetic algorithm. Important parameters for the genetic algorithm are the number of populations, the number of iterations, the number of pairs and the mutation rate. Additionally, the quantum annealer uses two parameters. The first parameter is the penalty value W_B, from $W_A \times H_A + W_B \times H_B$. The other parameter is the number of reads. The number of reads corresponds to the number states that have to be read from the solver.

The available time on the quantum annealer is limited. For this reason, the methods are chosen based on the simulated annealer, also provided by D-Wave. This simulated annealer has as advantage that the input problem also has to be in QUBO formulation and multiple solutions are returned. Also, for every solution that is returned by both annealers, not an objective value is given, but an energy level. This energy level represents both the objective value and the violation of constraints. Here the violation of constraints is measured by the slack variables from the QUBO formulation. The simulated annealer uses the same parameters as the quantum annealer.

In total, 18 instances have been created. Among these instances there are six instances based on the described graph 17 and graph 30. Graph 30 is quite a big instance and an extension of graph 17. For a run of the genetic algorithm with given parameters and methods, the computation time increases from 1173 to 3911 seconds by including the three instances corresponding to graph 30. So, the computation time increases significantly by including the graph 30 instances. To reduce the computation time, the methods are chosen and the parameters are tuned without including the three instances corresponding to graph 30.

Consider a single instance and let z^* denote the optimal objective value for this instance. Let z^H represent the found objective value by the genetic algorithm for this instance. The optimality gap can then be determined by Equation 10. The numerator in this equation is the absolute value of the difference between the values. The lower the optimality gap, the better the solution found by the genetic algorithm.

Unless indicated otherwise, all the optimality gaps are given in percentages. The optimality gap is a measure to determine the best methods and parameters for the genetic algorithm.

$$\text{Optimality Gap} = \frac{|z^H - z^*|}{z^*} \times 100\% \qquad (10)$$

First, the methods are chosen with the simulated annealer. The parameters are set to a certain value in this determination, because it becomes computationally intractable to tune the parameters for every combination of methods. Then, for the methods that are chosen, the parameters are tuned. The parameters are also tuned using the simulated annealer.

6.1 Determining the best methods

The parameters are set to a certain value for every combination of methods. Important parameters are the number of populations, the number of pairs and the number of iterations. It is reasonable to assume that a higher value for these parameters might lead to a better objective value against a higher computation time. The computation time of the genetic algorithm with given methods for 30 populations, 13 pairs and 20 iterations was 1751 seconds. There are 108 different methods. So, if these parameters were chosen, the computation time for all the different method combinations would have been around 52 hours. The values of the parameters are decreased to 26 populations, 12 pairs and 15 iterations, which gave a computation time of 1217 seconds for the same methods and parameters. From now on, all the computation times are given in seconds unless indicated otherwise.

Next to these parameters, there are some parameters within the annealing part of the algorithm. The annealing part consists of creating a QUBO model and solving it with the simulated annealer. The QUBO is of the form $min\ W_A \times H_A + W_B \times H_B$, where H_A represents the cost minimization and H_B the violation of constraints. W_A and W_B are objective and penalty weights respectively. The value of W_A is relative to the value of W_B. W_A is set to 1 and W_B is set to the sum of the cost of traveling by truck with every container. The value of W_B is tuned when the methods have been chosen. The current value is based on the fitness function and is meant to decrease the number of times the constraints are violated.

The problem that the simulated annealer solves is determining which containers travel by multimodal route and which containers travel by truck. The methods determine which multimodal routes are given as input for this problem and the problem itself does not change.

The simulated annealer has a number of reads and a number of steps. The number of reads corresponds to the number of output solutions that need to be read from the solver. The number of steps is how many steps are taken from a starting point to the found solution. The number of reads is set to 50 and the number of steps to the default value of 1000.

Another part that uses parameters is the mutation part. There is one mutation operator and two learning operators. For the mutation operator, the mutation rate is set to 10% per container. The first learning operator determines which containers often travel by multimodal route based on the best part of the returned solutions. There are four parameters for this learning operator. The first parameter is how many of the returned solutions are used to determine the number of times a route travels by multimodal route. The second parameter is the threshold that determines when a container is said to often travel by multimodal route. The third parameter is the mutation rate for containers that travel often by multimodal route. The last parameter is the mutation rate for containers that do not travel often by multimodal route. For the best 30 solutions it is calculated how often a container travels by multimodal route. The second parameter corresponds to the threshold and is set to 15. So, if in the best 30 solutions a container travels more than 15 times with the multimodal route, it is said that the multimodal route is selected often. The mutation rate for containers that travel often is set to 10% and the rate for containers that do not travel often is 20%. The second learning operator is applied or not. The chance of applying this operator is set to 10%.

When only a single combination of methods is considered, the optimality gap and computation time can be determined for every instance. An average optimality gap over these instances can be determined for every single combination of methods. Based on the optimality gaps for a single combination of methods, the standard deviation can be determined for this combination of methods as well. Next to the optimality gaps, the average computation time over the instances can also be computed for a single method combination. These values can be found in Appendix F for every combination of methods.

The combination of methods with the best average optimality gaps have been summarized in Table 3. The methods column represents which combination of methods was used. Methods 1, 1, 1, 1 means that the first initialization, the first selection, the first cross-over and the first mutation methods have been used. From Table 3 can be seen that the combination of the first initialization method, the first selection method, the third cross-over operator and the third mutation operator leads to the lowest average optimality gap. The corresponding standard deviation is not much higher than the standard deviation of the combination of methods with a slightly worse average optimality gap. The combination of the second initialization method, the first selection method, the third cross-over method and the third mutation method has the lowest standard deviation, but a higher average optimality gap. Furthermore, the computation times are comparable to each other.

Table 3: Combination of methods with best optimality gaps

Methods	Average optimality gap(%) (standard deviation)	Average computation time(s)
1, 1, 3, 3	27 (18)	53.60
2, 1, 3, 3	29 (16)	53.39
3, 1, 2, 3	29 (17)	52.94
4, 1, 2, 3	29 (17)	53.07
1, 1, 2, 3	29 (22)	52.42
1, 2, 3, 3	29 (23)	54.37

So, the (1, 1, 3, 3) methods lead to the lowest average optimality gap. This corresponds to the first initialization method, the first selection method, the third cross-over method and the third mutation method. Furthermore, the standard deviation is not much higher than the combination of methods with the lowest standard deviation. So, the first initialization method, the first selection method, the third cross-over method and the third mutation method are selected as best methods.

6.2 Parameter tuning

The first initialization method, the first selection method, the third cross-over method and the third mutation method have been chosen. Now, the parameters can be tuned. These parameters are also tuned using the simulated annealer. The parameters that need to be tuned are the number of populations, the number of pairs, the number of iterations, the W_B value, the number of reads and the mutation rate. First, the number of populations, the number of pairs and the number of iterations are examined. Then, the parameters used in the simulated annealer are tuned. Finally, the mutation rate is determined.

A higher number of iterations and a higher number of populations lead to a better solution against a higher computation time. Different values are tried for these parameters to show this with the simulated annealer. Also, it needs to be determined how many pairs have to be created to get the best result.

Three different numbers of populations are used, namely 20, 30 and 40 populations. For every number of populations, three different number of pairs have been tried. The number of populations selected in the pairs are based on approximately 50%, 75% and 100% of the number of populations. For 20 populations, 5, 8 and 10 pairs were made. This was 7, 11 and 15 for a total of 30 populations. For 40 populations, 10, 15 and 20 pairs were made. For every combination of number of populations and number of pairs three different numbers of iterations have been tried, these values were 15, 20 and 25 iterations. The average optimality gaps, the standard deviation of the optimality gaps and the average computation time per combination of these parameters can be found in Appendix G.

For a given number of populations and a given number of pairs applies that an increased number of iterations leads to a better optimality gap with a better standard deviation. This improvement can be made against a higher computation time. So, a trade-off has to be made. For every combination of given number of populations and number of pairs, this can be shown by comparing the different values for the number of iterations. Consider two randomly chosen number of populations and two randomly chosen number of pairs. Let 20 and 40 be the randomly chosen number of populations and 5 and 10 the randomly chosen number of pairs. The average optimality gap for $n_{pop} = 20, n_{pairs} = 5$ and $n_{pop} = 40, n_{pairs} = 10$ for every number of iterations value is given in Table 4. It can be seen that as the number of iterations increases, the average optimality gap and the corresponding standard deviation decreases against a higher average computation time.

For a given number of pairs, an average can be computed over the performance measures for the different number of populations and iterations. These values are shown in Table 5, and are thus based on the average over different values for n_{pop} and n_{iter}. The number of pairs are chosen relative to the number of populations, so 50% means that half of the population is chosen for a cross-over operation. It

Table 4: Comparing the number of iterations

n_{pop}, n_{pairs}, n_{iter}	Average optimality gap(%) (standard deviation)	Average computation time(s)
20, 5, 15	32 (24)	21.90
20, 5, 20	30 (23)	26.08
20, 5, 25	29 (23)	29.98
40, 10, 15	30 (22)	46.16
40, 10, 20	28 (22)	58.22
40, 10, 25	24 (16)	69.16

can be seen that an increased number of pairs leads to a better optimality gap with a higher computation time no matter the number of populations and the number of iterations.

Table 5: Averages for different number of pairs

Relative number of pairs	Optimality gap(%) (standard deviation)	Computation time(s)
50%	29 (22)	40.99
75%	27 (21)	61.70
100%	26 (19)	81.70

However, when comparing the number of pairs relative to the number of populations it might also happen that fewer pairs give a better optimality gap and better computation times. Consider for example 40 populations and 15 iterations, the optimality gap and computation times for the different number of pairs are displayed in Table 6. The table shows that having 15 pairs instead of 20 leads to a better optimality gap in terms of average and standard deviation. The computation time is also better for 15 pairs instead of 20 pairs. An explanation for this is that the elitism selection mechanism is used. In this mechanism, the best $n_{pairs} \times 2$ populations are selected for a cross-over operation. The actual pairs are made randomly. So, when more pairs are made, more populations are selected for the cross-over operation. This means that pairs are made with populations with a worse fitness value. However, including some populations with a worse fitness value lead to a more diversity, which is an explanation for why having an intermediate number of pairs is the best solution.

Table 6: Optimality gap and computation time for $n_{pop} = 40,\ n_{iter} = 15$

n_{pairs}	Average optimality gap(%) (standard deviation)	Average computation time(s)
10	30 (22)	46.16
15	26 (16)	65.18
20	27 (19)	86.20

The n_{pop}, n_{iter} and n_{pairs} values with the best average optimality gaps are shown in Table 7. The optimality gaps lie closely together considering the average and the standard deviation. However, there are two values that are the best in terms of these measures. The combinations of having 40 populations, 25 iterations and 15 or 20 pairs lead to an average optimality gap of 23 with a standard deviation of 15. However, the computation time for the combination with 20 pairs is higher. This means that of all the tried combinations the one with the most populations, the most iterations and an intermediate number of pairs is the best.

Table 7: Comparing the number of populations, the number of pairs and the number of iterations

n_{pop}, n_{pairs}, n_{iter}	Average optimality gap(%) (standard deviation)	Average computation time(s)
30, 15, 25	23 (17)	99.33
40, 10, 25	24 (16)	69.16
40, 15, 20	24 (16)	83.09
40, 15, 25	23 (15)	100.46
40, 20, 25	23 (15)	135.73

So, the higher the number of populations and the higher the number of iterations, the better the solution. For a number of pairs of 75% of the populations a good solution is given with a lower computation time than when a number of pairs of 100% is chosen.

The number of pairs is set to 75% of the number of populations. The number of populations and the number of iterations are based on a reasonable quantum annealer time. The simulated annealer is used to estimate the used quantum annealer time. Using 40 populations, 15 pairs and 30 iterations leads to 18,206 simulated annealer runs when all the instances are considered. The actual quantum annealer time differs per instance and is at most 0.05 seconds for a number reads of 100. This means that for 40 populations, 15 pairs and 30 iterations the used quantum time is at most $18,206 \times 0.05 = 910.3$ seconds, which corresponds to a bit more than 15 minutes to find a solution to all the instances.

The simulated and quantum annealer determine which containers travel by a given multimodal route and which travel by truck. This problem remains the same, no matter the values of the other parameters. That is why the parameters used in the annealers can be tuned separately. The parameters used in both the quantum and the simulated annealer are the number of reads and the value of W_B from the QUBO $\min W_A \times H_A + W_B \times H_B$.

The annealer is used multiple times within the genetic algorithm. This can either be the simulated or the quantum annealer. An optimality gap can be determined by comparing the best solution returned by the annealer and the optimal solution to the problem that was solved by the annealer. The quality of the best solution returned by an annealer is dependent on the value of this solution and whether it is feasible. The solution returned by the annealer is infeasible when the capacity is exceeded. The optimality gap can be determined using the fitness value of the best solution that is returned by the annealer. This means that, if the returned solution is infeasible a penalty is added to the found solution value and this value is compared to the optimal solution value. Since the annealer is used multiple times within the algorithm, an average gap can be determined for every instances.

Next to the optimality gap, timing is also important when measuring the quality of an algorithm. The annealing part of the genetic algorithm consists of three parts. The first part considers the creation of the QUBO model. The second part solves the QUBO model. Finally, every returned solution needs to processed to retrieve the needed values. For example, the objective value needs to be determined from the solution that was returned by the annealer. Let the total time the annealing part takes be denoted as the anneal time and the time that the annealer uses to actually solve the problem as the annealer time. The annealing part is described in Figure 11.

$$\begin{tikzcd}\boxed{\text{Construct QUBO}} \arrow[r] & \boxed{\text{Annealer part: Solve QUBO}} \arrow[r] & \boxed{\text{Post-process solution}}\end{tikzcd}$$

Figure 11: Annealing part

How much computation time the annealer part takes is depending on the number of reads value, because the number of reads determines the number of output samples from the solver and more sampling means more computation time. This is both for the simulated and the quantum annealer. Different values for the number of reads have been tried for the simulated annealer. For every instance an average optimality gap can be determined for a given number of reads. Additionally, an average can be determined by taking the average over the instances. These values are displayed in Table 8. The best optimality gap is attained for 100 number of reads. It appears that having a higher number of reads does not lead to a better optimality gap.

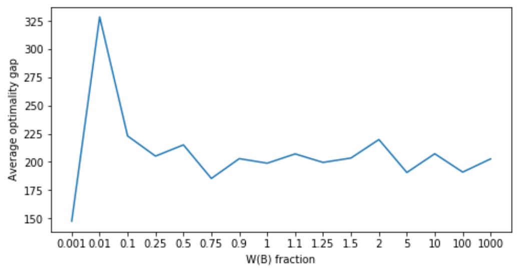
Table 8: Averages for number of reads tune

Number of reads	Average optimality gap(%) (st dev)	Anneal time(s)	Annealer time(s)
10	150 (1.18)	0.07	0.02
50	120 (1.21)	0.14	0.07
100	113 (1.2)	0.18	0.10
300	123 (1.21)	0.40	0.29
500	119 (1.21)	0.61	0.47
800	113 (1.21)	0.84	0.68

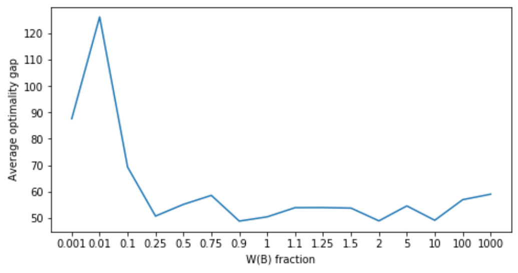
For the simulated annealer, the genetic algorithm has been applied for different values of W_B. The value of W_B is taken as a portion of the sum of the costs of traveling by truck with every container. This sum is different for every instance, so by taking a ratio the actual W_B value differs per instance. The average optimality gaps have been displayed for different W_B values per instance in Appendix H. The average optimality gap is determined by adding a penalty value to the found solution value if the returned solution is infeasible.

To get a better view of the results for the different W_B values, a plot has been made per instance. These can also be found in Appendix H. An interesting observation is that the graph has a similar form for quite some instances. An example of such an instance is the random network with 15 nodes and the third container set. The corresponding plot is displayed in Figure 12a. It can be seen that the average optimality gap for 0.001% percent of the sum of the costs of traveling by truck gives the best value. Then, there is a quite high value for 0.01% and then it seems to stabilize at an average optimality gap of around the 200%.

Another instance with such a plot is the instance of the dutch network with the third container set. The average optimality gaps are plotted against the different W_B rate values for this instance in Figure 12b. Now, the value corresponding to the 0.001% fraction is higher than the average optimality where it converges to as the W_B rate increases. The average optimality goes to a value around the 50%.



(a) Random 15, 3



(b) Dutch, 3

Figure 12: Parameter tune of the W_B value

The instances that have such a shape are the random network with 15 nodes and the third container set, the random network with 20 nodes and the third container set, the random network with 25 nodes and the third container set, the dutch network with the third container set, all the instances corresponding to the graph 17 network and the graph 30 network with the first and the third container set.

Another remarkable observation is the average optimality gap for the rate of 0.01% of the sum of the costs of traveling by truck. The average optimality gap for this rate lies high relative to the other rates for all the instances with exception of the random network with 25 nodes and the first container set. Furthermore, this rate corresponds to the worst average optimality gap of almost all the instances. The only instances where the worst average optimality gap corresponds to another value are the random network with 25 nodes and the first container set, the random network with 15 nodes and the second container set and the dutch network with the first container set.

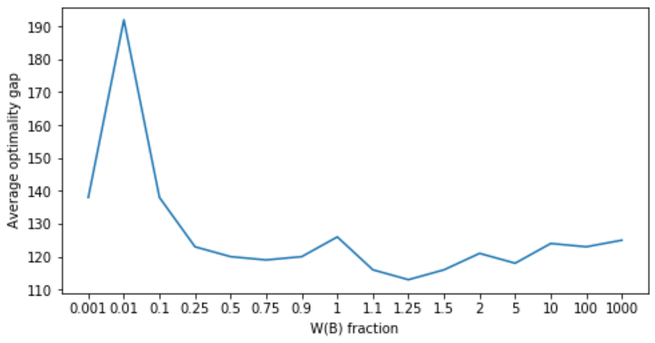


Figure 13: Average optimality gap for multiple W_B values

To pick a single W_B rate that suits every instance the average can be taken over all the instances. This leads to the values displayed in Table 9. When the average optimality gap is plotted against the different W_B values, Figure 13 is created. The best optimality gap is attained for a W_B rate of 1.25 times the sum of the costs of traveling by truck. This value is used from now on in both the simulated and quantum annealer.

The last parameter that needs to be tuned is the mutation rate. This rate represents the probability that a population is subject to change after the cross-over operation is performed. To find the best

Table 9: Average W_B values

W_B rate	Average optimality gap(%) (st dev)	Anneal time(s)	Annealer time(s)
0.001	138 (0.32)	0.16	0.08
0.01	192 (0.45)	0.16	0.09
0.1	138 (1.20)	0.18	0.11
0.25	123 (1.21)	0.17	0.10
0.5	120 (1.20)	0.17	0.10
0.75	119 (1.20)	0.17	0.10
0.9	120 (1.20)	0.17	0.10
1	126 (1.20)	0.17	0.10
1.1	116 (1.20)	0.17	0.10
1.25	113 (1.20)	0.16	0.10
1.5	116 (1.19)	0.17	0.10
2	121 (1.20)	0.16	0.10
5	118 (1.20)	0.16	0.09
10	124 (1.20)	0.16	0.09
100	123 (1.19)	0.15	0.09
1000	125 (1.22)	0.15	0.08

mutation rate multiple values have been tried for the genetic algorithm that uses the simulated annealer. The number of populations is set to 40, the number of pairs to 30, the number of iterations to 30, the number of reads is set to 100 and the W_B value is set to 1.25 times the costs of traveling by truck with every container.

For every instance and every option for the mutation rate, an optimality gap can be determined. For every mutation rate, an average optimality gap can be determined by taking the average over the instances. This average is not given in percentage. This average is displayed for the different mutation rates in Figure 14. The lowest average optimality gap is attained for the mutation rates 10%, 12.5% and 15%.

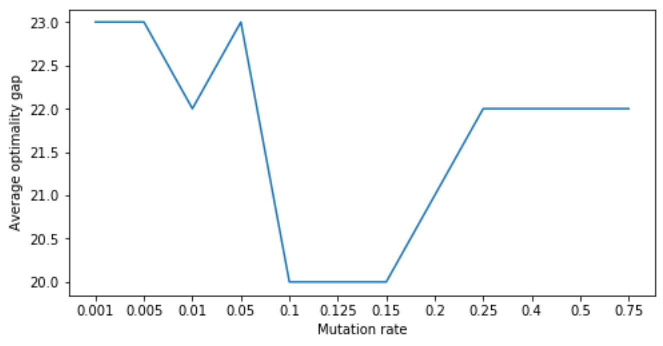


Figure 14: Mutation rate tune

Based on the optimality gaps, the standard deviation can also be calculated. Additionally, an average computation time can be determined. These values are displayed in Table 10. As was clear from Figure 14, the best average optimality gap is attained for the mutation rates 10%, 12.5% and 15%. The standard deviation is the smallest for the mutation rate of 10%. Furthermore, the average computation time increases when the mutation rate increases. So, $mutRate = 0.1$ gives the best mutation rate with the lowest standard deviation and lowest computation time.

So, the first initialization method, first selection method, third cross-over method and third mutation methods have been chosen. The number of pairs is set to 75% of the number of populations. A higher number of iterations and a higher number of populations lead to a better average optimality gap. The number of iterations is set to 30 and the number of populations is set to 40, such that the quantum annealer can find a solution for every instance without using too much time. This means that $n_{pairs} = 30$. The parameters used in the quantum and simulated annealer are set to the same value. The number of reads parameter is set to 100. The value for W_B is set to 1.25 times the sum of the cost of traveling by truck

Table 10: Average optimality gaps for different mutation rates

mutRate	Average optimality gap(%) (standard deviation)	Average computation time(s)
0.001	23 (18)	306.96
0.005	23 (15)	305.06
0.01	22 (16)	313.11
0.05	23 (14)	319.18
0.1	20 (13)	334.91
0.125	20 (15)	351.71
0.15	20 (14)	361.39
0.2	21 (14)	378.21
0.25	22 (18)	387.46
0.4	22 (19)	430.85
0.5	22 (16)	459.65
0.75	22 (19)	531.60

with every container. The mutation rate is set to 10%.

7 Results

The genetic algorithm uses the initialization method where the first population is created by using the first found shortest paths. The rest of the initial populations is created according to the first initialization method, which was described in Section 4.2.2. The first selection procedure is used. This procedure is called elitism and was explained in Section 4.4.1. The partial cross-over operator is used. This operator is the third cross-over method and was described in Section 4.5.3. The genetic algorithm uses the second learning operator, which was explained in Section 4.6.3.

The number of populations is set to 40, the number of pairs to 30, the number of iterations to 30 and the mutation rate to 10%. When an annealer is used, the value of W_B is set to 1.25 times the cost of traveling by truck with every container. The number of reads used by an annealer is set to 100. Unless indicated otherwise, the computation times are displayed in seconds.

To compare the results of the genetic algorithm when the quantum annealer is used, the genetic algorithm is also used in combination with the simulated annealer and in combination with the optimal solution to the problem that is usually solved with an annealer. First, the genetic algorithm is used in combination with the simulated annealer. Here, the results of the genetic algorithm are discussed and the performance of the simulated annealer is evaluated. Secondly, the results of the genetic algorithm that uses the optimal solution to determine which containers travel by truck and which by multimodal route are shown. Then, the results are displayed for the genetic algorithm that is used in combination with the quantum annealer. Again, the performance of the quantum annealer is shown. Afterwards, the parameters n_{pop}, n_{pairs} and n_{iter} are increased for the genetic algorithm that uses the optimal solution and the corresponding results are displayed. These parameters are increased to gain insight in the performance of the genetic algorithm when these parameters are less restricted by the limited available quantum time. Finally, the genetic algorithm is compared for the different methods that have been used to determine which container travels by truck and which by multimodal route.

7.1 Results using the simulated annealer

The found solutions for the genetic algorithm in combination with the simulated annealer, are given in Appendix J.1. The results are summarized in Table 11. The column with the best value describes the objective value of the solution that was found by the genetic algorithm. The optimality gap is displayed in the next column. This value compares the best value to the optimal solution. The iteration column describes in which iteration the best value was found. The column that displays the best initial value describes the best found objective value after the initialization part. The time column displays the total time that the genetic algorithm used to find the solution. This time includes the time it takes to find the optimal solution for every problem that is solved with the simulated annealer. This optimal value is used to measure the quality of the annealer by determining the optimality gap.

In Table 11, it can be seen that the solutions returned by the genetic algorithm differ quite a lot in terms of quality. The optimality gap ranges from 0 to 39%. The average optimality gap is 20%. The

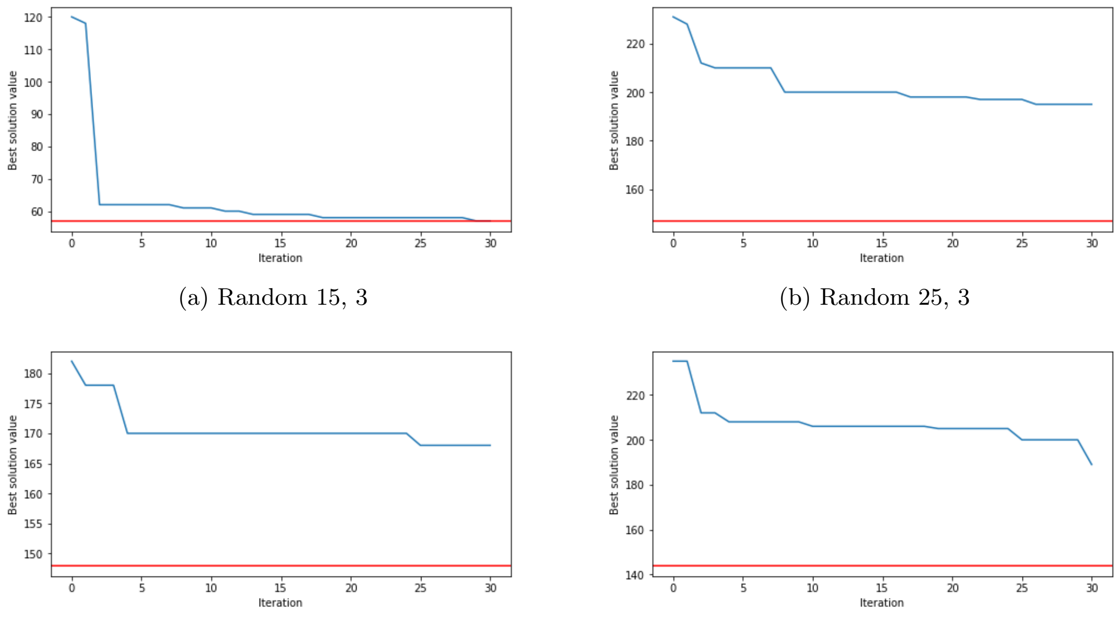
Table 11: Results genetic algorithm using the simulated annealer

Instance	Best value	Optimality gap(%)	Iteration	Best initial value	Time(s)
Random 15, 1	59	2	14	465	288.51
Random 15, 2	61	0	14	130	396.59
Random 15, 3	57	0	29	120	312.85
Random 20, 1	94	24	21	158	310.12
Random 20, 2	97	39	19	106	280.60
Random 20, 3	133	36	23	233	345.18
Random 25, 1	101	17	22	191	296.95
Random 25, 2	153	35	16	201	332.40
Random 25, 3	195	33	26	231	395.59
Dutch, 1	119	24	12	147	289.81
Dutch, 2	121	15	28	158	330.22
Dutch, 3	168	14	25	182	363.93
Graph 17, 1	167	16	25	230	417.63
Graph 17, 2	189	31	30	235	432.40
Graph 17, 3	234	12	29	278	463.16
Graph 30, 1	447	18	26	479	976.64
Graph 30, 2	782	28	19	811	2438.78
Graph 30, 3	808	14	23	825	2287.04

optimal solution is found for the instances random 15, 2 and random 15, 3. The random network with 15 locations and the first container set has a relatively low optimality gap of 2%. The instances with a relatively high optimality gap are: random 20, 2; random 20, 3; random 25, 2, random 25, 3 and graph 17, 2. The best initial value for the random network with 15 locations and the first container set is a lot higher than the corresponding best found value. Furthermore, this value is higher than the sum of costs of traveling by truck with every container. This means that the simulated annealer fails to find a feasible solution to every problem that is solved by the annealer in the initialization part of the genetic algorithm for this instance. This considers the performance of the simulated annealer and is discussed later on. Also, it can be seen that the difference between the best initial value and the best found value is relatively small for the instances corresponding to graph 30. Additionally, the computation times for the graph 30 are higher than the other computation times. For every instance, except the instances corresponding to the graph 30 network, 20 containers are used in the container sets. The first container set for graph 30 contains 30 containers and the last two have 50 containers. This is an explanation for the increased computation time for these instances. Another observation considers the iteration in which the best found value is attained. For quite some instances, the best found value is attained in an iteration relatively late in the algorithm.

How the solution value found by the genetic algorithm develops, can be shown by plotting the best found value in every iteration. The plot is made per instance and can be found in Appendix J.2. The optimal value is also displayed in this plot. A few of these plots are shown in Figure 15. In Figure 15a, the course of the algorithm is displayed for the random network with 15 locations and the third container set. The algorithm starts out with a high objective value. Then, a large improvement is found. The algorithm keeps finding a better value, until the optimum is found in iteration 29. The plot in Figure 15b shows the development of the algorithm for the random 25, 3 instance. The algorithm does not find large improvements for this instance. However, the objective value keeps improving with the iterations. Even so, the optimal solution is not found in the end. In Figure 41, the course of the genetic algorithm for the dutch network with the third container set is displayed. Here, a large improvement is made in the beginning. Thereafter, the objective value remains the same for a large number of iterations. Then, more in the end, the objective value improves again. The plot in Figure 15d shows the course of the algorithm for the graph 17, 2 instance. In the beginning of the algorithm and in the end, a relative large improvement is found. In between, the objective value also improves, but this improvement is less within a single iteration. Also, in the final iteration an improvement is still made.

The genetic algorithm gradually improves with the number of iterations for multiple instances, as was indicated by Figure 15. This might be because an improvement within the neighbourhood. So, a solution is found that is closer to the local optimum. Or, this improvement is made, because a local optimum is escaped. An indication of being stuck in a local optimum is when the solution value does not improve for a while. That the solution improves after a while, is an indication that the local optimum is



(a) Random 15, 3

(b) Random 25, 3

(c) Dutch, 3

(d) Graph 17, 2

Figure 15: Course of the genetic algorithm with the simulated annealer

escaped. This was the case for the Dutch, 3 instance. Having an improvement in an iteration just before termination is a sign that the algorithm has not converged to its best value yet, which was seen for the graph 17, 2 instance.

Multiple performance measures can be used to determine the quality of the used annealer. This can either be the simulated or the quantum annealer. The annealer is used multiple times per instance within the genetic algorithm. The measures to determine the quality of the annealer, only consider how well the annealer solves the problems that are sent to it not the impact on the final solution value that is returned by the genetic algorithm. The measures consider the best solution that was returned by the annealer, as the annealer returns multiple solutions every time it is called upon. The first measure is the percentage of solutions returned by the annealer that are different than the optimal solution. So, the percentage of not optimal solutions is given in the column denoted "Wrong". Secondly, the average optimality gap can be determined over all the problems that the annealer has solved for a single instance. To compute the optimality gap, the optimal solution is determined every time the annealer solves a problem. In this optimality gap, a penalty is included if the returned solution is infeasible. The standard deviation is displayed between the brackets. Additionally, a percentage of infeasible solutions can be determined. Thus, the number of times the best solution is infeasible is divided by the total number of times that the annealer is used. The column that displays the feasible gap considers the optimality gap when the best returned solution is feasible. Again, the standard deviation is displayed in brackets. This percentage represents the quality of the solution returned by the annealer, given that the solution is feasible. The column with the anneal time shows the average time it takes from having a combination of routes to an actual solution. The annealer time displays the average time it takes the annealer to solve the problem. The values for the measures considering the simulated annealer are displayed in Table 12.

From Table 12, it can be seen that, the percentage of wrong solutions is quite high for most of the instances. For more than half of the instances it holds that the optimal solution to the annealer problem is not found once by the simulated annealer. The instances with the lowest percentage of wrong solutions are all the instances corresponding to the random network with 15 locations, the random 20, 2 instance and the random 25, 1 instance. Remarkable is that, the average optimality gap for these instances is quite high. An explanation might be the relatively high standard deviation for these instances. A high standard deviation means that the difference between returned solution values is high. So, even though the optimalilty gap is zero sometimes, the optimality gap is very different from zero in other times. Furthermore, for every instance, multiple problems were sent to the annealer for which no feasible solution was found, as can be seen in the infeasible column. For the instance corresponding to the graph 30 network with the second container set, even applies that 84.56% of the solutions are not feasible. Another instance with a relatively high infeasible fraction is the instance random 15, 2. On the other hand, the random 25, 3 and graph 17, 3 instances have a relatively low infeasible percentage. The optimality gap when only the feasible solutions are considered lies between 7 and 21%. The random 15, 1 and the graph 30, 3 instances have the lowest feasible gap. The random 25, 3, the Dutch, 1 and the Dutch, 2 have the highest feasible gaps. Additionally, the average optimality gap including a penalty for infeasible solutions lies between 31 and 232 percent. The random 25, 3, the Dutch, 3 and the graph 17, 3 instances have the lowest average optimality gap. The instances corresponding to the random network with 15 locations and the second and third container sets have the highest optimality gap.

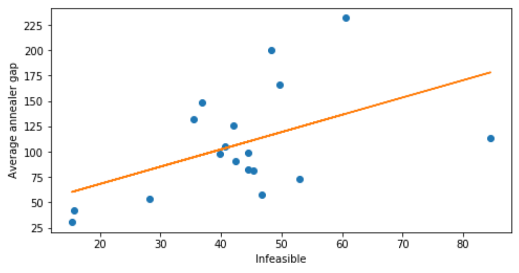
Table 12: Performance of the simulated annealer

Instance	Wrong(%)	Optimality gap(%)	Infeasible(%)	Feasible gap(%)	Anneal time(s)	Annealer time(s)
Random 15, 1	40.94	149 (2.00)	36.83	7 (0.09)	0.11	0.08
Random 15, 2	73.49	232 (1.93)	60.60	12 (0.11)	0.17	0.11
Random 15, 3	83.40	200 (2.06)	48.24	12 (0.10)	0.13	0.09
Random 20, 1	100.00	166 (1.58)	49.70	14 (0.09)	0.14	0.09
Random 20, 2	84.19	132 (1.71)	35.40	11 (0.08)	0.15	0.10
Random 20, 3	99.61	91 (0.92)	42.47	14 (0.06)	0.16	0.10
Random 25, 1	68.11	126 (1.46)	42.11	14 (0.09)	0.15	0.10
Random 25, 2	100.00	105 (1.08)	40.76	16 (0.07)	0.16	0.10
Random 25, 3	100.00	42 (0.59)	15.72	17 (0.06)	0.21	0.12
Dutch, 1	100.00	98 (1.01)	39.89	17 (0.06)	0.14	0.09
Dutch, 2	100.00	99 (0.88)	44.58	21 (0.07)	0.16	0.09
Dutch, 3	99.90	54 (0.64)	28.21	14 (0.06)	0.18	0.10
Graph 17, 1	100.00	83 (0.81)	44.44	11 (0.05)	0.23	0.12
Graph 17, 2	99.51	81 (0.77)	45.33	11 (0.05)	0.24	0.12
Graph 17, 3	100.00	31 (0.50)	15.36	10 (0.04)	0.28	0.14
Graph 30, 1	100.00	73 (0.60)	52.91	9 (0.03)	0.72	0.29
Graph 30, 2	100.00	113 (0.43)	84.56	14 (0.03)	2.07	0.59
Graph 30, 3	100.00	58 (0.55)	46.69	7 (0.02)	1.93	0.61

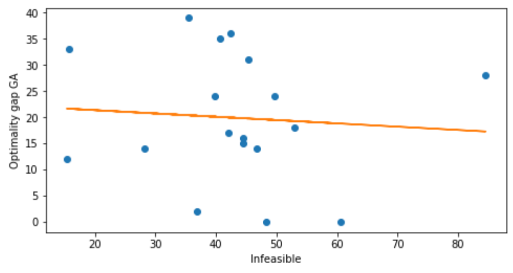
An explanation for the fact that the simulated annealer finds it hard to find feasible solutions is the required equality constraint. Slack variables have been added to ensure that the capacity of the arcs is not exceeded. However, the value of these variables also need to be determined by the annealer. So, if the slack variables do not contain the right value, it might happen that the capacity of an arc is exceeded while the annealer thinks it is not exceeded. This is also an issue the other way around. Hence, if the slack variables imply that the capacity is exceeded while this is not the case, the solution will be considered as a bad solution.

Some interesting observations can be made from Figure 16. Here, different performance measures of the simulated annealer are plotted against each other. Also, a regression line is plotted to indicate a relation between the performance measures. In Figure 16a, the infeasibility rate is plotted against the average annealer optimality gap. It can be seen that, the average optimality gap for the simulated annealer increases as the infeasible rate increases. This is as expected, because a penalty is added to the optimality gap if the returned solution is infeasible. However, when the infeasible rate is plotted against the optimality gap of the genetic algorithm in Figure 16b, a different relation emerges. It appears that as the infeasibility rate increases, the optimality gap of the genetic algorithm decreases. However, this is only a small negative correlation, as the data are far apart from each other. In the Figure 16c, the fraction of wrong solutions is plotted against the optimality gap of the genetic algorithm. It can be seen that, as the fraction of wrong solutions returned by the annealer increases, the optimality gap of the genetic algorithm also increases. However, for most instances, 100% of the solutions returned by the annealer are not equal to the optimal solution. This means that the shown correlation is based on only a few data points. Nonetheless, it appears that if the annealer occasionally returns the optimal solution, the optimality gap of the genetic algorithm decreases. Another remarkable relation is shown in Figure 16d. It appears that, as the average annealer gap increases, the optimality gap of the genetic algorithm in combination with the simulated annealer decreases.

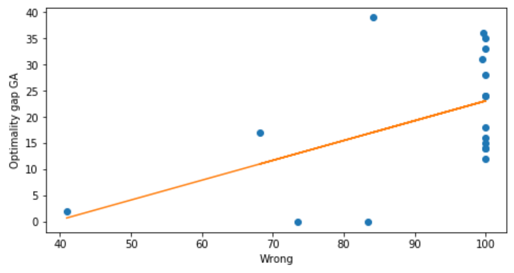
The solution to the problem that is solved with the simulated annealer is usually not the optimal solution, it is difficult to determine the quality of a population. The results of the genetic algorithm are dependent on this quality and it raises the question if a good solution to the entire problem can be found, if the fitness value is very different from the optimal solution. As the average optimality gaps for the problems sent to the simulated annealer are quite high, the quality of the population is wrongly



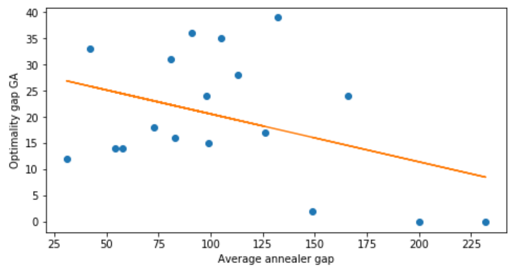
(a) Infeasible against the annealer gap



(b) Infeasible against the genetic algorithm gap



(c) Wrong against the genetic algorithm gap



(d) Annealer gap against the genetic algorithm gap

Figure 16: Performance of the simulated annealer

represented by the returned fitness value. Only for the instances corresponding to the random network with 15 locations this does not seem to be a problem. However, the simulated annealer occasionally returns the optimal solution for these instances, as is shown by the fraction of wrong solutions.

So, using the simulated annealer in the genetic algorithm gives an optimality gap between 0 and 39 percent. The average optimality gap is 20%. The instances for which the lowest optimality gap was achieved are: random 15, 1; random 15, 2 and random 15, 3. What is remarkable is that all the instances corresponding to the random network with 15 locations have a relatively low percentage of wrong solutions. Furthermore, the random 15, 1 instance has one of the lowest feasible gaps. However, the instances random 15, 2 and random 15, 3 have the highest average optimality gaps for the problems that are solved with the annealer. Additionally, random 15, 2 has a high infeasible rate. This is an explanation for the high average optimality gap of this instance. Moreover, the instances with the lowest average optimality gaps to the problem that is solved with the annealer are: graph 17, 3; random 25, 3; Dutch, 3 and graph 30, 3. The optimality gaps of the value found by the genetic algorithm are: 12; 33; 14 and 14 percent respectively, for these instances. With the exception of the random 25, 3 instance, these gaps are the lowest after the instances corresponding to the random network with 15 locations. Thus, even though the relation between the average annealer optimality gap and the optimal gap for the genetic algorithm appears negative, most of the instances with the a low average annealer optimality gap have a relatively low optimality gap for the entire genetic algorithm. However, the instances corresponding to the random network with 15 locations have the lowest optimal gap in the genetic algorithm and one of the highest average optimality gaps of the problem that is solved with the annealer. So, having a high average optimality gap for the problem that is solved with the simulated annealer does not necessarily mean a high optimality gap for the genetic algorithm, as was shown in Figure 16d. However, if an instance has a low average annealer optimality gap, this instance is likely to have a below average optimal gap for the best found value of the genetic algorithm.

7.2 Results using the optimal solution

The results of the genetic algorithm that uses the optimal solution to the problem that is otherwise send to an annealer are displayed in Table 13. The optimal solution to this problem is determined with the PuLP modeler (version 2.4). The solver that is used, is the default CBC solver (version 2.9.0). The found solutions are given in Appendix K.1. The optimal time column contains the time it takes to determine the optimal solution to the problem that is usually solved by an annealer. What stands out in the table is that, the optimality gap for more than half of the instances is 0. However, the optimality gap corresponding to the random network with 20 locations and the third container set is a lot higher than the rest. The optimality gap for this instance is 26% and the second highest optimality gap is 8% for the instance that uses the graph 17 network with the first container set. The average optimality gap is 3%. It is remarkable that, the optimal value has not been found for any of the instances corresponding to the graph 30 network. Furthermore, the best solution value for these instances is found in a late iteration of the algorithm, as can be seen in the iteration column. The same applies for the other instances where the optimal solution has not been found. It is interesting to see that, the optimal solution to the Dutch, 1 and Dutch, 3 instances were found in the beginning of the algorithm. The computation times are comparable to each other. However, the graph 30, 2 and graph 30, 3 instances do take a bit more time than the other instances. This can be explained through the bigger container sets. Additionally, the time it takes to solve the problem that is usually solved with the annealer is a large part of the total computation time.

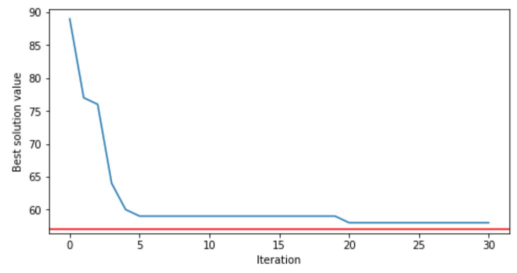
Table 13: Results genetic algorithm using the optimal solution

Instance	Best value	Optimality gap(%)	Iteration	Best initial value	Time(s)	Optimal time(s)
Random 15, 1	58	0	16	80	174.53	171.85
Random 15, 2	61	0	11	92	241.70	238.49
Random 15, 3	58	2	20	89	231.98	228.19
Random 20, 1	76	0	17	99	204.25	200.04
Random 20, 2	70	0	16	88	193.47	187.33
Random 20, 3	123	26	22	196	235.56	228.99
Random 25, 1	86	0	13	119	207.57	198.80
Random 25, 2	113	0	7	156	228.84	218.56
Random 25, 3	156	6	18	213	268.81	259.19
Dutch, 1	96	0	1	116	265.16	260.45
Dutch, 2	105	0	11	137	228.77	225.42
Dutch, 3	148	0	2	152	206.90	202.90
Graph 17, 1	155	8	24	209	199.35	193.88
Graph 17, 2	150	4	23	208	182.41	176.69
Graph 17, 3	209	0	15	246	177.31	170.89
Graph 30, 1	388	2	25	420	228.25	203.27
Graph 30, 2	634	4	28	697	304.22	254.64
Graph 30, 3	727	3	24	751	292.78	242.97

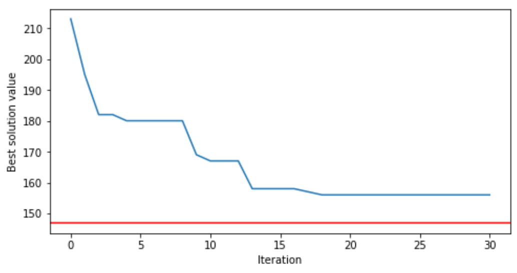
How the best solution value develops during the algorithm is shown per instance in Appendix K.2. Some of these plots are shown Figure 17. Figure 17a displays the development of the best solution value throughout the algorithm for the random 15, 3 instance. Here, two large improvements are found in the first 3 iterations. Thereafter, the solution value improves a bit until for numerous iterations no improvement has been found. However, in iteration 20 an improvement is made again. In Figure 17b, the course of the solution value during the algorithm is shown of the random 25, 3 instance. In some iterations a relatively large improvement is made, while in other iterations this improvement is a bit smaller. In the end, no improvement is made anymore. The plot in Figure 17c presents the development of the solution value for the graph 17, 2 instance. In the first ten iterations, the made improvements are relatively large. Then, for a while no improvements are made again, until a small improvement is made. Thereafter, multiple improvements are made again until the solution value goes to 150 and remains there. Figure 17d displays the course of the algorithm for the graph 30, 2 instance. It can be seen that throughout the entire algorithm improvements are being made. The last improvement is made in iteration 28, just before the algorithm is terminated.

Until the optimal solution is found, the genetic algorithm finds improvement in a lot of iterations, as was indicated by Figure 17. An improvement is made when either the solution comes closer to a local optimum or a neighbourhood is escaped and the new neighbourhood has a lower solution value. Additionally, if no improvement has been found for a while, the algorithm is still able to find an improvement. This was shown by the random 15, 3 instance. This is an indication that the algorithm is able to escape a local optimum.

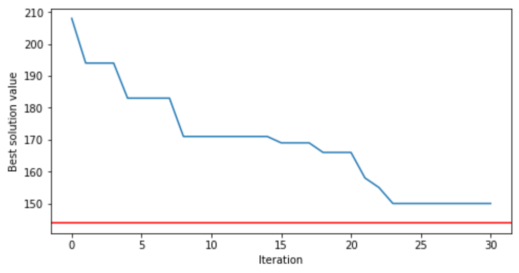
So, when the genetic algorithm is used in combination with the optimal solution to the annealer problem, the optimal solution to the container assignment problem is found for most of the instances. However, there are still some instances for which the optimal solution has not been found. The optimality gap lies between the 0 and 26 percent and is on average 3%. The optimality gap the random 20, 3 instance is with 26% higher than the other optimality gaps. It is remarkable that, the best found solution value for the instances where the optimum has not been attained, has been found in a late iteration of the



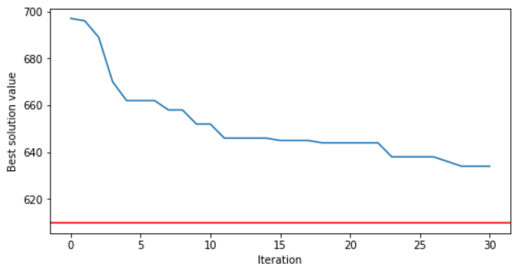
(a) Random 15, 3



(b) Random 25, 3



(c) Graph 17, 2



(d) Graph 30, 2

Figure 17: Course of the genetic algorithm using the optimal solution

algorithm.

7.3 Results using the quantum annealer

In this section, the results of the genetic algorithm in combination with the quantum annealer are discussed. A preliminary note is that the computation times of the random 15, 1 and random 15, 2 instances cannot be compared to the other instances. The genetic algorithm used too much memory to retrieve results, with the exception of the random 15, 1 and random 15, 2 instances. After the issue was fixed, the computation time decreased for the other instances. The time on the quantum annealer is limited and only the computation time is influenced by the adjustment. That is why the genetic algorithm has not been run again with the adjustment for the random 15, 1 and random 15, 2 instances. However, now these computation times cannot be compared to the other instances. Additionally, the computation time was really high for the instances corresponding to the graph 30 network. Unfortunately, the results for the graph 30, 3 instance could not be retrieved.

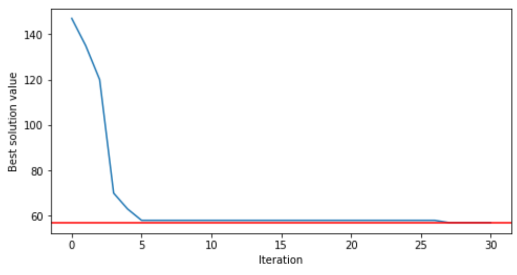
The found solutions for the genetic algorithm in combination with the quantum annealer are given in Appendix L.1. The results are displayed in Table 14. The optimality gap lies between 0 and 65 percent. The average optimality gap is 25%. For all the instances corresponding to the random network with 15 locations the optimal solution has been found. The Dutch network with the third container set has a relatively low optimality gap of 8%. The highest optimality gaps are attained for the instances corresponding to the random network with 20 locations. Other instances that have an optimality gap larger than 30% are: random 25, 1; random 25, 3; graph 17, 1 and graph 17, 3. The best solution value found by the genetic algorithm has been found in iteration 20 or later for half of the instances. Remarkable is that the genetic algorithm stops improving the solution value quite soon for the random 25, 1 and random 25, 2 instances. It can be seen that the computation times for the graph 30, 1 and graph 30, 2 instances lie significantly higher than the computation times of the other instances. Furthermore, the best initial value for the graph 30, 2 instance is higher than the cost of traveling by truck with every container for this instance. This means that no feasible solution has been found in the initialization part of the algorithm for this instance.

How the solution value develops during the algorithm when the quantum annealer is used is plotted per instance in Appendix L.2. A few of these plots are given in Figure 18. Figure 18a shows the best solution value in every iteration for the random network with 15 locations and the third container set. In the beginning, large improvements are made. Then, the best solution value remains the same for the major part of the algorithm. However, in one of the last iterations, the objective value improves again

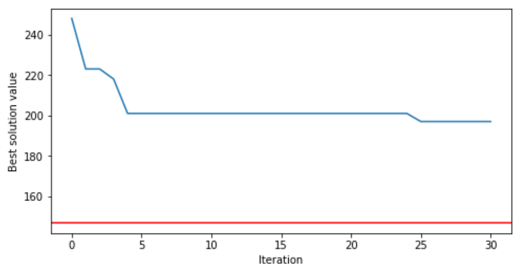
Table 14: Results genetic algorithm using the quantum annealer

Instance	Best value	Optimality gap(%)	Iteration	Best initial value	Time(s)
Random 15, 1	58	0	15	151	6157.06
Random 15, 2	61	0	24	130	7091.61
Random 15, 3	57	0	27	147	2836.01
Random 20, 1	107	41	19	159	2272.68
Random 20, 2	97	39	18	138	3276.64
Random 20, 3	162	65	20	228	4343.54
Random 25, 1	118	37	8	166	3069.78
Random 25, 2	146	29	4	185	4211.40
Random 25, 3	197	34	25	248	6383.60
Dutch, 1	120	25	14	144	3582.23
Dutch, 2	122	16	12	162	4076.97
Dutch, 3	160	8	24	175	5106.95
Graph 17, 1	190	32	27	243	5364.11
Graph 17, 2	193	34	30	243	5251.02
Graph 17, 3	241	15	13	274	7456.58
Graph 30, 1	430	13	23	462	32123.15
Graph 30, 2	814	33	11	1582	170887.41

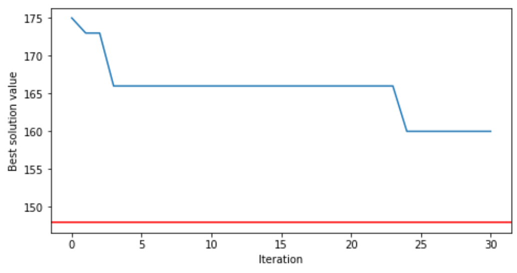
and the optimal solution is found. The development of the solution value for the random 25, 3 instance is shown in Figure 18b. From the figure can be seen that, in the beginning, some improvements are made. Then, the best solution value remains the same for a while. The best solution value improves a bit again in one of the last iterations. Figure 18c displays the course of the algorithm for the Dutch, 3 instance. The algorithm finds relative large improvements. Nonetheless, these improvements are not found often. The best solution value remains the same for most of the iterations. The plot in Figure 18d shows the development of the best solution value for the graph 17, 2 instance. The algorithm keeps finding a better solution throughout the iterations. Even in the last iteration, the solution value is improved.



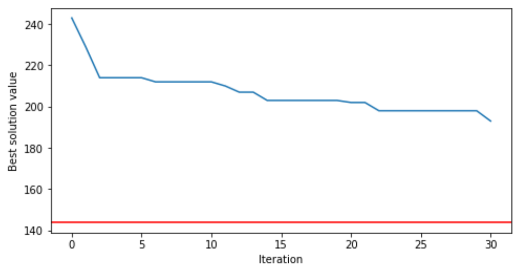
(a) Random 15, 3



(b) Random 25, 3



(c) Dutch, 3



(d) Graph 17, 2

Figure 18: Course of the genetic algorithm using the quantum annealer

When the genetic algorithm is used in combination with the quantum annealer, the genetic algorithm does not often find improvements for quite a lot instances. Most of the plots in Appendix L.2, have the same best solution value for multiple iterations. However, there are exceptions, as is shown in Figure 18d. Most best solution values are found in an iteration later in the algorithm. This is an indication that the algorithm has not converged yet.

The performance measures for the quantum annealer are given in Table 15. The timing measures are not included in this table. For all the instances with the exception of the instances corresponding to the network with 15 locations, the quantum annealer always returns a different solution than the optimal solution to the problem that is solved with the annealer. The average optimality gap for the instances corresponding to the random 15 network is still quite high. This can be explained through the high corresponding standard deviation. The average optimality gap lies between 17 and 103 percent. The instances with the lowest average optimality gaps are: random 25, 3; Dutch, 3 and all the instances corresponding to graph 17. Remarkable is that, the fraction of returned infeasible solutions is low for all the instances, except the instances corresponding to the graph 30 network. The high infeasible rates explain the high average annealer optimality gap for these instances. Also, almost all solutions returned by the quantum annealer for the graph 30, 2 instance are infeasible. This explains why no feasible solution has been found in the initialization part of the algorithm. What is interesting is that, the feasible optimality gap is quite high. This gap is between the 12 and 72%.

Table 15: Performance of the quantum annealer

Instance	Wrong(%)	Optimality gap(%)	Infeasible(%)	Feasible gap(%)
Random 15, 1	49.47	39 (42)	0.0	39 (42)
Random 15, 2	74.25	50 (35)	0.0	50 (35)
Random 15, 3	85.94	57 (31)	0.0	57 (31)
Random 20, 1	100.00	65 (17)	0.0	65 (17)
Random 20, 2	100.00	72 (21)	0.0	72 (21)
Random 20, 3	100.00	28 (15)	1.08	27 (11)
Random 25, 1	100.00	61 (16)	0.0	61 (16)
Random 25, 2	100.00	36 (12)	0.0	36 (12)
Random 25, 3	99.90	17 (10)	0.39	16 (7)
Dutch, 1	100.00	37 (10)	0.0	37 (10)
Dutch, 2	100.00	29 (9)	0.0	29 (9)
Dutch, 3	99.90	20 (19)	2.25	18 (7)
Graph 17, 1	100.00	21 (7)	0.10	21 (6)
Graph 17, 2	100.00	21 (6)	0.0	21 (6)
Graph 17, 3	100.00	17 (16)	1.75	14 (5)
Graph 30, 1	100.00	103 (47)	78.63	12 (4)
Graph 30, 2	100.00	126 (6)	99.90	14 (0)

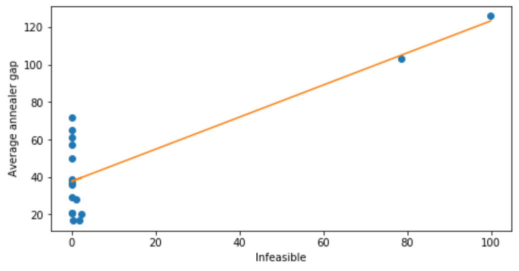
The performance measures considering the computation times of the quantum annealer are given in Table 16. The quantum time column displays the average actually used quantum time per instance in seconds. It can be seen that, the used quantum time remains quite the same for every instance. It is remarkable that the quantum time does not increase for the graph 30, 1 and graph 30, 2 instances, but the time it takes to solve the problem is quite higher than for the other instances. This can be explained, because these instances contain more variables. When the quantum annealer is used, every qubit represents a variable. In the used quantum annealer, every qubit is connected to 15 other qubits. If more than 15 variables are used, multiple qubits need to represent the same variable. In the problem that is solved with the annealer, every container has a variable. This variable represents whether this container travels by truck an which by multimodal route. The instances corresponding to the graph 30 network have more containers. Furthermore, there are more possible edges in the graph 30 network. The edges represent transportation services. When more transportation services are used, more slack variables need to be added to create equality constraints. So, for the graph 30 network, more variables are included in the problem that is solved with the annealer and all of these variables need to be assigned to qubits. An explanation for the higher computation time for the instances corresponding to the graph 30 network, is that a longer time is needed to find a good assignment of the variables to the qubits. Furthermore, the problem becomes more difficult. Because, not just all the slack variables need to be the exact right values, but the qubits representing these variables also need to be the exact right value.

From Figure 19 some observations can be made about the performance of the quantum annealer. In Figure 19a, the fraction of infeasible solutions is plotted against the average optimal annealer gap. For most of the instances, the quantum annealer was able to find a feasible solution. From Figure 19b, it can be seen that no clear relation can be determined between the fraction of infeasible solutions and the solution found by the genetic algorithm. Figure 19c shows the fraction of wrong solutions plotted

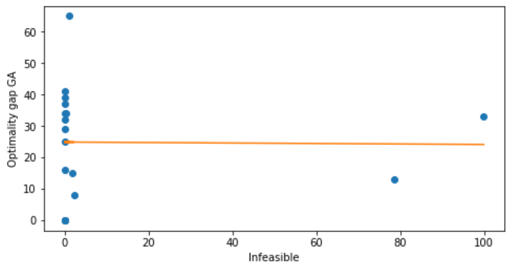
Table 16: Performance of the quantum annealer, computation times

Instance	Anneal time(s)	Annealer time(s)	Quantum time(s)
Random 15, 1	7.05	7.02	0.03
Random 15, 2	8.02	7.98	0.04
Random 15, 3	2.65	1.27	0.04
Random 20, 1	2.19	1.33	0.04
Random 20, 2	3.11	1.62	0.04
Random 20, 3	4.07	2.47	0.04
Random 25, 1	2.98	1.44	0.04
Random 25, 2	3.89	2.23	0.04
Random 25, 3	6.06	4.39	0.04
Dutch, 1	3.33	1.73	0.04
Dutch, 2	3.75	2.06	0.04
Dutch, 3	4.80	3.08	0.04
Graph 17, 1	4.95	3.33	0.04
Graph 17, 2	4.92	3.13	0.04
Graph 17, 3	7.07	5.32	0.04
Graph 30, 1	30.82	27.59	0.04
Graph 30, 2	162.59	158.55	0.04

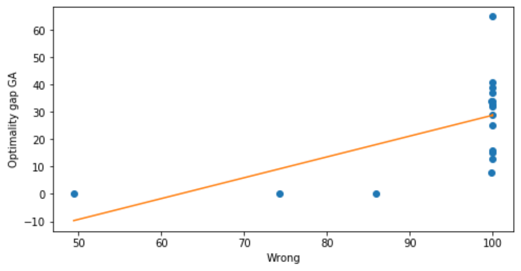
against the optimality gap determined with the final solution of the genetic algorithm. For most of the instances, the quantum annealer returned a solution different than the optimal solution value. However, if the annealer occasionally returns the optimal solution, the optimal solution is found. The average optimality gap for the quantum annealer is plotted against the optimality gap of the genetic algorithm in Figure 19d. The data points are far apart and no clear correlation can be determined.



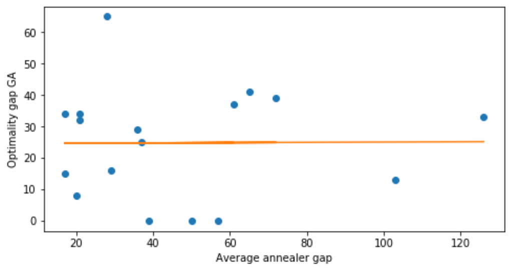
(a) Infeasible against the annealer gap



(b) Infeasible against the genetic algorithm gap



(c) Wrong against the genetic algorithm gap



(d) Annealer gap against the genetic algorithm gap

Figure 19: Performance of the quantum annealer

Thus, using the quantum annealer in the genetic algorithm gives an optimality gap between the 0 and the 65%. The average optimality gap is 25 percent. The optimal solution has been found to all the instances corresponding to the random network with 15 locations. For the Dutch, 3 instance, a relatively low optimality gap has been achieved. Also, the genetic algorithm finds a better solution after not improving for a while for quite a few instances. The average optimality gaps of the problems that are solved with the quantum annealer lie between the 17 and 126 percent. Interestingly, the quantum annealer returns mostly feasible solutions with a high objective value. This is an indication that too much pressure is set to find a feasible solution. How important it is to find a feasible solution relative to a good objective value, is determined by the W_B value. Since this parameter has been tuned with the simulated annealer, it is possible that the best value for this parameter is different when the quantum annealer is used. This explains how most of the returned solutions are feasible, but not good in terms of objective value. Furthermore, for the large instances corresponding to the graph 30 network, usually no feasible solution is found. A likely explanation for this phenomenon, is the limited connectivity of the qubits in the used quantum annealer.

7.4 Comparison

In Table 17, the optimality gaps of the solution values found by the genetic algorithm are shown for the different methods to determine which containers travel by truck and which by the given multimodal route. In the optimality gap Opt column, the optimality gaps are displayed when the genetic algorithm is used in combination with the optimal solution to the problem that is usually solved with an annealer. Remarkable is that, the genetic algorithm in combination with the optimal solution to the annealer problem gives the best solution for every instance except the random 15, 3 instance. Furthermore, the quantum annealer gives an at least as bad solution as the simulated annealer for most instances. The exceptions are: the random 25, 2; the Dutch, 3 and the graph 30, 1 instances. An explanation for the fact that the quantum annealer performs worse than the simulated annealer is that the parameters have been tuned with the simulated annealer. It is indicated that the W_B value is set too high for the quantum annealer, which gives bad results. Furthermore, the feasible gap of the simulated annealer lies between 7 and 21 percent, while the feasible gap of the quantum annealer lies between 12 and 72%. So, even though the simulated annealer returns an infeasible solution more often, if the solution is feasible, the solution value lies closer to the optimal solution value.

Table 17: Compare the optimality gap

Instance	Optimality gap SA(%)	Optimality gap Opt(%)	Optimality gap QA(%)
Random 15, 1	2	0	0
Random 15, 2	0	0	0
Random 15, 3	0	2	0
Random 20, 1	24	0	41
Random 20, 2	39	0	39
Random 20, 3	36	26	65
Random 25, 1	17	0	37
Random 25, 2	35	0	29
Random 25, 3	33	6	34
Dutch, 1	24	0	25
Dutch, 2	15	0	16
Dutch, 3	14	0	8
Graph 17, 1	16	8	32
Graph 17, 2	31	4	34
Graph 17, 3	12	0	15
Graph 30, 1	18	2	13
Graph 30, 2	28	4	33
Graph 30, 3	14	3	NA

The computation times for the different methods to solve the annealer problem are displayed in Table 18. It can be seen that the quantum annealer uses the most computation time. However, this computation time also includes the waiting time till the annealer is available, time for internet latency and the time it takes to implement the problem on the quantum annealer. The genetic algorithm in combination with the optimal solution uses the least computation time.

7.5 Long run with the optimal solution

The parameters n_{pop} and n_{iter} were based on the limited quantum time. The genetic algorithm is run again in combination with the optimal solution to the problem that determines which container travels by truck and which by the given multimodal route for $n_{pop} = 120$, $n_{pairs} = 90$ and $n_{iter} = 90$. This way it can be seen how the genetic algorithm performs when these parameters are not that limited.

Table 18: Compare the computation times

Instance	Time SA(s)	Time Opt(s)	Time QA(s)
Random 15, 1	288.51	171.85	6157.06
Random 15, 2	396.59	238.49	7091.61
Random 15, 3	312.85	228.19	2836.01
Random 20, 1	310.12	200.04	2272.68
Random 20, 2	280.6	187.33	3276.64
Random 20, 3	345.18	228.99	4343.54
Random 25, 1	296.95	198.8	3069.78
Random 25, 2	332.4	218.56	4211.4
Random 25, 3	395.59	259.19	6383.6
Dutch, 1	289.81	260.45	3582.23
Dutch, 2	330.22	225.42	4076.97
Dutch, 3	363.93	202.9	5106.95
Graph 17, 1	417.63	193.88	5364.11
Graph 17, 2	432.4	176.69	5251.02
Graph 17, 3	463.16	170.89	7456.58
Graph 30, 1	976.64	203.27	32123.15
Graph 30, 2	2438.78	254.64	170887.41
Graph 30, 3	2287.04	242.97	NA

The found routes are given in Appendix M.1. The results are summarized in Table 19. It can be seen that, the optimal solution is found for most of the instances, but not for all of them. The final improvement was made in iteration 51 considering all the instances. However, for most instances, the best solution value is found in the beginning of the algorithm. It is remarkable that, for all the instances corresponding to the Dutch network, the optimal solution has been found in one of the first iterations. The optimality gap for the random 20, 3 instance is still relatively high. However, the found solution improved from 123 to 105, which is an improvement of approximately 15%. The computation times corresponding to the graph 30, 2 and graph 30, 3 instances are a bit higher than the computation times for the other instances. Those instances have more containers in their container set, which explains the higher computation time.

Table 19: Results genetic algorithm using the optimal solution long run

Instance	Best value	Optimality gap(%)	Iteration	Best initial value	Time(s)	Optimal time(s)
Random 15, 1	58	0	11	66	1234.28	1226.00
Random 15, 2	61	0	7	92	1413.56	1402.56
Random 15, 3	57	0	9	89	1422.04	1411.23
Random 20, 1	76	0	11	87	1305.44	1288.46
Random 20, 2	70	0	11	79	1379.93	1350.73
Random 20, 3	105	7	45	196	1215.60	1196.15
Random 25, 1	86	0	12	119	1431.22	1402.03
Random 25, 2	113	0	11	147	1457.29	1415.87
Random 25, 3	148	1	22	210	1449.98	1416.65
Dutch, 1	96	0	4	114	1414.01	1399.58
Dutch, 2	105	0	5	134	1475.56	1462.09
Dutch, 3	148	0	2	152	1481.52	1464.67
Graph 17, 1	146	1	27	180	870.85	853.52
Graph 17, 2	144	0	48	206	1400.52	1377.19
Graph 17, 3	209	0	26	235	1544.10	1506.62
Graph 30, 1	385	1	24	412	1720.91	1565.40
Graph 30, 2	626	3	51	697	2429.17	2139.22
Graph 30, 3	718	2	41	751	2276.64	1969.68

For almost all of the instances for which the optimal solution was not found for $n_{pop} = 40$ and $n_{iter} = 30$, the optimal solution is still not found. The exceptions are the random 15, 3 and the graph 17, 2 instances. It is remarkable that, the optimal solution to the random 15, 3 instance has been found in the ninth iteration while the optimal solution was not found previously. The best found solution did decrease quite a bit for the instances for which the optimal solution still has not been found.

8 Discussion

The quantum annealer did not perform well for the given parameters. If the annealer performs better it is likely that the genetic algorithm finds better solutions to the container assignment problem. This better performance can likely be achieved by two changes considering the quantum annealer. The first change is the availability of the quantum annealer. Currently, a problem is sent to the quantum annealer, where it has to wait in line until it can be solved. This waiting time creates a high computation time, while the time it takes to actually solve the problem remains small. Furthermore, the quantum annealer can only be used for a limited time, such that the number of runs are restricted. The second change is the better connectivity of the qubits with each other. The limited connectivity makes it more difficult to find good and feasible solutions to the problem that is solved with the quantum annealer, especially when a lot of variables are used.

Having unlimited access to the quantum annealer that solves the received problems fast, creates different opportunities. The first opportunity is that every parameter can be tuned with the quantum annealer. Especially for the W_B value, it is indicated that this will created a difference. Because, the quantum annealer often returns feasible solutions with a high objective value. The indication is that too much pressure is set on finding a feasible solution, such that the cost minimization is less important. Having tuned parameters for the quantum annealer will improve the performance of the genetic algorithm in combination with the quantum annealer. The second opportunity is that the number of populations and number of iterations can be increased. By increasing the value for these parameters, it can be examined to which objective value the genetic algorithm converges to in combination with the quantum annealer. If the access to the quantum annealer is only expanded a bit, the number of iterations should be the first parameter that is increased. Most of the best solution values were found in iterations late in the algorithm. This is an indication that the genetic algorithm has not yet converged to the best achievable value and will improve when more iterations are available. However, the rate of improvement will decrease after more and more iterations are used. That is the point where the number of populations should be increased. The last opportunity is that larger instances can be solved faster and better with the quantum annealer when the connectivity increases. The instances corresponding to the graph 30 instance have more variables. The quantum annealer often returns infeasible solutions for these instances, which is an indication that it is hard for the quantum annealer to find the right value for all the variables. Since multiple qubits represent the same variable, it is extra hard to find the right values. A better connectivity can help solve this problem.

Additionally, for every problem that is sent to the quantum annealer, variables need to be assigned to qubits and connections need to be created between them. This is called an embedding. D-Wave has provided a tool that creates an embedding for a given QUBO formulation. However, it might be that an improvement can be made when the embedding is made manually.

Furthermore, there are also changes that might improve the performance of the genetic algorithm. Currently, the genetic algorithm only considers one multimodal route option per container. It might be interesting to adjust the genetic algorithm such that more options can be incorporated in the problem that is solved with an annealer.

Additionally, the optimal solution has not been found for some instances when the number of populations and the number of iterations are increased. This is an indication that an improvement might occur when more diversity is added to the genetic algorithm. This diversity could help prevent the genetic algorithm to get stuck in a local optimum. An idea to introduce more diversity is to construct a more diverse initial populations. This can be achieved by not using the shortest paths in the initial populations, but other paths that might have a higher cost. It would be interesting to investigate the influence of these initial populations on the performance of the genetic algorithm.

9 Conclusion

The aim of this thesis was to solve the container assignment problem, while using a quantum annealer. A hybrid quantum-classical genetic algorithm has been proposed to solve this problem. The quantum annealer has been used to determine which container travels by truck and which by the given multimodal route. The problem that remains, is to determine this multimodal route for every container. A genetic algorithm has been used as the solution approach to this problem. The genetic algorithm determined a multimodal route for every container that is used as input for the problem that is solved with the annealer.

For some parts of the genetic algorithm, multiple options were introduced. There were four different initialization methods, three different selection mechanisms, three different cross-over operators and three different mutation operators. Different instances have been constructed to evaluate the performance of the genetic algorithm. To construct the instances, different networks have been generated. Three networks have been created randomly, while three others have been generated based on an existing transportation network. For every network, three different container sets have been created. The time on the quantum annealer is limited, so the simulated annealer has been used to determine which of the introduced methods is actually used to retrieve the results with the quantum annealer. Additionally, the simulated annealer has been used to tune the parameters used in the genetic algorithm. Some of the results have been set to a restrictive value to control the time that the quantum annealer uses to retrieve the results. To provide a benchmark to the genetic algorithm used in combination with the quantum annealer, the genetic algorithm has also been used in combination with the simulated annealer and in combination with an exact solution method. This exact solution method finds the optimal solution to the problem that is usually sent to the annealer. To get an idea about the performance of the genetic algorithm when certain parameters are not restricted, the value of these parameters has been increased for the genetic algorithm in combination with the exact solution method.

First, consider the performance of the simulated annealer. The simulated annealer does often not find a feasible solution that it is solving. Furthermore, the optimal solution to the problem that is solved is often not found. In conclusion, the simulated annealer does not perform well. This might be the reason that, with the exception of the instances corresponding to the random network with 15 locations, the genetic algorithm also does not perform well. The optimal solution has not been found for most instances. Furthermore, the gap between the found solution value and the optimal solution value is on average 20%.

Then, the genetic algorithm is used with the optimal solution to the problem that determines which container travels by truck and which by multimodal route. For most instances, a good solution has been found. However, the solution for the random 20, 3 instance is less good than the solution for the other instances. Having good optimality gaps for most of the instances implies that the genetic algorithm is good in finding the global optimum when the optimal solution to the problem that is usually solved with the annealer is used.

Thereafter, the genetic algorithm is used in combination with the quantum annealer. Most of the returned solutions by the quantum annealer are feasible, but have a high objective value. To conclude, the quantum annealer does not give good results considering the problem of determining which container travels by truck and which by the given multimodal route. When the genetic algorithm is used in combination with the quantum annealer, the optimality gap is quite high. The average optimality gap is 25%. Even though the optimal solution has been found for the instances corresponding to the random network with 15 locations, for most of the instances applies that the found value is quite different than the optimal solution.

Finally, to see if the performance of the genetic algorithm improves if time is not a limiting factor, the number of populations, number of pairs and number of iterations have been increased. The optimal solution still has not found for some of the instances. This is an indication that the algorithm becomes stuck in a local optimum. However, for most of the instances, a good solution has been found. The found solutions did improve quite a bit regarding the solutions for the restricted parameter values.

To summarize, the genetic algorithm gives the best results in combination with the exact solution method. The simulated annealer does not perform well, because a lot of the returned solutions are infeasible. The results are better than the results obtained with the genetic algorithm in combination with the quantum annealer. This annealer also does not perform well, but now the reason is the high objective value for the returned solutions. So, the designed genetic algorithm is currently not able to find good solutions to the container assignment problem, while using a quantum annealer.

For future research, it is useful to improve the performance of the quantum annealer. When the performance of the quantum annealer improves, the results are likely to be closer to the results of the genetic algorithm in combination with the exact solution method. It is shown that, the genetic algorithm in combination with the exact solution method gives good results. So, when the quantum annealer often returns the optimal solution, the results will likely also be good. Especially when the parameter values are less restricted by the limited availability of the quantum annealer. To improve the performance of the quantum annealer, it is interesting to investigate whether a better parameter tuning can be used. Currently, it is indicated that the best W_B value is different for the quantum annealer with respect to the simulated annealer. A more profound parameter tuning for the quantum annealer might improve the performance of the quantum annealer and the final solution returned by the genetic algorithm enormously.

References

S. G. Ahmad, C. S. Liew, E. U. Munir, T. F. Ang, and S. U. Khan. A hybrid genetic algorithm for optimization of scheduling workflow applications in heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 87:80–90, 2016.

C. W. Ahn and R. S. Ramakrishna. A genetic algorithm for shortest path routing problem and the sizing of populations. *IEEE transactions on evolutionary computation*, 6(6):566–579, 2002.

R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. Network flows. 1988.

A. Ajagekar, T. Humble, and F. You. Quantum computing based hybrid solution strategies for large-scale discrete-continuous optimization problems. *Computers & Chemical Engineering*, 132:106630, 2020.

K. Altendorfer and S. Minner. Optimal transportation mode decision for deterministic transportation times, stochastic container arrivals and an intermodal option. In *Operations Research Proceedings 2015*, pages 201–207. Springer, 2017.

B. Ayar and H. Yaman. An intermodal multicommodity routing problem with scheduled services. *Computational optimization and applications*, 53(1):131–153, 2012.

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

E. Ballot, B. Montreuil, and R. D. Meller. *The Physical Internet.* La Documentation Française, Oct. 2014. URL `https://hal-mines-paristech.archives-ouvertes.fr/hal-01113648`.

C. Barnhart, N. Krishnan, and P. H. Vance. *Multicommodity flow problemsMulticommodity Flow Problems*, pages 1583–1591. Springer US, Boston, MA, 2001. ISBN 978-0-306-48332-5. doi: 10.1007/0-306-48332-7_316. URL `https://doi.org/10.1007/0-306-48332-7_316`.

B. Behdani, Y. Fan, B. Wiegmans, and R. Zuidwijk. Multimodal schedule design for synchromodal freight transport systems. *Behdani, B., Fan, Y., Wiegmans, B., & Zuidwijk*, pages 424–444, 2014.

M. G. Bell, X. Liu, P. Angeloudis, A. Fonzone, and S. H. Hosseinloo. A frequency-based maritime container assignment model. *Transportation Research Part B: Methodological*, 45(8):1152–1161, 2011.

S. Bock. Real-time control of freight forwarder transportation networks by integrating multimodal transport chains. *European Journal of Operational Research*, 200(3):733–746, 2010.

A. Caris, C. Macharis, and G. K. Janssens. Planning problems in intermodal freight transport: accomplishments and prospects. *Transportation Planning and Technology*, 31(3):277–302, 2008.

T.-S. Chang. Best routes selection in international intermodal networks. *Computers & operations research*, 35(9):2877–2891, 2008.

B.-C. Choi, K. Lee, J. Y.-T. Leung, M. L. Pinedo, and D. Briskorn. Container scheduling: Complexity and algorithms. *Production and Operations Management*, 21(1):115–128, 2012.

T. G. Crainic and K. H. Kim. Intermodal transportation. *Handbooks in operations research and management science*, 14:467–537, 2007.

T. G. Crainic and G. Laporte. Planning models for freight transportation. *Design and operation of civil and environmental engineering systems. Wiley-Interscience, New York*, pages 343–394, 1997.

M. A. De Juncker, D. Huizing, M. O. del Vecchyo, F. Phillipson, and A. Sangers. Framework of synchromodal transportation problems. In *International Conference on Computational Logistics*, pages 383–403. Springer, 2017.

E. W. Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1): 269–271, 1959.

Y. Ding, X. Chen, L. Lamata, E. Solano, and M. Sanz. Implementation of a hybrid classical-quantum annealing algorithm for logistic network design, 2020.

A. L. Erera, J. C. Morales, and M. Savelsbergh. Global intermodal tank container management for the chemical industry. *Transportation Research Part E: Logistics and Transportation Review*, 41(6): 551–566, 2005.

S. Feld, C. Roch, T. Gabor, C. Seidel, F. Neukart, I. Galter, W. Mauerer, and C. Linnhoff-Popien. A hybrid solution method for the capacitated vehicle routing problem using a quantum annealer. *Frontiers in ICT*, 6:13, 2019.

A. Ghosh and S. Mukherjee. Quantum annealing and computation: a brief documentary note. *arXiv preprint arXiv:1310.1339*, 2013.

F. Glover, G. Kochenberger, and Y. Du. Quantum bridge analytics i: a tutorial on formulating and using qubo models. *4OR*, 17(4):335–371, 2019.

J. Gu, X. Gu, and B. Jiao. A quantum genetic based scheduling algorithm for stochastic flow shop scheduling problem with random breakdown. *IFAC Proceedings Volumes*, 41(2):63–68, 2008a.

J. Gu, X. Gu, and B. Jiao. Solving stochastic earliness and tardiness parallel machine scheduling using quantum genetic algorithm. In *2008 7th World Congress on Intelligent Control and Automation*, pages 4154–4159. IEEE, 2008b.

J. Gu, M. Gu, C. Cao, and X. Gu. A novel competitive co-evolutionary quantum genetic algorithm for stochastic job shop scheduling problem. *Computers & Operations Research*, 37(5):927–937, 2010.

L. Gyongyosi and S. Imre. A survey on quantum computing technology. *Computer Science Review*, 31: 51–71, 2019.

A. Haghani and S.-C. Oh. Formulation and solution of a multi-commodity, multi-modal network flow model for disaster relief operations. *Transportation Research Part A: Policy and Practice*, 30(3):231–250, 1996.

K.-H. Han and J.-H. Kim. Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. *IEEE transactions on evolutionary computation*, 6(6):580–593, 2002.

K.-H. Han, K.-H. Park, C.-H. Lee, and J.-H. Kim. Parallel quantum-inspired genetic algorithm for combinatorial optimization problem. In *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546)*, volume 2, pages 1422–1429. IEEE, 2001.

P. Hauke, H. G. Katzgraber, W. Lechner, H. Nishimori, and W. D. Oliver. Perspectives of quantum annealing: Methods and implementations. *Reports on Progress in Physics*, 83(5):054401, 2020.

B. Heim, T. F. Rønnow, S. V. Isakov, and M. Troyer. Quantum versus classical annealing of ising spin glasses. *Science*, 348(6231):215–217, 2015.

J. H. Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* MIT press, 1992.

J. Inagaki, M. Haseyama, and H. Kitajima. A genetic algorithm for determining multiple routes and its applications. In *1999 IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 6, pages 137–140. IEEE, 1999.

M. W. Johnson, M. H. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A. J. Berkley, J. Johansson, P. Bunyk, et al. Quantum annealing with manufactured spins. *Nature*, 473(7346): 194–198, 2011.

K. Kalicharan, F. Phillipson, A. Sangers, and M. d. Juncker. Reduction of variables for solving logistic flow problems. In *6th International Physical Internet Conference (IPIC), 2019.*, 2019.

R. M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

J. King, M. Mohseni, W. Bernoudy, A. Fréchette, H. Sadeghi, S. V. Isakov, H. Neven, and M. H. Amin. Quantum-assisted genetic algorithm. *arXiv preprint arXiv:1907.00707*, 2019.

G. Kochenberger, J.-K. Hao, F. Glover, M. Lewis, Z. Lü, H. Wang, and Y. Wang. The unconstrained binary quadratic programming problem: a survey. *Journal of Combinatorial Optimization*, 28(1):58–81, 2014.

J.-C. Lee, W.-M. Lin, G.-C. Liao, and T.-P. Tsao. Quantum genetic algorithm for dynamic economic dispatch with valve-point effects and including wind power system. *International Journal of Electrical Power & Energy Systems*, 33(2):189–197, 2011.

B.-B. Li and L. Wang. A hybrid quantum-inspired genetic algorithm for multiobjective flow shop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(3):576–591, 2007.

M. E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations research*, 53(6): 1007–1023, 2005.

A. Lucas. Ising formulations of many np problems. *Frontiers in Physics*, 2:5, 2014.

S. Mankabady. The multimodal transport of goods convention: a challenge to unimodal transport conventions. *International and Comparative Law Quarterly*, pages 120–140, 1983.

U. Maulik and S. Bandyopadhyay. Genetic algorithm-based clustering technique. *Pattern recognition*, 33 (9):1455–1465, 2000.

Y. Miao and A. Ni. Vulnerability analysis of intercity multimode transportation networks; a case study of the yangtze river delta. *Sustainability*, 11(8), 2019. ISSN 2071-1050. doi: 10.3390/su11082237. URL `https://www.mdpi.com/2071-1050/11/8/2237`.

L. Moccia, J.-F. Cordeau, G. Laporte, S. Ropke, and M. P. Valentini. Modeling and solving a multimodal transportation problem with flexible-time and scheduled services. *Networks*, 57(1):53–68, 2011.

F. Neukart, G. Compostella, C. Seidel, D. Von Dollen, S. Yarkoni, and B. Parney. Traffic flow optimization using a quantum annealer. *Frontiers in ICT*, 4:29, 2017.

F. Neukart, D. Von Dollen, and C. Seidel. Quantum-assisted cluster analysis. *arXiv preprint arXiv:1803.02886*, 2018a.

F. Neukart, D. Von Dollen, C. Seidel, and G. Compostella. Quantum-enhanced reinforcement learning for finite-episode games with discrete state spaces. *Frontiers in physics*, 5:71, 2018b.

N. M. Neumann, P. B. de Heer, I. Chiscop, and F. Phillipson. Multi-agent reinforcement learning using simulated quantum annealing. In *International Conference on Computational Science*, pages 562–575. Springer, 2020.

M. Ortega del Vecchyo, F. Phillipson, and A. Sangers. Alternative performance indicators for optimizing container assignment in a synchromodal transportation network. 2018.

M. Paulinas and A. Ušinskas. A survey of genetic algorithms applications for image enhancement and segmentation. *Information Technology and control*, 36(3), 2007.

S. Pfoser, H. Treiblmaier, and O. Schauer. Critical success factors of synchromodality: Results from a case study and literature review. *Transportation Research Procedia*, 14:1463–1471, 2016.

F. Phillipson and I. Chiscop. Multimodal container planning: a qubo formulation and implementation on a quantum annealer. In *International Conference on Computational Science*, pages 1–14. Springer, 2021.

J. L. Ribeiro Filho, P. C. Treleaven, and C. Alippi. Genetic-algorithm programming environments. *Computer*, 27(6):28–43, 1994.

A. P. Rivera and M. Mes. Service and transfer selection for freights in a synchromodal network. In *International Conference on Computational Logistics*, pages 227–242. Springer, 2016.

K. Sastry, D. Goldberg, and G. Kendall. Genetic algorithms. In *Search methodologies*, pages 97–125. Springer, 2005.

P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

M. SteadieSeifi, N. P. Dellaert, W. Nuijten, T. Van Woensel, and R. Raoufi. Multimodal freight transportation planning: A literature review. *European journal of operational research*, 233(1):1–15, 2014.

Y. Sun and M. Lang. Modeling the multicommodity multimodal routing problem with schedule-based services and carbon dioxide emission costs. *Mathematical Problems in Engineering*, 2015, 2015.

Y. Sun, M. Lang, and J. Wang. On solving the fuzzy customer information problem in multicommodity multimodal routing with schedule-based services. *Information*, 7(1):13, 2016.

H. Talbi, A. Draa, and M. Batouche. A new quantum-inspired genetic algorithm for solving the travelling salesman problem. In *2004 IEEE International Conference on Industrial Technology, 2004. IEEE ICIT'04.*, volume 3, pages 1192–1197. IEEE, 2004.

H. Tikani and M. Setak. Efficient solution algorithms for a time-critical reliable transportation problem in multigraph networks with fifo property. *Applied Soft Computing*, 74:504–528, 2019.

V. Toğan and A. T. Daloğlu. An improved genetic algorithm with initial population strategy and self-adaptive member grouping. *Computers & Structures*, 86(11-12):1204–1218, 2008.

T. T. Tran, M. Do, E. G. Rieffel, J. Frank, Z. Wang, B. O'Gorman, D. Venturelli, and J. C. Beck. A hybrid quantum-classical approach to solving scheduling problems. In *SOCS*, pages 98–106, 2016.

H. Ushijima-Mwesigwa, R. Shaydulin, C. F. Negre, S. M. Mniszewski, Y. Alexeev, and I. Safro. Multilevel combinatorial optimization across quantum architectures. *arXiv preprint arXiv:1910.09985*, 2019.

B. van Riessen, R. R. Negenborn, and R. Dekker. Synchromodal container transportation: an overview of current topics and research opportunities. In *International conference on computational logistics*, pages 386–397. Springer, 2015.

D. van Vreumingen, F. Neukart, D. Von Dollen, C. Othmer, M. Hartmann, A.-C. Voigt, and T. Bäck. Quantum-assisted finite-element design optimization. *arXiv preprint arXiv:1908.03947*, 2019.

L. M. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. H. Sherwood, and I. L. Chuang. Experimental realization of shor's quantum factoring algorithm using nuclear magnetic resonance. *Nature*, 414(6866):883–887, 2001.

I.-L. Wang. Multicommodity network flows: A survey, part i: Applications and formulations. *International Journal of Operations Research*, 15(4):145–153, 2018a.

I.-L. Wang. Multicommodity network flows: A survey, part ii: Solution methods. *International Journal of Operations Research*, 15(4):155–173, 2018b.

L. Wang, F. Tang, and H. Wu. Hybrid genetic algorithm based on quantum computing for numerical optimization and parameter estimation. *Applied Mathematics and Computation*, 171(2):1141–1156, 2005.

D. Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

G. Xiong and Y. Wang. Best routes selection in multimodal networks using multi-objective genetic algorithm. *Journal of Combinatorial Optimization*, 28(3):655–673, 2014.

H. Zhi and S. Liu. Face recognition based on genetic algorithm. *Journal of Visual Communication and Image Representation*, 58:495–502, 2019.

A. Ziliaskopoulos and W. Wardell. An intermodal optimum path algorithm for multimodal networks with dynamic arc travel times and switching delays. *European Journal of Operational Research*, 125 (3):486–502, 2000.

B. G. Zweers, S. Bhulai, and R. D. van der Mei. Optimizing barge utilization in hinterland container transportation. *Naval Research Logistics (NRL)*, 66(3):253–271, 2019.

Appendix

A Quantum annealing

The quantum annealer starts with every qubit in its superposition. A qubit corresponds to a variable of the problem. When the quantum annealer runs, an energy barrier is raised between the zero and the one state. At the end of the anneal, every qubit will have fallen either to the zero state or the one state. This is a classical state and corresponds to the found solution to the problem.

The probability of ending in either state can be influenced by two things. An external magnetic field can be applied to the qubits, such that either the zero state or the one state will have a lower energy level. A lower energy level increases the probability that it will end up in the corresponding state. This is called the bias. The probability for each qubit to end up in either the zero state or the one state can also be influenced by the other qubits. Qubits can be linked together and influence each other. The qubits are linked together by something called a coupler. The coupler defines how qubits influence each other. For example, it might assign lower energy levels when the qubits are in the same state and thereby increasing the probability that the qubits will end up in the same state. This is called entanglement. When two qubits are entangled they have to be considered as a single object. This object now has four possible states. These states are all the combinations of being in the zero state or the one state for each qubit. The biases and couplers define the energy landscape.

To describe the energy of a physical system, a Hamiltonian is used. A Hamiltonian is a mathematical description of a physical system in terms of energy. The physical system corresponds to the states the qubits are in. Note that a qubit can still be in its superposition. So, when the current state of the system is put into the Hamiltonian, it gives the energy of that state. A classical example of a Hamiltonian is to compare the energy level of the states of an apple on a table and an apple on the floor. The Hamiltonian will assign a higher energy level to the state where the apple is on the table and a lower energy level to the state where the apple is on the floor.

The Hamiltonian in quantum annealing is defined by biases and couplers. The Hamiltonian is described by an Ising model, which is equivalent to a Quadratic Unconstrained Binary Optimization model. To solve a problem with the quantum annealer the input has to be in either one of those forms.

The Hamiltonian used in the quantum annealer exists of a combination of the initial Hamiltonian and the final Hamiltonian. The initial Hamiltonian corresponds to all the qubits being in their superposition. The final Hamiltonian corresponds to the solution to the problem. The final Hamiltonian is defined by the biases and couplers corresponding to the problem. The quantum annealer tries to find the state of all the qubits that gives the lowest energy when put into the Hamiltonian. The quantum annealer starts using the initial Hamiltonian and slowly introduces the problem Hamiltonian. The initial Hamiltonian is removed at the same rate as the problem Hamiltonian is introduced.

To visualize the possible energy levels during the quantum annealing procedure, an eigenspectrum could be used. An eigenspectrum is a graph of energy against time. The lowest energy state is at the bottom and higher excited states are above it, as can be seen in Figure 20. The higher level energy states correspond to non-optimal solutions. To find the optimal solution you want the system to stay in the lowest energy level during the entire anneal. The system starts at the lowest energy state at the left bottom, corresponding to the initial Hamiltonian. As the problem Hamiltonian is introduced, there are other energy levels that get closer to this lowest energy level. The energy state at the right bottom corresponds to the optimal solution. There is a probability that the system jumps out of the lowest energy level and into a higher energy level, meaning that the optimal solution might not be found. The harder a problem is, the higher the probability that the system will jump into a higher energy state. Each problem has a different Hamiltonian and a different eigenspectrum.

So, the quantum annealer tries to find the lowest energy level in a landscape defined by the Hamiltonian. This Hamiltonian is defined by biases and couplers. This is how quantum annealing leverages quantum physics to solve optimization problems[7]. Johnson et al. (2011) give a more technical description of the quantum annealing process.

[6]Source: `https://docs.dwavesys.com/docs/latest/c_gs_2.html`

[7]`https://www.youtube.com/user/dwavesystems/videos`

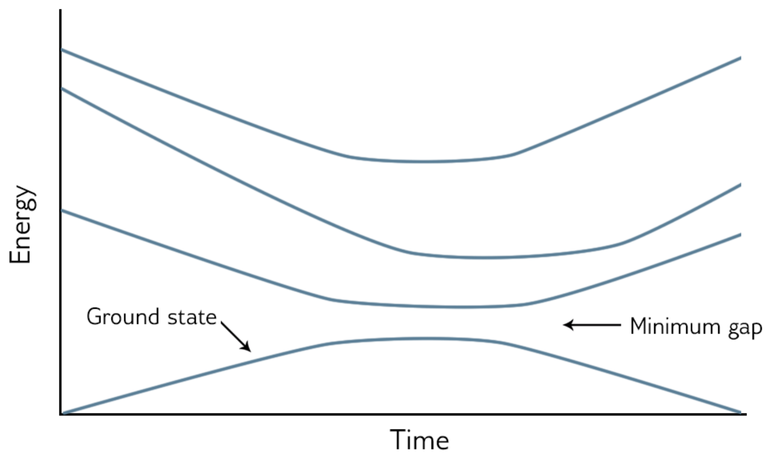


Figure 20: Eigenspectrum[6]

B Complexity of the problem that is solved with the quantum annealer

The set packing problem can be polynomially reduced to the problem that is solved with the quantum annealer, to be referred to as the container quantum problem. Polynomial reduction means that every instance of the set packing problem can be transferred to an equivalent instance of the container quantum problem in polynomial time. The set packing problem is NP-complete, as proven in the paper of Karp (1972). If a NP-complete problem can be polynomially reduced to another problem, it means that the other problem is NP-complete as well. It remains to show that indeed every instance of the set packing problem can be transferred to an instance of the container quantum problem in polynomial time. To do so, first the set packing problem is described. Then, it is described how an instance of the set packing problem can be transformed to an instance of the container quantum problem. Finally, the connection is established.

In the set packing problem a set of elements is given. This set is called the universe. Also, multiple subsets of the universe are given. The union of these subsets gives the universe. The set packing problem tries to find the maximum number of subsets, such that all the selected subsets are pairwise disjoint. This means that an element cannot appear more than once in the selected subsets. The decision problem that corresponds to the set packing problem is: can at least k subsets be selected, such that the selected subsets are pairwise disjoint?

Let U be the universe, so the set containing all the elements. Let S be the set containing all the subsets. A set packing is defined as a selection of a few of those subsets, such that every element is contained at most once in the packing. The problem is finding the set packing that contains the most subsets, while still containing each element at most once. The binary parameter a_{su}, $s \in S$ and $u \in U$, is given and contains the value 1 if subset s contains element u. Let the binary variable x_s, $s \in S$, contain the value 1 if subset s is selected in the set packing. The objective is to maximize the number of subsets that are selected in the set packing under the constraints that every element appears at most once in the subsets that are selected in the packing.

The container quantum problem chooses for every container if it is traveling by a multimodal route or traveling by truck. This multimodal route is represented as a binary string corresponding to arcs in a time-space graph. Each arc has a corresponding capacity. This capacity cannot be exceeded by the routes that are selected. The multimodal route is assumed to be feasible, meaning that the container travels from its origin to its destination and arrives there before its due date by using the arcs that the binary string represents.

Construct the instance for the container quantum problem from an instance of the set packing problem by letting every element in the universe correspond to an arc in a different time period. Let the order of the elements correspond to a chronological time order. So, element 1 corresponds to an arc in the first time period, element 2 to an arc in the second time period and so on. Assume that every arc is connected to the consecutive arc. So, the end of the arc corresponding to the first element is the same as the beginning of the arc corresponding to the second element. Let there be as many containers as there are subsets. Let a single subset correspond to arcs that are used in the multimodal route for a container.

Let the elements within a subset also be ordered, such that the elements correspond to chronological order.

Consider a subset that contains non-consecutive elements, for example 1,3,.... This means that the arc in the first time and third period are used in the route for a container, but the second arc is not. It follows that it might occur that the arcs in a route are not connected when only the elements are included as arcs. To solve this problem extra arcs can be added to the graph.

Let those extra arcs have a capacity equal to the number of containers, such that the capacity can never be exceeded for those arcs. An arc connects two nodes and has a begin and an end node. Create extra arcs between the end of every arc in the universe and the beginning of every higher arc in the universe. In Figure 21 the extra arcs for the first and second element have been displayed. The end of the first arc needs to be connected with every beginning of every other element in the universe, with the exception of the second element. Because, the end of the first arc is the same as the beginning of the second element. In total there are $n-2$ extra arcs needed for the first element. There is a subtraction of 2, because from the n elements there are two elements that do not have to be connected to the first element. The first element does not have to be connected to itself, so that is one element less. The beginning of the second arc is already connected to the end of the first arc, so that is another element less. Since the elements are chronologically ordered there are only $n-3$ extra arcs needed for the second element. The total number of extra arcs needed is thus $(n-2)+(n-3)+...+1$. This can be written as $\sum_{i=2}^{n-1}(n-i) = \sum_{j=1}^{n-2} j = \frac{(n-2)(n-1)}{2} = \frac{1}{2}n^2 - \frac{3}{2}n + 1 \leq n^2$ for $n \geq 1$. So, the number of arcs needed is always less or equal to n^2. Meaning that there is a polynomial number of extra arcs needed.

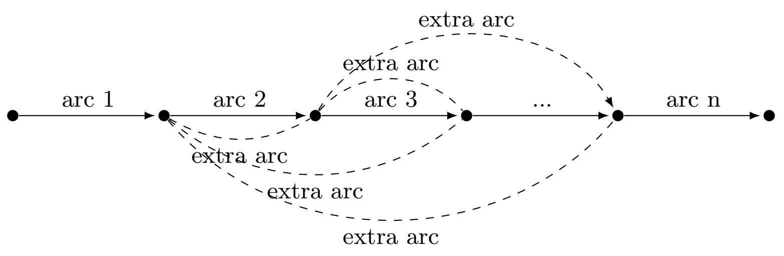


Figure 21: Elements in the universe

Let the capacity of each arc in the universe be 1. As mentioned, the capacity of the extra arcs is equal to the number of containers. Furthermore, let the cost of using the truck always be 1 and the cost of using the multimodal route for each container be 0. This means that you want to maximize the number of containers that travel by route to minimize the costs. Since the capacity is 1 for each arc in the universe, it means that the routes that are selected to be in the packing have to be pairwise disjoint. This can be formulated as the decision problem: can at least k subsets be selected such that the selected subsets are pairwise disjoint?

Construct a binary parameter a_{su}, which contains the value 1 if the route for container s uses arc u. This parameter is based on the instance of the set packing problem that was given, because that instance contains the subsets corresponding to the routes of each container. Let the binary variable x_s be 1 if container s uses the multimodal route. The objective is maximize the number of subsets that are selected under the constraints that every element appears at most once in the subsets that are selected in the packing.

This means that the answer to the set packing instance is yes if and only if the solution for the container quantum problem is yes. This shows that the container quantum problem is polynomially reducible to the set packing problem, which implies that the container quantum problem is a NP-complete problem as well.

C Network descriptions

C.1 Random graph

Three different random graphs have been created. Each network is defined for 4 time periods. There is a graph with 15 nodes and 40 edges. The edges of this network are displayed in Table 20. Another graph has been created with 20 nodes and 70 edges and the edges can be seen in Table 21. The last graph that has been created has 25 nodes and 90 edges. The edges are shown in Table 22.

Table 20: Random graph 15 nodes

Begin node	End node	Cost	Capacity
10	5	3	2
5	10	3	3
11	9	1	2
9	11	1	3
1	3	1	2
3	1	2	1
7	9	1	3
9	7	1	1
12	4	2	2
4	12	1	2
3	13	1	1
13	3	3	3
8	3	1	1
3	8	1	1
13	4	1	1
4	13	2	2
4	9	3	3
9	4	1	1
15	7	2	1
7	15	2	2
3	15	2	1
15	3	2	2
14	10	3	1
10	14	3	3
12	6	1	2
6	12	2	2
8	12	2	1
12	8	2	2
12	3	1	2
3	12	1	3
6	14	2	1
14	6	3	2
6	7	3	1
7	6	2	1
10	4	1	1
4	10	2	2
9	6	3	2
6	9	2	1
10	12	2	2
12	10	1	2
2	4	2	3
4	2	3	2
6	5	3	3
5	6	1	2
11	6	2	1
6	11	1	3

Continued on next page

Table 20 – continued from previous page

Begin node	End node	Cost	Capacity
12	5	2	1
5	12	3	1
2	10	3	2
10	2	1	1
5	8	3	2
8	5	1	1
15	11	1	2
11	15	2	1
10	3	3	2
3	10	1	1
14	7	2	1
7	14	2	3
14	3	3	2
3	14	2	3
7	3	1	2
3	7	3	1
11	8	2	1
8	11	1	2
5	9	3	3
9	5	2	1
13	15	2	3
15	13	3	3
1	13	2	2
13	1	3	3
11	5	2	2
5	11	3	3
11	3	3	1
3	11	2	3
13	5	2	3
5	13	2	2
5	14	3	3
14	5	3	2
12	14	2	2
14	12	1	3

Table 21: Random graph 20 nodes

Begin node	End node	Cost	Capacity
18	12	2	1
12	18	1	3
13	19	2	1
19	13	1	3
3	12	3	1
12	3	2	1
11	19	3	3
19	11	1	2
14	20	1	2
20	14	2	2
13	20	1	3
20	13	3	2
10	16	2	2
16	10	1	2
10	5	2	1
5	10	1	3
20	15	1	2

Continued on next page

Table 21 – continued from previous page

Begin node	End node	Cost	Capacity
15	20	1	1
13	1	2	3
1	13	3	2
8	16	1	1
16	8	2	2
20	16	3	3
16	20	3	3
4	1	1	2
1	4	2	3
11	10	3	3
10	11	1	2
12	20	3	3
20	12	3	1
2	1	2	3
1	2	2	3
4	18	1	1
18	4	2	2
20	19	3	3
19	20	3	3
16	13	2	2
13	16	3	3
7	10	2	1
10	7	2	1
14	19	2	3
19	14	2	2
5	6	2	3
6	5	3	2
11	18	1	2
18	11	3	3
2	3	3	1
3	2	2	1
3	1	2	3
1	3	2	1
2	4	1	1
4	2	3	3
4	5	2	3
5	4	2	1
6	20	2	1
20	6	1	3
7	2	3	1
2	7	3	2
3	9	1	3
9	3	3	1
13	5	1	2
5	13	1	2
16	12	3	2
12	16	1	1
19	17	2	3
17	19	3	3
8	9	1	3
9	8	1	2
10	17	1	1
17	10	2	2
13	6	1	3
6	13	1	3

Continued on next page

Table 21 – continued from previous page

Begin node	End node	Cost	Capacity
11	3	3	3
3	11	1	1
4	17	3	2
17	4	1	2
15	16	2	1
16	15	3	1
18	13	1	1
13	18	2	3
8	4	1	3
4	8	2	1
12	8	2	2
8	12	3	3
12	13	2	1
13	12	3	2
10	14	2	3
14	10	1	3
19	7	3	1
7	19	2	1
2	6	1	1
6	2	1	2
1	14	2	2
14	1	3	2
11	15	2	1
15	11	1	1
6	3	2	2
3	6	2	1
18	9	1	2
9	18	1	2
8	2	2	1
2	8	2	2
11	2	3	1
2	11	3	2
16	5	1	3
5	16	2	2
15	8	3	1
8	15	1	1
1	19	1	3
19	1	1	3
10	6	2	2
6	10	2	1
15	6	1	1
6	15	3	1
2	14	3	2
14	2	1	2
2	15	3	2
15	2	1	1
2	18	3	3
18	2	2	1
3	8	2	1
8	3	2	1
2	20	2	2
20	2	2	2
7	6	3	3
6	7	3	2
7	18	1	1

Continued on next page

Table 21 – continued from previous page

Begin node	End node	Cost	Capacity
18	7	2	1
3	18	1	3
18	3	1	3
7	1	2	3
1	7	2	2
13	17	3	1
17	13	3	1
10	15	1	3
15	10	1	1
13	11	1	1
11	13	3	1
3	14	3	3
14	3	1	2

Table 22: Random graph 25 nodes

Begin node	End node	Cost	Capacity
18	16	2	3
16	18	1	1
5	14	3	1
14	5	1	1
2	19	1	2
19	2	3	2
13	15	1	1
15	13	2	1
10	18	3	2
18	10	1	2
12	8	3	1
8	12	3	2
13	12	1	1
12	13	1	2
13	2	1	1
2	13	2	1
25	23	2	1
23	25	2	2
22	16	3	3
16	22	1	1
13	23	1	1
23	13	3	3
19	20	1	2
20	19	1	2
22	1	3	2
1	22	3	2
17	16	3	3
16	17	3	2
15	19	3	3
19	15	3	1
18	5	2	2
5	18	3	1
22	25	3	1
25	22	2	2
3	1	1	2
1	3	3	3
14	10	1	2
10	14	3	1

Continued on next page

Table 22 – continued from previous page

Begin node	End node	Cost	Capacity
3	2	1	3
2	3	2	2
23	24	3	3
24	23	3	1
3	4	1	2
4	3	3	2
13	1	3	2
1	13	2	1
25	21	3	1
21	25	3	3
11	13	2	1
13	11	3	3
23	12	1	1
12	23	1	2
10	9	3	1
9	10	1	1
21	14	1	1
14	21	3	1
19	21	3	3
21	19	3	2
14	8	2	2
8	14	3	3
8	10	2	2
10	8	1	3
11	6	2	1
6	11	1	1
10	19	3	3
19	10	3	1
14	7	2	3
7	14	1	3
23	11	3	1
11	23	1	2
5	17	1	2
17	5	1	2
21	13	1	3
13	21	3	2
4	1	1	3
1	4	1	1
3	11	1	2
11	3	3	1
19	25	2	3
25	19	2	1
23	18	3	3
18	23	1	2
7	11	2	1
11	7	2	3
24	25	1	1
25	24	2	3
20	12	1	3
12	20	1	2
24	3	2	3
3	24	3	2
16	12	3	1
12	16	2	2
5	10	2	3

Continued on next page

Table 22 – continued from previous page

Begin node	End node	Cost	Capacity
10	5	3	1
19	4	2	2
4	19	2	1
17	14	1	1
14	17	2	1
9	22	3	1
22	9	1	2
4	18	2	2
18	4	1	3
13	8	1	2
8	13	2	1
25	5	2	1
5	25	1	2
18	15	2	3
15	18	1	1
7	15	1	2
15	7	3	1
9	20	3	2
20	9	3	1
2	6	2	3
6	2	2	3
22	21	3	3
21	22	3	1
7	1	2	3
1	7	2	2
25	6	2	1
6	25	2	1
14	3	2	1
3	14	2	2
7	9	1	2
9	7	2	2
21	4	1	2
4	21	1	1
22	13	1	2
13	22	1	1
21	1	3	2
1	21	2	3
15	5	2	1
5	15	1	2
20	1	1	1
1	20	2	1
9	5	3	2
5	9	2	1
17	23	3	1
23	17	2	1
10	12	2	3
12	10	3	2
8	19	1	2
19	8	1	3
1	24	2	1
24	1	1	2
22	23	1	1
23	22	1	1
20	8	2	1
8	20	3	1

Continued on next page

Table 22 – continued from previous page

Begin node	End node	Cost	Capacity
2	23	2	2
23	2	1	3
10	15	2	1
15	10	3	1
12	19	3	2
19	12	3	3
20	13	2	2
13	20	1	3
22	3	3	2
3	22	1	3
7	17	3	2
17	7	1	3
12	14	1	2
14	12	2	1
16	1	3	3
1	16	1	3
15	2	1	3
2	15	1	1
11	5	2	2
5	11	2	2
16	5	3	1
5	16	2	2
9	13	1	3
13	9	1	3
9	19	2	1
19	9	3	1
24	12	3	1
12	24	1	2
20	16	3	3
16	20	1	1
19	14	1	2
14	19	3	3

C.2 Dutch graph

The Dutch network is based on the network in Figure 22. Rotterdam and Den Haag have been merged to one node under the name Rotterdam. The network is described in Table 23. The number of the node is displayed in brackets after the city name. Every connection goes both ways. So, the values for Alkmaar(1) and Amsterdam(3) are the same as for Amsterdam(3) and Alkmaar(1). In every time period a service departs with the mentioned properties. The Dutch network is defined for 6 time periods.

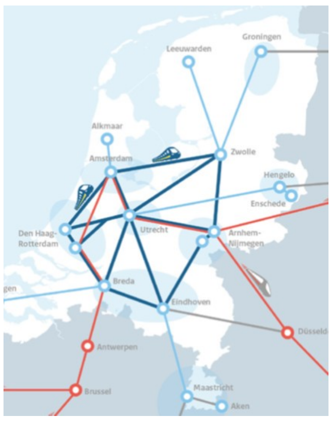


Figure 22: Dutch network[8]

Table 23: Dutch network characteristics

Node 1	Node 2	Travel time	Cost	Capacity
Alkmaar (1)	Amsterdam (3)	1	1	1
Groningen(2)	Zwolle (4)	2	1	1
Amsterdam(3)	Zwolle(4)	2	2	2
Amsterdam(3)	Utrecht (6)	1	3	3
Amsterdam(3)	Rotterdam(8)	2	3	3
Zwolle(4)	Utrecht(6)	2	3	2
Zwolle(4)	Arnhem(7)	2	1	1
Hengelo(5)	Utrecht(6)	3	1	1
Utrecht(6)	Arnhem(7)	1	2	1
Utrecht(6)	Rotterdam(8)	1	3	3
Utrecht(6)	Breda(9)	2	3	2
Utecht(6)	Eindhoven(10)	2	2	2
Arnhem(7)	Eindhoven(10)	3	1	1
Rotterdam(8)	Breda(9)	1	2	2
Breda(9)	Eindhoven(10)	1	2	1
Eindhoven(10)	Maastricht(11)	2	1	1

D Container sets

D.1 Container set random graph 15 nodes

The container sets, that have been used in combination with the randomly generated graph with 15 nodes, are described in Tables 24,25 and 26. The overlap is 25%, 63% and 73% for container set 1,2 and 3 respectively.

Table 24: Random graph 15 nodes, container set 1

Container	Origin	Destination	Release date	Due date	Cost truck
1	8	6	1	4	17
2	8	6	1	4	19
3	8	6	2	4	16
4	8	6	2	4	19
5	3	5	1	4	17
6	3	5	1	4	18
7	3	5	2	4	17
8	3	5	2	4	20
9	2	13	1	4	20
10	2	13	1	4	20
11	2	13	2	4	18
12	2	13	2	4	16
13	10	14	1	4	19
14	10	14	1	4	19
15	10	14	2	4	20
16	10	14	2	4	18
17	5	11	1	4	19
18	5	11	1	4	20
19	5	11	2	4	17
20	5	11	2	4	20

[8] Source: `https://www.eltis.org/discover/news/netherlands-develop-metro-urban-rail-system-handle-40-percent-growth-public-transport`

Table 25: Random graph 15 nodes, container set 2

Container	Origin	Destination	Release date	Due date	Cost truck
1	1	5	1	4	16
2	1	5	1	4	19
3	1	5	1	4	18
4	1	5	1	4	19
5	11	3	1	4	16
6	11	3	1	4	20
7	11	3	1	4	16
8	11	3	1	4	20
9	12	1	1	4	18
10	12	1	1	4	16
11	12	1	1	4	16
12	12	1	1	4	15
13	15	3	2	4	15
14	15	3	2	4	17
15	15	3	2	4	16
16	15	3	2	4	18
17	4	10	2	4	15
18	4	10	2	4	18
19	4	10	2	4	18
20	4	10	2	4	17

Table 26: Random graph 15 nodes, container set 3

Container	Origin	Destination	Release date	Due date	Cost truck
1	7	12	1	4	19
2	7	12	1	4	15
3	7	12	1	4	19
4	7	12	1	4	16
5	7	12	1	4	16
6	7	12	1	4	20
7	7	12	1	4	17
8	7	12	1	4	15
9	7	12	1	4	17
10	7	12	1	4	18
11	5	10	1	4	19
12	5	10	1	4	17
13	5	10	1	4	19
14	5	10	1	4	17
15	5	10	1	4	17
16	5	10	1	4	16
17	5	10	1	4	18
18	5	10	1	4	18
19	5	10	1	4	19
20	5	10	1	4	20

D.2 Container set random graph 20 nodes

The container sets, that have been used in combination with the randomly generated graph with 20 nodes, are described in Tables 27,28 and 29. The overlap is 16%, 44% and 83% for container set 1,2 and 3 respectively.

Table 27: Random graph 20 nodes, container set 1

Container	Origin	Destination	Release date	Due date	Cost truck
1	2	17	1	4	20
2	2	17	1	4	16
3	2	17	2	4	15
4	2	17	2	4	20
5	7	11	1	4	17
6	7	11	1	4	15
7	7	11	2	4	15
8	7	11	2	4	18
9	11	14	1	4	19
10	11	14	1	4	15
11	11	14	2	4	17
12	11	14	2	4	20
13	5	20	1	4	15
14	5	20	1	4	18
15	5	20	2	4	16
16	5	20	2	4	20
17	1	3	1	4	20
18	1	3	1	4	18
19	1	3	2	4	17
20	1	3	2	4	15

Table 28: Random graph 20 nodes, container set 2

Container	Origin	Destination	Release date	Due date	Cost truck
1	13	19	1	4	16
2	13	19	1	4	17
3	13	19	1	4	15
4	13	19	1	4	17
5	6	5	1	4	17
6	6	5	1	4	18
7	6	5	1	4	20
8	6	5	1	4	17
9	10	2	1	4	15
10	10	2	1	4	19
11	10	2	1	4	20
12	10	2	1	4	19
13	1	2	2	4	15
14	1	2	2	4	15
15	1	2	2	4	16
16	1	2	2	4	17
17	14	9	2	4	19
18	14	9	2	4	15
19	14	9	2	4	16
20	14	9	2	4	19

D.3 Container set random graph 25 nodes

The container sets, that have been used in combination with the randomly generated graph with 25 nodes, are described in Tables 30,31 and 32. The overlap is 23%, 53% and 85% for container set 1,2 and

Table 29: Random graph 20 nodes, container set 3

Container	Origin	Destination	Release date	Due date	Cost truck
1	14	5	1	4	17
2	14	5	1	4	16
3	14	5	1	4	17
4	14	5	1	4	17
5	14	5	1	4	15
6	14	5	1	4	16
7	14	5	1	4	19
8	14	5	1	4	19
9	14	5	1	4	16
10	14	5	1	4	20
11	11	12	1	4	18
12	11	12	1	4	16
13	11	12	1	4	18
14	11	12	1	4	19
15	11	12	1	4	18
16	11	12	1	4	16
17	11	12	1	4	18
18	11	12	1	4	15
19	11	12	1	4	20
20	11	12	1	4	16

3 respectively.

Table 30: Random graph 25 nodes, container set 1

Container	Origin	Destination	Release date	Due date	Cost truck
1	20	15	1	4	19
2	20	15	1	4	19
3	20	15	2	4	16
4	20	15	2	4	19
5	23	4	1	4	20
6	23	4	1	4	15
7	23	4	2	4	20
8	23	4	2	4	20
9	19	3	1	4	15
10	19	3	1	4	15
11	19	3	2	4	20
12	19	3	2	4	18
13	10	22	1	4	19
14	10	22	1	4	19
15	10	22	2	4	20
16	10	22	2	4	15
17	21	23	1	4	17
18	21	23	1	4	15
19	21	23	2	4	18
20	21	23	2	4	17

D.4 Container sets Dutch graph

The container sets, that have been used in combination with the network based on the Netherlands, are described in Tables 33,34 and 35. The overlap is 32%, 59% and 69% for container set 1,2 and 3 respectively.

Table 31: Random graph 25 nodes, container set 2

Container	Origin	Destination	Release date	Due date	Cost truck
1	9	16	1	4	17
2	9	16	1	4	17
3	9	16	1	4	15
4	9	16	1	4	17
5	17	23	1	4	18
6	17	23	1	4	18
7	17	23	1	4	18
8	17	23	1	4	16
9	21	7	1	4	17
10	21	7	1	4	17
11	21	7	1	4	19
12	21	7	1	4	16
13	15	3	2	4	16
14	15	3	2	4	17
15	15	3	2	4	15
16	15	3	2	4	20
17	14	16	2	4	19
18	14	16	2	4	19
19	14	16	2	4	20
20	14	16	2	4	18

Table 32: Random graph 25 nodes, container set 3

Container	Origin	Destination	Release date	Due date	Cost truck
1	6	8	1	4	16
2	6	8	1	4	18
3	6	8	1	4	16
4	6	8	1	4	20
5	6	8	1	4	19
6	6	8	1	4	18
7	6	8	1	4	19
8	6	8	1	4	17
9	6	8	1	4	15
10	6	8	1	4	16
11	18	9	1	4	17
12	18	9	1	4	17
13	18	9	1	4	19
14	18	9	1	4	15
15	18	9	1	4	16
16	18	9	1	4	17
17	18	9	1	4	18
18	18	9	1	4	17
19	18	9	1	4	19
20	18	9	1	4	17

Table 33: Dutch container set 1

Container	Origin	Destination	Release date	Due date	Cost truck
1	3	1	1	6	13
2	3	2	1	6	15
3	3	2	1	6	15
4	3	5	2	6	15
5	3	1	1	6	10
6	3	1	1	6	10
7	3	2	1	6	15
8	3	5	2	6	10
9	3	7	1	6	12
10	3	7	1	6	10
11	8	1	1	6	10
12	8	3	1	6	15
13	8	2	1	6	12
14	8	5	1	6	14
15	8	7	1	6	11
16	8	2	1	6	10
17	8	9	2	6	14
18	8	10	2	6	13
19	8	6	1	6	11
20	8	11	1	6	15

Table 34: Dutch container set 2

Container	Origin	Destination	Release date	Due date	Cost truck
1	3	1	1	6	12
2	3	1	1	6	15
3	3	1	1	6	12
4	3	2	1	6	10
5	3	2	1	6	13
6	3	2	1	6	12
7	3	5	1	6	10
8	3	5	1	6	12
9	3	7	1	6	10
10	3	7	1	6	12
11	8	5	1	6	15
12	8	11	1	6	12
13	8	5	1	6	13
14	8	5	1	6	13
15	8	10	1	6	10
16	8	6	1	6	10
17	8	7	1	6	12
18	8	7	1	6	10
19	8	11	1	6	11
20	8	11	1	6	10

Table 35: Dutch container set 3

Container	Origin	Destination	Release date	Due date	Cost truck
1	3	2	1	6	10
2	3	2	1	6	15
3	3	2	1	6	14
4	3	5	1	6	13
5	3	5	1	6	10
6	3	7	1	6	13
7	3	7	1	6	13
8	3	7	1	6	10
9	3	7	1	6	11
10	3	7	1	6	13
11	8	11	1	6	10
12	8	11	1	6	14
13	8	11	1	6	14
14	8	5	1	6	12
15	8	5	1	6	15
16	8	5	1	6	10
17	8	5	1	6	15
18	8	11	1	6	14
19	8	5	1	6	12
20	8	11	1	6	12

D.5 Container set graph 17

The container sets, that have been used in combination with the Chinese network with 17 nodes, are described in Tables 36,37 and 38. The overlap is 30%, 57% and 84% for container set 1,2 and 3 respectively.

Table 36: Graph 17 container set 1

Container	Origin	Destination	Release date	Due date	Cost truck
1	1	15	1	7	15
2	2	15	1	7	18
3	3	15	1	7	18
4	3	12	1	7	15
5	4	12	1	7	16
6	6	12	1	7	16
7	5	17	1	7	15
8	7	17	1	7	19
9	2	17	1	7	19
10	3	17	1	7	15
11	1	15	2	7	15
12	2	15	2	7	20
13	3	15	2	7	20
14	3	12	2	7	19
15	4	12	2	7	18
16	6	12	2	7	18
17	5	17	2	7	15
18	7	17	2	7	17
19	2	17	2	7	17
20	3	17	2	7	19

D.6 Container set graph 30

The container sets, that have been used in combination with the Chinese network with 30 nodes, are described in Tables 39,40 and 38. The overlap is 42%, 56% and 83% for container set 1,2 and 3

Table 37: Graph 17 container set 2

Container	Origin	Destination	Release date	Due date	Cost truck
1	1	15	1	7	18
2	2	15	1	7	18
3	3	15	1	7	15
4	3	12	1	7	16
5	4	12	1	7	19
6	6	12	1	7	16
7	5	17	1	7	18
8	7	17	1	7	20
9	2	17	1	7	20
10	3	17	1	7	16
11	1	15	1	7	16
12	2	15	1	7	17
13	3	15	1	7	20
14	3	12	1	7	19
15	4	12	1	7	15
16	6	12	1	7	16
17	5	17	1	7	17
18	7	17	1	7	15
19	2	17	1	7	20
20	3	17	1	7	18

Table 38: Graph 17 container set 3

Container	Origin	Destination	Release date	Due date	Cost truck
1	1	15	1	7	17
2	1	15	1	7	18
3	1	15	1	7	19
4	1	15	1	7	19
5	1	15	1	7	20
6	1	17	1	7	19
7	1	17	1	7	17
8	1	17	1	7	17
9	1	17	1	7	16
10	1	17	1	7	20
11	1	16	1	7	20
12	1	16	1	7	17
13	1	16	1	7	18
14	1	16	1	7	18
15	1	16	1	7	18
16	1	12	1	7	20
17	1	12	1	7	19
18	1	12	1	7	15
19	1	12	1	7	16
20	1	12	1	7	19

respectively.

Table 39: Graph 30 container set 1

Container	Origin	Destination	Release date	Due date	Cost truck
1	1	30	1	9	16
2	2	30	1	9	18
3	3	30	1	9	20
4	4	30	1	9	20
5	5	30	1	9	15
6	6	30	1	9	18
7	7	30	1	9	16
8	8	30	1	9	16
9	9	30	1	9	16
10	10	30	1	9	16
11	1	12	3	9	20
12	2	12	3	9	15
13	3	12	3	9	20
14	4	12	3	9	18
15	5	12	3	9	17
16	1	26	2	9	19
17	2	26	2	9	18
18	3	26	2	9	20
19	4	26	2	9	16
20	5	26	2	9	20
21	6	26	1	9	18
22	7	26	1	9	15
23	8	26	1	9	17
24	9	26	1	9	19
25	10	26	2	9	17
26	1	30	2	9	20
27	2	30	1	9	17
28	3	30	2	9	20
29	4	30	2	9	15
30	5	30	1	9	19

Table 40: Graph 30 container set 2

Container	Origin	Destination	Release date	Due date	Cost truck
1	1	30	1	9	20
2	2	30	1	9	19
3	3	30	1	9	15
4	4	30	1	9	16
5	5	30	1	9	15
6	6	30	1	9	17
7	7	30	1	9	19
8	8	30	1	9	19
9	9	30	1	9	18
10	10	30	1	9	20
11	1	12	1	9	18
12	2	12	1	9	19
13	3	12	1	9	19
14	4	12	1	9	17
15	5	12	1	9	19
16	1	26	2	9	17
17	2	26	2	9	15
18	3	26	2	9	17
19	4	26	2	9	16
20	5	26	2	9	19
21	1	26	1	9	18
22	2	26	1	9	17
23	3	26	1	9	18
24	4	26	1	9	17
25	5	26	1	9	16
26	6	26	1	9	17
27	7	26	1	9	20
28	8	26	1	9	16
29	9	26	1	9	20
30	10	26	2	9	15
31	1	30	2	9	15
32	2	30	1	9	18
33	3	30	2	9	16
34	4	30	2	9	17
35	5	30	1	9	18
36	6	30	2	9	20
37	7	30	2	9	17
38	8	30	2	9	18
39	9	30	2	9	19
40	10	30	2	9	20
41	1	12	2	9	18
42	2	12	2	9	17
43	3	12	2	9	20
44	4	12	2	9	19
45	5	12	2	9	19
46	6	12	2	9	17
47	7	12	2	9	20
48	8	12	2	9	20
49	9	12	2	9	19
50	10	12	2	9	15

Table 41: Graph 30 container set 3

Container	Origin	Destination	Release date	Due date	Cost truck
1	1	30	1	9	16
2	1	30	1	9	19
3	1	30	1	9	18
4	1	30	1	9	20
5	1	30	1	9	15
6	1	30	1	9	19
7	1	30	1	9	18
8	1	30	1	9	18
9	1	30	1	9	17
10	1	30	1	9	15
11	1	12	1	9	16
12	1	12	1	9	19
13	1	12	1	9	15
14	1	12	1	9	18
15	1	12	1	9	15
16	1	26	2	9	20
17	1	26	2	9	16
18	1	26	2	9	16
19	1	26	2	9	16
20	1	26	2	9	19
21	1	26	1	9	16
22	1	26	1	9	20
23	1	26	1	9	18
24	1	26	1	9	20
25	1	26	1	9	15
26	2	26	1	9	18
27	2	26	1	9	18
28	2	26	1	9	20
29	2	26	1	9	16
30	2	26	2	9	20
31	1	30	2	9	17
32	1	30	2	9	15
33	1	30	2	9	16
34	1	30	2	9	20
35	1	30	2	9	16
36	1	30	2	9	19
37	1	30	2	9	15
38	1	30	2	9	20
39	1	30	2	9	19
40	1	30	2	9	17
41	4	12	2	9	18
42	4	12	2	9	19
43	4	12	2	9	17
44	1	12	2	9	16
45	1	12	2	9	20
46	1	12	2	9	15
47	1	12	2	9	17
48	1	12	2	9	19
49	1	12	2	9	19
50	1	12	2	9	19

E Optimal solutions

E.1 Random graph 15 nodes

The optimal solution value of the random graph with 15 nodes and an overlap of 25% is 58. The total time to find the optimal solution was 5.74 seconds. The optimal paths are described in Table 42.

Table 42: Random graph 15 nodes, container set 1

Container	Optimal path	Cost
Container 1	[0, 8, 20, 36, 61]	2
Container 2	[0, 8, 26, 36, 61]	3
Container 3	[0, 23, 41, 51, 61]	3
Container 4	[0, 23, 42, 51, 61]	3
Container 5	[0, 3, 23, 35, 61]	2
Container 6	[0, 18, 43, 50, 61]	3
Container 7	[0, 18, 42, 50, 61]	3
Container 8	[0, 18, 38, 50, 61]	2
Container 9	[0, 2, 19, 43, 61]	4
Container 10	[0, 2, 19, 43, 61]	4
Container 11	[0, 17, 34, 58, 61]	4
Container 12	[0, 17, 34, 58, 61]	4
Container 13	[0, 40, 59, 61]	3
Container 14	[0, 10, 29, 61]	3
Container 15	[0, 40, 59, 61]	3
Container 16	[0, 40, 59, 61]	3
Container 17	[0, 5, 21, 41, 61]	2
Container 18	[0, 5, 21, 41, 61]	2
Container 19	[0, 20, 41, 61]	3
Container 20	[0, 20, 36, 56, 61]	2

The optimal solution value of the random graph with 15 nodes and an overlap of 63% is 61. The total time to find the optimal solution was 4.63 seconds. The optimal paths are described in Table 43.

Table 43: Random graph 15 nodes, container set 2

Container	Optimal path	Cost
Container 1	[0, 1, 18, 38, 50, 61]	3
Container 2	[0, 16, 43, 50, 61]	4
Container 3	[0, 16, 43, 50, 61]	4
Container 4	[0, 1, 28, 35, 61]	4
Container 5	[0, 11, 18, 61]	3
Container 6	[0, 41, 48, 61]	3
Container 7	[0, 26, 33, 61]	3
Container 8	[0, 26, 38, 48, 61]	3
Container 9	[0, 12, 18, 31, 61]	3
Container 10	[0, 12, 19, 43, 46, 61]	7
Container 11	[0, 27, 33, 46, 61]	3
Container 12	[0, 12, 18, 43, 46, 61]	5
Container 13	[0, 45, 48, 61]	2
Container 14	[0, 30, 33, 61]	2
Container 15	[0, 45, 48, 61]	2
Container 16	[0, 30, 33, 61]	2
Container 17	[0, 34, 55, 61]	2
Container 18	[0, 19, 42, 55, 61]	2
Container 19	[0, 34, 55, 61]	2
Container 20	[0, 19, 42, 55, 61]	2

The optimal solution value of the random graph with 15 nodes and an overlap of 73% is 57. The total time to find the optimal solution was 12.65 seconds. The optimal paths are described in Table 44.

Table 44: Random graph 15 nodes, container set 3

Container	Optimal path	Cost
Container 1	[0, 22, 44, 57, 61]	3
Container 2	[0, 22, 33, 57, 61]	2
Container 3	[0, 7, 18, 42, 61]	2
Container 4	[0, 22, 44, 57, 61]	3
Container 5	[0, 7, 18, 42, 61]	2
Container 6	[0, 7, 29, 42, 61]	3
Container 7	[0, 22, 44, 57, 61]	3
Container 8	[0, 22, 33, 57, 61]	2
Container 9	[0, 7, 29, 42, 61]	3
Container 10	[0, 7, 29, 42, 61]	3
Container 11	[0, 5, 25, 61]	3
Container 12	[0, 5, 27, 40, 61]	4
Container 13	[0, 20, 40, 61]	3
Container 14	[0, 35, 55, 61]	3
Container 15	[0, 5, 25, 61]	3
Container 16	[0, 35, 55, 61]	3
Container 17	[0, 20, 40, 61]	3
Container 18	[0, 35, 55, 61]	3
Container 19	[0, 20, 40, 61]	3
Container 20	[0, 5, 25, 61]	3

E.2 Random graph 20 nodes

The optimal solution value of the random graph with 20 nodes and an overlap of 16% is 76. The total time to find the optimal solution was 10.33 seconds. The optimal paths are described in Table 45.

Table 45: Random graph 20 nodes, container set 1

Container	Optimal path	Cost
Container 1	[0, 2, 26, 50, 77, 81]	4
Container 2	[0, 2, 24, 57, 81]	4
Container 3	truck	15
Container 4	[0, 22, 44, 77, 81]	4
Container 5	[0, 7, 30, 51, 81]	3
Container 6	[0, 7, 39, 51, 81]	3
Container 7	[0, 27, 50, 71, 81]	3
Container 8	[0, 27, 59, 71, 81]	3
Container 9	[0, 31, 50, 74, 81]	5
Container 10	[0, 11, 39, 54, 81]	5
Container 11	[0, 31, 50, 74, 81]	5
Container 12	[0, 31, 59, 74, 81]	5
Container 13	[0, 5, 33, 60, 81]	2
Container 14	[0, 5, 33, 60, 81]	2
Container 15	[0, 25, 53, 80, 81]	2
Container 16	[0, 25, 53, 80, 81]	2
Container 17	[0, 1, 34, 43, 81]	3
Container 18	[0, 1, 23, 81]	2
Container 19	[0, 21, 43, 81]	2
Container 20	[0, 41, 63, 81]	2

The optimal solution value of the random graph with 20 nodes and an overlap of 44% is 70. The total time to find the optimal solution was 16.44 seconds. The optimal paths are described in Table 46.

The optimal solution value of the random graph with 20 nodes and an overlap of 83% is 98. The total time to find the optimal solution was 12.74 seconds. The optimal paths are described in Table 47.

Table 46: Random graph 20 nodes, container set 2

Container	Optimal path	Cost
Container 1	[0, 53, 79, 81]	2
Container 2	[0, 33, 59, 81]	2
Container 3	[0, 13, 39, 81]	2
Container 4	[0, 13, 21, 59, 81]	3
Container 5	[0, 26, 53, 65, 81]	2
Container 6	[0, 26, 53, 65, 81]	2
Container 7	[0, 6, 33, 45, 81]	2
Container 8	[0, 6, 33, 45, 81]	2
Container 9	[0, 30, 54, 62, 81]	3
Container 10	[0, 30, 55, 62, 81]	2
Container 11	[0, 10, 35, 42, 81]	2
Container 12	[0, 10, 34, 42, 81]	3
Container 13	[0, 41, 62, 81]	2
Container 14	[0, 21, 42, 81]	2
Container 15	[0, 41, 62, 81]	2
Container 16	[0, 21, 42, 81]	2
Container 17	[0, 34, 43, 69, 81]	2
Container 18	truck	15
Container 19	truck	16
Container 20	[0, 34, 43, 69, 81]	2

Table 47: Random graph 20 nodes, container set 3

Container	Optimal path	Cost
Container 1	[0, 34, 50, 65, 81]	3
Container 2	[0, 14, 39, 53, 65, 81]	4
Container 3	[0, 14, 30, 56, 65, 81]	4
Container 4	[0, 14, 30, 56, 65, 81]	4
Container 5	[0, 14, 39, 53, 65, 81]	4
Container 6	[0, 14, 40, 56, 65, 81]	5
Container 7	[0, 14, 30, 45, 81]	3
Container 8	[0, 14, 22, 44, 65, 81]	4
Container 9	[0, 14, 22, 46, 65, 81]	5
Container 10	[0, 14, 40, 46, 65, 81]	5
Container 11	[0, 11, 38, 53, 72, 81]	5
Container 12	[0, 11, 23, 48, 72, 81]	8
Container 13	[0, 31, 58, 72, 81]	3
Container 14	[0, 11, 33, 52, 81]	6
Container 15	[0, 11, 23, 52, 81]	6
Container 16	[0, 11, 22, 48, 72, 81]	8
Container 17	[0, 31, 53, 72, 81]	6
Container 18	[0, 11, 38, 52, 81]	3
Container 19	[0, 11, 35, 60, 72, 81]	6
Container 20	[0, 31, 43, 72, 81]	6

E.3 Random graph 25 nodes

The optimal solution value of the random graph with 25 nodes and an overlap of 23% is 86. The total time to find the optimal solution was 18.45 seconds. The optimal paths are described in Table 48.

Table 48: Random graph 25 nodes, container set 1

Container	Optimal path	Cost
Container 1	[0, 20, 38, 65, 101]	3
Container 2	[0, 20, 44, 65, 101]	4
Container 3	[0, 45, 63, 90, 101]	3
Container 4	[0, 45, 69, 90, 101]	4
Container 5	[0, 23, 43, 54, 101]	4
Container 6	[0, 48, 68, 79, 101]	4
Container 7	[0, 48, 68, 79, 101]	4
Container 8	[0, 48, 68, 79, 101]	4
Container 9	[0, 19, 39, 53, 101]	3
Container 10	[0, 19, 29, 53, 101]	5
Container 11	[0, 44, 64, 78, 101]	3
Container 12	[0, 44, 52, 78, 101]	5
Container 13	[0, 10, 37, 63, 97, 101]	4
Container 14	[0, 10, 37, 73, 97, 101]	4
Container 15	[0, 35, 59, 97, 101]	6
Container 16	truck	15
Container 17	[0, 21, 38, 73, 101]	2
Container 18	[0, 21, 38, 62, 98, 101]	3
Container 19	[0, 46, 72, 98, 101]	4
Container 20	[0, 46, 63, 98, 101]	2

The optimal solution value of the random graph with 25 nodes and an overlap of 53% is 113. The total time to find the optimal solution was 12.35 seconds. The optimal paths are described in Table 49.

Table 49: Random graph 25 nodes, container set 2

Container	Optimal path	Cost
Container 1	[0, 9, 38, 62, 91, 101]	4
Container 2	[0, 9, 30, 66, 101]	5
Container 3	[0, 34, 55, 91, 101]	5
Container 4	[0, 9, 30, 66, 101]	5
Container 5	[0, 42, 73, 101]	3
Container 6	[0, 67, 98, 101]	3
Container 7	[0, 17, 48, 101]	3
Container 8	[0, 17, 30, 61, 98, 101]	4
Container 9	[0, 21, 39, 57, 101]	3
Container 10	[0, 21, 29, 51, 82, 101]	4
Container 11	[0, 21, 29, 51, 82, 101]	4
Container 12	[0, 46, 64, 82, 101]	3
Container 13	truck	16
Container 14	[0, 40, 52, 78, 101]	3
Container 15	truck	15
Container 16	[0, 40, 52, 78, 101]	3
Container 17	[0, 39, 62, 91, 101]	4
Container 18	[0, 39, 55, 91, 101]	3
Container 19	[0, 39, 67, 91, 101]	5
Container 20	truck	18

The optimal solution value of the random graph with 25 nodes and an overlap of 85% is 147. The total time to find the optimal solution was 19.56 seconds. The optimal paths are described in Table 50.

Table 50: Random graph 25 nodes, container set 3

Container	Optimal path	Cost
Container 1	truck	16
Container 2	[0, 6, 50, 69, 83, 101]	5
Container 3	truck	16
Container 4	[0, 6, 27, 69, 83, 101]	4
Container 5	[0, 6, 36, 63, 83, 101]	4
Container 6	[0, 6, 27, 69, 83, 101]	4
Container 7	[0, 6, 27, 63, 83, 101]	5
Container 8	truck	17
Container 9	truck	15
Container 10	truck	16
Container 11	[0, 18, 41, 72, 84, 101]	4
Container 12	[0, 18, 48, 72, 84, 101]	3
Container 13	[0, 18, 40, 57, 84, 101]	6
Container 14	[0, 43, 60, 84, 101]	4
Container 15	[0, 18, 35, 59, 101]	4
Container 16	[0, 18, 40, 63, 84, 101]	5
Container 17	[0, 18, 48, 63, 84, 101]	5
Container 18	[0, 43, 55, 84, 101]	4
Container 19	[0, 18, 30, 59, 101]	4
Container 20	[0, 18, 41, 70, 84, 101]	6

E.4 Dutch graph

The optimal solution value of the graph based on the Netherlands and an overlap of 32% is 96. The total time to find the optimal solution was 3.47 seconds. The optimal paths are described in Table 51.

Table 51: Dutch graph, container set 1

Container	Optimal path	Cost
Container 1	[0, 47, 56, 67]	1
Container 2	[0, 3, 26, 46, 67]	3
Container 3	truck	15
Container 4	[0, 14, 28, 38, 67]	4
Container 5	[0, 25, 34, 67]	1
Container 6	[0, 14, 23, 67]	1
Container 7	[0, 14, 37, 57, 67]	3
Container 8	truck	10
Container 9	[0, 14, 37, 62, 67]	3
Container 10	[0, 3, 26, 51, 67]	3
Container 11	[0, 19, 36, 45, 67]	4
Container 12	[0, 41, 58, 67]	3
Container 13	truck	12
Container 14	[0, 8, 17, 27, 67]	4
Container 15	[0, 41, 50, 62, 67]	5
Container 16	truck	10
Container 17	[0, 19, 31, 67]	2
Container 18	[0, 41, 53, 65, 67]	4
Container 19	[0, 19, 28, 67]	3
Container 20	[0, 8, 20, 32, 44, 67]	5

The optimal solution value of the graph based on the Netherlands and an overlap of 59% is 105. The total time to find the optimal solution was 3.74 seconds. The optimal paths are described in Table 52.

The optimal solution value of the graph based on the Netherlands and an overlap of 85% is 148. The total time to find the optimal solution was 2.63 seconds. The optimal paths are described in Table 53.

Table 52: Dutch graph, container set 2

Container	Optimal path	Cost
Container 1	[0, 14, 23, 67]	1
Container 2	[0, 47, 56, 67]	1
Container 3	[0, 3, 12, 67]	1
Container 4	truck	10
Container 5	[0, 3, 26, 46, 67]	3
Container 6	[0, 14, 37, 57, 67]	3
Container 7	truck	10
Container 8	truck	12
Container 9	[0, 14, 37, 62, 67]	3
Container 10	[0, 3, 26, 51, 67]	3
Container 11	[0, 19, 28, 38, 67]	4
Container 12	[0, 19, 31, 43, 55, 67]	5
Container 13	[0, 8, 17, 27, 67]	4
Container 14	truck	13
Container 15	[0, 41, 53, 65, 67]	4
Container 16	[0, 52, 61, 67]	3
Container 17	[0, 30, 39, 51, 67]	5
Container 18	[0, 41, 50, 62, 67]	5
Container 19	[0, 8, 20, 32, 44, 67]	5

Table 53: Dutch graph, container set 3

Container	Optimal path	Cost
Container 1	truck	10
Container 2	[0, 3, 26, 46, 67]	3
Container 3	[0, 14, 37, 57, 67]	3
Container 4	truck	13
Container 5	truck	10
Container 6	[0, 3, 26, 51, 67]	3
Container 7	[0, 25, 39, 51, 67]	5
Container 8	[0, 36, 50, 62, 67]	5
Container 9	[0, 3, 17, 29, 67]	5
Container 10	[0, 14, 37, 62, 67]	3
Container 11	truck	10
Container 12	[0, 19, 31, 43, 55, 67]	5
Container 13	truck	14
Container 14	truck	12
Container 15	[0, 19, 28, 38, 67]	4
Container 16	truck	10
Container 17	[0, 8, 17, 27, 67]	4
Container 18	[0, 8, 20, 32, 44, 67]	5
Container 19	truck	12
Container 20	truck	12

E.5 Graph 17

The optimal solution value of the graph based on a part of the Chinese network with 17 nodes and an overlap of 30% is 144. The total time to find the optimal solution was 110.25 seconds. The optimal paths are described in Table 54.

Table 54: Graph 17, container set 1

Container	Optimal path	Cost
Container 1	[0, 1, 21, 44, 66, 120]	6
Container 2	[0, 2, 25, 45, 65, 83, 120]	8
Container 3	[0, 20, 40, 61, 83, 120]	6
Container 4	[0, 3, 19, 42, 63, 120]	6
Container 5	[0, 4, 18, 36, 56, 75, 97, 120]	10
Container 6	[0, 6, 26, 45, 59, 80, 120]	8
Container 7	[0, 5, 24, 46, 67, 85, 120]	8
Container 8	[0, 7, 29, 50, 68, 120]	6
Container 9	[0, 2, 20, 43, 65, 85, 120]	8
Container 10	[0, 3, 26, 48, 68, 120]	6
Container 11	[0, 35, 55, 78, 100, 120]	6
Container 12	[0, 36, 54, 74, 95, 117, 120]	8
Container 13	[0, 54, 77, 99, 117, 120]	6
Container 14	[0, 54, 70, 93, 114, 120]	6
Container 15	[0, 21, 35, 53, 73, 92, 114, 120]	10
Container 16	[0, 23, 37, 53, 76, 97, 120]	8
Container 17	[0, 39, 58, 80, 101, 119, 120]	8
Container 18	[0, 41, 63, 84, 102, 120]	6
Container 19	[0, 36, 59, 79, 99, 119, 120]	8
Container 20	[0, 37, 60, 82, 102, 120]	6

The optimal solution value of the graph based on a part of the Chinese network with 17 nodes and an overlap of 57% is 144. The total time to find the optimal solution was 153.20 seconds. The optimal paths are described in Table 55.

Table 55: Graph 17, container set 2

Container	Optimal path	Cost
Container 1	[0, 35, 55, 78, 100, 120]	6
Container 2	[0, 36, 59, 79, 99, 117, 120]	8
Container 3	[0, 20, 40, 61, 83, 120]	6
Container 4	[0, 37, 53, 73, 92, 114, 120]	8
Container 5	[0, 21, 35, 53, 76, 97, 120]	8
Container 6	[0, 6, 20, 36, 56, 75, 97, 120]	10
Container 7	[0, 5, 25, 45, 65, 85, 120]	8
Container 8	[0, 7, 29, 50, 68, 120]	6
Container 9	[0, 36, 54, 77, 99, 119, 120]	8
Container 10	[0, 3, 26, 48, 68, 120]	6
Container 11	[0, 1, 21, 44, 66, 120]	6
Container 12	[0, 2, 20, 43, 65, 83, 120]	8
Container 13	[0, 54, 74, 95, 117, 120]	6
Container 14	[0, 3, 19, 42, 63, 120]	6
Container 15	[0, 38, 52, 70, 93, 114, 120]	8
Container 16	[0, 6, 26, 45, 59, 80, 120]	8
Container 17	[0, 22, 41, 63, 84, 102, 120]	8
Container 18	[0, 58, 80, 101, 119, 120]	6
Container 19	[0, 2, 25, 46, 67, 85, 120]	8
Container 20	[0, 37, 60, 82, 102, 120]	6

The optimal solution value of the graph based on a part of the Chinese network with 17 nodes and an overlap of 84% is 209. The total time to find the optimal solution was 188.02 seconds. The optimal paths are described in Table 56.

Table 56: Graph 17, container set 3

Container	Optimal path	Cost
Container 1	[0, 1, 20, 43, 65, 83, 120]	7
Container 2	[0, 35, 55, 78, 100, 120]	6
Container 3	[0, 18, 38, 61, 83, 120]	6
Container 4	[0, 52, 72, 95, 117, 120]	6
Container 5	[0, 1, 21, 44, 66, 120]	6
Container 6	[0, 18, 37, 60, 82, 102, 120]	7
Container 7	truck	17
Container 8	truck	17
Container 9	truck	16
Container 10	[0, 35, 54, 77, 99, 119, 120]	7
Container 11	[0, 1, 19, 42, 63, 84, 120]	8
Container 12	truck	17
Container 13	[0, 35, 53, 76, 97, 118, 120]	8
Container 14	truck	18
Container 15	truck	18
Container 16	[0, 18, 37, 53, 73, 92, 114, 120]	9
Container 17	[0, 18, 36, 56, 75, 97, 120]	8
Container 18	truck	15
Container 19	[0, 1, 20, 36, 59, 80, 120]	7
Container 20	[0, 52, 70, 93, 114, 120]	6

E.6 Graph 30

The optimal solution value of the graph based on a part of the Chinese network with 30 nodes and an overlap of 42% is 380. The total time to find the optimal solution was 2311.85 seconds. The optimal paths are described in Table 57.

The optimal solution value of the graph based on a part of the Chinese network with 30 nodes and an overlap of 56% is 610. The total time to find the optimal solution was 12002.85 seconds. The optimal paths are described in Table 58.

The optimal solution value of the graph based on a part of the Chinese network with 30 nodes and an overlap of 83% is 707. The total time to find the optimal solution was 9288.02 seconds. The optimal paths are described in Table 59.

Table 57: Graph 30, container set 1

Container	Optimal path	Cost
Container 1	truck	19
Container 2	truck	17
Container 3	[0, 3, 39, 74, 107, 142, 174, 209, 240, 271]	14
Container 4	truck	17
Container 5	truck	20
Container 6	[0, 6, 40, 75, 108, 145, 179, 210, 271]	12
Container 7	[0, 37, 72, 106, 139, 171, 204, 239, 270, 271]	14
Container 8	truck	19
Container 9	truck	15
Container 10	[0, 10, 45, 78, 115, 149, 180, 271]	10
Container 11	[0, 91, 122, 158, 192, 271]	6
Container 12	[0, 62, 98, 132, 271]	4
Container 13	[0, 123, 152, 188, 222, 271]	6
Container 14	[0, 124, 151, 182, 218, 252, 271]	8
Container 15	[0, 185, 217, 252, 271]	4
Container 16	[0, 61, 92, 128, 162, 196, 230, 266, 271]	12
Container 17	[0, 32, 68, 102, 136, 170, 206, 271]	10
Container 18	[0, 33, 69, 104, 137, 172, 204, 237, 266, 271]	14
Container 19	[0, 34, 70, 105, 138, 174, 203, 236, 271]	11
Container 20	[0, 65, 97, 132, 166, 200, 236, 271]	10
Container 21	truck	15
Container 22	[0, 7, 42, 76, 110, 146, 271]	8
Container 23	[0, 8, 41, 73, 106, 140, 176, 271]	10
Container 24	[0, 9, 44, 77, 112, 144, 173, 206, 271]	11
Container 25	[0, 100, 135, 168, 204, 233, 266, 271]	9
Container 26	truck	16
Container 27	truck	18
Container 28	truck	17
Container 29	truck	16
Container 30	truck	18

Table 58: Graph 30, container set 2

Container	Optimal path	Cost
Container 1	truck	19
Container 2	truck	17
Container 3	[0, 3, 39, 74, 107, 142, 174, 209, 240, 271]	14
Container 4	truck	17
Container 5	truck	20
Container 6	[0, 6, 40, 75, 108, 144, 179, 210, 271]	12
Container 7	truck	20
Container 8	[0, 8, 41, 73, 106, 139, 171, 204, 239, 270, 271]	16
Container 9	truck	15
Container 10	[0, 10, 45, 78, 115, 149, 180, 271]	10
Container 11	[0, 91, 122, 158, 192, 271]	6
Container 12	[0, 2, 35, 67, 102, 271]	6
Container 13	[0, 3, 32, 68, 102, 271]	6
Container 14	[0, 4, 31, 62, 98, 132, 271]	8
Container 15	[0, 155, 187, 222, 271]	4
Container 16	truck	17
Container 17	truck	18
Container 18	truck	16
Container 19	[0, 34, 70, 105, 138, 174, 203, 236, 271]	11
Container 20	[0, 95, 127, 162, 196, 230, 266, 271]	10
Container 21	truck	15
Container 22	[0, 2, 38, 71, 103, 136, 170, 206, 271]	12
Container 23	truck	16
Container 24	truck	15
Container 25	[0, 65, 97, 132, 166, 200, 236, 271]	10
Container 26	[0, 36, 69, 104, 137, 172, 204, 237, 266, 271]	14
Container 27	[0, 7, 42, 76, 110, 146, 271]	8
Container 28	[0, 8, 42, 72, 106, 140, 176, 271]	9
Container 29	[0, 9, 44, 77, 112, 144, 173, 206, 271]	11
Container 30	[0, 100, 135, 168, 204, 233, 266, 271]	9
Container 31	truck	16
Container 32	truck	15
Container 33	truck	16
Container 34	truck	19
Container 35	truck	19
Container 36	truck	18
Container 37	truck	16
Container 38	truck	16
Container 39	truck	15
Container 40	truck	15
Container 41	[0, 151, 182, 218, 252, 271]	6
Container 42	[0, 152, 188, 222, 271]	4
Container 43	[0, 63, 92, 125, 157, 192, 271]	8
Container 44	[0, 34, 61, 92, 128, 162, 271]	8
Container 45	[0, 185, 217, 252, 271]	4
Container 46	[0, 96, 129, 164, 197, 226, 252, 271]	10
Container 47	[0, 37, 72, 271]	2
Container 48	[0, 38, 72, 271]	2
Container 49	[0, 69, 101, 133, 166, 192, 271]	8
Container 50	[0, 40, 66, 99, 134, 167, 196, 222, 271]	12

Table 59: Graph 30, container set 3

Container	Optimal path	Cost
Container 1	truck	19
Container 2	truck	17
Container 3	truck	20
Container 4	truck	17
Container 5	[0, 31, 64, 100, 135, 168, 204, 239, 270, 271]	14
Container 6	[0, 1, 34, 70, 105, 138, 175, 209, 240, 271]	14
Container 7	truck	20
Container 8	truck	19
Container 9	truck	15
Container 10	truck	18
Container 11	[0, 61, 93, 129, 164, 197, 226, 252, 271]	11
Container 12	[0, 1, 32, 68, 102, 271]	6
Container 13	truck	15
Container 14	[0, 1, 33, 62, 95, 127, 162, 271]	9
Container 15	[0, 121, 152, 185, 217, 252, 271]	8
Container 16	truck	17
Container 17	truck	18
Container 18	truck	16
Container 19	truck	18
Container 20	[0, 31, 63, 99, 134, 167, 196, 230, 266, 271]	13
Container 21	truck	15
Container 22	truck	19
Container 23	truck	16
Container 24	truck	15
Container 25	[0, 1, 33, 69, 104, 137, 172, 204, 233, 266, 271]	14
Container 26	truck	16
Container 27	[0, 2, 38, 72, 106, 140, 176, 271]	10
Container 28	[0, 2, 35, 67, 102, 136, 170, 206, 271]	12
Container 29	truck	16
Container 30	[0, 32, 65, 97, 132, 166, 200, 236, 271]	12
Container 31	truck	16
Container 32	truck	15
Container 33	truck	16
Container 34	truck	19
Container 35	truck	19
Container 36	truck	18
Container 37	truck	16
Container 38	truck	16
Container 39	truck	15
Container 40	truck	15
Container 41	truck	16
Container 42	truck	16
Container 43	[0, 34, 66, 99, 131, 163, 196, 222, 271]	12
Container 44	[0, 61, 93, 122, 155, 187, 222, 271]	9
Container 45	[0, 151, 182, 218, 252, 271]	6
Container 46	[0, 91, 123, 152, 188, 222, 271]	7
Container 47	[0, 31, 63, 92, 125, 157, 192, 271]	9
Container 48	[0, 91, 122, 158, 192, 271]	6
Container 49	[0, 31, 62, 98, 132, 271]	6
Container 50	[0, 61, 92, 128, 162, 271]	6

F Values for every possible combination of methods

For every combination of methods the average optimality gap of the instances has been determined. Based on the optimality gaps of the instances, also the standard deviation of the gaps could be determined. Next to the optimality gaps, the average computation time is determined for every combination of methods. These values are displayed in Table 60. The methods column represents which combination of methods was used. Methods 1, 1, 1, 1 means that the first initialization, the first selection, the first cross-over and the first mutation methods have been used. The average computation time is given in seconds. The combination of methods with the best average optimality gap is highlighted.

Table 60: Values for every combination of methods

Methods	Average optimality gap (standard deviation)	Average computation time
1, 1, 1, 1	35 (24)	95.58
1, 1, 1, 2	32 (20)	83.48
1, 1, 1, 3	32 (26)	56.40
1, 1, 2, 1	38 (27)	92.25
1, 1, 2, 2	34 (23)	83.56
1, 1, 2, 3	29 (22)	52.42
1, 1, 3, 1	33 (26)	92.19
1, 1, 3, 2	30 (22)	82.71
1, 1, 3, 3	27 (18)	53.60
1, 2, 1, 1	33 (25)	91.87
1, 2, 1, 2	37 (20)	84.80
1, 2, 1, 3	33 (26)	55.41
1, 2, 2, 1	36 (24)	90.68
1, 2, 2, 2	33 (23)	83.10
1, 2, 2, 3	30 (23)	52.32
1, 2, 3, 1	39 (25)	93.95
1, 2, 3, 2	36 (28)	84.43
1, 2, 3, 3	29 (23)	54.37
1, 3, 1, 1	36 (24)	91.75
1, 3, 1, 2	34 (23)	84.02
1, 3, 1, 3	34 (25)	54.11
1, 3, 2, 1	40 (23)	93.93
1, 3, 2, 2	33 (25)	83.18
1, 3, 2, 3	34 (29)	53.21
1, 3, 3, 1	63 (101)	92.39
1, 3, 3, 2	32 (19)	84.57
1, 3, 3, 3	31 (28)	54.40
2, 1, 1, 1	34 (16)	97.67
2, 1, 1, 2	35 (21)	82.51
2, 1, 1, 3	30 (15)	51.91
2, 1, 2, 1	34 (14)	92.42
2, 1, 2, 2	39 (14)	82.13
2, 1, 2, 3	36 (22)	51.70
2, 1, 3, 1	34 (21)	91.74
2, 1, 3, 2	35 (15)	82.83
2, 1, 3, 3	29 (16)	53.39
2, 2, 1, 1	35 (17)	90.49
2, 2, 1, 2	36 (17)	82.59
2, 2, 1, 3	32 (15)	55.59
2, 2, 2, 1	39 (16)	91.42
2, 2, 2, 2	38 (20)	82.85
2, 2, 2, 3	30 (16)	52.07
2, 2, 3, 1	38 (17)	93.56
2, 2, 3, 2	35 (15)	83.81
2, 2, 3, 3	30 (13)	53.29
Continued on next page		

Table 60 – continued from previous page

Methods	Average optimality gap (standard deviation)	Average computation time
2, 3, 1, 1	38 (19)	91.81
2, 3, 1, 2	36 (18)	82.16
2, 3, 1, 3	31 (14)	52.79
2, 3, 2, 1	38 (23)	90.45
2, 3, 2, 2	37 (19)	82.85
2, 3, 2, 3	34 (17)	54.40
2, 3, 3, 1	37 (21)	92.77
2, 3, 3, 2	35 (17)	83.78
2, 3, 3, 3	30 (14)	54.06
3, 1, 1, 1	34 (18)	94.30
3, 1, 1, 2	36 (15)	83.54
3, 1, 1, 3	59 (101)	52.73
3, 1, 2, 1	33 (17)	90.65
3, 1, 2, 2	32 (16)	83.23
3, 1, 2, 3	29 (17)	52.94
3, 1, 3, 1	35 (16)	91.72
3, 1, 3, 2	32 (14)	82.14
3, 1, 3, 3	31 (14)	55.37
3, 2, 1, 1	36 (22)	91.14
3, 2, 1, 2	34 (17)	82.26
3, 2, 1, 3	30 (14)	52.61
3, 2, 2, 1	34 (16)	90.94
3, 2, 2, 2	36 (17)	83.49
3, 2, 2, 3	30 (14)	53.40
3, 2, 3, 1	37 (20)	93.42
3, 2, 3, 2	38 (17)	85.03
3, 2, 3, 3	30 (15)	54.33
3, 3, 1, 1	37 (22)	90.67
3, 3, 1, 2	36 (16)	83.82
3, 3, 1, 3	56 (102)	51.90
3, 3, 2, 1	32 (15)	91.18
3, 3, 2, 2	36 (18)	191.54
3, 3, 2, 3	32 (17)	53.73
3, 3, 3, 1	34 (16)	92.40
3, 3, 3, 2	34 (19)	83.23
3, 3, 3, 3	30 (15)	54.47
4, 1, 1, 1	34 (18)	92.39
4, 1, 1, 2	36 (15)	83.65
4, 1, 1, 3	59 (101)	52.74
4, 1, 2, 1	33 (17)	90.65
4, 1, 2, 2	32 (16)	83.26
4, 1, 2, 3	29 (17)	53.07
4, 1, 3, 1	35 (16)	91.84
4, 1, 3, 2	32 (14)	82.63
4, 1, 3, 3	31 (14)	53.06
4, 2, 1, 1	36 (22)	91.20
4, 2, 1, 2	34 (17)	82.35
4, 2, 1, 3	30 (14)	52.64
4, 2, 2, 1	34 (16)	91.26
4, 2, 2, 2	36 (17)	84.00
4, 2, 2, 3	30 (14)	55.49
4, 2, 3, 1	37 (20)	95.55
4, 2, 3, 2	38 (17)	84.49
4, 2, 3, 3	30 (15)	53.98
4, 3, 1, 1	37 (22)	90.31

Continued on next page

Table 60 – continued from previous page

Methods	Average optimality gap (standard deviation)	Average computation time
4, 3, 1, 2	36 (16)	83.95
4, 3, 1, 3	56 (102)	52.70
4, 3, 2, 1	32 (15)	101.34
4, 3, 2, 2	36 (18)	84.60
4, 3, 2, 3	32 (17)	53.64
4, 3, 3, 1	34 (16)	92.28
4, 3, 3, 2	34 (19)	83.20
4, 3, 3, 3	30 (15)	54.75

G Comparing number of pairs and iterations

Different values have been tried for the number of populations, the number of pairs and the number of iterations. The average optimality gap, standard deviation of the optimality gaps and the average computation times have been determined and are shown in Table 61. The n_{pop} value describes the number of populations. The n_{pairs} value represents the number of pairs and the n_{iter} value corresponds to the number of iterations. The average computation time is given in seconds. The combinations with the best average optimality gap are highlighted.

Table 61: Comparing number of populations, pairs and iterations

n_{pop}, n_{pairs}, n_{iter}	Average optimality gap (standard deviation)	Average computation time
20, 5, 15	32 (24)	21.90
20, 5, 20	30 (23)	26.08
20, 5, 25	29 (23)	29.98
20, 8, 15	31 (22)	32.85
20, 8, 20	28 (20)	41.55
20, 8, 25	26 (19)	49.46
20, 10, 15	28 (21)	41.27
20, 10, 20	26 (20)	52.32
20, 10, 25	25 (18)	63.86
30, 7, 15	32 (24)	31.80
30, 7, 20	31 (23)	39.18
30, 7, 25	26 (19)	46.43
30, 11, 15	32 (28)	48.19
30, 11, 20	28 (25)	60.96
30, 11, 25	27 (25)	73.57
30, 15, 15	28 (22)	63.27
30, 15, 20	25 (19)	81.71
30, 15, 25	23 (17)	99.33
40, 10, 15	30 (22)	46.16
40, 10, 20	28 (22)	58.22
40, 10, 25	24 (16)	69.16
40, 15, 15	26 (16)	65.18
40, 15, 20	24 (16)	83.09
40, 15, 25	23 (15)	100.46
40, 20, 15	27 (19)	86.20
40, 20, 20	25 (18)	111.59
40, 20, 25	23 (15)	135.73

H Average optimality gaps for different annealing parameters

The times are given in seconds.

H.1 Random network 15 nodes container set 1

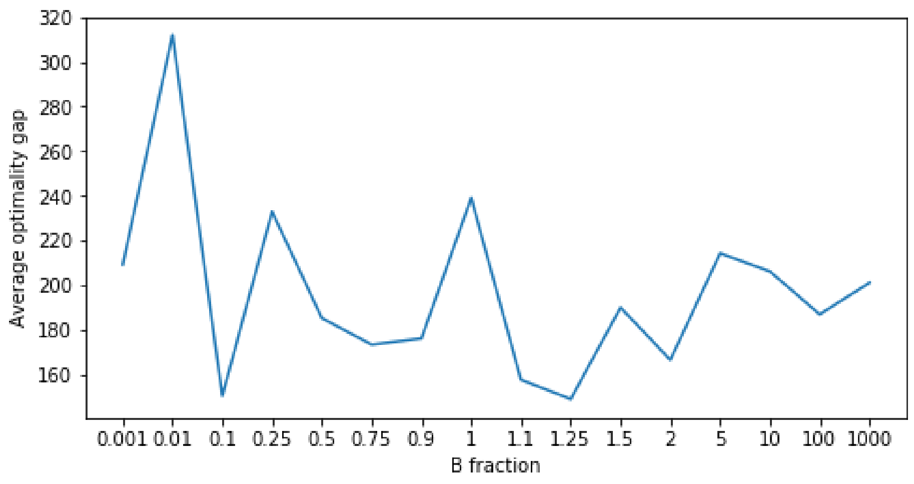


Figure 23: Parameter tune of the W_B value

Table 62: Parameter tune of the W_B value

W_B rate	Average optimality gap (standard deviation)	Anneal time	Annealer time
0.001	209 (0.59)	0.14	0.07
0.01	312 (1.86)	0.08	0.05
0.1	150 (2.01)	0.08	0.06
0.25	233 (2.21)	0.1	0.07
0.5	185 (2.16)	0.09	0.06
0.75	173 (2.09)	0.09	0.06
0.9	176 (2.04)	0.1	0.07
1	239 (2.2)	0.1	0.07
1.1	158 (2.01)	0.09	0.06
1.25	149 (2.0)	0.09	0.06
1.5	190 (2.14)	0.09	0.07
2	166 (2.08)	0.09	0.06
5	214 (2.17)	0.09	0.07
10	206 (2.15)	0.09	0.06
100	187 (2.11)	0.09	0.06
1000	201 (2.14)	0.1	0.07

H.2 Random network 15 nodes container set 2

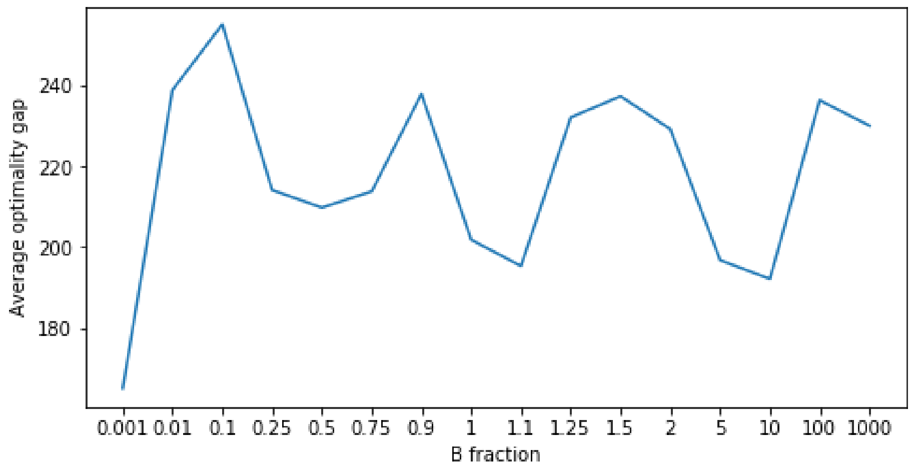


Figure 24: Parameter tune of the W_B value

Table 63: Parameter tune of the W_B value

W_B rate	Average optimality gap (standard deviation)	Anneal time	Annealer time
0.001	165 (0.58)	0.14	0.07
0.01	239 (0.66)	0.14	0.08
0.1	255 (1.84)	0.11	0.08
0.25	214 (1.87)	0.11	0.07
0.5	210 (1.87)	0.12	0.08
0.75	214 (1.87)	0.12	0.08
0.9	238 (1.82)	0.12	0.08
1	202 (1.88)	0.11	0.08
1.1	195 (1.81)	0.12	0.08
1.25	232 (1.93)	0.11	0.08
1.5	237 (1.81)	0.12	0.08
2	229 (1.84)	0.12	0.08
5	197 (1.84)	0.11	0.08
10	192 (1.81)	0.11	0.07
100	236 (1.83)	0.12	0.08
1000	230 (1.91)	0.11	0.08

H.3 Random network 15 nodes container set 3

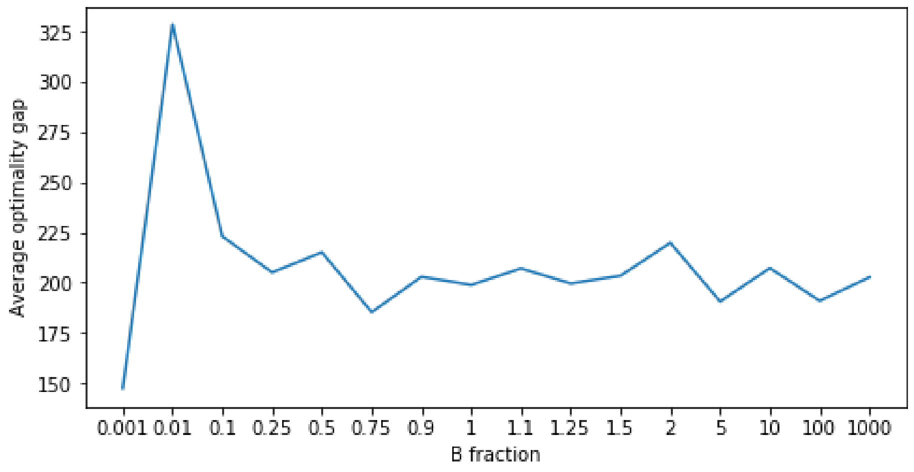


Figure 25: Parameter tune of the W_B value

Table 64: Parameter tune of the W_B value

W_B rate	Average optimality gap (standard deviation)	Anneal time	Annealer time
0.001	147 (0.6)	0.12	0.07
0.01	329 (1.3)	0.1	0.07
0.1	223 (1.93)	0.12	0.09
0.25	205 (2.03)	0.12	0.09
0.5	215 (1.98)	0.12	0.09
0.75	185 (2.01)	0.12	0.08
0.9	203 (2.0)	0.12	0.09
1	199 (2.01)	0.12	0.08
1.1	207 (2.02)	0.12	0.08
1.25	200 (2.06)	0.12	0.09
1.5	203 (2.03)	0.12	0.08
2	220 (2.0)	0.12	0.09
5	191 (1.99)	0.11	0.08
10	207 (1.99)	0.12	0.08
100	191 (1.99)	0.12	0.08
1000	203 (2.05)	0.11	0.07

H.4 Random network 20 nodes container set 1

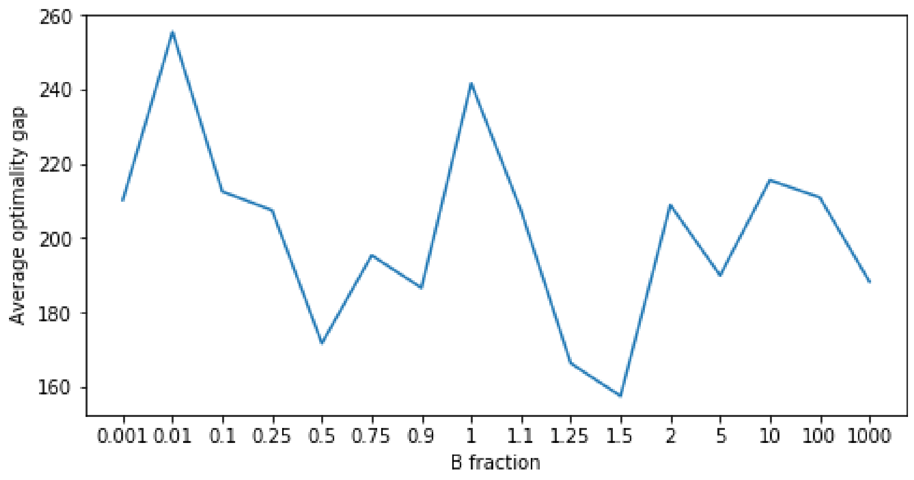


Figure 26: Parameter tune of the W_B value

Table 65: Parameter tune of the W_B value

W_B rate	Average optimality gap (standard deviation)	Anneal time	Annealer time
0.001	210 (0.48)	0.12	0.06
0.01	255 (0.44)	0.11	0.07
0.1	213 (1.64)	0.12	0.08
0.25	207 (1.65)	0.13	0.08
0.5	172 (1.66)	0.12	0.08
0.75	195 (1.59)	0.13	0.08
0.9	187 (1.66)	0.13	0.09
1	242 (1.4)	0.13	0.09
1.1	207 (1.6)	0.13	0.09
1.25	166 (1.58)	0.13	0.09
1.5	157 (1.59)	0.12	0.08
2	209 (1.53)	0.13	0.09
5	190 (1.63)	0.12	0.08
10	216 (1.56)	0.13	0.09
100	211 (1.51)	0.13	0.08
1000	188 (1.56)	0.12	0.08

H.5 Random network 20 nodes container set 2

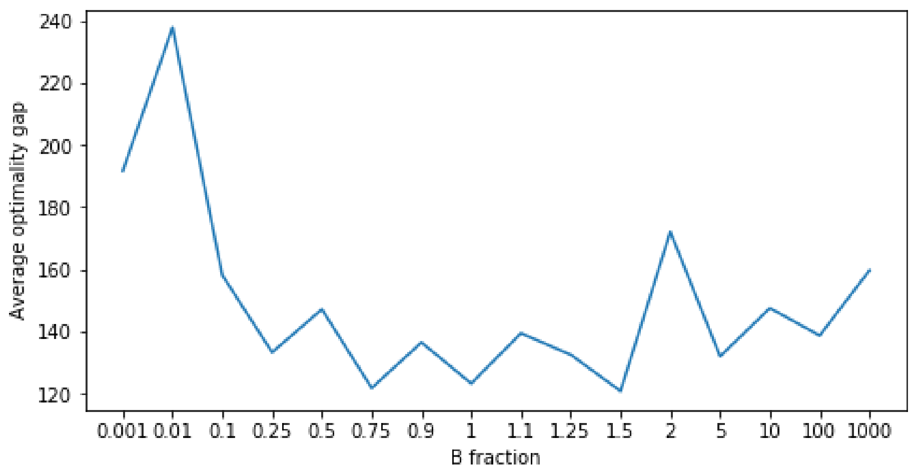


Figure 27: Parameter tune of the W_B value

Table 66: Parameter tune of the W_B value

W_B rate	Average optimality gap (standard deviation)	Anneal time	Annealer time
0.001	192 (0.5)	0.13	0.07
0.01	238 (0.55)	0.12	0.07
0.1	158 (1.71)	0.15	0.1
0.25	133 (1.67)	0.15	0.1
0.5	147 (1.72)	0.15	0.1
0.75	122 (1.64)	0.15	0.1
0.9	136 (1.67)	0.15	0.1
1	123 (1.67)	0.15	0.1
1.1	139 (1.73)	0.15	0.1
1.25	132 (1.71)	0.14	0.1
1.5	121 (1.63)	0.14	0.1
2	172 (1.75)	0.14	0.1
5	132 (1.63)	0.14	0.09
10	147 (1.73)	0.14	0.09
100	139 (1.72)	0.13	0.09
1000	160 (1.79)	0.13	0.09

H.6 Random network 20 nodes container set 3

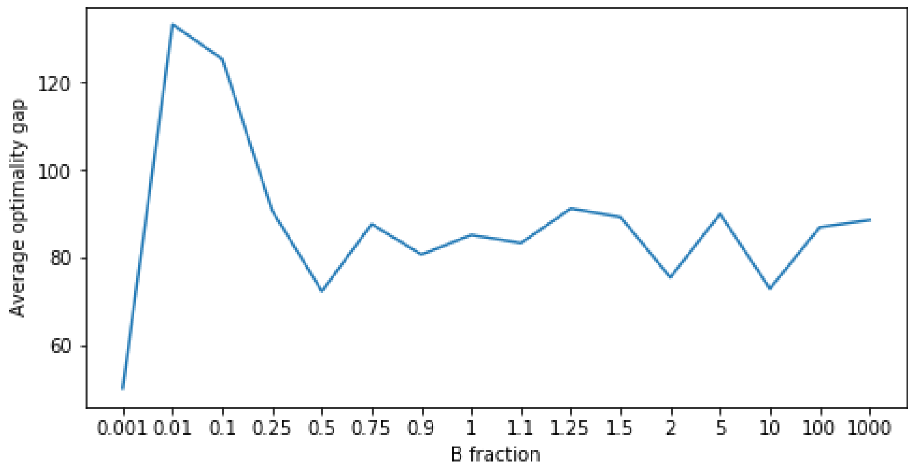


Figure 28: Parameter tune of the W_B value

Table 67: Parameter tune of the W_B value

W_B rate	Average optimality gap (standard deviation)	Anneal time	Annealer time
0.001	50 (0.14)	0.15	0.08
0.01	133 (0.18)	0.15	0.09
0.1	125 (0.83)	0.18	0.11
0.25	91 (0.85)	0.17	0.11
0.5	72 (0.87)	0.17	0.1
0.75	88 (0.87)	0.17	0.1
0.9	81 (0.85)	0.17	0.1
1	85 (0.91)	0.16	0.1
1.1	83 (0.88)	0.17	0.1
1.25	91 (0.92)	0.16	0.09
1.5	89 (0.85)	0.16	0.09
2	75 (0.89)	0.16	0.09
5	90 (0.88)	0.16	0.09
10	73 (0.83)	0.16	0.09
100	87 (0.86)	0.15	0.08
1000	89 (0.88)	0.14	0.07

H.7 Random network 25 nodes container set 1

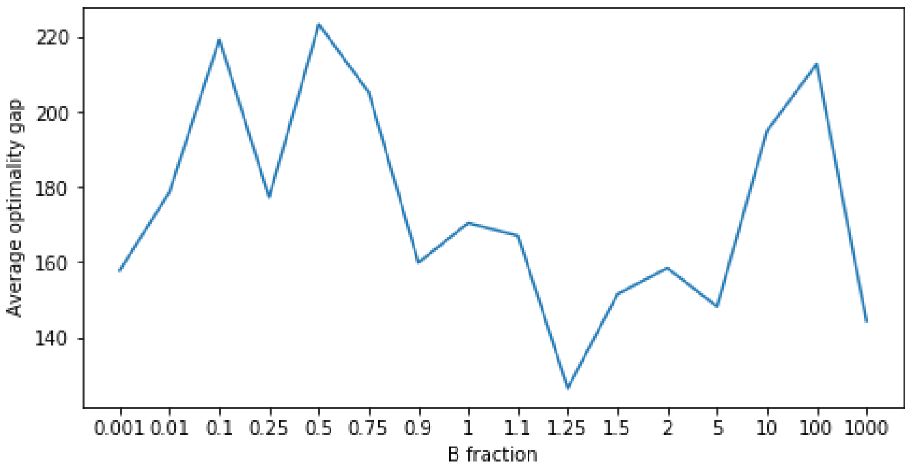


Figure 29: Parameter tune of the W_B value

Table 68: Parameter tune of the W_B value

W_B rate	Average optimality gap (standard deviation)	Anneal time	Annealer time
0.001	158 (0.43)	0.15	0.07
0.01	179 (0.43)	0.14	0.07
0.1	219 (1.31)	0.15	0.1
0.25	177 (1.52)	0.15	0.1
0.5	223 (1.47)	0.15	0.1
0.75	205 (1.5)	0.15	0.1
0.9	160 (1.51)	0.15	0.1
1	170 (1.45)	0.16	0.1
1.1	167 (1.51)	0.15	0.1
1.25	126 (1.46)	0.15	0.1
1.5	151 (1.5)	0.15	0.1
2	158 (1.5)	0.15	0.09
5	148 (1.47)	0.15	0.09
10	195 (1.49)	0.14	0.09
100	213 (1.45)	0.14	0.09
1000	144 (1.53)	0.14	0.09

H.8 Random network 25 nodes container set 2

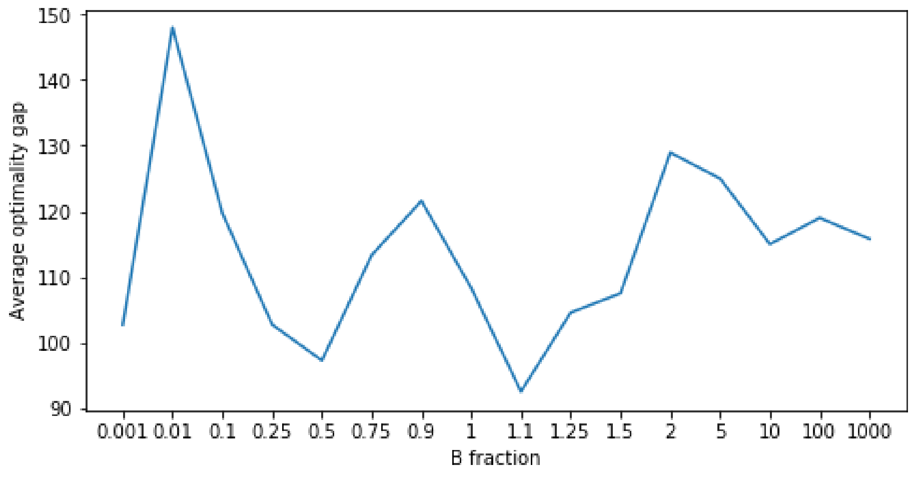


Figure 30: Parameter tune of the W_B value

Table 69: Parameter tune of the W_B value

W_B rate	Average optimality gap (standard deviation)	Anneal time	Annealer time
0.001	103 (0.31)	0.16	0.09
0.01	148 (0.25)	0.14	0.07
0.1	120 (1.09)	0.18	0.11
0.25	103 (1.09)	0.17	0.11
0.5	97 (1.09)	0.17	0.11
0.75	113 (1.12)	0.16	0.1
0.9	122 (1.12)	0.17	0.1
1	108 (1.13)	0.16	0.1
1.1	93 (1.05)	0.16	0.1
1.25	105 (1.08)	0.16	0.1
1.5	107 (1.1)	0.16	0.1
2	129 (1.12)	0.16	0.1
5	125 (1.1)	0.16	0.09
10	115 (1.09)	0.15	0.09
100	119 (1.1)	0.15	0.09
1000	116 (1.08)	0.14	0.08

H.9 Random network 25 nodes container set 3

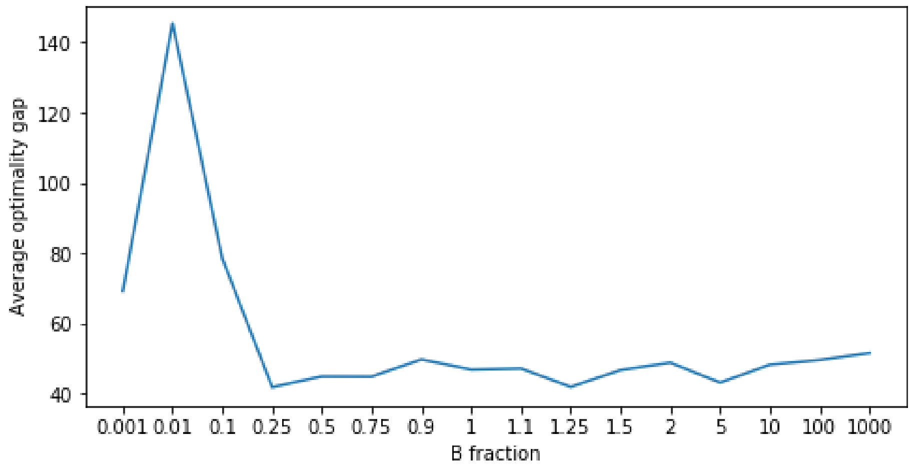


Figure 31: Parameter tune of the W_B value

Table 70: Parameter tune of the W_B value

W_B rate	Average optimality gap (standard deviation)	Anneal time	Annealer time
0.001	69 (0.14)	0.18	0.1
0.01	145 (0.12)	0.22	0.13
0.1	78 (0.79)	0.24	0.14
0.25	42 (0.61)	0.23	0.13
0.5	45 (0.64)	0.22	0.12
0.75	45 (0.63)	0.22	0.12
0.9	50 (0.67)	0.22	0.12
1	47 (0.64)	0.21	0.12
1.1	47 (0.64)	0.21	0.12
1.25	42 (0.59)	0.21	0.12
1.5	47 (0.63)	0.21	0.12
2	49 (0.66)	0.2	0.11
5	43 (0.61)	0.2	0.11
10	48 (0.65)	0.19	0.1
100	50 (0.66)	0.18	0.09
1000	51 (0.67)	0.18	0.09

H.10 Dutch network container set 1

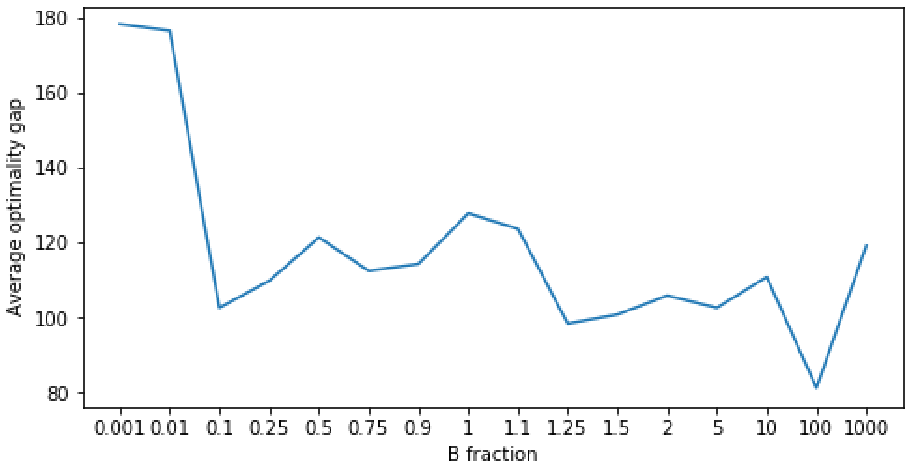


Figure 32: Parameter tune of the W_B value

Table 71: Parameter tune of the W_B value

W_B rate	Average optimality gap (standard deviation)	Anneal time	Annealer time
0.001	178 (0.25)	0.12	0.06
0.01	176 (0.28)	0.13	0.08
0.1	104 (1.02)	0.17	0.1
0.25	110 (1.05)	0.15	0.09
0.5	121 (1.04)	0.15	0.09
0.75	112 (1.02)	0.15	0.09
0.9	114 (1.02)	0.15	0.09
1	128 (1.03)	0.15	0.09
1.1	124 (1.05)	0.14	0.09
1.25	98 (1.01)	0.14	0.09
1.5	101 (0.99)	0.14	0.09
2	106 (1.04)	0.14	0.09
5	103 (1.01)	0.14	0.09
10	111 (1.03)	0.14	0.08
100	81 (0.93)	0.13	0.08
1000	119 (1.03)	0.13	0.08

H.11 Dutch network container set 2

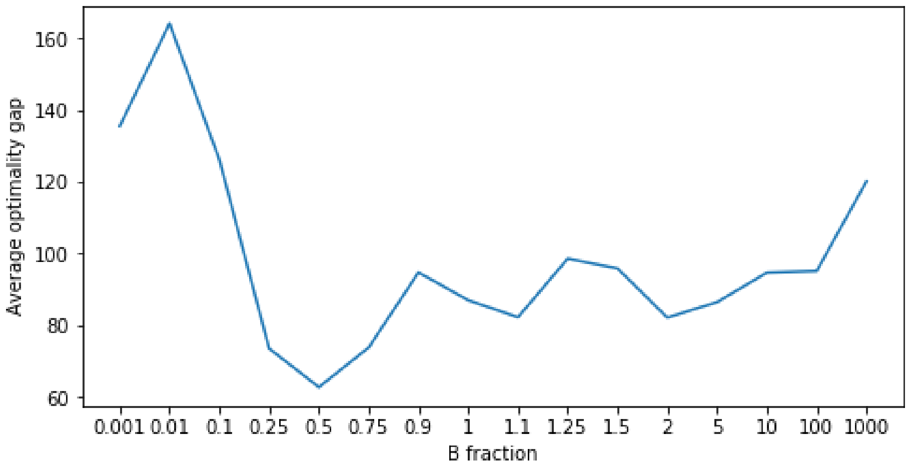


Figure 33: Parameter tune of the W_B value

Table 72: Parameter tune of the W_B value

W_B rate	Average optimality gap (standard deviation)	Anneal time	Annealer time
0.001	136 (0.21)	0.15	0.08
0.01	164 (0.22)	0.14	0.07
0.1	126 (0.84)	0.18	0.11
0.25	74 (0.81)	0.17	0.1
0.5	63 (0.78)	0.16	0.1
0.75	74 (0.83)	0.16	0.09
0.9	95 (0.88)	0.16	0.09
1	87 (0.86)	0.16	0.09
1.1	82 (0.85)	0.15	0.09
1.25	99 (0.88)	0.15	0.09
1.5	96 (0.88)	0.16	0.09
2	82 (0.86)	0.15	0.09
5	87 (0.87)	0.16	0.09
10	95 (0.87)	0.15	0.08
100	95 (0.89)	0.14	0.08
1000	120 (0.87)	0.14	0.07

H.12 Dutch network container set 3

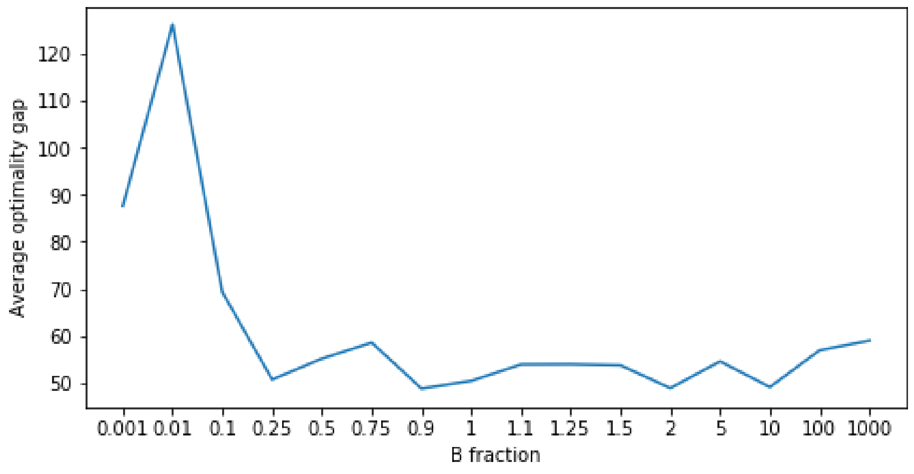


Figure 34: Parameter tune of the W_B value

Table 73: Parameter tune of the W_B value

W_B rate	Average optimality gap (standard deviation)	Anneal time	Annealer time
0.001	88 (0.09)	0.18	0.09
0.01	126 (0.09)	0.19	0.11
0.1	69 (0.7)	0.21	0.12
0.25	51 (0.64)	0.2	0.11
0.5	55 (0.65)	0.19	0.1
0.75	59 (0.67)	0.18	0.1
0.9	49 (0.63)	0.19	0.1
1	50 (0.62)	0.19	0.1
1.1	54 (0.65)	0.17	0.09
1.25	54 (0.64)	0.18	0.1
1.5	54 (0.64)	0.18	0.1
2	49 (0.62)	0.18	0.1
5	55 (0.66)	0.17	0.09
10	49 (0.63)	0.17	0.09
100	57 (0.66)	0.16	0.08
1000	59 (0.67)	0.15	0.07

H.13 Graph 17 container set 1

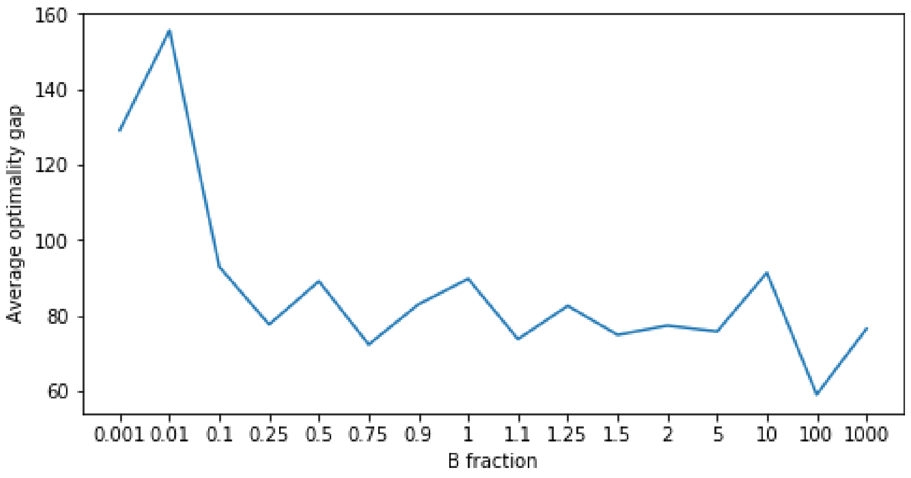


Figure 35: Parameter tune of the W_B value

Table 74: Parameter tune of the W_B value

W_B rate	Average optimality gap (standard deviation)	Anneal time	Annealer time
0.001	129 (0.16)	0.21	0.1
0.01	156 (0.15)	0.19	0.09
0.1	93 (0.77)	0.24	0.13
0.25	78 (0.81)	0.21	0.11
0.5	89 (0.8)	0.23	0.12
0.75	72 (0.76)	0.22	0.11
0.9	83 (0.79)	0.22	0.11
1	90 (0.8)	0.24	0.12
1.1	74 (0.8)	0.21	0.11
1.25	83 (0.81)	0.22	0.11
1.5	75 (0.79)	0.23	0.11
2	77 (0.79)	0.22	0.11
5	76 (0.8)	0.21	0.1
10	91 (0.83)	0.21	0.1
100	59 (0.76)	0.2	0.1
1000	76 (0.79)	0.19	0.09

H.14 Graph 17 container set 2

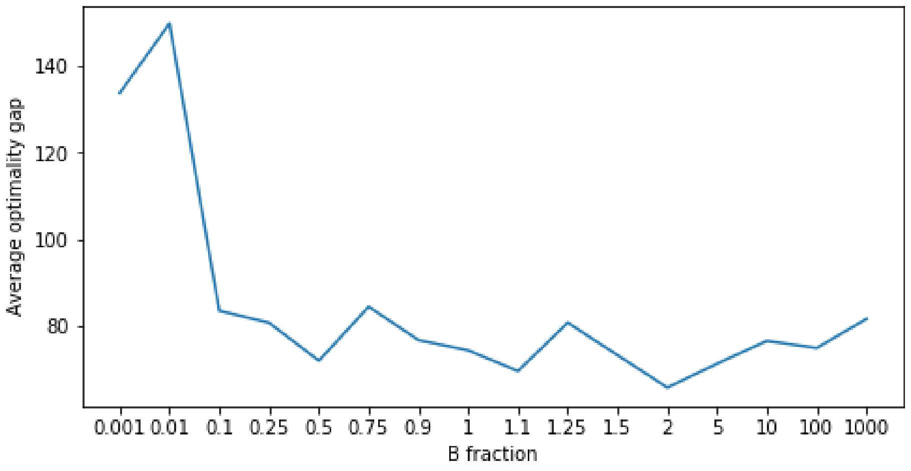


Figure 36: Parameter tune of the W_B value

Table 75: Parameter tune of the W_B value

W_B rate	Average optimality gap (standard deviation)	Anneal time	Annealer time
0.001	134 (0.16)	0.22	0.1
0.01	150 (0.14)	0.2	0.1
0.1	83 (0.82)	0.24	0.13
0.25	81 (0.78)	0.25	0.13
0.5	72 (0.77)	0.24	0.12
0.75	84 (0.79)	0.24	0.12
0.9	77 (0.77)	0.25	0.12
1	74 (0.78)	0.23	0.12
1.1	70 (0.77)	0.23	0.12
1.25	81 (0.77)	0.24	0.12
1.5	73 (0.77)	0.23	0.11
2	66 (0.75)	0.22	0.11
5	71 (0.77)	0.22	0.11
10	77 (0.79)	0.22	0.1
100	75 (0.78)	0.22	0.1
1000	82 (0.78)	0.21	0.09

H.15 Graph 17 container set 3

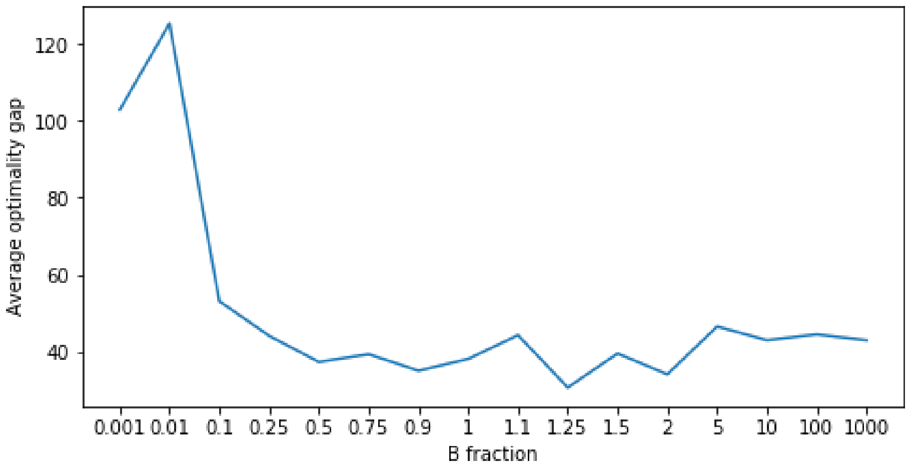


Figure 37: Parameter tune of the W_B value

Table 76: Parameter tune of the W_B value

W_B rate	Average optimality gap (standard deviation)	Anneal time	Annealer time
0.001	103 (0.11)	0.27	0.12
0.01	125 (0.09)	0.28	0.14
0.1	53 (0.64)	0.32	0.17
0.25	44 (0.6)	0.3	0.15
0.5	37 (0.54)	0.29	0.14
0.75	39 (0.55)	0.29	0.14
0.9	35 (0.52)	0.28	0.14
1	38 (0.55)	0.28	0.14
1.1	44 (0.59)	0.28	0.13
1.25	31 (0.5)	0.27	0.13
1.5	40 (0.57)	0.27	0.13
2	34 (0.52)	0.27	0.12
5	47 (0.6)	0.26	0.12
10	43 (0.58)	0.26	0.12
100	45 (0.59)	0.25	0.11
1000	43 (0.58)	0.24	0.1

I Mutation rate tune

Table 77: Random network 15 nodes container set 1

Mutation rate	Optimality gap	Computation time
0.001	0	264.9
0.005	2	239.45
0.01	2	231.16
0.05	2	229.66
0.1	2	224.76
0.125	0	246.94
0.15	0	263.52
0.2	2	278.38
0.25	0	262.29
0.4	0	259.90
0.5	0	254.43
0.75	0	291.24

Table 78: Random network 15 nodes container set 2

Mutation rate	Optimality gap	Computation time
0.001	0	272.51
0.005	2	272.55
0.01	0	277.28
0.05	2	243.66
0.1	0	271.84
0.125	0	282.44
0.15	0	305.79
0.2	0	327.84
0.25	0	341.33
0.4	0	343.71
0.5	0	370.09
0.75	0	417.14

Table 79: Random network 15 nodes container set 3

Mutation rate	Optimality gap	Computation time
0.001	2	275.9
0.005	0	270.68
0.01	0	276.03
0.05	2	292.05
0.1	0	299.22
0.125	2	306.97
0.15	0	309.8
0.2	2	325.34
0.25	2	333.82
0.4	0	349.0
0.5	0	370.48
0.75	0	439.44

Table 80: Random network 20 nodes container set 1

Mutation rate	Optimality gap	Computation time
0.001	25	272.97
0.005	33	275.12
0.01	29	282.6
0.05	29	267.6
0.1	24	307.61
0.125	24	312.16
0.15	24	303.3
0.2	25	321.83
0.25	28	336.47
0.4	29	351.74
0.5	29	394.52
0.75	25	464.26

Table 81: Random network 20 nodes container set 2

Mutation rate	Optimality gap	Computation time
0.001	39	273.71
0.005	40	285.45
0.01	40	289.68
0.05	39	292.79
0.1	39	278.44
0.125	40	319.28
0.15	40	326.53
0.2	39	348.13
0.25	39	363.16
0.4	39	409.24
0.5	40	435.25
0.75	46	500.40

Table 82: Random network 20 nodes container set 3

Mutation rate	Optimality gap	Computation time
0.001	68	308.57
0.005	48	314.66
0.01	53	317.22
0.05	43	324.43
0.1	36	345.24
0.125	53	357.36
0.15	44	365.34
0.2	44	386.05
0.25	70	398.92
0.4	70	433.82
0.5	57	471.02
0.75	68	553.59

Table 83: Random network 25 nodes container set 1

Mutation rate	Optimality gap	Computation time
0.001	15	253.3
0.005	15	256.92
0.01	14	283.81
0.05	16	277.05
0.1	17	298.62
0.125	16	325.32
0.15	13	318.66
0.2	13	349.37
0.25	15	349.78
0.4	13	389.87
0.5	14	441.75
0.75	13	466.2

Table 84: Random network 25 nodes container set 2

Mutation rate	Optimality gap	Computation time
0.001	35	305.52
0.005	36	272.42
0.01	35	311.76
0.05	36	302.86
0.1	35	327.52
0.125	37	350.26
0.15	36	383.02
0.2	37	387.11
0.25	35	399.64
0.4	35	447.83
0.5	35	483.38
0.75	36	568.96

Table 85: Random network 25 nodes container set 3

Mutation rate	Optimality gap	Computation time
0.001	33	350.47
0.005	33	349.78
0.01	34	353.91
0.05	33	372.41
0.1	33	392.65
0.125	33	408.53
0.15	34	420.86
0.2	36	429.77
0.25	34	441.46
0.4	35	502.18
0.5	35	545.35
0.75	38	622.33

Table 86: Dutch network container set 1

Mutation rate	Optimality gap	Computation time
0.001	24	279.59
0.005	24	276.51
0.01	24	290.15
0.05	24	291.11
0.1	24	289.22
0.125	24	326.88
0.15	24	328.38
0.2	24	337.39
0.25	24	328.71
0.4	27	403.56
0.5	30	436.11
0.75	26	509.29

Table 87: Dutch network container set 2

Mutation rate	Optimality gap	Computation time
0.001	14	281.83
0.005	14	286.17
0.01	16	300.0
0.05	15	319.38
0.1	15	328.55
0.125	15	331.63
0.15	15	354.27
0.2	15	377.09
0.25	18	389.15
0.4	14	436.51
0.5	18	453.40
0.75	18	534.0

Table 88: Dutch network container set 3

Mutation rate	Optimality gap	Computation time
0.001	14	321.45
0.005	15	318.17
0.01	16	326.04
0.05	16	343.8
0.1	14	360.14
0.125	15	375.16
0.15	16	380.0
0.2	14	397.34
0.25	15	401.48
0.4	14	467.77
0.5	14	489.38
0.75	16	582.9

Table 89: Graph 17 container set 1

Mutation rate	Optimality gap	Computation time
0.001	33	344.92
0.005	34	345.42
0.01	17	339.92
0.05	35	394.68
0.1	16	413.65
0.125	15	416.51
0.15	17	425.39
0.2	23	437.89
0.25	19	454.25
0.4	22	532.47
0.5	23	550.12
0.75	11	623.16

Table 90: Graph 17 container set 2

Mutation rate	Optimality gap	Computation time
0.001	34	372.49
0.005	35	385.52
0.01	38	382.45
0.05	30	393.0
0.1	31	429.62
0.125	17	428.69
0.15	26	448.61
0.2	20	442.99
0.25	18	477.33
0.4	15	541.90
0.5	22	568.88
0.75	13	659.53

Table 91: Graph 17 container set 3

Mutation rate	Optimality gap	Computation time
0.001	14	425.18
0.005	12	425.95
0.01	19	433.59
0.05	21	442.21
0.1	12	455.45
0.125	14	486.33
0.15	13	486.21
0.2	17	525.39
0.25	19	532.92
0.4	15	592.06
0.5	15	629.36
0.75	16	740.41

J Results using the simulated annealer

J.1 Found solutions

Table 92: Solution random 15, 1

Container	Used path	Cost
Container 1	[0, 8, 20, 36, 61]	2
Container 2	[0, 8, 26, 36, 61]	3
Container 3	[0, 23, 41, 51, 61]	3
Container 4	[0, 23, 42, 51, 61]	3
Container 5	[0, 18, 38, 50, 61]	2
Container 6	[0, 3, 23, 35, 61]	2
Container 7	[0, 18, 43, 50, 61]	3
Container 8	[0, 18, 42, 50, 61]	3
Container 9	[0, 2, 19, 43, 61]	4
Container 10	[0, 2, 19, 43, 61]	4
Container 11	[0, 17, 34, 58, 61]	4
Container 12	[0, 17, 34, 58, 61]	4
Container 13	[0, 25, 44, 61]	3
Container 14	[0, 40, 59, 61]	3
Container 15	[0, 25, 44, 61]	3
Container 16	[0, 40, 59, 61]	3
Container 17	[0, 5, 21, 41, 61]	2
Container 18	[0, 20, 41, 61]	3
Container 19	[0, 35, 56, 61]	3
Container 20	[0, 20, 36, 56, 61]	2

Table 93: Solution random 15, 2

Container	Used path	Cost
Container 1	[0, 1, 28, 35, 61]	4
Container 2	[0, 1, 28, 35, 61]	4
Container 3	[0, 1, 18, 38, 50, 61]	3
Container 4	[0, 1, 18, 42, 50, 61]	4
Container 5	[0, 26, 33, 61]	3
Container 6	[0, 26, 38, 48, 61]	3
Container 7	[0, 11, 24, 37, 48, 61]	3
Container 8	[0, 11, 23, 33, 61]	3
Container 9	[0, 12, 18, 31, 61]	3
Container 10	[0, 12, 18, 43, 46, 61]	5
Container 11	[0, 12, 20, 43, 46, 61]	7
Container 12	[0, 27, 33, 46, 61]	3
Container 13	[0, 30, 33, 61]	2
Container 14	[0, 45, 48, 61]	2
Container 15	[0, 45, 48, 61]	2
Container 16	[0, 30, 33, 61]	2
Container 17	[0, 19, 40, 61]	2
Container 18	[0, 34, 55, 61]	2
Container 19	[0, 19, 42, 55, 61]	2
Container 20	[0, 19, 40, 61]	2

Table 94: Solution random 15, 3

Container	Used path	Cost
Container 1	[0, 22, 33, 57, 61]	2
Container 2	[0, 7, 24, 34, 57, 61]	3
Container 3	[0, 7, 29, 42, 61]	3
Container 4	[0, 7, 18, 42, 61]	2
Container 5	[0, 7, 29, 42, 61]	3
Container 6	[0, 22, 44, 57, 61]	3
Container 7	[0, 7, 29, 42, 61]	3
Container 8	[0, 7, 18, 42, 61]	2
Container 9	[0, 22, 44, 57, 61]	3
Container 10	[0, 22, 33, 57, 61]	2
Container 11	[0, 20, 40, 61]	3
Container 12	[0, 20, 40, 61]	3
Container 13	[0, 35, 55, 61]	3
Container 14	[0, 35, 55, 61]	3
Container 15	[0, 20, 40, 61]	3
Container 16	[0, 5, 25, 61]	3
Container 17	[0, 5, 27, 40, 61]	4
Container 18	[0, 5, 25, 61]	3
Container 19	[0, 35, 55, 61]	3
Container 20	[0, 5, 25, 61]	3

Table 95: Solution random 20, 1

Container	Used path	Cost
Container 1	[0, 2, 26, 50, 77, 81]	4
Container 2	[0, 2, 24, 57, 81]	4
Container 3	[0, 22, 44, 77, 81]	4
Container 4	truck	20
Container 5	[0, 7, 39, 51, 81]	3
Container 6	[0, 7, 30, 51, 81]	3
Container 7	[0, 27, 50, 71, 81]	3
Container 8	[0, 27, 59, 71, 81]	3
Container 9	[0, 31, 50, 74, 81]	5
Container 10	[0, 11, 38, 43, 74, 81]	5
Container 11	[0, 31, 50, 74, 81]	5
Container 12	[0, 31, 59, 74, 81]	5
Container 13	truck	15
Container 14	[0, 5, 33, 60, 81]	2
Container 15	[0, 25, 53, 80, 81]	2
Container 16	[0, 25, 53, 80, 81]	2
Container 17	[0, 41, 63, 81]	2
Container 18	[0, 1, 23, 81]	2
Container 19	[0, 21, 54, 63, 81]	3
Container 20	[0, 21, 43, 81]	2

Table 96: Solution random 20, 2

Container	Used path	Cost
Container 1	[0, 13, 21, 59, 81]	3
Container 2	[0, 33, 59, 81]	2
Container 3	[0, 13, 39, 81]	2
Container 4	[0, 53, 79, 81]	2
Container 5	[0, 26, 53, 65, 81]	2
Container 6	[0, 6, 33, 45, 81]	2
Container 7	[0, 6, 33, 45, 81]	2
Container 8	truck	17
Container 9	truck	15
Container 10	[0, 30, 55, 62, 81]	2
Container 11	[0, 10, 35, 42, 81]	2
Container 12	[0, 30, 54, 62, 81]	3
Container 13	[0, 41, 62, 81]	2
Container 14	[0, 41, 62, 81]	2
Container 15	[0, 41, 62, 81]	2
Container 16	[0, 21, 42, 81]	2
Container 17	[0, 34, 43, 69, 81]	2
Container 18	truck	15
Container 19	truck	16
Container 20	[0, 34, 43, 69, 81]	2

Table 97: Solution random 20, 3

Container	Used path	Cost
Container 1	[0, 14, 40, 46, 65, 81]	5
Container 2	[0, 14, 39, 53, 65, 81]	4
Container 3	[0, 14, 22, 44, 65, 81]	4
Container 4	[0, 14, 30, 56, 65, 81]	4
Container 5	[0, 14, 30, 45, 81]	3
Container 6	[0, 14, 30, 56, 65, 81]	4
Container 7	[0, 14, 39, 53, 65, 81]	4
Container 8	[0, 14, 40, 46, 65, 81]	5
Container 9	[0, 34, 50, 65, 81]	3
Container 10	truck	20
Container 11	truck	18
Container 12	[0, 11, 33, 52, 81]	6
Container 13	[0, 11, 39, 53, 72, 81]	7
Container 14	[0, 11, 23, 52, 81]	6
Container 15	[0, 31, 53, 72, 81]	6
Container 16	[0, 31, 58, 72, 81]	3
Container 17	[0, 31, 43, 72, 81]	6
Container 18	[0, 11, 35, 60, 72, 81]	6
Container 19	[0, 11, 38, 52, 81]	3
Container 20	truck	16

Table 98: Solution random 25, 1

Container	Used path	Cost
Container 1	[0, 20, 38, 65, 101]	3
Container 2	[0, 20, 44, 65, 101]	4
Container 3	[0, 45, 63, 90, 101]	3
Container 4	[0, 45, 69, 90, 101]	4
Container 5	[0, 48, 68, 79, 101]	4
Container 6	truck	15
Container 7	[0, 48, 68, 79, 101]	4
Container 8	[0, 48, 68, 79, 101]	4
Container 9	[0, 19, 39, 64, 78, 101]	4
Container 10	[0, 19, 39, 53, 101]	3
Container 11	[0, 44, 52, 78, 101]	5
Container 12	[0, 44, 54, 78, 101]	5
Container 13	[0, 10, 34, 72, 101]	6
Container 14	[0, 10, 33, 63, 97, 101]	4
Container 15	[0, 35, 59, 97, 101]	6
Container 16	truck	15
Container 17	[0, 21, 29, 68, 98, 101]	4
Container 18	[0, 21, 38, 73, 101]	2
Container 19	[0, 46, 63, 98, 101]	2
Container 20	[0, 46, 72, 98, 101]	4

Table 99: Solution random 25, 2

Container	Used path	Cost
Container 1	[0, 9, 38, 72, 91, 101]	5
Container 2	[0, 9, 30, 66, 101]	5
Container 3	[0, 34, 55, 91, 101]	5
Container 4	truck	17
Container 5	[0, 42, 73, 101]	3
Container 6	[0, 67, 98, 101]	3
Container 7	truck	18
Container 8	[0, 17, 48, 101]	3
Container 9	[0, 21, 39, 57, 101]	3
Container 10	truck	17
Container 11	[0, 46, 64, 82, 101]	3
Container 12	[0, 21, 29, 51, 82, 101]	4
Container 13	truck	16
Container 14	[0, 40, 52, 78, 101]	3
Container 15	truck	15
Container 16	[0, 40, 52, 78, 101]	3
Container 17	[0, 39, 67, 91, 101]	5
Container 18	[0, 39, 62, 91, 101]	4
Container 19	[0, 39, 55, 91, 101]	3
Container 20	truck	18

Table 100: Solution random 25, 3

Container	Used path	Cost
Container 1	truck	16
Container 2	[0, 6, 27, 69, 83, 101]	4
Container 3	truck	16
Container 4	[0, 6, 27, 69, 83, 101]	4
Container 5	[0, 6, 36, 63, 83, 101]	4
Container 6	[0, 6, 27, 63, 83, 101]	5
Container 7	truck	19
Container 8	[0, 6, 50, 69, 83, 101]	5
Container 9	truck	15
Container 10	truck	16
Container 11	[0, 43, 55, 84, 101]	4
Container 12	[0, 18, 41, 72, 84, 101]	4
Container 13	[0, 18, 48, 63, 84, 101]	5
Container 14	truck	15
Container 15	truck	16
Container 16	[0, 18, 35, 59, 101]	4
Container 17	[0, 43, 60, 84, 101]	4
Container 18	[0, 18, 48, 72, 84, 101]	3
Container 19	truck	19
Container 20	truck	17

Table 101: Solution Dutch, 1

Container	Used path	Cost
Container 1	[0, 14, 23, 67]	1
Container 2	[0, 3, 26, 46, 67]	3
Container 3	[0, 14, 37, 57, 67]	3
Container 4	truck	15
Container 5	[0, 36, 45, 67]	1
Container 6	[0, 25, 34, 67]	1
Container 7	truck	15
Container 8	[0, 14, 28, 38, 67]	4
Container 9	[0, 3, 17, 29, 67]	5
Container 10	[0, 3, 26, 51, 67]	3
Container 11	truck	10
Container 12	[0, 30, 47, 67]	3
Container 13	truck	12
Container 14	[0, 8, 17, 27, 67]	4
Container 15	[0, 41, 50, 62, 67]	5
Container 16	truck	10
Container 17	[0, 52, 64, 67]	2
Container 18	truck	13
Container 19	[0, 19, 28, 67]	3
Container 20	[0, 8, 17, 43, 55, 67]	6

Table 102: Solution Dutch, 2

Container	Used path	Cost
Container 1	[0, 47, 56, 67]	1
Container 2	[0, 25, 34, 67]	1
Container 3	[0, 3, 12, 67]	1
Container 4	truck	10
Container 5	[0, 3, 26, 46, 67]	3
Container 6	[0, 14, 37, 57, 67]	3
Container 7	truck	10
Container 8	[0, 14, 28, 38, 67]	4
Container 9	[0, 3, 26, 51, 67]	3
Container 10	[0, 14, 37, 62, 67]	3
Container 11	truck	15
Container 12	[0, 8, 17, 43, 55, 67]	6
Container 13	[0, 8, 17, 27, 67]	4
Container 14	truck	13
Container 15	[0, 30, 42, 54, 67]	4
Container 16	[0, 8, 17, 67]	3
Container 17	truck	12
Container 18	truck	10
Container 19	[0, 8, 20, 32, 44, 67]	5
Container 20	truck	10

Table 103: Solution Dutch, 3

Container	Used path	Cost
Container 1	[0, 3, 26, 46, 67]	3
Container 2	[0, 14, 37, 57, 67]	3
Container 3	truck	14
Container 4	truck	13
Container 5	truck	10
Container 6	truck	13
Container 7	[0, 14, 37, 62, 67]	3
Container 8	truck	10
Container 9	[0, 3, 17, 29, 67]	5
Container 10	[0, 3, 26, 51, 67]	3
Container 11	truck	10
Container 12	[0, 8, 20, 32, 44, 67]	5
Container 13	truck	14
Container 14	truck	12
Container 15	truck	15
Container 16	truck	10
Container 17	[0, 8, 17, 27, 67]	4
Container 18	[0, 19, 31, 43, 55, 67]	5
Container 19	[0, 19, 28, 38, 67]	4
Container 20	truck	12

Table 104: Solution Graph 17, 1

Container	Used path	Cost
Container 1	[0, 1, 21, 44, 66, 120]	6
Container 2	[0, 2, 18, 38, 61, 83, 120]	8
Container 3	[0, 20, 43, 65, 83, 120]	6
Container 4	[0, 20, 36, 56, 75, 97, 120]	8
Container 5	truck	16
Container 6	[0, 40, 54, 70, 93, 114, 120]	8
Container 7	[0, 5, 24, 46, 67, 85, 120]	8
Container 8	[0, 7, 29, 50, 68, 120]	6
Container 9	[0, 2, 25, 45, 65, 85, 120]	8
Container 10	[0, 3, 26, 48, 68, 120]	6
Container 11	[0, 35, 55, 78, 100, 120]	6
Container 12	[0, 36, 59, 79, 99, 117, 120]	8
Container 13	[0, 37, 60, 82, 100, 120]	6
Container 14	truck	19
Container 15	[0, 21, 35, 53, 76, 97, 120]	8
Container 16	[0, 23, 37, 53, 73, 92, 114, 120]	10
Container 17	[0, 22, 42, 62, 82, 102, 120]	8
Container 18	[0, 41, 63, 84, 102, 120]	6
Container 19	[0, 19, 39, 58, 80, 101, 119, 120]	10
Container 20	[0, 54, 77, 99, 119, 120]	6

Table 105: Solution Graph 17, 2

Container	Used path	Cost
Container 1	[0, 52, 72, 95, 117, 120]	6
Container 2	[0, 2, 18, 38, 61, 83, 120]	8
Container 3	[0, 3, 26, 48, 66, 120]	6
Container 4	[0, 37, 53, 73, 92, 114, 120]	8
Container 5	[0, 4, 18, 36, 56, 75, 97, 120]	10
Container 6	[0, 40, 54, 70, 93, 114, 120]	8
Container 7	[0, 5, 19, 37, 60, 82, 102, 120]	10
Container 8	[0, 58, 80, 101, 119, 120]	6
Container 9	[0, 2, 22, 41, 63, 84, 102, 120]	10
Container 10	truck	16
Container 11	[0, 1, 21, 44, 66, 120]	6
Container 12	truck	17
Container 13	truck	20
Container 14	[0, 20, 36, 59, 80, 120]	6
Container 15	truck	15
Container 16	[0, 6, 20, 35, 53, 76, 97, 120]	9
Container 17	[0, 5, 24, 46, 67, 85, 120]	8
Container 18	[0, 7, 29, 50, 68, 120]	6
Container 19	[0, 36, 54, 77, 99, 119, 120]	8
Container 20	[0, 20, 43, 65, 85, 120]	6

Table 106: Solution Graph 17, 3

Container	Used path	Cost
Container 1	[0, 35, 55, 74, 95, 117, 120]	8
Container 2	truck	18
Container 3	[0, 18, 38, 61, 83, 120]	6
Container 4	[0, 1, 21, 44, 66, 120]	6
Container 5	truck	20
Container 6	[0, 1, 20, 43, 65, 85, 120]	7
Container 7	[0, 35, 54, 77, 99, 119, 120]	7
Container 8	truck	17
Container 9	truck	16
Container 10	[0, 18, 37, 60, 82, 102, 120]	7
Container 11	[0, 1, 19, 39, 58, 80, 101, 120]	10
Container 12	[0, 1, 20, 36, 59, 79, 98, 118, 120]	11
Container 13	[0, 18, 36, 56, 75, 97, 118, 120]	10
Container 14	truck	18
Container 15	truck	18
Container 16	[0, 35, 53, 73, 92, 114, 120]	8
Container 17	truck	19
Container 18	truck	15
Container 19	[0, 52, 70, 93, 114, 120]	6
Container 20	[0, 18, 37, 53, 76, 97, 120]	7

Table 107: Solution Graph 30, 1

Container	Used path	Cost
Container 1	truck	16
Container 2	truck	18
Container 3	truck	20
Container 4	[0, 4, 40, 75, 108, 145, 179, 210, 271]	12
Container 5	[0, 5, 38, 71, 104, 135, 168, 205, 239, 270, 271]	16
Container 6	[0, 36, 70, 105, 138, 175, 209, 240, 271]	12
Container 7	truck	16
Container 8	truck	16
Container 9	truck	16
Container 10	truck	16
Container 11	[0, 121, 152, 188, 222, 271]	6
Container 12	truck	15
Container 13	truck	20
Container 14	[0, 124, 151, 182, 218, 252, 271]	8
Container 15	[0, 155, 187, 222, 271]	4
Container 16	[0, 31, 62, 92, 128, 162, 196, 230, 266, 271]	13
Container 17	truck	18
Container 18	truck	20
Container 19	truck	16
Container 20	[0, 35, 67, 102, 136, 170, 206, 271]	10
Container 21	truck	18
Container 22	truck	15
Container 23	[0, 8, 42, 76, 110, 146, 271]	8
Container 24	[0, 9, 44, 77, 109, 140, 176, 271]	10
Container 25	truck	17
Container 26	truck	20
Container 27	truck	17
Container 28	truck	20
Container 29	truck	15
Container 30	truck	19

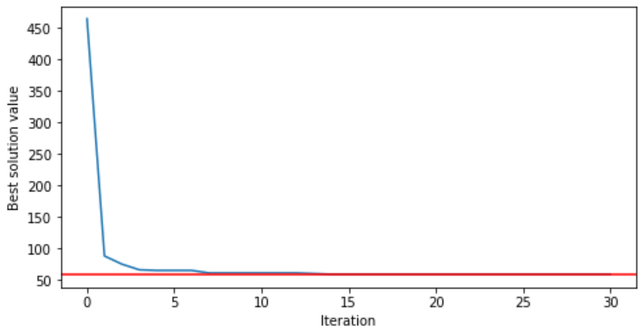
Table 108: Solution Graph 30, 2

Container	Used path	Cost
Container 1	truck	20
Container 2	truck	19
Container 3	truck	15
Container 4	[0, 4, 40, 75, 108, 145, 179, 210, 271]	12
Container 5	truck	15
Container 6	truck	17
Container 7	truck	19
Container 8	truck	19
Container 9	truck	18
Container 10	truck	20
Container 11	[0, 1, 32, 65, 97, 132, 271]	8
Container 12	truck	19
Container 13	truck	19
Container 14	truck	17
Container 15	truck	19
Container 16	truck	17
Container 17	truck	15
Container 18	truck	17
Container 19	[0, 34, 70, 105, 138, 174, 203, 236, 271]	11
Container 20	[0, 35, 67, 102, 136, 170, 206, 271]	10
Container 21	truck	18
Container 22	truck	17
Container 23	truck	18
Container 24	truck	17
Container 25	truck	16
Container 26	truck	17
Container 27	truck	20
Container 28	[0, 128, 162, 196, 230, 266, 271]	8
Container 29	truck	20
Container 30	truck	15
Container 31	truck	15
Container 32	truck	18
Container 33	truck	16
Container 34	truck	17
Container 35	truck	18
Container 36	truck	20
Container 37	truck	17
Container 38	truck	18
Container 39	truck	19
Container 40	truck	20
Container 41	[0, 31, 62, 95, 127, 162, 271]	8
Container 42	[0, 32, 68, 102, 271]	4
Container 43	[0, 63, 91, 122, 158, 192, 271]	7
Container 44	[0, 34, 61, 92, 125, 157, 192, 271]	10
Container 45	truck	19
Container 46	truck	17
Container 47	truck	20
Container 48	[0, 38, 72, 271]	2
Container 49	[0, 69, 93, 122, 155, 187, 222, 271]	10
Container 50	truck	15

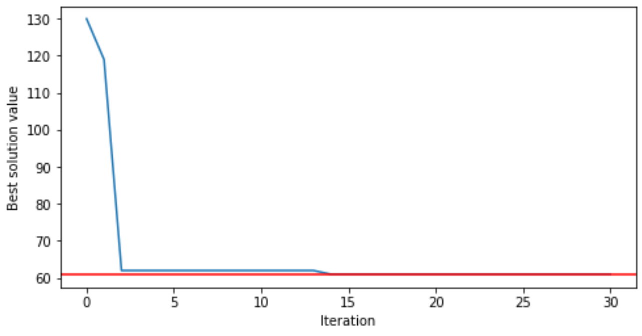
Table 109: Solution Graph 30, 3

Container	Used path	Cost
Container 1	truck	16
Container 2	truck	19
Container 3	truck	18
Container 4	truck	20
Container 5	truck	15
Container 6	truck	19
Container 7	truck	18
Container 8	truck	18
Container 9	truck	17
Container 10	[0, 1, 34, 70, 105, 138, 174, 209, 240, 271]	14
Container 11	truck	16
Container 12	truck	19
Container 13	truck	15
Container 14	[0, 31, 62, 95, 127, 162, 271]	8
Container 15	truck	15
Container 16	truck	20
Container 17	truck	16
Container 18	truck	16
Container 19	truck	16
Container 20	[0, 31, 63, 99, 134, 167, 199, 230, 266, 271]	13
Container 21	truck	16
Container 22	[0, 1, 32, 68, 102, 136, 170, 206, 271]	12
Container 23	truck	18
Container 24	truck	20
Container 25	truck	15
Container 26	truck	18
Container 27	truck	18
Container 28	truck	20
Container 29	truck	16
Container 30	truck	20
Container 31	truck	17
Container 32	truck	15
Container 33	truck	16
Container 34	truck	20
Container 35	truck	16
Container 36	truck	19
Container 37	truck	15
Container 38	truck	20
Container 39	truck	19
Container 40	truck	17
Container 41	[0, 34, 66, 99, 131, 158, 192, 271]	10
Container 42	[0, 94, 121, 152, 188, 222, 271]	8
Container 43	[0, 64, 91, 122, 155, 187, 222, 271]	10
Container 44	[0, 91, 123, 159, 191, 218, 252, 271]	9
Container 45	truck	20
Container 46	truck	15
Container 47	truck	17
Container 48	[0, 61, 92, 128, 162, 271]	6
Container 49	truck	19
Container 50	truck	19

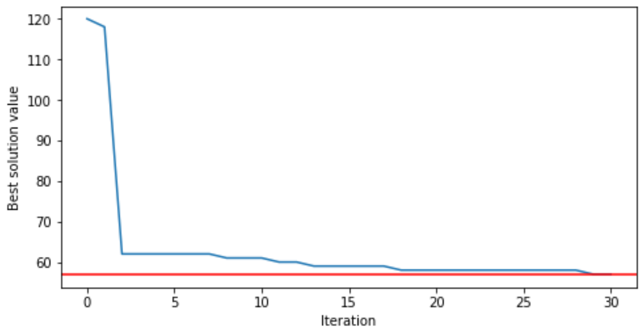
J.2 Course of the algorithm using the simulated annealer



(a) Random 15, 1

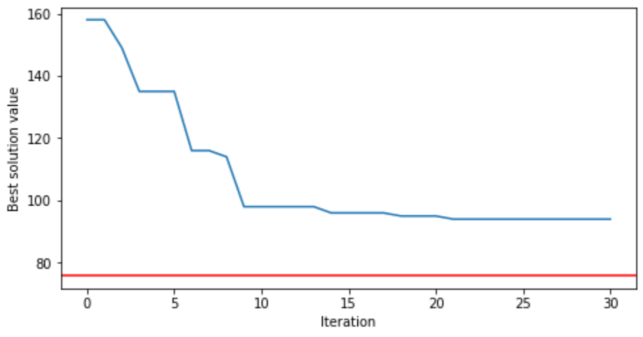


(b) Random 15, 2

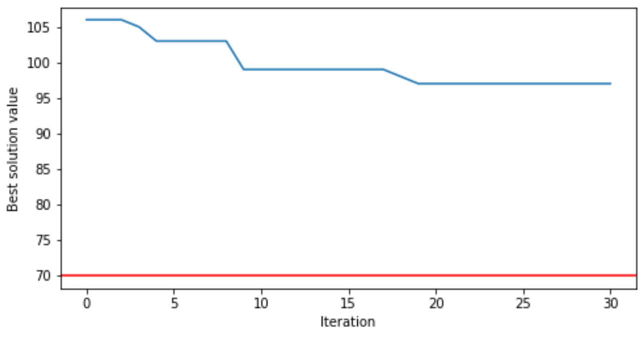


(c) Random 15, 3

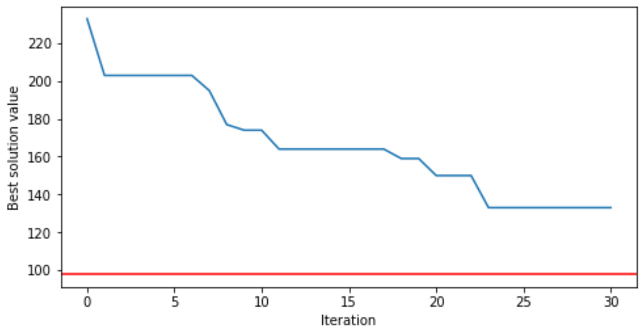
Figure 38: Random 15



(a) Random 20, 1

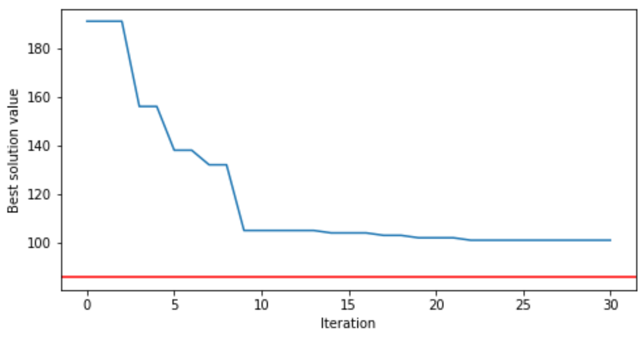


(b) Random 20, 2

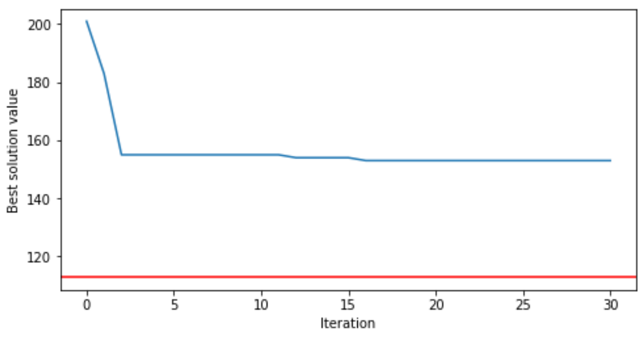


(c) Random 20, 3

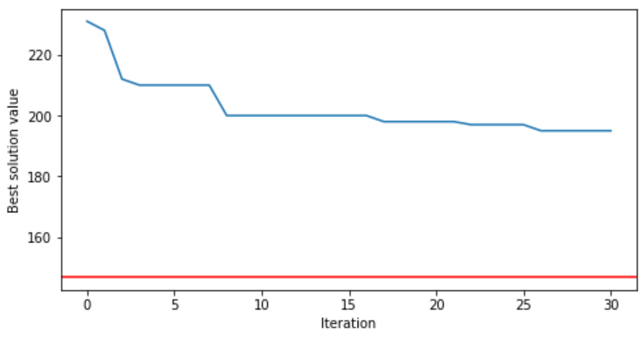
Figure 39: Random 20



(a) Random 25, 1

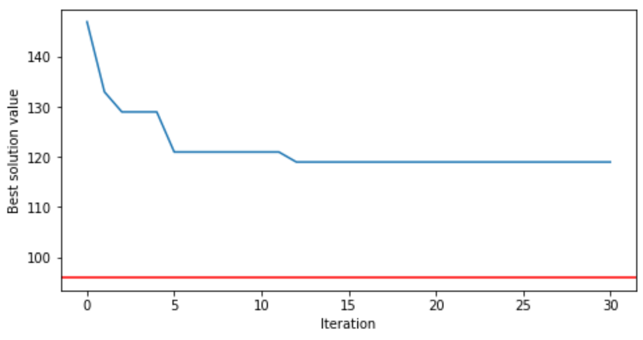


(b) Random 25, 2

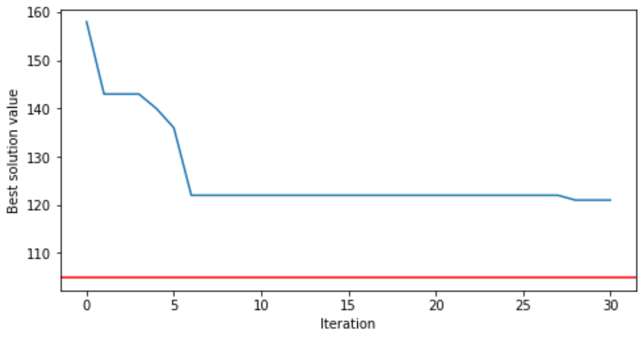


(c) Random 25, 3

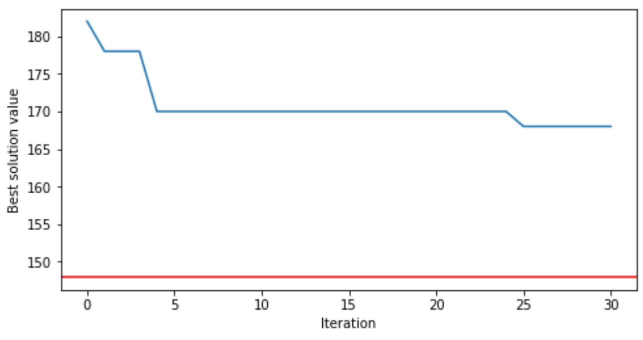
Figure 40: Random 15



(a) Dutch, 1

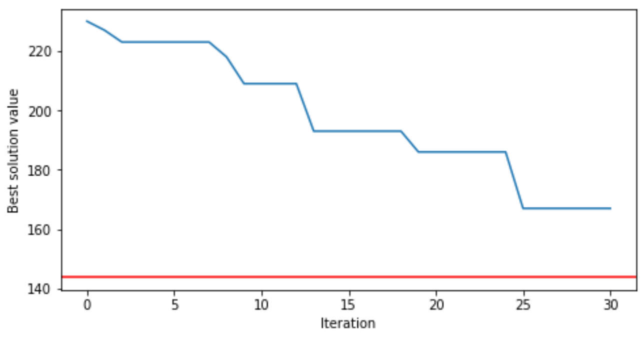


(b) Dutch, 2

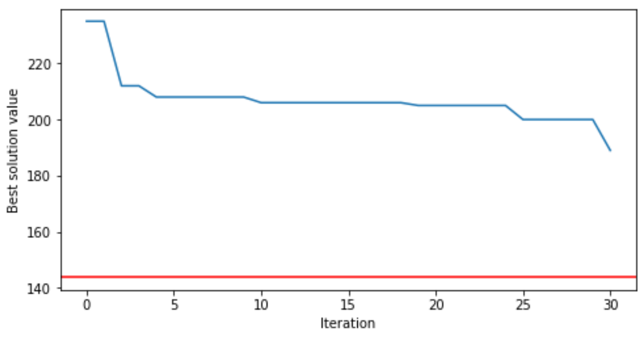


(c) Dutch, 3

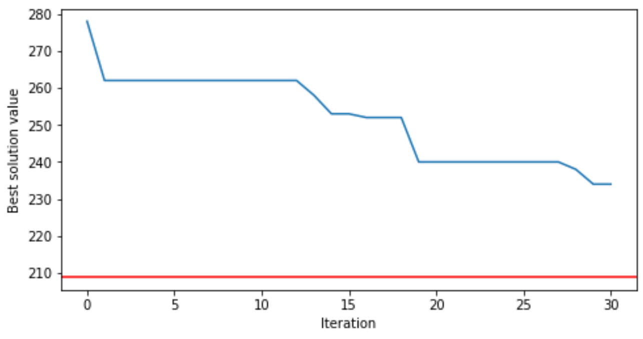
Figure 41: Dutch



(a) Graph 17, 1

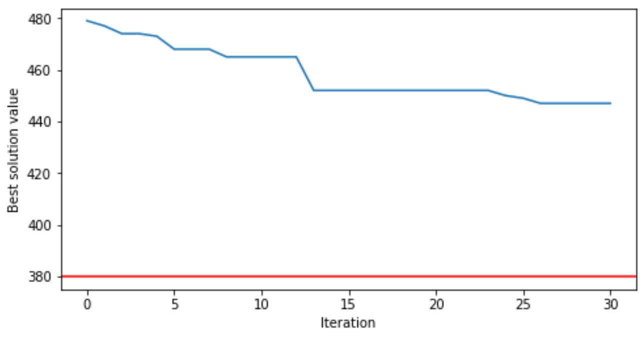


(b) Graph 17, 2

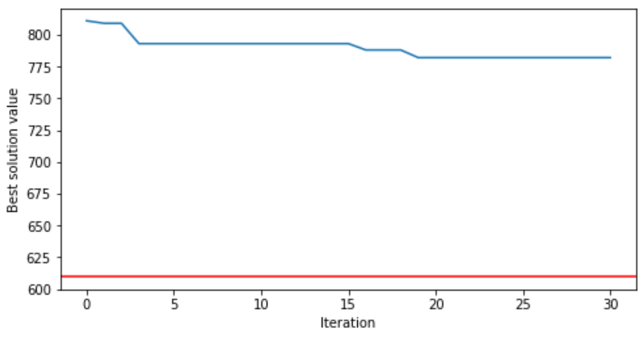


(c) Graph 17, 3

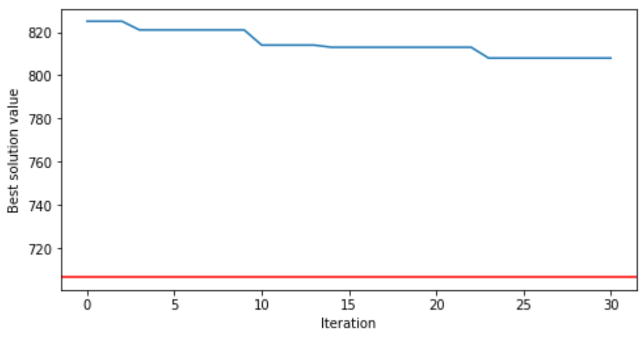
Figure 42: Graph 17



(a) Graph 30, 1



(b) Graph 30, 2



(c) Graph 30, 3

Figure 43: Graph 30

K Results using the optimal solution

K.1 Found solutions using the optimal solution

Table 110: Solution random 15, 1

Container	Used path	Cost
Container 1	[0, 8, 20, 36, 61]	2
Container 2	[0, 8, 26, 36, 61]	3
Container 3	[0, 23, 35, 51, 61]	2
Container 4	[0, 23, 42, 51, 61]	3
Container 5	[0, 3, 28, 35, 61]	3
Container 6	[0, 18, 38, 50, 61]	2
Container 7	[0, 18, 42, 50, 61]	3
Container 8	[0, 18, 43, 50, 61]	3
Container 9	[0, 2, 19, 43, 61]	4
Container 10	[0, 2, 19, 43, 61]	4
Container 11	[0, 17, 34, 58, 61]	4
Container 12	[0, 17, 34, 58, 61]	4
Container 13	[0, 10, 29, 61]	3
Container 14	[0, 40, 59, 61]	3
Container 15	[0, 40, 59, 61]	3
Container 16	[0, 40, 59, 61]	3
Container 17	[0, 5, 21, 41, 61]	2
Container 18	[0, 5, 21, 41, 61]	2
Container 19	[0, 20, 41, 61]	3
Container 20	[0, 20, 36, 56, 61]	2

Table 111: Solution random 15, 2

Container	Used path	Cost
Container 1	[0, 1, 28, 35, 61]	4
Container 2	[0, 16, 43, 50, 61]	4
Container 3	[0, 1, 18, 38, 50, 61]	3
Container 4	[0, 1, 28, 35, 61]	4
Container 5	[0, 26, 33, 61]	3
Container 6	[0, 11, 23, 33, 61]	3
Container 7	[0, 41, 48, 61]	3
Container 8	[0, 11, 18, 61]	3
Container 9	[0, 27, 33, 46, 61]	3
Container 10	[0, 12, 18, 43, 46, 61]	5
Container 11	[0, 12, 19, 43, 46, 61]	7
Container 12	[0, 12, 18, 31, 61]	3
Container 13	[0, 45, 48, 61]	2
Container 14	[0, 30, 33, 61]	2
Container 15	[0, 45, 48, 61]	2
Container 16	[0, 30, 33, 61]	2
Container 17	[0, 19, 40, 61]	2
Container 18	[0, 34, 55, 61]	2
Container 19	[0, 34, 55, 61]	2
Container 20	[0, 19, 40, 61]	2

Table 112: Solution random 15, 3

Container	Used path	Cost
Container 1	[0, 7, 18, 42, 61]	2
Container 2	[0, 7, 18, 42, 61]	2
Container 3	[0, 7, 29, 42, 61]	3
Container 4	[0, 7, 24, 34, 57, 61]	3
Container 5	[0, 7, 29, 42, 61]	3
Container 6	[0, 22, 44, 57, 61]	3
Container 7	[0, 22, 33, 57, 61]	2
Container 8	[0, 22, 33, 57, 61]	2
Container 9	[0, 22, 44, 57, 61]	3
Container 10	[0, 7, 29, 42, 61]	3
Container 11	[0, 5, 27, 40, 61]	4
Container 12	[0, 20, 40, 61]	3
Container 13	[0, 35, 55, 61]	3
Container 14	[0, 35, 55, 61]	3
Container 15	[0, 20, 40, 61]	3
Container 16	[0, 5, 25, 61]	3
Container 17	[0, 20, 42, 55, 61]	4
Container 18	[0, 5, 25, 61]	3
Container 19	[0, 5, 25, 61]	3
Container 20	[0, 35, 55, 61]	3

Table 113: Solution random 20, 1

Container	Used path	Cost
Container 1	[0, 2, 24, 57, 81]	4
Container 2	[0, 2, 26, 50, 77, 81]	4
Container 3	truck	15
Container 4	[0, 22, 44, 77, 81]	4
Container 5	[0, 7, 39, 51, 81]	3
Container 6	[0, 7, 38, 53, 71, 81]	3
Container 7	[0, 27, 50, 71, 81]	3
Container 8	[0, 27, 59, 71, 81]	3
Container 9	[0, 11, 38, 43, 74, 81]	5
Container 10	[0, 11, 39, 54, 81]	5
Container 11	[0, 31, 59, 74, 81]	5
Container 12	[0, 31, 50, 74, 81]	5
Container 13	[0, 5, 33, 60, 81]	2
Container 14	[0, 5, 33, 60, 81]	2
Container 15	[0, 25, 53, 80, 81]	2
Container 16	[0, 25, 53, 80, 81]	2
Container 17	[0, 21, 43, 81]	2
Container 18	[0, 1, 23, 81]	2
Container 19	[0, 21, 54, 63, 81]	3
Container 20	[0, 41, 63, 81]	2

Table 114: Solution random 20, 2

Container	Used path	Cost
Container 1	[0, 13, 39, 81]	2
Container 2	[0, 33, 41, 79, 81]	3
Container 3	[0, 33, 59, 81]	2
Container 4	[0, 53, 79, 81]	2
Container 5	[0, 6, 33, 45, 81]	2
Container 6	[0, 26, 53, 65, 81]	2
Container 7	[0, 6, 33, 45, 81]	2
Container 8	[0, 26, 53, 65, 81]	2
Container 9	[0, 10, 35, 42, 81]	2
Container 10	[0, 30, 55, 62, 81]	2
Container 11	[0, 30, 54, 62, 81]	3
Container 12	[0, 30, 46, 62, 81]	3
Container 13	[0, 21, 42, 81]	2
Container 14	[0, 41, 62, 81]	2
Container 15	[0, 21, 42, 81]	2
Container 16	[0, 21, 42, 81]	2
Container 17	[0, 34, 43, 69, 81]	2
Container 18	truck	15
Container 19	truck	16
Container 20	[0, 34, 43, 69, 81]	2

Table 115: Solution random 20, 3

Container	Used path	Cost
Container 1	[0, 14, 39, 53, 65, 81]	4
Container 2	[0, 14, 30, 45, 81]	3
Container 3	[0, 14, 30, 56, 65, 81]	4
Container 4	[0, 34, 50, 65, 81]	3
Container 5	truck	15
Container 6	[0, 14, 39, 53, 65, 81]	4
Container 7	[0, 14, 40, 46, 65, 81]	5
Container 8	[0, 14, 40, 46, 65, 81]	5
Container 9	[0, 14, 30, 56, 65, 81]	4
Container 10	[0, 14, 22, 44, 65, 81]	4
Container 11	[0, 11, 38, 53, 72, 81]	5
Container 12	[0, 11, 23, 52, 81]	6
Container 13	[0, 11, 33, 52, 81]	6
Container 14	[0, 31, 43, 72, 81]	6
Container 15	[0, 31, 53, 72, 81]	6
Container 16	[0, 11, 38, 52, 81]	3
Container 17	[0, 31, 58, 72, 81]	3
Container 18	truck	15
Container 19	[0, 11, 35, 60, 72, 81]	6
Container 20	truck	16

Table 116: Solution random 25, 1

Container	Used path	Cost
Container 1	[0, 20, 38, 65, 101]	3
Container 2	[0, 20, 44, 65, 101]	4
Container 3	[0, 45, 63, 90, 101]	3
Container 4	[0, 45, 69, 90, 101]	4
Container 5	[0, 23, 43, 54, 101]	4
Container 6	[0, 23, 27, 69, 79, 101]	4
Container 7	[0, 48, 68, 79, 101]	4
Container 8	[0, 48, 68, 79, 101]	4
Container 9	[0, 44, 64, 78, 101]	3
Container 10	[0, 19, 39, 53, 101]	3
Container 11	[0, 44, 54, 78, 101]	5
Container 12	[0, 44, 54, 78, 101]	5
Container 13	[0, 10, 37, 73, 97, 101]	4
Container 14	[0, 10, 33, 63, 97, 101]	4
Container 15	[0, 35, 59, 97, 101]	6
Container 16	truck	15
Container 17	[0, 21, 38, 73, 101]	2
Container 18	[0, 21, 38, 62, 98, 101]	3
Container 19	[0, 46, 63, 98, 101]	2
Container 20	[0, 46, 72, 98, 101]	4

Table 117: Solution random 25, 2

Container	Used path	Cost
Container 1	[0, 9, 30, 66, 101]	5
Container 2	[0, 9, 38, 70, 91, 101]	5
Container 3	[0, 9, 30, 66, 101]	5
Container 4	[0, 9, 38, 62, 91, 101]	4
Container 5	[0, 42, 73, 101]	3
Container 6	[0, 67, 98, 101]	3
Container 7	[0, 17, 30, 61, 98, 101]	4
Container 8	[0, 17, 48, 101]	3
Container 9	[0, 21, 39, 57, 101]	3
Container 10	[0, 46, 64, 82, 101]	3
Container 11	[0, 21, 38, 59, 82, 101]	4
Container 12	[0, 21, 38, 59, 82, 101]	4
Container 13	truck	16
Container 14	[0, 40, 52, 78, 101]	3
Container 15	truck	15
Container 16	[0, 40, 52, 78, 101]	3
Container 17	[0, 39, 62, 91, 101]	4
Container 18	[0, 39, 55, 91, 101]	3
Container 19	[0, 39, 67, 91, 101]	5
Container 20	truck	18

Table 118: Solution random 25, 3

Container	Used path	Cost
Container 1	truck	16
Container 2	[0, 6, 36, 63, 83, 101]	4
Container 3	truck	16
Container 4	[0, 6, 50, 69, 83, 101]	5
Container 5	[0, 6, 27, 69, 83, 101]	4
Container 6	[0, 6, 27, 69, 83, 101]	4
Container 7	[0, 6, 27, 63, 83, 101]	5
Container 8	truck	17
Container 9	truck	15
Container 10	truck	16
Container 11	[0, 18, 40, 63, 84, 101]	5
Container 12	[0, 18, 48, 72, 84, 101]	3
Container 13	[0, 43, 60, 84, 101]	4
Container 14	truck	15
Container 15	[0, 18, 48, 63, 84, 101]	5
Container 16	[0, 18, 41, 70, 84, 101]	6
Container 17	[0, 18, 35, 59, 101]	4
Container 18	[0, 18, 41, 72, 84, 101]	4
Container 19	[0, 18, 30, 59, 101]	4
Container 20	[0, 43, 55, 84, 101]	4

Table 119: Solution Dutch, 1

Container	Used path	Cost
Container 1	[0, 36, 45, 67]	1
Container 2	truck	15
Container 3	[0, 14, 37, 57, 67]	3
Container 4	[0, 14, 28, 38, 67]	4
Container 5	[0, 3, 12, 67]	1
Container 6	[0, 14, 23, 67]	1
Container 7	[0, 3, 26, 46, 67]	3
Container 8	truck	10
Container 9	[0, 3, 26, 51, 67]	3
Container 10	[0, 14, 37, 62, 67]	3
Container 11	[0, 8, 25, 34, 67]	4
Container 12	[0, 41, 58, 67]	3
Container 13	truck	12
Container 14	[0, 8, 17, 27, 67]	4
Container 15	[0, 30, 39, 51, 67]	5
Container 16	truck	10
Container 17	[0, 30, 42, 67]	2
Container 18	[0, 41, 53, 65, 67]	4
Container 19	[0, 30, 39, 67]	3
Container 20	[0, 19, 31, 43, 55, 67]	5

Table 120: Solution Dutch, 2

Container	Used path	Cost
Container 1	[0, 14, 23, 67]	1
Container 2	[0, 25, 34, 67]	1
Container 3	[0, 36, 45, 67]	1
Container 4	truck	10
Container 5	[0, 14, 37, 57, 67]	3
Container 6	[0, 3, 26, 46, 67]	3
Container 7	truck	10
Container 8	truck	12
Container 9	[0, 14, 37, 62, 67]	3
Container 10	[0, 3, 26, 51, 67]	3
Container 11	[0, 19, 28, 38, 67]	4
Container 12	[0, 19, 31, 43, 55, 67]	5
Container 13	[0, 8, 17, 27, 67]	4
Container 14	truck	13
Container 15	[0, 41, 53, 65, 67]	4
Container 16	[0, 8, 17, 67]	3
Container 17	[0, 41, 50, 62, 67]	5
Container 18	[0, 8, 17, 29, 67]	5
Container 19	[0, 8, 20, 32, 44, 67]	5
Container 20	truck	10

Table 121: Solution Dutch, 3

Container	Used path	Cost
Container 1	truck	10
Container 2	[0, 14, 37, 57, 67]	3
Container 3	[0, 3, 26, 46, 67]	3
Container 4	truck	13
Container 5	truck	10
Container 6	[0, 14, 28, 40, 67]	5
Container 7	[0, 36, 50, 62, 67]	5
Container 8	[0, 3, 26, 51, 67]	3
Container 9	[0, 3, 17, 29, 67]	5
Container 10	[0, 14, 37, 62, 67]	3
Container 11	truck	10
Container 12	[0, 8, 20, 32, 44, 67]	5
Container 13	truck	14
Container 14	truck	12
Container 15	[0, 8, 17, 27, 67]	4
Container 16	truck	10
Container 17	[0, 19, 28, 38, 67]	4
Container 18	[0, 19, 31, 43, 55, 67]	5
Container 19	truck	12
Container 20	truck	12

Table 122: Solution Graph17, 1

Container	Used path	Cost
Container 1	[0, 1, 21, 44, 66, 120]	6
Container 2	[0, 2, 25, 45, 62, 79, 99, 117, 120]	10
Container 3	[0, 20, 43, 65, 83, 120]	6
Container 4	[0, 3, 19, 42, 63, 120]	6
Container 5	[0, 38, 52, 70, 93, 114, 120]	8
Container 6	[0, 6, 26, 45, 59, 80, 120]	8
Container 7	[0, 5, 24, 46, 67, 85, 120]	8
Container 8	[0, 7, 29, 50, 68, 120]	6
Container 9	[0, 36, 54, 77, 99, 119, 120]	8
Container 10	[0, 3, 26, 48, 68, 120]	6
Container 11	[0, 18, 38, 61, 83, 120]	6
Container 12	[0, 19, 35, 55, 78, 100, 120]	8
Container 13	[0, 54, 74, 95, 117, 120]	6
Container 14	[0, 20, 36, 56, 75, 97, 120]	8
Container 15	[0, 21, 35, 53, 73, 92, 114, 120]	10
Container 16	[0, 23, 37, 53, 76, 97, 120]	8
Container 17	[0, 39, 58, 80, 101, 119, 120]	8
Container 18	[0, 41, 63, 84, 102, 120]	6
Container 19	truck	17
Container 20	[0, 37, 60, 82, 102, 120]	6

Table 123: Solution Graph17, 2

Container	Used path	Cost
Container 1	[0, 1, 21, 44, 66, 120]	6
Container 2	[0, 2, 18, 38, 61, 83, 120]	8
Container 3	[0, 20, 43, 65, 83, 120]	6
Container 4	[0, 20, 35, 53, 73, 92, 114, 120]	9
Container 5	[0, 4, 18, 36, 59, 80, 120]	8
Container 6	[0, 6, 26, 45, 64, 81, 101, 114, 120]	11
Container 7	[0, 39, 59, 79, 99, 119, 120]	8
Container 8	[0, 7, 29, 50, 68, 120]	6
Container 9	[0, 2, 25, 45, 65, 85, 120]	8
Container 10	[0, 37, 60, 82, 102, 120]	6
Container 11	[0, 35, 55, 78, 100, 120]	6
Container 12	[0, 19, 37, 54, 77, 99, 117, 120]	9
Container 13	[0, 37, 57, 78, 95, 117, 120]	7
Container 14	[0, 20, 36, 56, 75, 97, 120]	8
Container 15	[0, 38, 52, 70, 93, 114, 120]	8
Container 16	[0, 23, 37, 53, 76, 97, 120]	8
Container 17	[0, 5, 24, 46, 67, 85, 120]	8
Container 18	[0, 58, 80, 101, 119, 120]	6
Container 19	[0, 19, 42, 63, 84, 102, 120]	8
Container 20	[0, 3, 26, 48, 68, 120]	6

Table 124: Solution Graph17, 3

Container	Used path	Cost
Container 1	truck	17
Container 2	[0, 1, 21, 44, 66, 120]	6
Container 3	[0, 52, 72, 95, 117, 120]	6
Container 4	[0, 18, 38, 61, 83, 120]	6
Container 5	[0, 35, 55, 78, 100, 120]	6
Container 6	[0, 35, 54, 77, 99, 119, 120]	7
Container 7	truck	17
Container 8	[0, 18, 37, 60, 82, 102, 120]	7
Container 9	truck	16
Container 10	[0, 1, 20, 43, 65, 85, 120]	7
Container 11	[0, 18, 36, 59, 80, 101, 120]	8
Container 12	truck	17
Container 13	[0, 35, 53, 76, 97, 118, 120]	8
Container 14	[0, 1, 19, 42, 63, 84, 120]	8
Container 15	truck	18
Container 16	[0, 1, 20, 36, 56, 75, 97, 120]	9
Container 17	[0, 18, 37, 53, 73, 92, 114, 120]	9
Container 18	truck	15
Container 19	truck	16
Container 20	[0, 52, 70, 93, 114, 120]	6

Table 125: Solution Graph30, 1

Container	Used path	Cost
Container 1	truck	16
Container 2	truck	18
Container 3	[0, 33, 69, 104, 137, 172, 204, 239, 270, 271]	14
Container 4	[0, 34, 70, 105, 138, 175, 209, 240, 271]	12
Container 5	truck	15
Container 6	truck	18
Container 7	truck	16
Container 8	truck	16
Container 9	truck	16
Container 10	[0, 10, 45, 78, 115, 149, 180, 271]	10
Container 11	[0, 121, 152, 185, 217, 252, 271]	8
Container 12	[0, 92, 128, 162, 271]	4
Container 13	[0, 93, 122, 158, 192, 271]	6
Container 14	[0, 124, 151, 182, 218, 252, 271]	8
Container 15	[0, 95, 127, 162, 271]	4
Container 16	[0, 31, 64, 100, 135, 168, 204, 233, 266, 271]	13
Container 17	[0, 62, 98, 132, 166, 200, 236, 271]	10
Container 18	[0, 63, 99, 134, 167, 199, 230, 266, 271]	12
Container 19	truck	16
Container 20	[0, 35, 67, 102, 136, 170, 203, 236, 271]	12
Container 21	truck	18
Container 22	[0, 37, 72, 106, 140, 176, 271]	8
Container 23	[0, 8, 42, 76, 110, 146, 271]	8
Container 24	[0, 39, 74, 107, 139, 170, 206, 271]	10
Container 25	[0, 40, 75, 108, 144, 173, 206, 271]	9
Container 26	truck	20
Container 27	truck	17
Container 28	truck	20
Container 29	truck	15
Container 30	truck	19

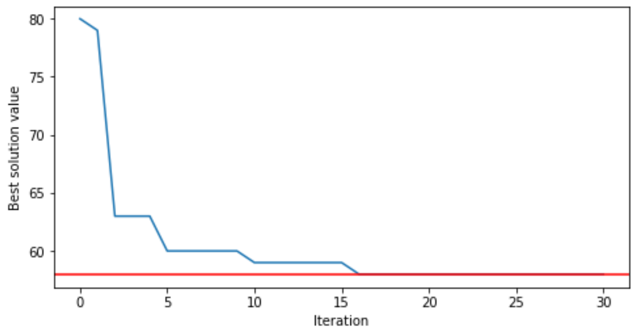
Table 126: Solution Graph30, 2

Container	Used path	Cost
Container 1	truck	20
Container 2	truck	19
Container 3	truck	15
Container 4	truck	16
Container 5	truck	15
Container 6	truck	17
Container 7	truck	19
Container 8	truck	19
Container 9	[0, 39, 74, 107, 142, 174, 209, 240, 271]	12
Container 10	[0, 10, 45, 78, 115, 149, 180, 271]	10
Container 11	[0, 31, 62, 95, 127, 162, 271]	8
Container 12	[0, 92, 128, 162, 271]	4
Container 13	[0, 3, 32, 65, 97, 132, 271]	8
Container 14	[0, 124, 151, 182, 218, 252, 271]	8
Container 15	[0, 5, 37, 72, 271]	4
Container 16	truck	17
Container 17	truck	15
Container 18	truck	17
Container 19	[0, 64, 100, 135, 168, 204, 233, 266, 271]	11
Container 20	[0, 35, 67, 102, 136, 170, 206, 271]	10
Container 21	[0, 31, 63, 99, 134, 167, 199, 230, 266, 271]	13
Container 22	truck	17
Container 23	[0, 33, 69, 104, 137, 169, 200, 236, 271]	12
Container 24	[0, 34, 70, 105, 138, 174, 203, 236, 271]	11
Container 25	truck	16
Container 26	truck	17
Container 27	[0, 7, 42, 76, 110, 146, 271]	8
Container 28	truck	16
Container 29	[0, 9, 44, 77, 109, 140, 176, 271]	10
Container 30	truck	15
Container 31	truck	15
Container 32	truck	18
Container 33	truck	16
Container 34	truck	17
Container 35	truck	18
Container 36	truck	20
Container 37	truck	17
Container 38	truck	18
Container 39	truck	19
Container 40	[0, 40, 75, 108, 145, 179, 210, 271]	10
Container 41	[0, 91, 122, 155, 187, 222, 271]	8
Container 42	[0, 32, 68, 102, 271]	4
Container 43	[0, 93, 122, 158, 192, 271]	6
Container 44	[0, 34, 61, 92, 125, 157, 192, 271]	10
Container 45	[0, 155, 188, 222, 271]	4
Container 46	[0, 96, 129, 164, 197, 226, 252, 271]	10
Container 47	[0, 217, 252, 271]	2
Container 48	[0, 38, 72, 271]	2
Container 49	[0, 39, 71, 98, 132, 271]	6
Container 50	truck	15

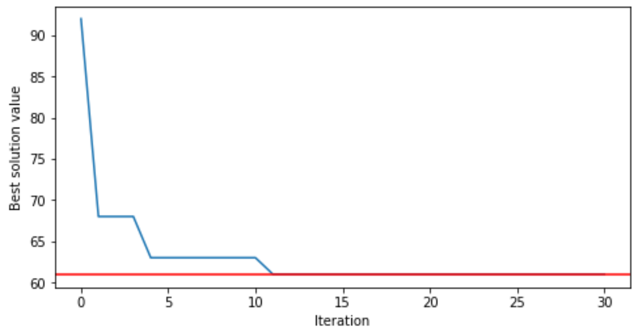
Table 127: Solution Graph30, 3

Container	Used path	Cost
Container 1	truck	16
Container 2	truck	19
Container 3	truck	18
Container 4	[0, 1, 33, 69, 104, 137, 172, 204, 239, 270, 271]	15
Container 5	truck	15
Container 6	truck	19
Container 7	truck	18
Container 8	truck	18
Container 9	truck	17
Container 10	truck	15
Container 11	truck	16
Container 12	[0, 61, 92, 128, 162, 271]	6
Container 13	[0, 151, 182, 218, 252, 271]	6
Container 14	[0, 1, 33, 62, 98, 132, 271]	7
Container 15	[0, 1, 32, 68, 102, 271]	6
Container 16	[0, 31, 63, 99, 134, 167, 199, 230, 266, 271]	13
Container 17	truck	16
Container 18	truck	16
Container 19	truck	16
Container 20	truck	19
Container 21	truck	16
Container 22	[0, 1, 34, 70, 105, 138, 174, 203, 236, 271]	13
Container 23	truck	18
Container 24	[0, 31, 64, 100, 135, 168, 204, 233, 266, 271]	13
Container 25	truck	15
Container 26	truck	18
Container 27	[0, 2, 38, 72, 106, 140, 176, 271]	10
Container 28	[0, 2, 35, 67, 102, 136, 170, 206, 271]	12
Container 29	truck	16
Container 30	[0, 32, 65, 97, 132, 166, 200, 236, 271]	12
Container 31	truck	17
Container 32	truck	15
Container 33	truck	16
Container 34	truck	20
Container 35	truck	16
Container 36	truck	19
Container 37	truck	15
Container 38	truck	20
Container 39	truck	19
Container 40	truck	17
Container 41	truck	18
Container 42	[0, 64, 91, 122, 158, 192, 271]	8
Container 43	[0, 64, 96, 129, 161, 193, 226, 252, 271]	12
Container 44	truck	16
Container 45	[0, 61, 93, 123, 152, 188, 222, 271]	8
Container 46	truck	15
Container 47	truck	17
Container 48	[0, 31, 62, 95, 127, 162, 271]	8
Container 49	[0, 61, 93, 122, 155, 187, 222, 271]	9
Container 50	[0, 121, 152, 185, 217, 252, 271]	8

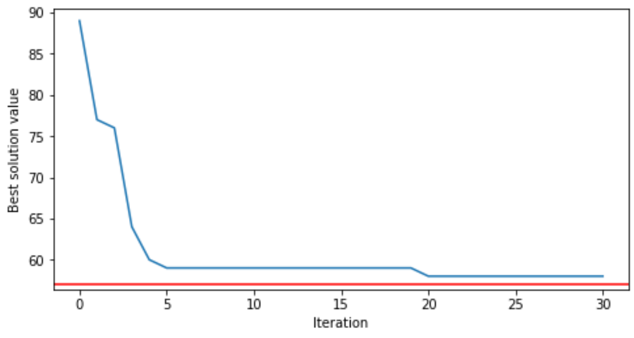
K.2 Course of the algorithm using the optimal solution



(a) Random 15, 1

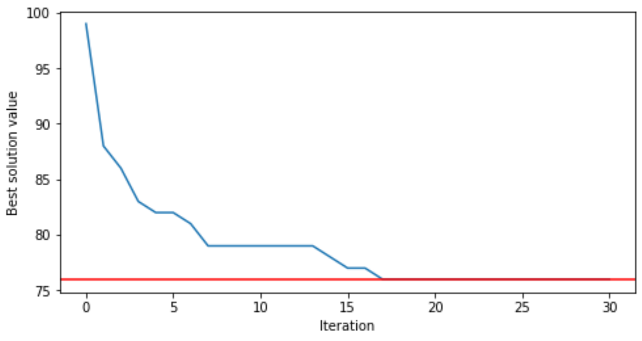


(b) Random 15, 2

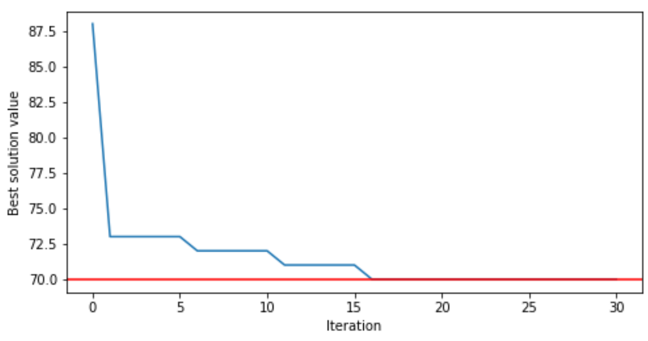


(c) Random 15, 3

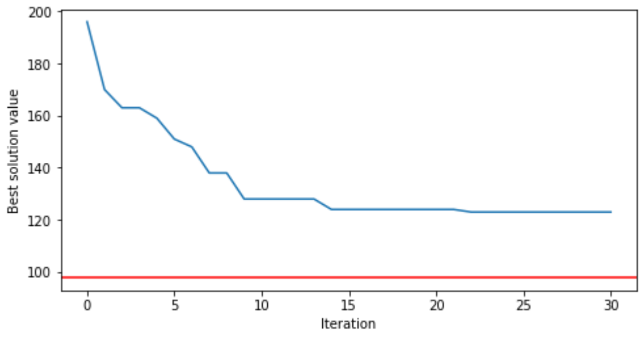
Figure 44: Random 15



(a) Random 20, 1

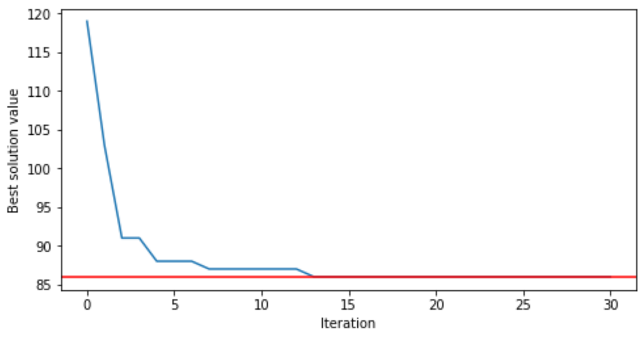


(b) Random 20, 2

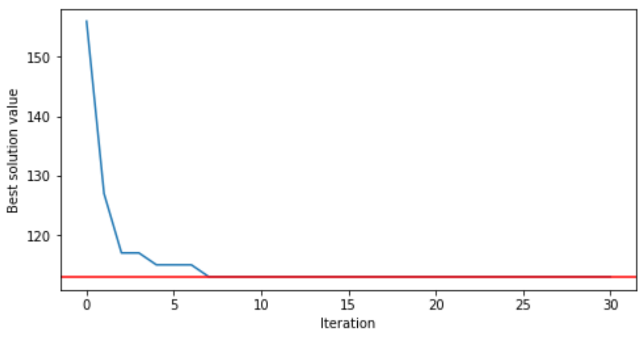


(c) Random 20, 3

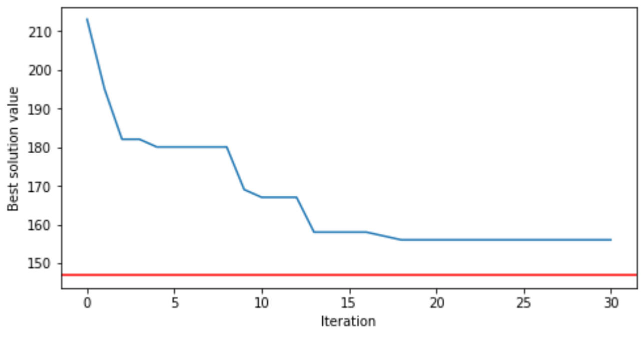
Figure 45: Random 20



(a) Random 25, 1

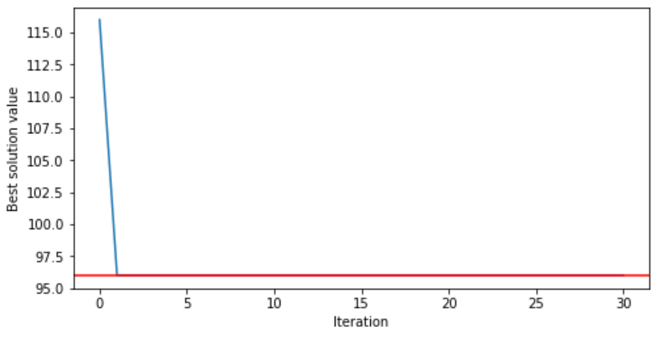


(b) Random 25, 2

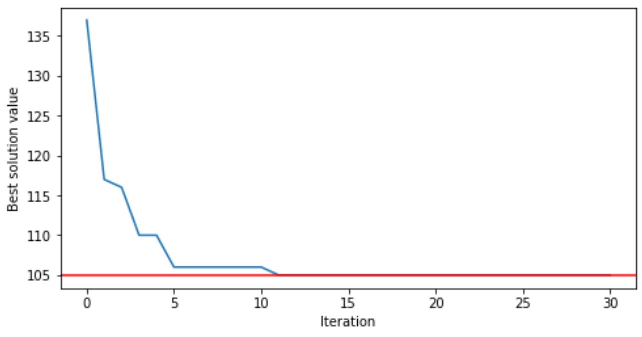


(c) Random 25, 3

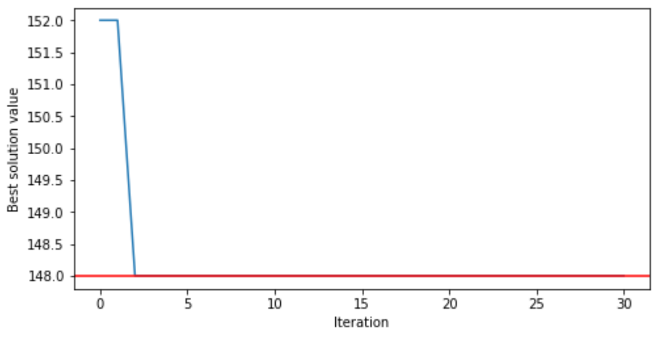
Figure 46: Random 15



(a) Dutch, 1

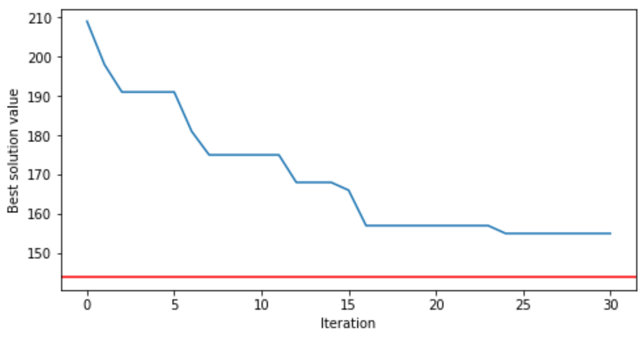


(b) Dutch, 2

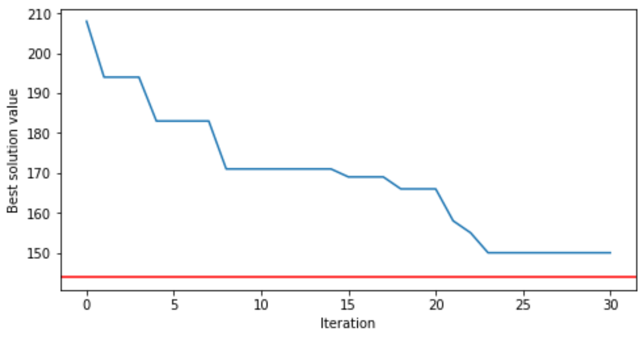


(c) Dutch, 3

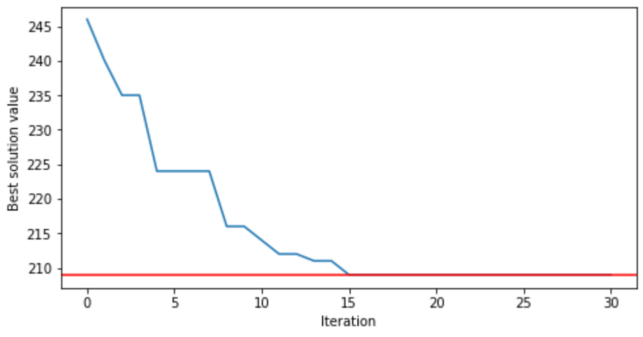
Figure 47: Dutch



(a) Graph 17, 1

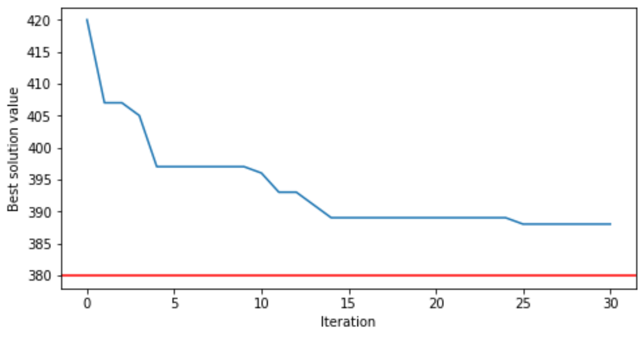


(b) Graph 17, 2

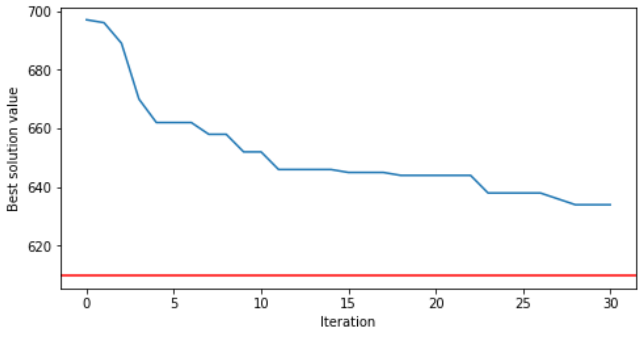


(c) Graph 17, 3

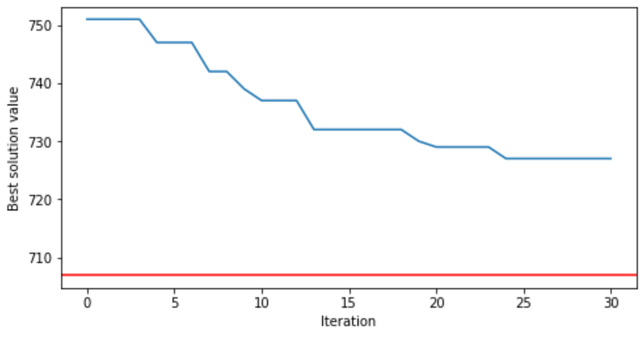
Figure 48: Graph 17



(a) Graph 30, 1



(b) Graph 30, 2



(c) Graph 30, 3

Figure 49: Graph 30

L Results using the quantum annealer

L.1 Found solution using the quantum annealer

Table 128: Solution random 15, 1

Container	Used path	Cost
Container 1	[0, 8, 20, 36, 61]	2
Container 2	[0, 8, 26, 36, 61]	3
Container 3	[0, 23, 41, 51, 61]	3
Container 4	[0, 23, 42, 51, 61]	3
Container 5	[0, 3, 28, 35, 61]	3
Container 6	[0, 3, 23, 35, 61]	2
Container 7	[0, 18, 42, 50, 61]	3
Container 8	[0, 18, 38, 50, 61]	2
Container 9	[0, 2, 19, 43, 61]	4
Container 10	[0, 2, 19, 43, 61]	4
Container 11	[0, 17, 34, 58, 61]	4
Container 12	[0, 17, 34, 58, 61]	4
Container 13	[0, 10, 29, 61]	3
Container 14	[0, 40, 59, 61]	3
Container 15	[0, 40, 59, 61]	3
Container 16	[0, 40, 59, 61]	3
Container 17	[0, 5, 21, 41, 61]	2
Container 18	[0, 5, 21, 41, 61]	2
Container 19	[0, 20, 41, 61]	3
Container 20	[0, 20, 36, 56, 61]	2

Table 129: Solution random 15, 2

Container	Used path	Cost
Container 1	[0, 1, 28, 35, 61]	4
Container 2	[0, 16, 43, 50, 61]	4
Container 3	[0, 16, 43, 50, 61]	4
Container 4	[0, 1, 18, 38, 50, 61]	3
Container 5	[0, 41, 48, 61]	3
Container 6	[0, 26, 33, 61]	3
Container 7	[0, 11, 18, 61]	3
Container 8	[0, 11, 24, 37, 48, 61]	3
Container 9	[0, 27, 33, 46, 61]	3
Container 10	[0, 12, 18, 43, 46, 61]	5
Container 11	[0, 12, 20, 43, 46, 61]	7
Container 12	[0, 12, 18, 31, 61]	3
Container 13	[0, 45, 48, 61]	2
Container 14	[0, 45, 48, 61]	2
Container 15	[0, 30, 33, 61]	2
Container 16	[0, 30, 33, 61]	2
Container 17	[0, 19, 40, 61]	2
Container 18	[0, 34, 55, 61]	2
Container 19	[0, 34, 55, 61]	2
Container 20	[0, 19, 40, 61]	2

Table 130: Solution random 15, 3

Container	Used path	Cost
Container 1	[0, 22, 44, 57, 61]	3
Container 2	[0, 7, 24, 34, 57, 61]	3
Container 3	[0, 22, 33, 57, 61]	2
Container 4	[0, 7, 29, 42, 61]	3
Container 5	[0, 22, 33, 57, 61]	2
Container 6	[0, 7, 18, 42, 61]	2
Container 7	[0, 7, 18, 42, 61]	2
Container 8	[0, 7, 29, 42, 61]	3
Container 9	[0, 7, 29, 42, 61]	3
Container 10	[0, 22, 44, 57, 61]	3
Container 11	[0, 35, 55, 61]	3
Container 12	[0, 5, 25, 61]	3
Container 13	[0, 5, 25, 61]	3
Container 14	[0, 5, 25, 61]	3
Container 15	[0, 20, 42, 55, 61]	4
Container 16	[0, 35, 55, 61]	3
Container 17	[0, 20, 40, 61]	3
Container 18	[0, 20, 40, 61]	3
Container 19	[0, 35, 55, 61]	3
Container 20	[0, 20, 40, 61]	3

Table 131: Solution random 20, 1

Container	Used path	Cost
Container 1	[0, 2, 24, 57, 81]	4
Container 2	[0, 2, 21, 59, 77, 81]	5
Container 3	truck	15
Container 4	[0, 22, 44, 77, 81]	4
Container 5	[0, 7, 38, 53, 71, 81]	3
Container 6	[0, 7, 30, 51, 81]	3
Container 7	[0, 27, 59, 71, 81]	3
Container 8	[0, 27, 50, 71, 81]	3
Container 9	[0, 31, 59, 74, 81]	5
Container 10	[0, 11, 39, 54, 81]	5
Container 11	[0, 31, 43, 74, 81]	6
Container 12	[0, 31, 59, 74, 81]	5
Container 13	truck	15
Container 14	[0, 5, 33, 60, 81]	2
Container 15	[0, 25, 56, 80, 81]	5
Container 16	[0, 25, 53, 80, 81]	2
Container 17	[0, 41, 63, 81]	2
Container 18	[0, 1, 23, 81]	2
Container 19	[0, 21, 54, 63, 81]	3
Container 20	truck	15

Table 132: Solution random 20, 2

Container	Used path	Cost
Container 1	[0, 33, 41, 79, 81]	3
Container 2	[0, 33, 41, 79, 81]	3
Container 3	[0, 13, 21, 59, 81]	3
Container 4	[0, 13, 39, 81]	2
Container 5	[0, 46, 65, 81]	3
Container 6	[0, 46, 65, 81]	3
Container 7	[0, 6, 30, 45, 81]	4
Container 8	[0, 26, 53, 65, 81]	2
Container 9	[0, 10, 35, 46, 62, 81]	3
Container 10	[0, 10, 35, 42, 81]	2
Container 11	truck	20
Container 12	[0, 30, 46, 62, 81]	3
Container 13	[0, 21, 42, 81]	2
Container 14	[0, 41, 62, 81]	2
Container 15	[0, 21, 42, 81]	2
Container 16	[0, 21, 42, 81]	2
Container 17	[0, 34, 43, 69, 81]	2
Container 18	truck	15
Container 19	[0, 34, 43, 69, 81]	2
Container 20	truck	19

Table 133: Solution random 20, 3

Container	Used path	Cost
Container 1	[0, 34, 50, 65, 81]	3
Container 2	truck	16
Container 3	[0, 14, 40, 46, 65, 81]	5
Container 4	[0, 14, 30, 45, 81]	3
Container 5	[0, 14, 30, 56, 65, 81]	4
Container 6	truck	16
Container 7	[0, 14, 22, 44, 65, 81]	4
Container 8	[0, 14, 39, 53, 65, 81]	4
Container 9	[0, 14, 30, 56, 65, 81]	4
Container 10	[0, 14, 39, 53, 65, 81]	4
Container 11	truck	18
Container 12	[0, 11, 33, 52, 81]	6
Container 13	truck	18
Container 14	[0, 31, 53, 72, 81]	6
Container 15	[0, 11, 38, 52, 81]	3
Container 16	truck	16
Container 17	[0, 31, 43, 72, 81]	6
Container 18	truck	15
Container 19	[0, 11, 35, 60, 72, 81]	6
Container 20	[0, 11, 38, 53, 72, 81]	5

Table 134: Solution random 25, 1

Container	Used path	Cost
Container 1	[0, 20, 26, 57, 90, 101]	4
Container 2	[0, 20, 44, 65, 101]	4
Container 3	[0, 45, 63, 90, 101]	3
Container 4	[0, 45, 69, 90, 101]	4
Container 5	[0, 23, 47, 51, 79, 101]	5
Container 6	truck	15
Container 7	[0, 48, 68, 79, 101]	4
Container 8	[0, 48, 68, 79, 101]	4
Container 9	[0, 44, 64, 78, 101]	3
Container 10	[0, 19, 39, 53, 101]	3
Container 11	truck	20
Container 12	[0, 44, 54, 78, 101]	5
Container 13	[0, 10, 37, 73, 97, 101]	4
Container 14	[0, 10, 33, 63, 97, 101]	4
Container 15	[0, 35, 59, 97, 101]	6
Container 16	truck	15
Container 17	[0, 21, 47, 73, 101]	4
Container 18	[0, 21, 39, 62, 98, 101]	4
Container 19	[0, 46, 75, 98, 101]	5
Container 20	[0, 46, 63, 98, 101]	2

Table 135: Solution random 25, 2

Container	Used path	Cost
Container 1	[0, 34, 72, 91, 101]	6
Container 2	[0, 9, 30, 66, 101]	5
Container 3	[0, 9, 30, 66, 101]	5
Container 4	[0, 9, 38, 72, 91, 101]	5
Container 5	[0, 17, 48, 101]	3
Container 6	[0, 42, 73, 101]	3
Container 7	truck	18
Container 8	[0, 17, 30, 75, 98, 101]	4
Container 9	[0, 21, 39, 57, 101]	3
Container 10	[0, 21, 26, 57, 101]	5
Container 11	truck	19
Container 12	[0, 46, 64, 82, 101]	3
Container 13	truck	16
Container 14	[0, 40, 52, 78, 101]	3
Container 15	truck	15
Container 16	[0, 40, 52, 78, 101]	3
Container 17	[0, 39, 55, 91, 101]	3
Container 18	[0, 39, 62, 91, 101]	4
Container 19	[0, 39, 67, 91, 101]	5
Container 20	truck	18

Table 136: Solution random 25, 3

Container	Used path	Cost
Container 1	truck	16
Container 2	[0, 6, 27, 69, 83, 101]	4
Container 3	truck	16
Container 4	[0, 6, 36, 63, 83, 101]	4
Container 5	truck	19
Container 6	[0, 6, 27, 69, 83, 101]	4
Container 7	[0, 6, 27, 63, 83, 101]	5
Container 8	truck	17
Container 9	truck	15
Container 10	[0, 6, 50, 69, 83, 101]	5
Container 11	truck	17
Container 12	truck	17
Container 13	truck	19
Container 14	[0, 18, 48, 72, 84, 101]	3
Container 15	truck	16
Container 16	[0, 18, 35, 59, 101]	4
Container 17	[0, 43, 60, 84, 101]	4
Container 18	[0, 18, 41, 72, 84, 101]	4
Container 19	[0, 43, 55, 84, 101]	4
Container 20	[0, 18, 30, 59, 101]	4

Table 137: Solution Dutch, 1

Container	Used path	Cost
Container 1	[0, 3, 12, 67]	1
Container 2	[0, 3, 26, 46, 67]	3
Container 3	truck	15
Container 4	truck	15
Container 5	[0, 14, 23, 67]	1
Container 6	[0, 36, 45, 67]	1
Container 7	[0, 14, 37, 57, 67]	3
Container 8	[0, 14, 28, 38, 67]	4
Container 9	[0, 14, 37, 62, 67]	3
Container 10	truck	10
Container 11	truck	10
Container 12	[0, 8, 25, 67]	3
Container 13	truck	12
Container 14	[0, 8, 17, 27, 67]	4
Container 15	truck	11
Container 16	truck	10
Container 17	[0, 19, 31, 67]	2
Container 18	[0, 30, 42, 54, 67]	4
Container 19	[0, 30, 39, 67]	3
Container 20	[0, 8, 20, 32, 44, 67]	5

Table 138: Solution Dutch, 2

Container	Used path	Cost
Container 1	[0, 3, 12, 67]	1
Container 2	[0, 36, 45, 67]	1
Container 3	[0, 14, 23, 67]	1
Container 4	truck	10
Container 5	[0, 3, 26, 46, 67]	3
Container 6	[0, 14, 37, 57, 67]	3
Container 7	[0, 3, 17, 27, 67]	4
Container 8	truck	12
Container 9	truck	10
Container 10	[0, 3, 17, 29, 67]	5
Container 11	truck	15
Container 12	[0, 8, 20, 32, 44, 67]	5
Container 13	[0, 19, 28, 38, 67]	4
Container 14	truck	13
Container 15	[0, 19, 28, 39, 65, 67]	6
Container 16	[0, 41, 50, 67]	3
Container 17	[0, 30, 39, 50, 62, 67]	6
Container 18	[0, 19, 28, 40, 67]	5
Container 19	[0, 19, 31, 43, 55, 67]	5
Container 20	truck	10

Table 139: Solution Dutch, 3

Container	Used path	Cost
Container 1	truck	10
Container 2	[0, 3, 26, 46, 67]	3
Container 3	[0, 14, 37, 57, 67]	3
Container 4	truck	13
Container 5	truck	10
Container 6	[0, 14, 28, 40, 67]	5
Container 7	truck	13
Container 8	[0, 14, 37, 62, 67]	3
Container 9	[0, 25, 39, 51, 67]	5
Container 10	[0, 3, 26, 51, 67]	3
Container 11	truck	10
Container 12	[0, 8, 20, 32, 44, 67]	5
Container 13	truck	14
Container 14	truck	12
Container 15	truck	15
Container 16	truck	10
Container 17	[0, 8, 17, 27, 67]	4
Container 18	[0, 8, 17, 43, 55, 67]	6
Container 19	[0, 19, 28, 38, 67]	4
Container 20	truck	12

Table 140: Solution Graph 17, 1

Container	Used path	Cost
Container 1	[0, 1, 21, 44, 66, 120]	6
Container 2	[0, 36, 59, 79, 99, 117, 120]	8
Container 3	[0, 3, 26, 48, 66, 120]	6
Container 4	[0, 20, 36, 56, 75, 97, 120]	8
Container 5	truck	16
Container 6	truck	16
Container 7	truck	15
Container 8	[0, 7, 29, 50, 68, 120]	6
Container 9	[0, 19, 37, 60, 82, 99, 119, 120]	9
Container 10	truck	15
Container 11	[0, 18, 38, 61, 83, 120]	6
Container 12	[0, 36, 52, 72, 95, 117, 120]	8
Container 13	[0, 37, 57, 78, 100, 120]	6
Container 14	truck	19
Container 15	[0, 38, 52, 70, 93, 114, 120]	8
Container 16	[0, 23, 37, 53, 73, 92, 114, 120]	10
Container 17	[0, 39, 58, 80, 101, 119, 120]	8
Container 18	[0, 41, 63, 84, 102, 120]	6
Container 19	[0, 19, 42, 62, 82, 102, 120]	8
Container 20	[0, 20, 43, 65, 85, 120]	6

Table 141: Solution Graph 17, 2

Container	Used path	Cost
Container 1	[0, 35, 55, 78, 100, 120]	6
Container 2	[0, 19, 42, 62, 79, 99, 117, 120]	9
Container 3	[0, 37, 52, 72, 95, 117, 120]	7
Container 4	[0, 37, 53, 73, 92, 114, 120]	8
Container 5	[0, 4, 18, 36, 59, 80, 120]	8
Container 6	[0, 6, 26, 43, 62, 76, 97, 120]	9
Container 7	[0, 5, 25, 45, 65, 85, 120]	8
Container 8	[0, 7, 29, 50, 68, 120]	6
Container 9	truck	20
Container 10	truck	16
Container 11	[0, 1, 21, 44, 66, 120]	6
Container 12	truck	17
Container 13	[0, 3, 26, 48, 66, 120]	6
Container 14	[0, 3, 19, 39, 58, 80, 120]	8
Container 15	[0, 21, 35, 54, 70, 93, 114, 120]	9
Container 16	[0, 6, 20, 36, 56, 75, 97, 120]	10
Container 17	[0, 5, 24, 46, 67, 85, 120]	8
Container 18	[0, 41, 63, 84, 102, 120]	6
Container 19	[0, 19, 37, 60, 82, 102, 120]	8
Container 20	truck	18

Table 142: Solution Graph 17, 3

Container	Used path	Cost
Container 1	truck	17
Container 2	[0, 18, 38, 61, 83, 120]	6
Container 3	[0, 52, 72, 95, 117, 120]	6
Container 4	[0, 35, 55, 78, 100, 120]	6
Container 5	[0, 1, 21, 44, 66, 120]	6
Container 6	truck	19
Container 7	[0, 35, 54, 77, 99, 119, 120]	7
Container 8	truck	17
Container 9	truck	16
Container 10	[0, 18, 37, 60, 82, 102, 120]	7
Container 11	[0, 35, 53, 76, 97, 118, 120]	8
Container 12	truck	17
Container 13	[0, 1, 19, 39, 58, 80, 101, 120]	10
Container 14	truck	18
Container 15	truck	18
Container 16	truck	20
Container 17	[0, 1, 20, 37, 53, 73, 92, 114, 120]	10
Container 18	[0, 1, 20, 43, 65, 85, 101, 114, 120]	11
Container 19	truck	16
Container 20	[0, 52, 70, 93, 114, 120]	6

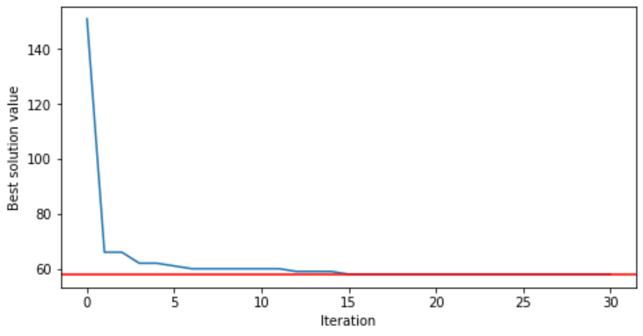
Table 143: Solution Graph 30, 1

Container	Used path	Cost
Container 1	truck	16
Container 2	truck	18
Container 3	[0, 3, 39, 74, 105, 138, 175, 209, 240, 271]	14
Container 4	[0, 4, 40, 75, 108, 145, 179, 210, 271]	12
Container 5	truck	15
Container 6	truck	18
Container 7	truck	16
Container 8	truck	16
Container 9	truck	16
Container 10	[0, 10, 45, 78, 115, 149, 180, 271]	10
Container 11	[0, 61, 92, 125, 157, 192, 271]	8
Container 12	[0, 62, 95, 127, 162, 271]	6
Container 13	[0, 123, 152, 188, 222, 271]	6
Container 14	[0, 94, 126, 159, 191, 218, 252, 271]	10
Container 15	[0, 65, 97, 132, 271]	4
Container 16	truck	19
Container 17	truck	18
Container 18	truck	20
Container 19	truck	16
Container 20	[0, 35, 67, 102, 136, 170, 206, 271]	10
Container 21	truck	18
Container 22	truck	15
Container 23	[0, 8, 42, 76, 110, 146, 271]	8
Container 24	truck	19
Container 25	truck	17
Container 26	[0, 31, 64, 100, 135, 168, 204, 239, 270, 271]	14
Container 27	truck	17
Container 28	truck	20
Container 29	truck	15
Container 30	truck	19

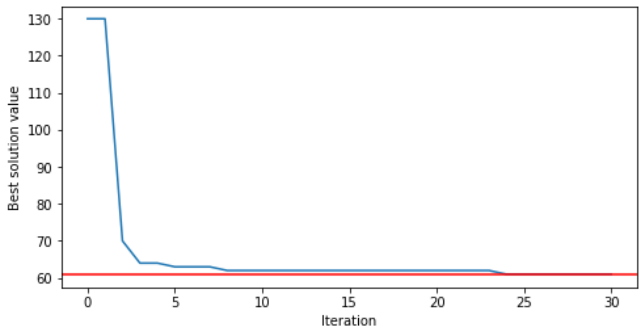
Table 144: Solution Graph 30, 2

Container	Used path	Cost
Container 1	[0, 1, 34, 70, 105, 138, 175, 209, 240, 271]	14
Container 2	[0, 2, 31, 64, 100, 135, 168, 205, 239, 270, 271]	16
Container 3	truck	15
Container 4	truck	16
Container 5	truck	15
Container 6	truck	17
Container 7	truck	19
Container 8	truck	19
Container 9	truck	18
Container 10	truck	20
Container 11	[0, 61, 92, 125, 157, 192, 271]	8
Container 12	truck	19
Container 13	[0, 3, 31, 62, 95, 127, 162, 271]	9
Container 14	truck	17
Container 15	truck	19
Container 16	truck	17
Container 17	truck	15
Container 18	truck	17
Container 19	truck	16
Container 20	truck	19
Container 21	truck	18
Container 22	[0, 32, 65, 97, 132, 166, 200, 236, 271]	12
Container 23	truck	18
Container 24	truck	17
Container 25	[0, 5, 38, 71, 103, 136, 170, 206, 271]	12
Container 26	truck	17
Container 27	truck	20
Container 28	truck	16
Container 29	[0, 9, 44, 77, 109, 140, 176, 271]	10
Container 30	truck	15
Container 31	truck	15
Container 32	truck	18
Container 33	truck	16
Container 34	truck	17
Container 35	truck	18
Container 36	truck	20
Container 37	truck	17
Container 38	truck	18
Container 39	truck	19
Container 40	[0, 40, 75, 108, 145, 179, 210, 271]	10
Container 41	truck	18
Container 42	truck	17
Container 43	truck	20
Container 44	truck	19
Container 45	truck	19
Container 46	truck	17
Container 47	[0, 37, 72, 271]	2
Container 48	truck	20
Container 49	truck	19
Container 50	truck	15

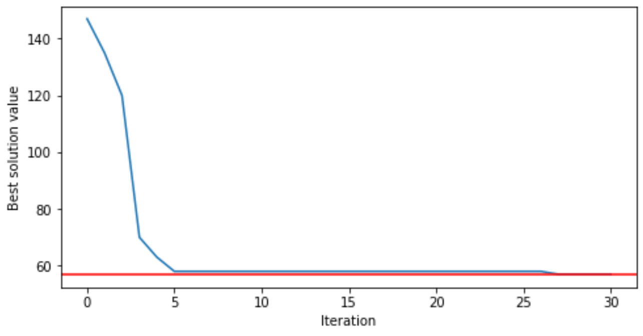
L.2 Course of the algorithm using the quantum annealer



(a) Random 15, 1

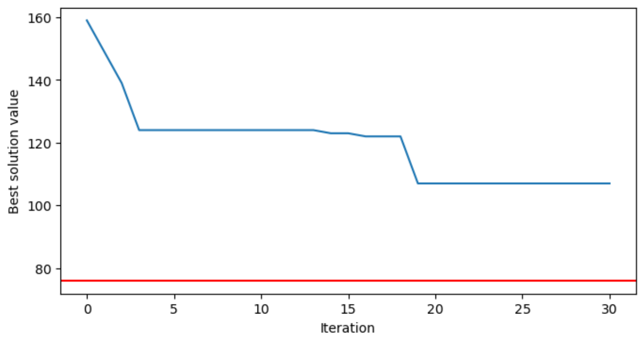


(b) Random 15, 2

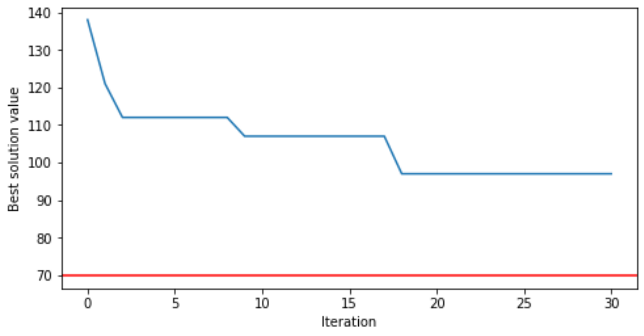


(c) Random 15, 3

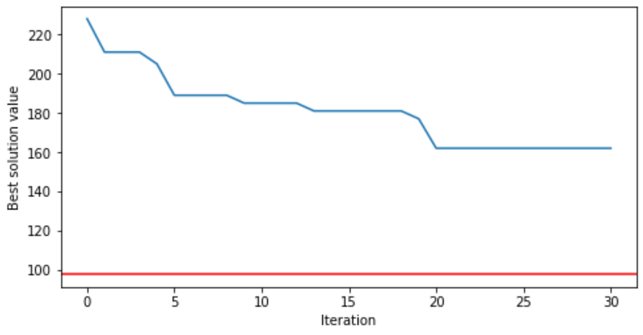
Figure 50: Random 15



(a) Random 20, 1

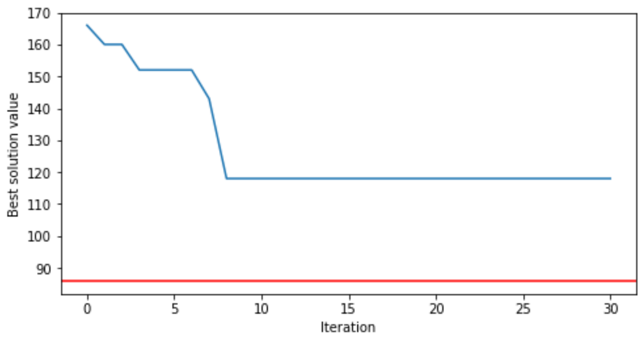


(b) Random 20, 2

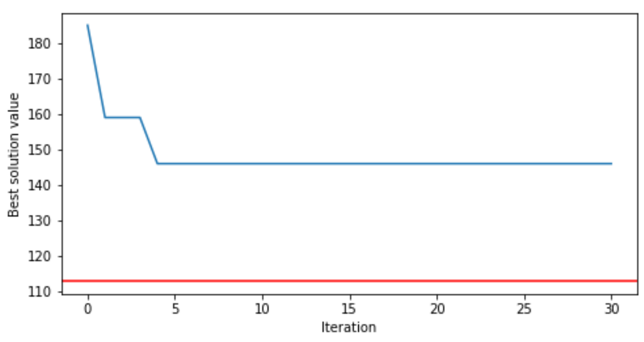


(c) Random 20, 3

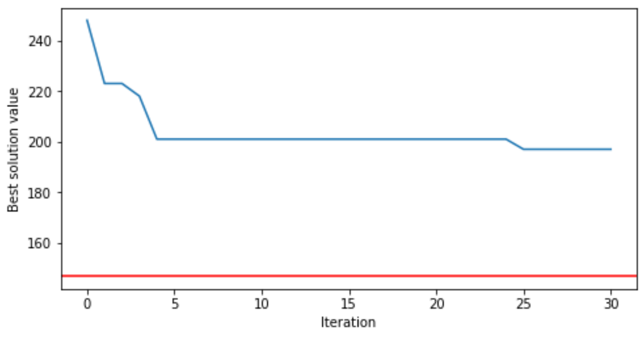
Figure 51: Random 20



(a) Random 25, 1

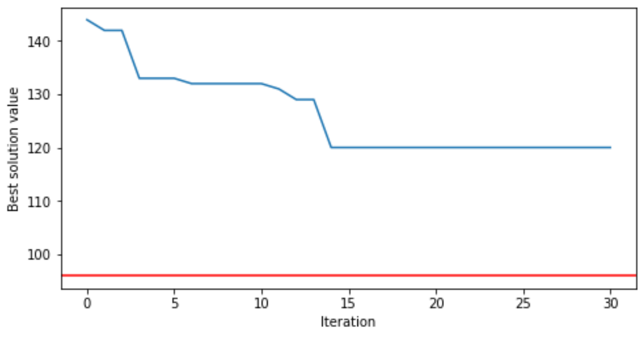


(b) Random 25, 2

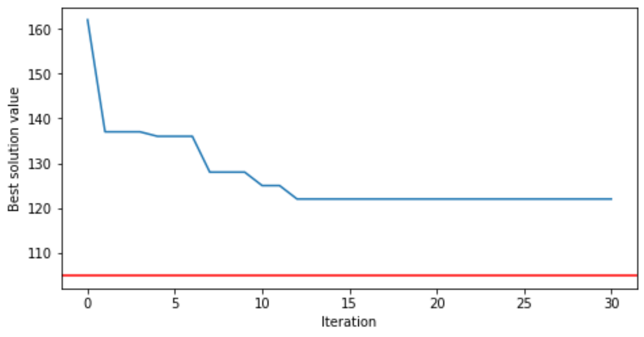


(c) Random 25, 3

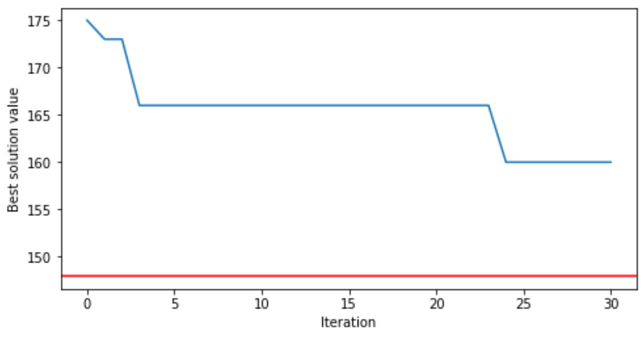
Figure 52: Random 15



(a) Dutch, 1

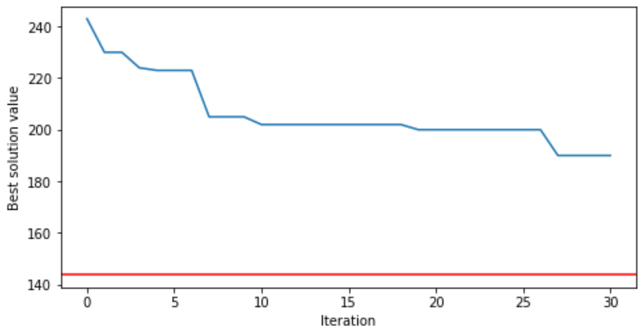


(b) Dutch, 2

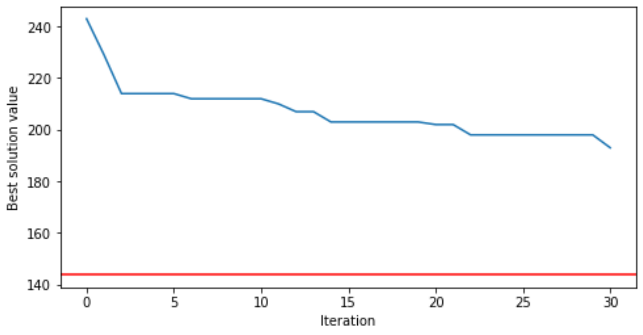


(c) Dutch, 3

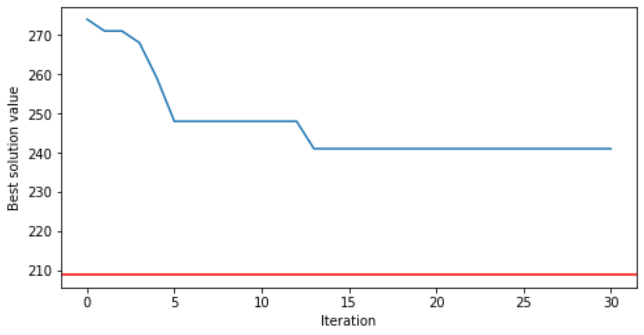
Figure 53: Dutch



(a) Graph 17, 1

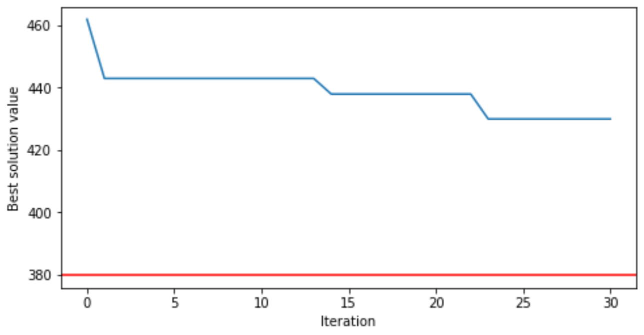


(b) Graph 17, 2

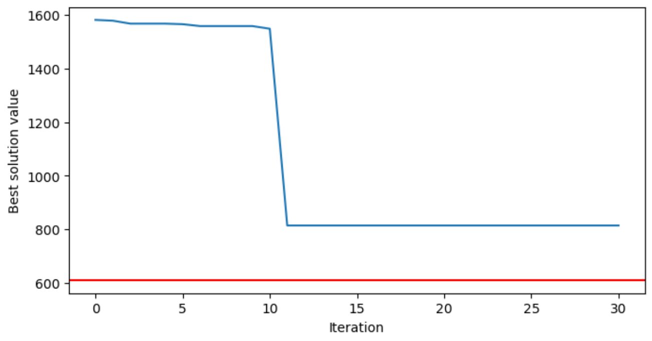


(c) Graph 17, 3

Figure 54: Graph 17



(a) Graph 30, 1



(b) Graph 30, 2

Figure 55: Graph 30

M Results using the optimal solution, long run

M.1 Found solutions

Table 145: Solution random 15, 1

Container	Used path	Cost
Container 1	[0, 8, 26, 36, 61]	3
Container 2	[0, 8, 20, 36, 61]	2
Container 3	[0, 23, 35, 51, 61]	2
Container 4	[0, 23, 41, 51, 61]	3
Container 5	[0, 3, 27, 35, 61]	3
Container 6	[0, 3, 28, 35, 61]	3
Container 7	[0, 18, 38, 50, 61]	2
Container 8	[0, 18, 43, 50, 61]	3
Container 9	[0, 2, 19, 43, 61]	4
Container 10	[0, 2, 19, 43, 61]	4
Container 11	[0, 17, 34, 58, 61]	4
Container 12	[0, 17, 34, 58, 61]	4
Container 13	[0, 40, 59, 61]	3
Container 14	[0, 10, 29, 61]	3
Container 15	[0, 25, 44, 61]	3
Container 16	[0, 25, 44, 61]	3
Container 17	[0, 5, 21, 41, 61]	2
Container 18	[0, 5, 21, 41, 61]	2
Container 19	[0, 20, 41, 61]	3
Container 20	[0, 20, 36, 56, 61]	2

Table 146: Solution random 15, 2

Container	Used path	Cost
Container 1	[0, 16, 43, 50, 61]	4
Container 2	[0, 1, 18, 38, 50, 61]	3
Container 3	[0, 1, 28, 35, 61]	4
Container 4	[0, 16, 43, 50, 61]	4
Container 5	[0, 26, 33, 61]	3
Container 6	[0, 11, 24, 37, 48, 61]	3
Container 7	[0, 41, 48, 61]	3
Container 8	[0, 11, 18, 61]	3
Container 9	[0, 12, 18, 43, 46, 61]	5
Container 10	[0, 12, 18, 31, 61]	3
Container 11	[0, 27, 33, 46, 61]	3
Container 12	[0, 12, 19, 43, 46, 61]	7
Container 13	[0, 45, 48, 61]	2
Container 14	[0, 30, 33, 61]	2
Container 15	[0, 30, 33, 61]	2
Container 16	[0, 45, 48, 61]	2
Container 17	[0, 19, 40, 61]	2
Container 18	[0, 19, 42, 55, 61]	2
Container 19	[0, 19, 40, 61]	2
Container 20	[0, 34, 55, 61]	2

Table 147: Solution random 15, 3

Container	Used path	Cost
Container 1	[0, 7, 18, 42, 61]	2
Container 2	[0, 22, 44, 57, 61]	3
Container 3	[0, 7, 29, 42, 61]	3
Container 4	[0, 22, 33, 57, 61]	2
Container 5	[0, 22, 44, 57, 61]	3
Container 6	[0, 22, 33, 57, 61]	2
Container 7	[0, 7, 24, 34, 57, 61]	3
Container 8	[0, 22, 44, 57, 61]	3
Container 9	[0, 7, 18, 42, 61]	2
Container 10	[0, 7, 29, 42, 61]	3
Container 11	[0, 20, 40, 61]	3
Container 12	[0, 20, 40, 61]	3
Container 13	[0, 20, 40, 61]	3
Container 14	[0, 5, 25, 61]	3
Container 15	[0, 5, 25, 61]	3
Container 16	[0, 5, 27, 40, 61]	4
Container 17	[0, 35, 55, 61]	3
Container 18	[0, 35, 55, 61]	3
Container 19	[0, 5, 25, 61]	3
Container 20	[0, 35, 55, 61]	3

Table 148: Solution random 20, 1

Container	Used path	Cost
Container 1	[0, 2, 26, 50, 77, 81]	4
Container 2	[0, 2, 24, 57, 81]	4
Container 3	truck	15
Container 4	[0, 22, 44, 77, 81]	4
Container 5	[0, 7, 30, 51, 81]	3
Container 6	[0, 7, 38, 53, 71, 81]	3
Container 7	[0, 27, 59, 71, 81]	3
Container 8	[0, 27, 50, 71, 81]	3
Container 9	[0, 11, 39, 54, 81]	5
Container 10	[0, 11, 35, 60, 74, 81]	5
Container 11	[0, 31, 50, 74, 81]	5
Container 12	[0, 31, 50, 74, 81]	5
Container 13	[0, 5, 33, 60, 81]	2
Container 14	[0, 5, 33, 60, 81]	2
Container 15	[0, 25, 53, 80, 81]	2
Container 16	[0, 25, 53, 80, 81]	2
Container 17	[0, 1, 23, 81]	2
Container 18	[0, 21, 54, 63, 81]	3
Container 19	[0, 41, 63, 81]	2
Container 20	[0, 21, 43, 81]	2

Table 149: Solution random 20, 2

Container	Used path	Cost
Container 1	[0, 33, 59, 81]	2
Container 2	[0, 53, 79, 81]	2
Container 3	[0, 13, 21, 59, 81]	3
Container 4	[0, 13, 39, 81]	2
Container 5	[0, 26, 53, 65, 81]	2
Container 6	[0, 6, 33, 45, 81]	2
Container 7	[0, 26, 53, 65, 81]	2
Container 8	[0, 6, 33, 45, 81]	2
Container 9	[0, 30, 46, 62, 81]	3
Container 10	[0, 10, 26, 42, 81]	3
Container 11	[0, 30, 55, 62, 81]	2
Container 12	[0, 10, 35, 42, 81]	2
Container 13	[0, 41, 62, 81]	2
Container 14	[0, 21, 42, 81]	2
Container 15	[0, 41, 62, 81]	2
Container 16	[0, 41, 62, 81]	2
Container 17	[0, 34, 43, 69, 81]	2
Container 18	truck	15
Container 19	truck	16
Container 20	[0, 34, 43, 69, 81]	2

Table 150: Solution random 20, 3

Container	Used path	Cost
Container 1	[0, 14, 40, 46, 65, 81]	5
Container 2	[0, 14, 40, 56, 65, 81]	5
Container 3	[0, 14, 22, 44, 65, 81]	4
Container 4	[0, 14, 30, 45, 81]	3
Container 5	[0, 34, 50, 65, 81]	3
Container 6	[0, 14, 39, 53, 65, 81]	4
Container 7	[0, 14, 30, 56, 65, 81]	4
Container 8	[0, 14, 39, 53, 65, 81]	4
Container 9	[0, 14, 30, 56, 65, 81]	4
Container 10	[0, 14, 22, 46, 65, 81]	5
Container 11	[0, 11, 33, 52, 81]	6
Container 12	[0, 11, 38, 52, 81]	3
Container 13	[0, 11, 38, 53, 72, 81]	5
Container 14	[0, 11, 23, 48, 72, 81]	8
Container 15	[0, 31, 58, 72, 81]	3
Container 16	[0, 31, 53, 72, 81]	6
Container 17	[0, 11, 23, 52, 81]	6
Container 18	truck	15
Container 19	[0, 31, 43, 72, 81]	6
Container 20	[0, 11, 35, 60, 72, 81]	6

Table 151: Solution random 25, 1

Container	Used path	Cost
Container 1	[0, 20, 38, 65, 101]	3
Container 2	[0, 20, 26, 57, 90, 101]	4
Container 3	[0, 45, 69, 90, 101]	4
Container 4	[0, 45, 63, 90, 101]	3
Container 5	[0, 48, 68, 79, 101]	4
Container 6	[0, 23, 43, 54, 101]	4
Container 7	[0, 48, 68, 79, 101]	4
Container 8	[0, 48, 68, 79, 101]	4
Container 9	[0, 19, 39, 53, 101]	3
Container 10	[0, 44, 52, 78, 101]	5
Container 11	[0, 44, 54, 78, 101]	5
Container 12	[0, 44, 64, 78, 101]	3
Container 13	[0, 10, 37, 73, 97, 101]	4
Container 14	[0, 10, 33, 63, 97, 101]	4
Container 15	[0, 35, 59, 97, 101]	6
Container 16	truck	15
Container 17	[0, 21, 38, 73, 101]	2
Container 18	[0, 21, 38, 62, 98, 101]	3
Container 19	[0, 46, 72, 98, 101]	4
Container 20	[0, 46, 63, 98, 101]	2

Table 152: Solution random 25, 2

Container	Used path	Cost
Container 1	[0, 9, 30, 66, 101]	5
Container 2	[0, 9, 30, 66, 101]	5
Container 3	[0, 9, 38, 62, 91, 101]	4
Container 4	[0, 34, 55, 91, 101]	5
Container 5	[0, 42, 73, 101]	3
Container 6	[0, 67, 98, 101]	3
Container 7	[0, 17, 48, 101]	3
Container 8	[0, 17, 30, 75, 98, 101]	4
Container 9	[0, 21, 38, 59, 82, 101]	4
Container 10	[0, 21, 39, 57, 101]	3
Container 11	[0, 46, 64, 82, 101]	3
Container 12	[0, 21, 38, 59, 82, 101]	4
Container 13	truck	16
Container 14	[0, 40, 52, 78, 101]	3
Container 15	truck	15
Container 16	[0, 40, 52, 78, 101]	3
Container 17	[0, 39, 62, 91, 101]	4
Container 18	[0, 39, 55, 91, 101]	3
Container 19	[0, 39, 67, 91, 101]	5
Container 20	truck	18

Table 153: Solution random 25, 3

Container	Used path	Cost
Container 1	truck	16
Container 2	[0, 6, 36, 63, 83, 101]	4
Container 3	truck	16
Container 4	[0, 6, 27, 69, 83, 101]	4
Container 5	[0, 6, 27, 69, 83, 101]	4
Container 6	[0, 6, 50, 69, 83, 101]	5
Container 7	[0, 6, 27, 63, 83, 101]	5
Container 8	truck	17
Container 9	truck	15
Container 10	truck	16
Container 11	[0, 18, 30, 59, 101]	4
Container 12	[0, 18, 48, 72, 84, 101]	3
Container 13	[0, 43, 55, 84, 101]	4
Container 14	[0, 18, 41, 72, 84, 101]	4
Container 15	[0, 18, 48, 63, 84, 101]	5
Container 16	[0, 18, 35, 69, 84, 101]	7
Container 17	[0, 43, 60, 84, 101]	4
Container 18	[0, 18, 41, 70, 84, 101]	6
Container 19	[0, 18, 40, 63, 84, 101]	5
Container 20	[0, 18, 35, 59, 101]	4

Table 154: Solution Dutch, 1

Container	Used path	Cost
Container 1	[0, 47, 56, 67]	1
Container 2	[0, 14, 37, 57, 67]	3
Container 3	truck	15
Container 4	[0, 14, 28, 38, 67]	4
Container 5	[0, 14, 23, 67]	1
Container 6	[0, 3, 12, 67]	1
Container 7	[0, 3, 26, 46, 67]	3
Container 8	truck	10
Container 9	[0, 3, 26, 51, 67]	3
Container 10	[0, 14, 37, 62, 67]	3
Container 11	[0, 8, 25, 34, 67]	4
Container 12	[0, 8, 25, 67]	3
Container 13	truck	12
Container 14	[0, 8, 17, 27, 67]	4
Container 15	[0, 30, 39, 51, 67]	5
Container 16	truck	10
Container 17	[0, 41, 53, 67]	2
Container 18	[0, 41, 53, 65, 67]	4
Container 19	[0, 52, 61, 67]	3
Container 20	[0, 8, 20, 32, 44, 67]	5

Table 155: Solution Dutch, 2

Container	Used path	Cost
Container 1	[0, 3, 12, 67]	1
Container 2	[0, 36, 45, 67]	1
Container 3	[0, 25, 34, 67]	1
Container 4	truck	10
Container 5	[0, 3, 26, 46, 67]	3
Container 6	[0, 14, 37, 57, 67]	3
Container 7	truck	10
Container 8	truck	12
Container 9	[0, 14, 37, 62, 67]	3
Container 10	[0, 3, 26, 51, 67]	3
Container 11	[0, 8, 17, 27, 67]	4
Container 12	[0, 8, 20, 32, 44, 67]	5
Container 13	truck	13
Container 14	[0, 19, 28, 38, 67]	4
Container 15	[0, 41, 53, 65, 67]	4
Container 16	[0, 41, 50, 67]	3
Container 17	[0, 19, 28, 40, 67]	5
Container 18	[0, 8, 17, 29, 67]	5
Container 19	[0, 19, 31, 43, 55, 67]	5
Container 20	truck	10

Table 156: Solution Dutch, 3

Container	Used path	Cost
Container 1	truck	10
Container 2	[0, 14, 37, 57, 67]	3
Container 3	[0, 3, 26, 46, 67]	3
Container 4	truck	13
Container 5	truck	10
Container 6	[0, 3, 26, 51, 67]	3
Container 7	[0, 14, 37, 62, 67]	3
Container 8	[0, 14, 28, 40, 67]	5
Container 9	[0, 36, 50, 62, 67]	5
Container 10	[0, 3, 17, 29, 67]	5
Container 11	truck	10
Container 12	truck	14
Container 13	[0, 19, 31, 43, 55, 67]	5
Container 14	truck	12
Container 15	[0, 19, 28, 38, 67]	4
Container 16	truck	10
Container 17	[0, 8, 17, 27, 67]	4
Container 18	[0, 8, 20, 32, 44, 67]	5
Container 19	truck	12
Container 20	truck	12

Table 157: Solution graph 17, 1

Container	Used path	Cost
Container 1	[0, 1, 21, 44, 66, 120]	6
Container 2	[0, 2, 18, 38, 61, 83, 120]	8
Container 3	[0, 54, 77, 99, 117, 120]	6
Container 4	[0, 3, 19, 42, 63, 120]	6
Container 5	[0, 21, 35, 53, 73, 92, 114, 120]	10
Container 6	[0, 23, 43, 62, 81, 101, 114, 120]	10
Container 7	[0, 5, 24, 46, 67, 85, 120]	8
Container 8	[0, 7, 29, 50, 68, 120]	6
Container 9	[0, 2, 25, 45, 65, 85, 120]	8
Container 10	[0, 3, 26, 48, 68, 120]	6
Container 11	[0, 35, 55, 78, 100, 120]	6
Container 12	[0, 36, 52, 72, 95, 117, 120]	8
Container 13	[0, 20, 43, 65, 83, 120]	6
Container 14	[0, 20, 36, 56, 75, 97, 120]	8
Container 15	[0, 38, 52, 70, 93, 114, 120]	8
Container 16	[0, 23, 37, 53, 76, 97, 120]	8
Container 17	[0, 39, 58, 80, 101, 119, 120]	8
Container 18	[0, 41, 63, 84, 102, 120]	6
Container 19	[0, 36, 59, 79, 99, 119, 120]	8
Container 20	[0, 37, 60, 82, 102, 120]	6

Table 158: Solution graph 17, 2

Container	Used path	Cost
Container 1	[0, 1, 21, 44, 66, 120]	6
Container 2	[0, 19, 35, 55, 78, 100, 120]	8
Container 3	[0, 20, 43, 65, 83, 120]	6
Container 4	[0, 54, 70, 93, 114, 120]	6
Container 5	[0, 21, 35, 53, 73, 92, 114, 120]	10
Container 6	[0, 23, 37, 53, 76, 97, 120]	8
Container 7	[0, 39, 58, 80, 101, 119, 120]	8
Container 8	[0, 7, 29, 50, 68, 120]	6
Container 9	[0, 2, 25, 45, 65, 85, 120]	8
Container 10	[0, 37, 60, 82, 102, 120]	6
Container 11	[0, 52, 72, 95, 117, 120]	6
Container 12	[0, 2, 18, 38, 61, 83, 120]	8
Container 13	[0, 54, 77, 99, 117, 120]	6
Container 14	[0, 3, 19, 42, 63, 120]	6
Container 15	[0, 4, 18, 36, 56, 75, 97, 120]	10
Container 16	[0, 6, 26, 45, 59, 80, 120]	8
Container 17	[0, 5, 24, 46, 67, 85, 120]	8
Container 18	[0, 41, 63, 84, 102, 120]	6
Container 19	[0, 36, 59, 79, 99, 119, 120]	8
Container 20	[0, 3, 26, 48, 68, 120]	6

Table 159: Solution graph 17, 3

Container	Used path	Cost
Container 1	truck	17
Container 2	[0, 1, 21, 44, 66, 120]	6
Container 3	[0, 52, 72, 95, 117, 120]	6
Container 4	[0, 18, 38, 61, 83, 120]	6
Container 5	[0, 35, 55, 78, 100, 120]	6
Container 6	[0, 18, 37, 60, 82, 102, 120]	7
Container 7	[0, 35, 54, 77, 99, 119, 120]	7
Container 8	truck	17
Container 9	truck	16
Container 10	[0, 1, 20, 43, 65, 85, 120]	7
Container 11	[0, 1, 20, 36, 59, 80, 101, 120]	9
Container 12	truck	17
Container 13	truck	18
Container 14	[0, 35, 53, 76, 97, 118, 120]	8
Container 15	truck	18
Container 16	[0, 18, 36, 56, 75, 97, 120]	8
Container 17	[0, 1, 19, 42, 63, 120]	6
Container 18	truck	15
Container 19	[0, 18, 37, 53, 73, 92, 114, 120]	9
Container 20	[0, 52, 70, 93, 114, 120]	6

Table 160: Solution graph 30, 1

Container	Used path	Cost
Container 1	truck	16
Container 2	truck	18
Container 3	[0, 3, 39, 74, 107, 142, 174, 209, 240, 271]	14
Container 4	[0, 4, 40, 75, 108, 145, 179, 210, 271]	12
Container 5	truck	15
Container 6	truck	18
Container 7	truck	16
Container 8	truck	16
Container 9	truck	16
Container 10	[0, 10, 45, 78, 115, 149, 180, 271]	10
Container 11	[0, 61, 92, 128, 162, 271]	6
Container 12	[0, 152, 188, 222, 271]	4
Container 13	[0, 153, 182, 218, 252, 271]	6
Container 14	[0, 64, 91, 122, 158, 192, 271]	8
Container 15	[0, 65, 97, 132, 271]	4
Container 16	[0, 31, 64, 100, 135, 168, 204, 233, 266, 271]	13
Container 17	[0, 62, 98, 132, 166, 200, 236, 271]	10
Container 18	[0, 63, 99, 134, 167, 199, 230, 266, 271]	12
Container 19	truck	16
Container 20	[0, 35, 67, 102, 136, 170, 206, 271]	10
Container 21	truck	18
Container 22	truck	15
Container 23	[0, 8, 42, 76, 110, 146, 271]	8
Container 24	[0, 9, 44, 77, 109, 140, 176, 271]	10
Container 25	[0, 70, 105, 138, 174, 203, 236, 271]	9
Container 26	truck	20
Container 27	truck	17
Container 28	[0, 33, 69, 104, 137, 172, 204, 239, 270, 271]	14
Container 29	truck	15
Container 30	truck	19

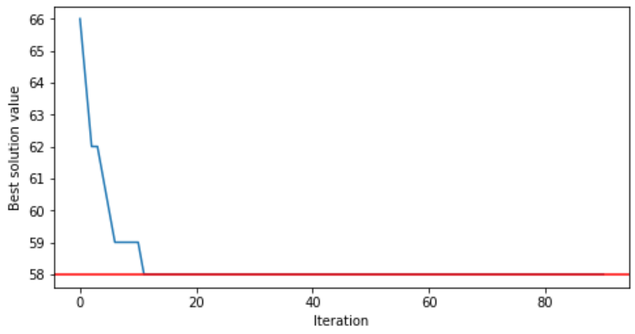
Table 161: Solution graph 30, 2

Container	Used path	Cost
Container 1	truck	20
Container 2	truck	19
Container 3	truck	15
Container 4	truck	16
Container 5	truck	15
Container 6	truck	17
Container 7	truck	19
Container 8	truck	19
Container 9	[0, 39, 74, 107, 142, 174, 209, 240, 271]	12
Container 10	[0, 10, 45, 78, 115, 149, 180, 271]	10
Container 11	[0, 1, 32, 65, 97, 132, 271]	8
Container 12	[0, 92, 128, 162, 271]	4
Container 13	[0, 123, 152, 188, 222, 271]	6
Container 14	[0, 34, 61, 92, 125, 157, 192, 271]	10
Container 15	[0, 185, 217, 252, 271]	4
Container 16	truck	17
Container 17	truck	15
Container 18	truck	17
Container 19	truck	16
Container 20	[0, 35, 67, 102, 136, 170, 206, 271]	10
Container 21	[0, 1, 33, 69, 104, 137, 169, 200, 236, 271]	13
Container 22	truck	17
Container 23	[0, 63, 99, 134, 167, 196, 230, 266, 271]	12
Container 24	[0, 34, 70, 105, 138, 174, 203, 236, 271]	11
Container 25	truck	16
Container 26	truck	17
Container 27	[0, 7, 42, 76, 110, 146, 271]	8
Container 28	truck	16
Container 29	[0, 9, 44, 77, 109, 140, 176, 271]	10
Container 30	truck	15
Container 31	truck	15
Container 32	truck	18
Container 33	truck	16
Container 34	truck	17
Container 35	truck	18
Container 36	[0, 66, 100, 135, 168, 205, 239, 270, 271]	12
Container 37	truck	17
Container 38	truck	18
Container 39	truck	19
Container 40	[0, 40, 75, 108, 145, 179, 210, 271]	10
Container 41	[0, 31, 62, 95, 127, 162, 271]	8
Container 42	[0, 32, 68, 102, 271]	4
Container 43	[0, 93, 122, 158, 192, 271]	6
Container 44	[0, 124, 151, 182, 218, 252, 271]	8
Container 45	[0, 155, 187, 222, 271]	4
Container 46	[0, 36, 69, 101, 133, 166, 192, 271]	10
Container 47	[0, 37, 72, 271]	2
Container 48	[0, 38, 72, 271]	2
Container 49	[0, 39, 71, 98, 132, 271]	6
Container 50	[0, 70, 96, 129, 161, 193, 226, 252, 271]	12

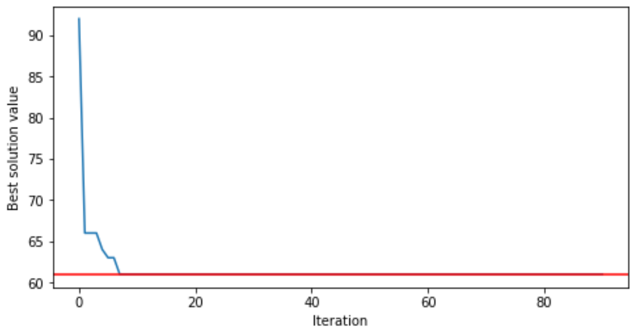
Table 162: Solution graph 30, 3

Container	Used path	Cost
Container 1	truck	16
Container 2	truck	19
Container 3	truck	18
Container 4	[0, 1, 33, 69, 104, 137, 172, 204, 239, 270, 271]	15
Container 5	truck	15
Container 6	truck	19
Container 7	truck	18
Container 8	truck	18
Container 9	truck	17
Container 10	truck	15
Container 11	truck	16
Container 12	[0, 1, 32, 68, 102, 271]	6
Container 13	truck	15
Container 14	[0, 31, 63, 92, 128, 162, 271]	7
Container 15	[0, 1, 33, 62, 98, 132, 271]	7
Container 16	[0, 31, 63, 99, 134, 167, 199, 230, 266, 271]	13
Container 17	truck	16
Container 18	truck	16
Container 19	truck	16
Container 20	[0, 31, 64, 100, 135, 168, 204, 233, 266, 271]	13
Container 21	truck	16
Container 22	[0, 1, 34, 70, 105, 138, 174, 203, 236, 271]	13
Container 23	truck	18
Container 24	truck	20
Container 25	truck	15
Container 26	truck	18
Container 27	[0, 2, 38, 72, 106, 140, 176, 271]	10
Container 28	[0, 2, 35, 67, 102, 136, 170, 206, 271]	12
Container 29	truck	16
Container 30	[0, 32, 65, 97, 132, 166, 200, 236, 271]	12
Container 31	truck	17
Container 32	truck	15
Container 33	truck	16
Container 34	truck	20
Container 35	truck	16
Container 36	truck	19
Container 37	truck	15
Container 38	truck	20
Container 39	truck	19
Container 40	truck	17
Container 41	truck	18
Container 42	[0, 94, 121, 152, 185, 217, 252, 271]	10
Container 43	[0, 64, 96, 129, 161, 193, 226, 252, 271]	12
Container 44	[0, 91, 122, 155, 187, 222, 271]	8
Container 45	[0, 91, 123, 152, 188, 222, 271]	7
Container 46	truck	15
Container 47	[0, 31, 62, 95, 127, 162, 271]	8
Container 48	[0, 151, 182, 218, 252, 271]	6
Container 49	[0, 61, 93, 122, 158, 192, 271]	7
Container 50	[0, 61, 92, 125, 157, 192, 271]	8

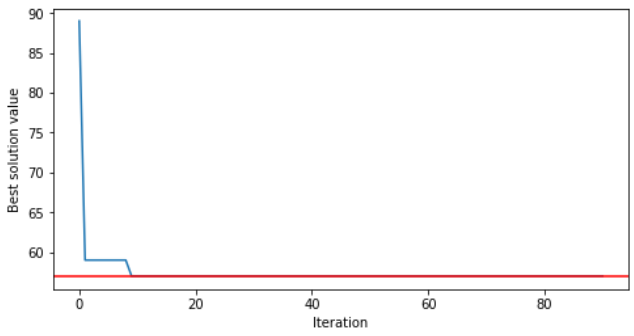
M.2 Course of the algorithm using the optimal solution, long run



(a) Random 15, 1

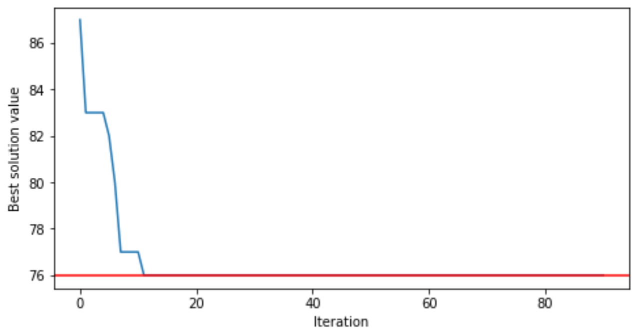


(b) Random 15, 2

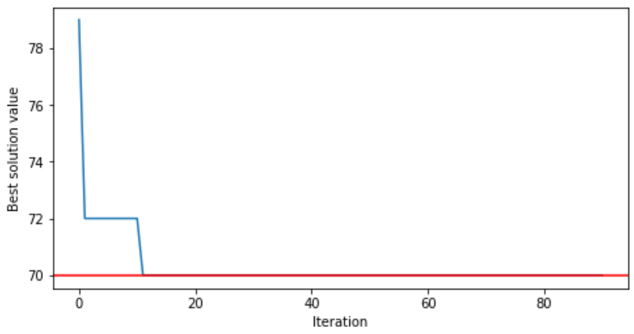


(c) Random 15, 3

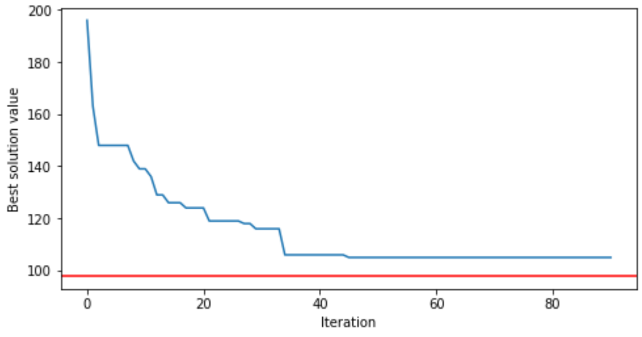
Figure 56: Random 15



(a) Random 20, 1

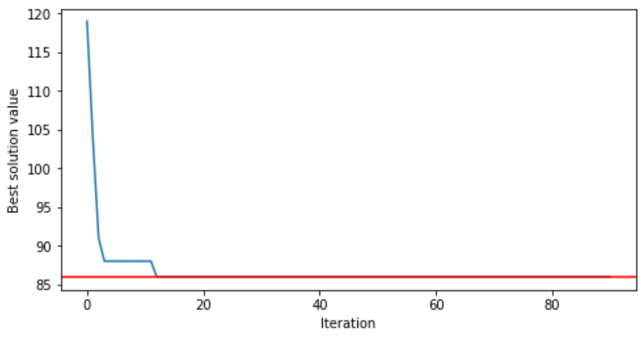


(b) Random 20, 2

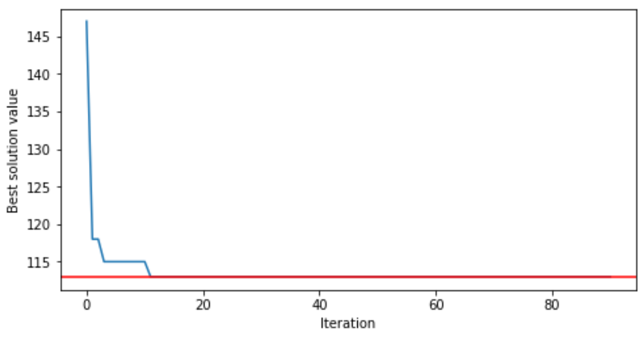


(c) Random 20, 3

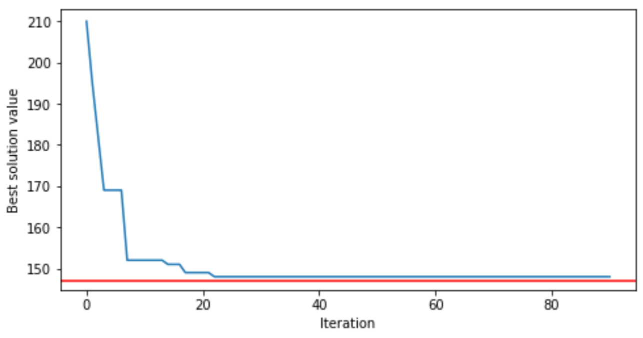
Figure 57: Random 20



(a) Random 25, 1

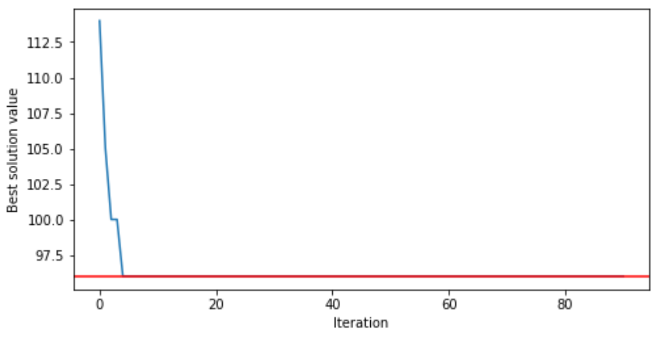


(b) Random 25, 2

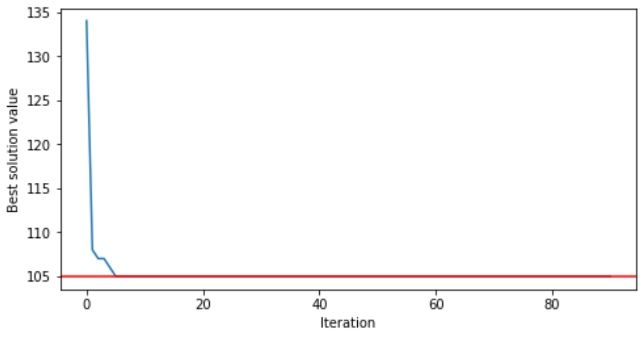


(c) Random 25, 3

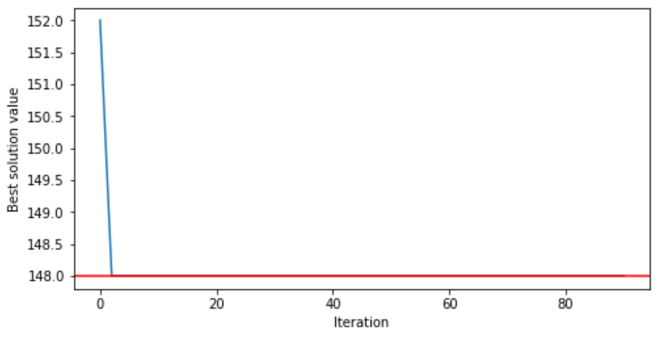
Figure 58: Random 15



(a) Dutch, 1

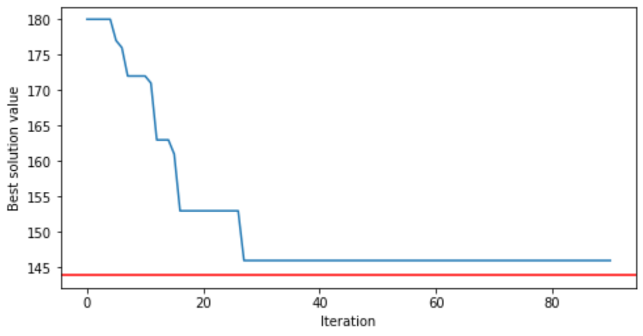


(b) Dutch, 2

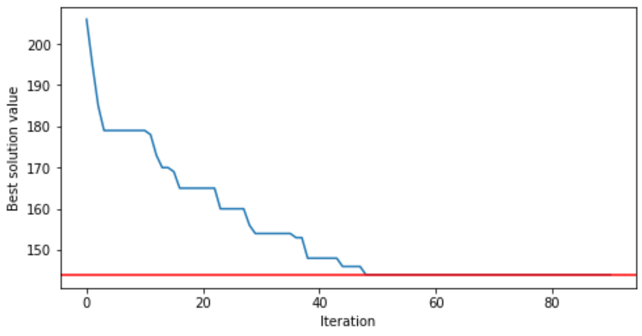


(c) Dutch, 3

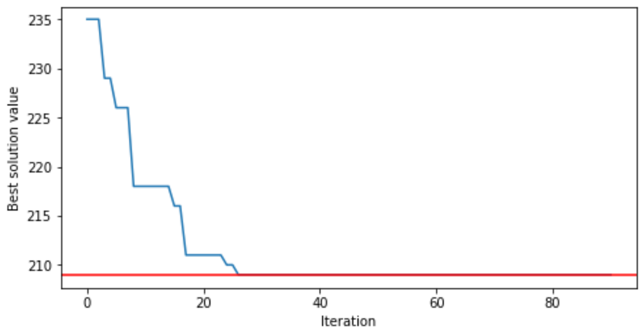
Figure 59: Dutch



(a) Graph 17, 1

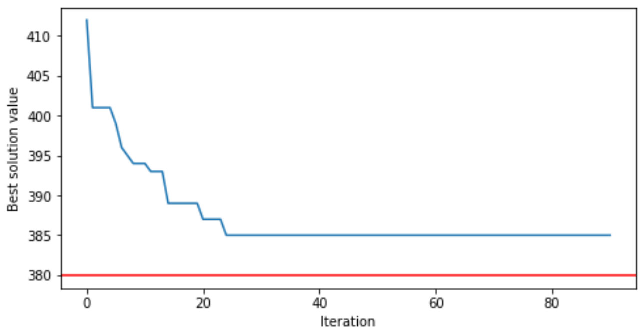


(b) Graph 17, 2

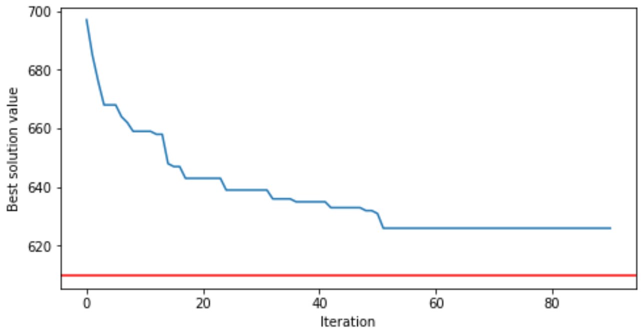


(c) Graph 17, 3

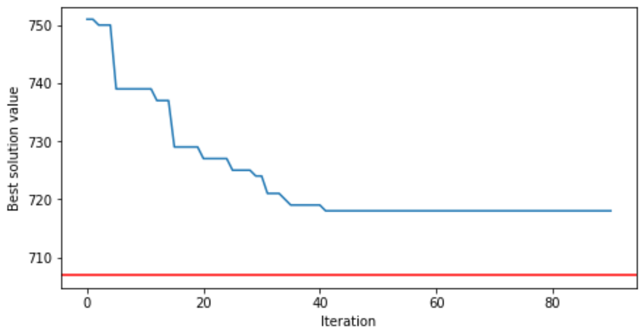
Figure 60: Graph 17



(a) Graph 30, 1



(b) Graph 30, 2



(c) Graph 30, 3

Figure 61: Graph 30

N Code description

Different scripts have been used to implement the genetic algorithm. The scripts are shortly explained in this section. First, the readFile script is explained. Secondly, the annealing script is described. Thirdly, the cross-over script is explained. Next, the initialize code is discussed. Then, the Mutation script is reviewed, followed by the Parameter_tuning script. Thereafter, the Population class is described. Next, the selection script is discussed. Finally, the solveOptimal script is explained.

The readFile script shows how the instances have been created. The annealing script contains the entire annealing part. So, the QUBO is constructed and solved. Then, the returned solution is post-processed. In the cross-over script, the implementation of all the described cross-over operators is given. The initialize code contains the implementation of all four different initialization methods. The mutation script consists of the three implemented mutation operators. The Parameter_tuning script is the main and contains the overview of the genetic algorithm. The other scripts are called upon from this script. The Population class contains all the values corresponding to a population. For example, the routes and the returned solution by the annealer. The selection script has an implementation for all the introduced selection methods. The solveOptimal script contains the code that is used to determine the optimal solution to the entire problem and the code that is used to determine the optimal solution to the problem that is sent to the annealer.