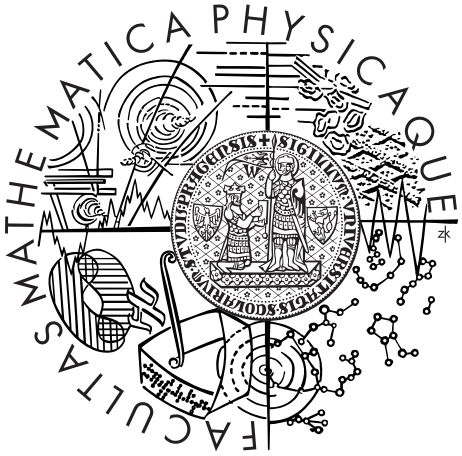
Charles University in Prague
Faculty of Mathematics and Physics

BACHELOR THESIS



Imrich Kuklis

Vybrané problémy související s vehicle routing

Department of Software and Computer Science Education

Supervisor of the bachelor thesis: RNDr. Martin Pergel Ph.D.
Study programme: Computer Science
Specialization: General Computer Science

Prague 2015

I would like to thank my supervisor Martin Pergel for his time, valuable suggestions and advices.

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In ..... date ..... signature of the author

Název práce: Vybrané problémy související s vehicle routing

Autor: Imrich Kuklis

Katedra: Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce: RNDr. Martin Pergel Ph.D., Katedra softwaru a výuky informatiky

Abstrakt: V této práci se zaměříme na známí problém, kterým se zabývá mnoho vedců z oblasti logistiky, teoretické informatiky a matematiky. Tento problém se jmenuje dopravní problém. Hlavně se soustředíme na speciální variantu s časovým oknem. Implementujeme několik rozvrhovacích algoritmů a porovnáváme ich výsledky.

Klíčová slova: dopravní problém VRP, časová složitost, rozvrhovací algoritmus, časové okno, optimalizace

Title: Particular Problems Related to the Vehicle Routing Problem

Author: Imrich Kuklis

Department: Department of Software and Computer Science Education

Supervisor: RNDr. Martin Pergel Ph.D., Department of Software and Computer Science Education

Abstract: In our thesis we concentrate on a well-known problem which is popular among the scientists from the field of logistics, theoretical computer science and applied mathematics. This problem is called the Vehicle Routing Problem. We concentrate mainly on the Vehicle Routing Problem with Time Window. We implement some scheduling algorithms and compare their results.

Keywords: vehicle routing problem (VRP), running time, scheduling algorithm, time window, optimization

Contents

Introduction

Every company owner's dream is to lead a company that is successful on the market. When a company wants to achieve such success, every day is a new challenge and only a well organized company can take on these everyday challenges. Such a company needs employees which are up to date in their profession and a strong customer base which is loyal to the company. When a company wants to gain the loyalty of its customers, it must always make sure that the products which are sold are the best choice for the customer. If the company wants to expand its customer base it needs to ensure services (online shopping, customer service, transportation) in addition to sales and purchases. Expenses associated with services must also be calculated into the sale prices. If the company wants to remain competitive in the market it must keep the expenses associated with services low.

In case of a transportation service we need to create a schedule, which takes advantage of the properties of trucks which are available for delivery, deliver the orders on time and keep the transportation at low cost. At any given time the schedule strictly specifies which truck is located where on the map, what is its destination, what is it transporting, how much does the transportation cost, what time does it arrive at its destination, and how much will be the income for the delivery. The company should choose the best possible schedule to minimize transportation costs.

Creating a transportation schedule is not an easy task. This problem belongs to the research field of logistics. Martin Christopher defined *logistics* in [1] as: *"Logistics is the process of strategically managing the procurement, movement and storage of materials, parts and finished inventory (and the related information flows) through the organization and its marketing channels in such a way that current and future profitability are maximized through the cost-effective fulfillment of orders."* Nowadays logistics is one of the most popular research fields and has a wide range of use in practice.

In our thesis we concentrate on a well-known problem which is popular among the scientists from the field of logistics, theoretical computer science and applied mathematics. The goal of our thesis is to solve this problem of logistics as efficiently as possible. We could describe our problem in the following way.

We have a large network of roads, which connects entities (cities, regions and states). Between these entities are tasks which we want to accomplish. This task we can image as an order (package of limited size and weight), which we must deliver from city A to city B. How do we accomplish such a task? With the help of the trucks, which are available for delivery. An important question is: Can we solve this problem by the help of some algorithm? This problem is well-known and called the Vehicle Routing Problem (VRP). The aim of this thesis is to create schedules with different algorithms for a given instance of the problem and compare their results.

In the first chapter we describe the basic definitions.

In the second chapter we define the problem VRP and list some popular versions of the problem VRP. After that we describe the relation between problem VRP and linear programming (LP) and problem VRP and constraint satisfaction

problem (CSP).

In the third chapter we explain why we created two applications in our thesis. We also explain the selection of algorithms that we implemented.

In the fourth chapter we describe the model of our problem and the scheduling algorithms which we implemented. The description contains pseudo code, explanation of pseudo code and running time complexity analysis.

In the fifth chapter we describe the user guide and the programmer guide of the implemented applications. The thesis consists of two applications. The first one is **Editor application**, which enables the user to create test cases. The second one is **Simulator application**, which contains the implemented algorithms and enables the user to run these algorithms on the created test cases.

In the sixth chapter we focus on testing the implemented scheduling algorithms on different instances of the VRP problem. We examine the output of the algorithms on different tests and special cases too. After testing we describe the observed advantages and disadvantages of the scheduling algorithms.

In the last chapter we summarize the result of testing the scheduling algorithms. We propose some possible extension of the thesis and extensions of the applications.

1. Basic Terms

In this chapter we introduce the notion of the problem and algorithm. After that we define running time complexity and describe the two most known classes of running time complexity: P and NP. At the end of this chapter we mention some well-known problems of the class NP. This chapter is inspired by book [2] and [3].

1.1 Problem, Instance of Problem, Algorithm

Let us begin with the definition of the notion *problem*. Its necessary to introduce it first, because without this notion we are unable to introduce the notion of *algorithm* and *running time complexity*. We take these notions from the book [2].

Problem, Instance of Problem

We use the word problem frequently in our every day life, without thinking about what the word problem means. There are lots of definitions for the notion *problem*. For our needs the best possible description is from the book [2]: *"For our purposes, a problem will be a general question to be answered, usually possessing several parameters, or free variables, whose values are left unspecified. A problem is described by giving: (1) a general description of all its parameters, and (2) a statement of what properties, the answer, or solution, is required to satisfy."*. The solution of a problem is always individual. Sometimes it's a simple short answer, in other cases it can be very complex. We would like to also mention a special class of problems. These problems are called *decision problems*. These problems differ from other problems in the way that their solution is one of the two possibilities **yes** or **no**.

Another important notion is the *instance* of problem. Garey and Johnson in [2] defined *instance of problem* as: *"An instance of problem is obtained by specifying particular values for all the problem parameters."*

Algorithm

Now that we have defined what *problem* and *instance* of problem is, we can introduce the definition of *algorithm*. According to book [2]: *"Algorithms are general, step-by-step procedures for solving problems. For concreteness, we can think of them simply as being computer programs, written in some precise computer language. An algorithm is said to solve problem* Π *if that algorithm can be applied to any instance I of* Π *and is guaranteed always to produce a solution for instance I."*

Algorithms have two important properties which we would like to mention. The first one is finiteness and the second one is effectiveness.An algorithm is finite if it solves the instance of problem in finite number of steps. Effectiveness of the algorithm is divided into two classes. These two classes are running time complexity and memory consumption. In most cases we focus on the running time complexity of the algorithm.

1.2 Running Time Complexity, Class P and NP

Nowadays when we are talking about algorithms, we are interested in their running time complexity. This does not necessarily mean that memory consumption is insignificant. There are lot of algorithms for which we do not have enough memory. One such algorithm is the Min-Max algorithm, which is used as artificial intelligence (AI) in board games. We are unable to store the data structure of the winning strategy of chess in any computers memory.

Running Time Complexity, O-notation

For better understanding of what running time complexity is we have some basic notations(Θ, O,Ω, o, ω). Thanks to these basic notations we can describe the asymptotic running time complexity as a function whose domain is equal to the set of natural numbers (N). For us the most important notation is the O-notation. We introduce the definition of *O-notation* from book [3]:*"For a given function g(n), we denote by O(g(n)) the set of functions O(g(n)) = { f(n): there exists positive constant c and n_0 such that $0 \leq f(n) \leq c * g(n)$ for all $n \geq n_0$}"*. This notation bounds the running time complexity of a given algorithm from above.

Class P

In computer science we divide problems by running time complexity into several basic classes. We would like to highlight two of these basic classes P and NP. In book [3] *class P* is defined by the following way: *"The class P consists of those problems that are solvable in polynomial time. More specifically, they are problems that can be solved in time $O(n^k)$ for some constant k, where n is the size of the input to the problem."* Another possible definition for *class P* is also from book [3]: *"We define the complexity class P as the set of decision problems that are polynomial-time solvable. A problem is polynomial-time solvable, if there exists an algorithm to solve it in time $O(n^k)$ for some constant k."*

Class P is considered a class of problems which we can solve effectively on large input data. Popular problems of class P are *sorting of numbers, matrix multiplication and searching for shortest paths in weighted graphs.* There exists problems in class P, which are solvable in polynomial time, but are unusable in practice. For example algorithms with running time complexity $O(n^{1000})$.

Before we introduce class NP it is necessary to introduce the notion of *language* and *alphabet*. In book [3] *alphabet* and *language* is defined as: *"An alphabet $\sum$ is a finite set of symbols. A language L over $\sum$ is any set of strings made up of symbols from $\sum$."* We also need to describe the correspondence between languages and decision problems. In book [2] the correspondence between language and decision problem is described in the following way: *"The correspondence between decision problems and languages is brought about by the encoding schemes we use for specifying problem instances whenever we intend to compute with them. Recall that an encoding scheme e for a problem Π provides a way of describing each instance of Π by an appropriate string of symbols over some fixed alphabet $\sum$. Thus the problem Π and the encoding scheme e for Π partition $\sum^*$ into three classes of strings: those that are not encoding of instances of Π, those that encode instances of Π for which the answer is "no," and those that encode instances of*

Π *for which the answer is "yes." This third class of strings is the language that we associate with* Π *and* e*, setting* $L[\Pi, e] = \{x \in \sum^* : \sum$ *is the alphabet used by* e*, and* x *is the encoding under* e *of an instance* $I \in Y_\Pi\}$*. Our formal theory is applied to decision problems by saying that, if a result holds for language* $L[\Pi, e]$*, then it holds for problem* Π *under the encoding scheme* e*."*

Class NP, NP-hard and NP-complete

Class NP is an extension of class P. We introduce the formal definition of *class NP* from book [3] *"The complexity class NP is the class of languages that can be verified by a polynomial-time algorithm. More precisely, a language L belongs to NP if and only if there exist a two-input polynomial-time algorithm A and constant c such that* $L = \{x \in \{0,1\}^* :$ *there exists a certificate* y *with* $|y| = O(|x|^c)$ *such that* $A(x, y) = 1\}$*."* There are other three notions which are important to understand class NP. The first notion is *polynomial-time reducible*, which is defined in [3] as: *"We say that language* L_1 *is polynomial-time reducible to a language* L_2*, written in* $L_1 \leq_P L_2$*, if there exists a polynomial-time computable function* $f : \{0,1\}^* \to \{0,1\}^*$ *such that for all* $x \in \{0,1\}^*, x \in L_1$ *if and only if* $f(x) \in L_2$*. We call* f *the reduction function."* The other two notions are *NP-complete* and *NP-hard*, which are defined in [3] as *"A language* $L \in \{0,1\}^*$ *is NP-complete if*

1. $L \in NP$*, and*
2. $L' \leq_P L$ *for every* $L' \in NP$*.*

If a language L satisfies property 2, but not necessarily property 1, we say that L is NP-hard."

1.3 Examples of NP-complete Problems

Class NP contains hundreds of problems. We mention some popular problems from it. Each of these problems are NP-complete.

Traveling-Salesman Problem (TSP)

We can describe the problem in the following way. We have a complete graph with n vertices. Every edge of the graph has a non-negative cost. In the graph there is a salesman present which would like to visit every vertex of the graph. The tour of the salesman must fulfill two conditions:

1. We can visit every vertex only once.
2. The salesman must end his tour in the same vertex where he started.

We search for the tour whose total cost is minimum. The TSP problem has also a decision problem version. We introduce the formal definition of decision problem of TSP from book [3]: *"TSP =*$\{ \langle G, c, k\rangle$*,* $G = (V, E)$ *is a complete graph,* c *is a function from* $V \times V \to Z$*,* $k \in Z$*, and* G *has a traveling-salesman tour with cost at most* $k\}$*."*

Clique Problem

We have a graph G(V,E) and a subset $V' \subseteq V$ of vertices. If every pair of vertices is connected in V', than V' is a clique. In other words, a clique is a complete sub-graph of graph G. The Clique problem is an optimization problem, where we search for the largest clique in graph G. We introduce the formal definition of the decision problem of Clique from book [3]: *"$CLIQUE = \{\langle G, k\rangle$, G is a graph containing a clique of size k$\}$."*

Vertex-Cover Problem

We have a graph G(V,E) and a subset $V' \subseteq V$ of vertices. In this graph G the following condition applies: If $(u, v) \in E$ then $u \in V'$ or $v \in V'$ or $u, v \in V'$. This means that each vertex covers its incident edges. A vertex cover of graph G is a subset of vertices which covers every edge of graph G. The size of graph's vertex cover is equal to the number of vertices which cover every edge of graph. The goal of this problem is to find the minimum size vertex cover of graph G. We introduce the formal definition of the decision problem of Vertex-Cover problem from book [3]: *"$VERTEX\text{-}COVER = \{\langle G, k\rangle$, graph G has a vertex cover of size k$\}$."*

Subset Sum Problem

The Subset sum problem is an arithmetic problem. In this problem we have a subset $S \subset N$ and a target value $t \in N$. We ask whether there is a subset $S' \subseteq S$ whose elements sum to t. We introduce the formal definition of the decision problem of SUBSET-Sum problem from book [3]: *"$SUBSET\text{-}SUM = \{\langle S, t\rangle$: there exists a subset $S' \subseteq S$ such that $t = \sum_{s \in S'} s$ $\}$."*

One of the most popular problems of theoretical computer science is whether class P is equal to class NP. Nowadays we only know that $P \subseteq NP$. This problem is one of the the Millennium Prize Problems.

2. VRP

In this chapter we describe the problem of VRP. We list some popular types of VRP. After that we describe the relation between VRP and CSP and VRP and LP. In LP we focus mainly on the mixed integer programming. This chapter is inspired by [4] - [9].

2.1 Desciption of VRP

The VRP problem focuses on transportation of orders to customers and companies. Transportation includes pickup and delivery activities which are performed by a fleet of trucks. The main task of VRP is to find a schedule, which utilizes the best possible way the fleet of available trucks by delivering orders and meeting the requirements of customers and companies.

We can describe the VRP problem in the following way. We have a graph, which contains directed and undirected edges. The vertices of the graph are cities where customers are located. The edges of the graph represent roads between cities. The objective of VRP is to find a set of routes, that serve the orders of all customers and the cost function of this set is minimized. Cost function is a special function, which assigns a real number to routes of truck.

VRP is divided into two groups of problems: static and dynamic problems. Static problems are such problems, that the requirements for services are known in advance and do not change during the execution of given services. We also assume, that trucks provide the same type of service. In case of dynamic problems, service demands can change during the delivery. In our bachelor thesis we deal with static problems. The main components of VRP are: road network, orders, trucks, global and local constraints.

2.2 Modelling of VRP, Global and Local Constraints

We use a graph for modelling the network of VRP. The edges of this graph represent roads between cities. Every road has two properties, which are essential information for VRP modelling. The first property is the direction of the road. A road can be one-way or two-way. The other property is the length of the road that we represent with a non-negative real number. If we have a two-way road, the cost is the same in both directions. Vertices of road network represent depots, warehouses and customers.

Trucks deliver goods to customers. We can describe them with different properties, such as speed, capacity, cost per unit distance and reliability. The truck has only two duties, picks up the order and delivers it to the customers. The most important information for the orders are the location of the warehouse of the order, location of costumer, quantity and size of goods, priority of order and desired delivery time.

We use global and local constraints for modelling VRP. Local constraints are such constraints that apply only on a subset of variables. For example, the loading

area of truck with identification number 111 has to be filled at least fifty percent. In book [4] *global constraint* is defined as: *"A global constraint is a constraint that can be applied over arbitrary subsets of variables."* For example, we minimize the global transportation cost. In general, local and global constraints are incompatible. In some cases, when we want to satisfy the given local constraint, it can happen that we dissatisfy the global constraint. For this reason we assign each constraint a priority, and we create a priority list. We have to decide in advance, which constraint we prefer more. We cannot choose constraints that are in conflict. If we do not have a priority list, the global constraints automatically have higher priority. The goal of modelling is to simplify the understanding of the problem.

2.3 Basic Types of VRP

According to the customer requirements VRP is divided into two main classes: node routing problem (NRP) and arc routing problem (ARP). In [5] ARP is defined as: *"ARPs concern the distribution of goods or materials along the arcs (edges) of a road network"*. In case of ARP, vertices of graph are irrelevant. In case of NRP, customers are located in cities. In this section we deal with the NRP problem. In many cases, NRP is called as vehicle scheduling problem. NRP belongs to the NP-hard problems. If we remove global and local constraints, the problem can be reduced to TSP problem, which we mentioned at the end of the first chapter.

If we want to reduce VRP to TSP, we have to add extra edges to build a complete graph. The complete graph is a graph with $\binom{n}{2}$ edges, where the number of vertices is n in a graph. In complete graphs, every vertex is connected with every other vertex. We choose one vertex from a set of vertices, which we denote as the depot of truck. The other ones are customers. We deliver every order by one truck. In addition, we allow infinity capacity for the truck. Infinite capacity enables the truck to transport every order at once. The goal of this problem is to visit every costumer and satisfy their demands provided that the truck starts its tour in the depot, visits every costumer only once and at the end of the tour it returns to the depot and the cost of the tour is minimal. This tour is called a Hamiltonian circuit. In [6] *Hamiltonian circuit* (cycle) is defined as: *"A Hamiltonian cycle in a graph G is a cycle containing all vertices of G."*

The following enumeration of VRP problems is inspired by [5] and [7]. On figure 2.1 we can see the relationship among VRP problems.

We extend TSP problem to multiple travelling-salesman problem (MTSP). The difference between MTSP and TSP is that we have trucks and every truck must serve at least one customer. "The solution of MTSP is minimum cost tours which start and end at the depot so that each customer is visited exactly once."[5]

We can extend MTSP to capacitated VRP (CVRP). The difference between MTSP and CVRP is that the capacity of trucks cannot be infinitive. Problem CVRP contains m identical trucks with capacity c. The other constraint is that customer demand is always smaller than c. The well-known modification of CVRP is distance constrained VRP (DCVRP) where in addition we have distance constraints. The solution of DCVRP is the minimum total length of Hamiltonian circuits. There exists a special case of problem CVRP, which is called VRP with

backhaul (VRPB). In case of VRPB we divide customers into two groups. The first group of customers are Linehaul Customers (LC), these customers need to deliver a required amount of goods. The second group of costumers are Backhaul Customers (BC), they require a quantity of goods to be picked up. There is an important constraint that holds for transport on the route: Every LC has precedence before BC.

VRP with time windows (TWVRP) is a variant of CVRP. In this problem, every customer has a time window. Moreover, the service time is given for all customers. If a truck arrives before the start of time window, it has to wait until time window start and then it can start to serve the costumer. We can only deliver the orders during the time window. The routes start at time 0. The solution of VRPTW is a set of k routes with minimum cost. For each route the following conditions apply: Each route must visit the depot. Each customer is visited only by a single route. The sum of demands does not exceed the capacity of truck. For each customer we provide service only in defined time interval.

The last type of VRP is VRP with pickup and delivery (VRPPD). In this case, every customer is associated with two quantities. The first demand consists of commodities that need to be delivered and the second demand consists of commodities that are needed to be picked up. For each customer we define the origin of delivery demand and the destination of pick up demand. For each customer the delivery is performed before the pick up. Until now we modelled these problems with graphs. In the next sections, we introduce other modelling techniques.

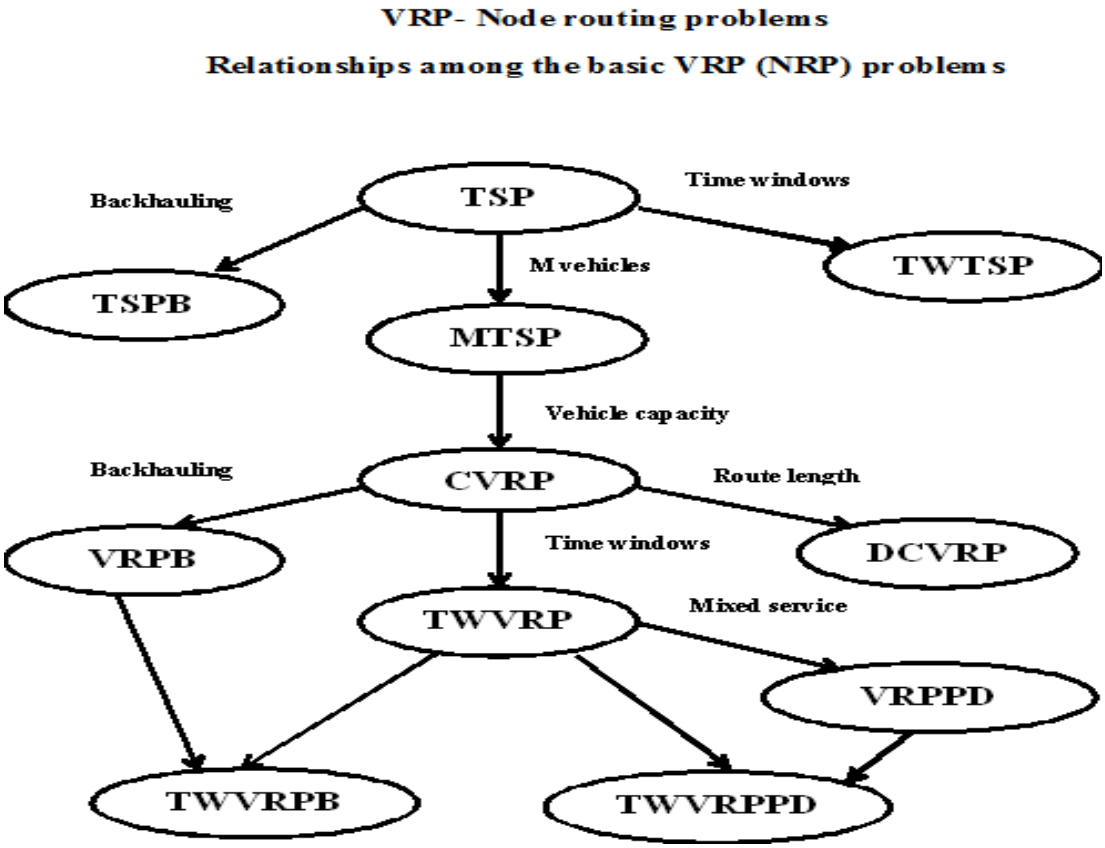


Figure 2.1: Relationship among VRP problems (source [5])

2.4 Modelling with Constraint Programming

In theoretical informatics there exists a discipline which specializes in solving problems with constraints. This discipline is called *constraint programming* (CP). In [8] CP is defined as: *"The idea of constraint programming is to solve problems by stating constraints (conditions, properties) which must be satisfied by the solution."*

One of the main tasks of CP is to model the problem as a *constraint satisfaction problem* (CSP). In [8] CSP is defined as: *"The Constraint Satisfaction Problem (CSP) is a problem where one is given: a finite set of variables, a function which maps every variable to a finite domain and a finite set of constraints."*

The advantage of modelling with constraints is that we can filter the domain of variables. In case the domain of a variable is empty after filtering, then this indicates that the instance of problem has no solution.

We can model VRP with CP. We can find a detailed CP model of VRP in book [4].

2.5 Modelling with Linear Programming

Another discipline that deals with modelling and solving problems is linear programming (LP). Linear programming is the discipline of optimization. Before we introduce the definition of LP we need to define the notion of *polyhedron* from book [9]: *"A set P of vectors in R^n is called a (convex) polyhedron if $P = \{x|Ax \leq b\}$"*. In book [9] LP is defined as: *"Linear programming, LP for short, concerns the problem of maximizing or minimizing a linear functional over a polyhedron. Examples are $max\{cx|Ax \leq b\}$, $min\{cx|x \geq 0; Ax \leq b\}$."*

Problem VRP is modelled with *integer linear programming*. In book [9] *integer linear programming* (ILP) is defined as: *"Integer linear programming (ILP) investigates linear programming problems in which the variables are restricted to integers: the general problem is to determine $max\{cx|Ax \leq b; xintegral\}$."* We can find an ILP formulation of VRP in book [4].

3. Analysis

In this chapter we explain the reason why we created two applications in our thesis. We explain the selection of scheduling algorithms and the selection of the third party libraries.

3.1 Applications

Before we started to work on our thesis, we created a summary what our thesis should contain. We decided that we are going to create two applications. We did not find any good reason to merge these two applications into a single application, because the functionality of these two applications differ completely. The first application that we created is the **Simulator** application. This application enables to run scheduling algorithms on TWVRP model instances. After we created the **Simulator** application, we wanted to test it on different TWVRP model instances with different scheduling algorithms. In order to facilitate the creation of TWVRP model instances we created the **Editor** application. This application enables to create TWVRP model instances manually or generate random model instances. After we created the **Editor** application, we needed to implement some scheduling algorithms that we are going to test and compare on the TWVRP model instances.

3.2 Algorithms

Before we selected the scheduling algorithms for the **Simulator** application, we consulted with our supervisor what kind of scheduling algorithms he recommends to implement. After the consultation we decided that the scheduling algorithms that we implement are from different algorithm families.

The first algorithm that we selected belongs to the family of greedy algorithms. The advantage of this algorithm is that it computes really fast, returns relatively good results on some special cases and it is easy to implement.

The second algorithm that we selected belongs the the family of tree search algorithms. The advantage if this algorithm is that in most cases it returns better results than the greedy algorithm. On the other hand, the running time of this algorithm is a lot longer than the running time of the greedy algorithm.

The last algorithm that we selected belongs to the family of iterated local search algorithms. The curiosity of this algorithm is that it approaches our problem from a completely different angle than the previous two algorithms.

3.3 Libraries

When we were implementing the applications, one of the largest problems that we faced was the visualization of graphs. We needed some library that visualizes graphs in Java. We tried out two libraries **JUNG** (`http://jung.sourceforge.net/`) and **GraphStream** (`http://graphstream-project.org/`). For our needs

in the **Editor** application the **JUNG** library was more suitable than the **GraphStream** library. On the other hand, the JUNG library is not very suited for the visualization of simulations on graphs. In the **Simulator** application we tried to visualize the movements of trucks with GraphStream. After two weeks of debugging we made a decision that we will write our own visualization.

4. Algorithms

In this chapter we introduce our model that represents the TWVRP problem, followed by the description of the three algorithms that we implemented. The description of each algorithm contains pseudo code, explanation of pseudo code and running time complexity analysis. For each algorithm we label the number of cities with c, the number of roads with r, the number of trucks with t and the number of orders with o.

4.1 Model of Problem

In the second chapter we have described some special types of VRP problems. For each type we focused on its special characteristics. The most important common characteristic of these problems is that each of them minimizes (maximizes) its objective function. Our model of the problem consists of four different object types: city, road, truck and order.

City Properties

The cities of our model have five basic properties. They represent the vertices of the graph. The five basic properties are:

1. **Identification number**: This property is unique for every city in the TWVRP model and we represent it by a unique positive integer number.
2. **Name**: This property is unique for every city in the TWVRP model and we represent it by a string.
3. **Number of neighbors**: This property tells us how many cities are connected with the given city. The value of this property is represented by an integer number. It can be equal to zero or a positive integer number.
4. **X coordinate**: This property contains the **x coordinate** of city in plane. The value of this property can be zero or a positive integer number.
5. **Y coordinate**: This property contains the **y coordinate** of city in plane. The value of this property can be zero or a positive integer number.

The last two properties represent the **position** of the city in plane. This position is fictitious and we only need it for graphical visualization.

For each city in the model we can add individual properties, which are represented by two strings. The first string is the **name** of property and the second string is its **value**.

Road Properties

The roads of our model have four basic properties. They represent the edges of the graph in our model. The four basic properties are:

1. **Identification number**: This property is unique for every road in the TWVRP model and we represent it by a unique positive integer number.

2. **Identification number of first city**: This property contains the identification number of the first city that the roads connects. The value of this property is represented by a positive integer number.

3. **Identification number of second city**: This property contains the identification number of the second city that the roads connects. The value of this property is represented by a positive integer number.

4. **Length**: This property contains the length of road. The value of this property is represented by a positive real number.

The second and third property specify the cities that the road connects. The value of these properties must be different. We do not allow loops in our model. Our model only allows one road between each pair of city.

For each road in the model we can add individual properties. Individual properties are represented in the same way as the individual properties of cities. The cities and roads represent the **network** of the TWVRP model.

Truck Properties

The trucks of our model have six basic properties. They deliver the orders from the warehouses to the costumers. The six basic properties are:

1. **Identification number**: This property is unique for every truck in the TWVRP model and we represent it by a unique positive integer number.

2. **Location**: Location is represented by an integer number. The value of this property can be equal to zero or a positive integer number. It is equal to zero when the truck is driving on a road, otherwise it is equal to the identification number of the city where it is located.

3. **Speed**: This property contains the speed of truck. The value of this property is represented by a positive real number.

4. **Cost**: This property represents the travelling expense of truck for a step in the simulation. The value of this property is represented by a positive real number.

5. **Waiting cost**: This property represents the waiting expense of truck. The value of this property is represented by a positive real number.

6. **Capacity**: This property contains the capacity of truck. The value of this property is represented by a positive real number.

For each truck in the model we can add individual properties. Individual properties are represented in the same way as the individual properties of cities.

Order Properties

The orders of our model have seven basic properties. These seven basic properties are:

1. **Identification number**: This property is unique for every order in the TWVRP model and we represent it by a unique positive integer number.

2. **Location of warehouse**: This property contains the identification number of city where the warehouse is located. The value of this property is represented by a positive integer number. It is always equal to the identification number of a city.

3. **Location of customer**: This property contains the identification number of city where the customer is located. The value of this property is represented by a positive integer number. It is always equal to the identification number of a city.

4. **Size**: This property contains the size of order. The value of this property is represented by a positive real number.

5. **Income**: This property contains the income of order. The value of this property is represented by a positive real number.

6. **Time window start**: This property specifies the time when we can start to deliver our order. The value of this property is represented by an integer number. It can be equal to zero or a positive integer number. If the value of this property equals to zero then it indicates that we can start our delivery immediately.

7. **Time window end**: This property specifies the time until the order must be delivered. The value of this property is represented by a positive integer number. The **time window end** property must always be larger than the **time window start** property.

The last two properties specify the **time window interval** of order. We cannot start the delivery before the time window start and we cannot deliver the order after the time window end.

For each order in the model we can add individual properties. Individual properties are represented in the same way as the individual properties of cities.

Time Unit Property

The last property in our model is its **time unit**. This property specifies how long does a step take in our model. The value of this property is represented by a positive integer number.

Objective Function

The objective of our scheduling algorithms is to create such schedules that maximize profit.

4.2 Greedy Algorithm

The first algorithm that we implemented is the modification of the greedy algorithm from [3], that solves the **interval-graph coloring problem**.

From the orders of our problem we can create an interval-graph, whose vertices represent the time windows of the orders and the edges connect orders that are incompatible. In our context *"incompatible"* means that none of the trucks is able to deliver the two orders together.

Pseudo Code

1. run the **Floyd-Warshall** algorithm to find shortest paths in the graph
2. initialize $s = 15$
3. **for each** truck create an empty cluster
4. sort orders by the time window end
5. **while** list of clusters not empty **do**
6. **begin**
7. **if** list of sorted orders is empty **then**
8. **break**
9. cluster = first cluster from list of clusters
10. remove cluster from list of clusters
11. **for each** sorted order **do**
12. **begin**
13. add order to cluster
14. **if** unable to add order to cluster **then**
15. add order to the list of unused orders
16. **if** cluster is full **then**
17. **break**
18. **end**
19. **if** cluster is not empty **then**
20. **begin**
21. create a plan for truck
22. add cluster at the end of the list of clusters
23. **end**

24. **for each** order from the list of unused orders **do**

25. add order to the list of sorted orders

26. **end**

Explanation of Pseudo Code

In line 1 we run the **Floyd-Warshall** algorithm to find the shortest path between every pair of vertices in the graph.

In line 2 we initialize the s parameter of the algorithm, which limits the size of each cluster.

In line 3 we create a cluster for every available truck and on line 4 we sort every available order by their time window end.

In line 5 we can see the main loop of our algorithm. The algorithm runs until the list of clusters or the list of sorted orders is not empty.

In line 7 we test whether the list of sorted orders is not empty. In case the list is empty the algorithm terminates.

In line 9 we select the first cluster and remove it from the list of clusters.

In line 11 we iterate through the list of sorted orders and try to add the order to the cluster. In case we are unable to add the order to the cluster we add the order to the list of unused orders.

In line 16 we test whether the cluster is full. In case the cluster is full we terminate the loop of line 11.

In line 19 we test whether the cluster is empty. In case the cluster is not empty, we create a plan for the truck of cluster and insert the cluster at the end of the list of clusters.

In line 24 we insert every order from the list of unused orders to the list of sorted orders and clear the list of unused orders.

Running Time Complexity Analysis

The operation in line 1 takes $O(c^3)$ time. We can limit the time consumption of operation in line 3 by $O(t)$. We create for each truck only one cluster and the creation of this cluster takes $O(1)$ time. In line 4 we sort the available orders. This operation takes $O(o * \log o)$ time. The operation in line 11 and 24 takes $O(o)$ time. The main loop in line 5 takes $O(t * o)$ time, because we try to add each order to every cluster only once. The **total time consumption** of the algorithm is of order $O(c^3 + o * (\log o + t))$.

4.3 Beam Search Algorithm

The second algorithm that we implemented is the modification of the algorithm from [10]. We changed the behavior of the **k - means procedure (Algorithm 1)** in our algorithm. The description of these changes will follow later.

Pseudo Code

1. run the **Floyd-Warshall** algorithm to find shortest paths in the graph

2. initialize $\omega = 3$ and $n = 8$
3. **for each** truck create an empty cluster
4. **for each** order choose a cluster
5. **while** list of clusters is not empty **do**
6. **begin**
7. **for each** cluster **in** list **do**
8. **begin**
9. select n orders from cluster
10. create beam tree from selected orders
11. select leaf with highest profit in tree
12. insert unused orders of beam tree back to the cluster
13. **if** cluster is empty remove it from the list of clusters
14. **end**
15. **end**

Explanation of Pseudo Code

In line 1 we run the **Floyd-Warshall** algorithm to find the shortest path between every pair of vertices in the graph.

In line 2 we initialize the parameters of the beam search tree. The first constant n limits the depth of beam search tree, whereas second constant ω limits the number of descendants of each node in the beam search tree.

In line 3 we create empty clusters for each truck. Every cluster has a city that is the center of the cluster and a truck that is associated with it.

In line 4 we choose a cluster for every order. For each order we choose the cluster in the following way. In the first round we filter the trucks that have smaller capacity than the weight of order. In the second round we filter the trucks that are unable to deliver the order in time. In the third round we filter the trucks that are far away from the order. In the fourth round we test the number of remaining trucks. In case the number of remaining trucks is larger than one, we select the truck with the cluster that has the smallest number of orders.

In line 5 we can see the main loop of our algorithm. The algorithm runs until the list of clusters is not empty.

In line 9 we select n orders from the given cluster. We delete these orders from the given cluster and create the beam search tree from the selected orders.

In line 11 we put every leaf of the beam search tree into the list. We select the leaf with the largest profit from the list.

In line 12 we test whether we can insert back the unused orders of beam search tree to the cluster. In case we can still deliver the orders in time we insert the unused orders into the current cluster.

In line 13 we test whether the cluster is empty. In case the cluster is empty we remove it from the list of clusters.

Running Time Complexity Analysis

The operation in line 1 takes $O(c^3)$ time. We can limit the time consumption of operation in line 3 by $O(t)$. We create for each truck only one cluster and the creation of this cluster takes $O(1)$ time. We can limit the time consumption of operation in line 4 with $O(t * r)$. We choose for each order one cluster and never change it. Choosing a cluster takes $O(t)$ time. The operation in line 7 takes $O(t)$ time, because the number of clusters equals to the number of trucks. We can limit the time consumption of operation in line 9 by $O(n)$. In worst case the creation of the beam search tree takes $O(\omega^n)$ time. In our case it is only a large constant. In the worst case the operation on line 12 takes $O(n)$ time. The main loop in line 5 takes $O(r * t * \omega^n)$ time. The **total time consumption** of the algorithm is order of $O(c^3 + r * t * \omega^n)$.

4.4 Iterated Local Search Algorithm

The last algorithm that we implemented is the modification of the iterated local search (ILS) algorithm from [11]. The original algorithm does not limit the number of orders that the truck can deliver at once. In our algorithm we limit the number of orders to n that a truck can deliver.

Pseudo Code

1. run the **Floyd-Warshall** algorithm to find shortest paths in the graph
2. initialize $n = 10$
3. **while** list of available orders is not empty **do**
4. **begin**
5. **if** list of available trucks is empty **then**
6. **break**
7. **for each** truck from list of available trucks **do**
8. **begin**
9. select two orders that are furthest apart from each other
10. **if** pair found **then**
11. **begin**
12. call the ILS algorithm on the pair of orders

13. create plan for truck
14. **continue**
15. **end**
16. search order for truck
17. **if** no order has been found for truck **then**
18. remove truck from the list of available trucks
19. **end**

Explanation of Pseudo Code

In line 1 we run the **Floyd-Warshall** algorithm to find the shortest path between every pair of vertices in the graph.

In line 2 we initialize the parameter n to 10, which limits the number of orders that the truck can serve at once.

In line 3 we can see the main loop of the algorithm. The algorithm runs until the list of available orders or the list of available trucks is not empty.

In line 5 we test whether the list of available trucks is not empty. In case the list is empty the algorithm terminates.

In line 7 we try to create a schedule for each available truck. First we calculate the *distance* between each pair of available orders, that the truck can deliver in time. In our case the *distance* equals to the time that the truck waits before it can start the delivery of the second order. We select the pair of orders with the largest distance.

In line 12 we call on the pair of orders the original ILS algorithm with the restriction that the maximum number of orders that the truck can deliver at once equals to the parameter n.

In line 16 if no pair of orders has been found the algorithm searches for a single order that the truck could deliver in time. If no such an order exists the algorithm deletes the truck from the list of available trucks.

Running Time Complexity Analysis

The operation in line 1 takes $O(c^3)$ time. The operation in line 7 takes $O(o^2)$ time. With the restriction that we added to the algorithm the insertion step and the shake step of the ILS algorithm runs in $O(o)$ time. The operation in line 7 takes $O(t * o^2)$ time. The main loop in line 3 takes $O(t * o^3)$ time. The **total time consumption** of the algorithm is of order $O(c^3 + t * o^3)$.

5. Documentation

In this chapter we present the user guide and programmer guide of **Editor** and **Simulator** application. In the user guide we describe what are the system requirements of these applications. After that we describe how can the user work with these applications. In the programmer guide we describe the main modules of the two applications, important data structures and third-party libraries which we used.

Both of these applications were developed in the **Java** programming language. The main reason why we chose this language is that the language is cross-platform. Cross-platform means that we can run our compiled code on any platform that is supported by Java without recompilation. Another advantage of this language is that it has lots of built-in modules and it is easy to learn.

System Requirements of Editor and Simulator Application

Both of these applications were developed in **Java 1.8**. In order to run these applications on our operating system it is mandatory to have **Java Runtime Environment 1.8 (JRE)** installed on our operating system. We can check out the version of JRE on our system by the following way. We simply open a command line and type the command **java -version**. If the command line displays the following message *"java is not recognized as an internal or external command, operable program or batch file"*, then we do not have the Java JRE installed on our system. In that case we can download and install the appropriate version of Java from the following web page `http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html`.

5.1 User Guide of Editor Application

Earlier we mentioned that the main purpose of the Editor application is to create and modify TWVRP model instances. Now we describe in detail how to work with the Editor application. First we describe how to run the application and then we describe the menu items and tabs of the application.

Running Application Editor

In order to run the Editor application first we need to open a command line on our system. In the command line we need to find the location of the file **Editor.jar**. After that we type the following command **java -jar Editor.jar** and the application starts. The application always starts with an empty TWVRP model instance.

Menu Items of Editor Application

The first menu item that we describe is the **Open TWVRP** menu item. This menu item enables to select a text file which contains a TWVRP model instance and load it to the Editor application.

The second menu item that we describe is the **Save TWVRP** menu item. This menu item enables to select a text file, where the application saves the TWVRP model instance.

The third menu item that we describe is the **Close** menu item. This menu item enables to close the Editor application. Before the application is closed it asks whether we really would like to close the application. If the answer is **yes**, then the application asks whether we would like to save the current model instance from application to a text file. If the answer is **no** the application closes and current data of the application will be lost, otherwise we need to select a text file where the TWVRP model instance will be saved.

The fourth menu item that we describe is the **Change layout** menu item. This menu item enables to change layout of the graph in the **Graphical Tab**. After we click on this menu item a dialog box shows up. In this dialog box we can choose from three graph layout algorithms. The first algorithm is the Fruchtermann Reingold algorithm [12]. This algorithm works best with graphs up to 30 vertices. The second algorithm is the Kamada Kawai algorithm [13]. This is the only algorithm from the three implemented algorithms which works with road distances. The last algorithm is the Walshaw algorithm [14]. Changing the graph layout can be computationally demanding.

The fifth menu item that we describe is the **Time unit** menu item. This menu item enables to change the time unit of the current TWVRP model instance. In the TWVRP model time unit expresses numerically how long does a step takes in the simulation. The application enables only positive non-zero integer numbers as time unit.

The sixth menu item that we describe is the **Information** menu item. This menu item show information about the Editor application in a dialog box.

The last menu item we describe is the **Mouse Mode** menu item. This menu item enables to change the behavior of the mouse in the **Graphical Tab**.

Tabs of Editor Application

The application has eight different tabs **City Tab, Road Tab, Truck Tab, Order Tab, Graphical Tab, Generator Tab, Modified Tab** and **Tester Tab**. In these tabs we can modify the properties of the cities, roads, truck and orders that the current TWVRP model instance contains. We can remove cities, roads, trucks and orders from then current TWVRP model instance or add them to the current model. The tabs also enable generation of new TWVRP model instances.

City Tab

The first tab that we describe is the **City Tab** (see Figure 5.1). In this tab we can create new cities with the **Add City** button. Before we add a new city to the current model instance we must specify the name of the city. The name of the city must be a single-word. It must start with a capital letter and can be followed by any number of characters of the English alphabet. The characters can be followed by any number of digits. For example *City1, City* or *City12345* are all valid city names but the city name *C1ty15* is not allowed. In addition to **Add**

City button the tab also contains the **Load City, Add Property, Remove Property, Edit City, Remove City, Show City** and **Clear** buttons.

In order to load the basic properties of the selected city into the **City Tab** property fields we need to click on the **Load City** button.

In order to add a special property to the selected city we need to click on the **Add Property** button. The same rules must be applied in case of the special property name as in case of the city name.

In order to remove a special property of the selected city we need to click on the **Remove Property** button.

In order to edit the basic properties of a selected city, we need to fill in the new name of city into the city name text field and click on the **Edit City** button.

In order to remove the city from the TWVRP model, we need to click on the **Remove City** button. After we clicked on this button the application asks whether we really would like to remove the selected city. If our answer is **yes**, then the application removes the selected city, the roads which connect the city with other cities, trucks that are located in the city and orders whose costumer is located in the city or the warehouse of the order is located in the city.

In order to list every property of the selected city, we need to click on the **Show City** button. After we clicked on this button the information is shown in the text area. We can clear the content of the text area with the **Clear** button.

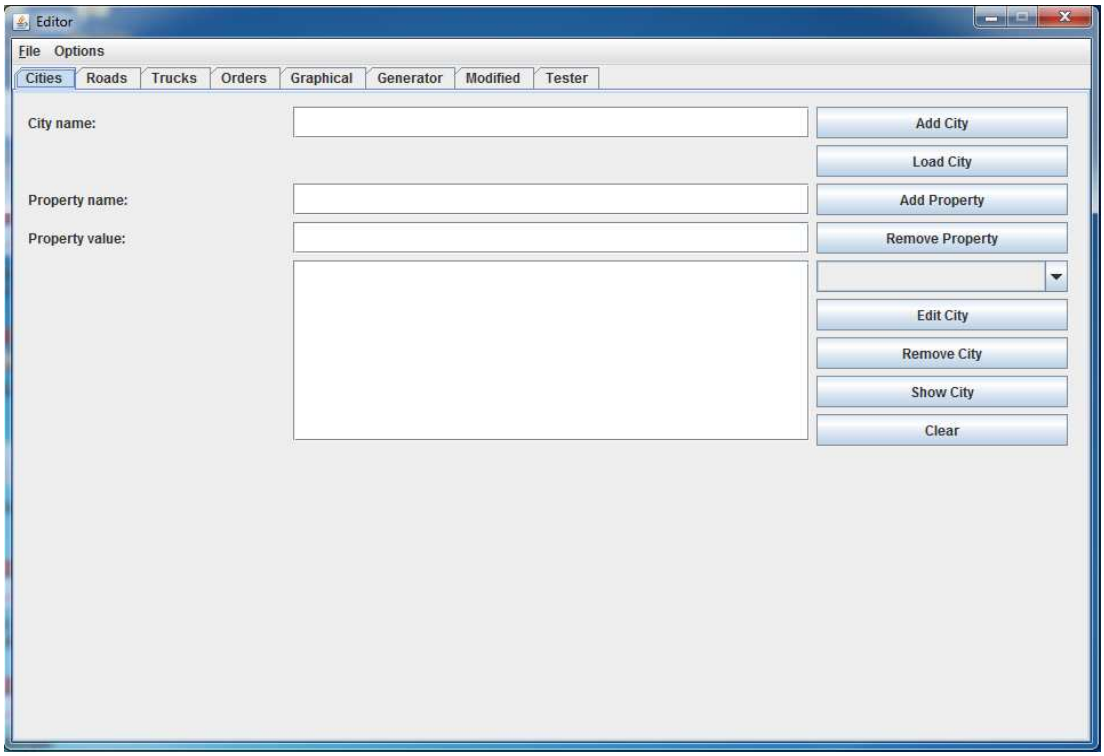


Figure 5.1: City tab of Editor application

Road Tab

The second tab that we describe is the **Road Tab** (see Figure 5.2). In this tab we can create new roads with the **Add Road** button. Before we add a new road to the current model instance we must specify three basic properties of the

new road. The first two properties are the pair of cities which the new road will connect. The third basic property that we must specify is the length of the new road. In addition to **Add Road** button the tab has the **Load Road, Add Property, Remove Property, Edit Road, Remove Road, Show Road** and **Clear** buttons.

In order to load the basic properties of the selected road into the **Road tab** property fields, we need to click on the **Load Road** button.

In order to add a special property to the selected road instance, we need to click on the **Add Property** button. The same rules must be applied in case of the special property name as in case of the city name.

In order to remove a special property of the selected road instance, we need to click on the **Remove Property** button.

In order to edit the basic properties of the selected road instance, we need to choose which cities will be connected by the edited road. We also need to fill in the road length text field. After that we need to click on the **Edit Road** button.

In order to remove the road from the TWVRP model instance, we need to click on the **Remove Road** button. After we clicked on this button the application removes the road from the current TWVRP model instance and removes the edge of graph from **Graphical Tab**.

In order to list every property of the selected road, we need to click on the **Show Road** button. After we clicked on this button the information is shown in the text area. We can clear the content of the text area with the **Clear** button.

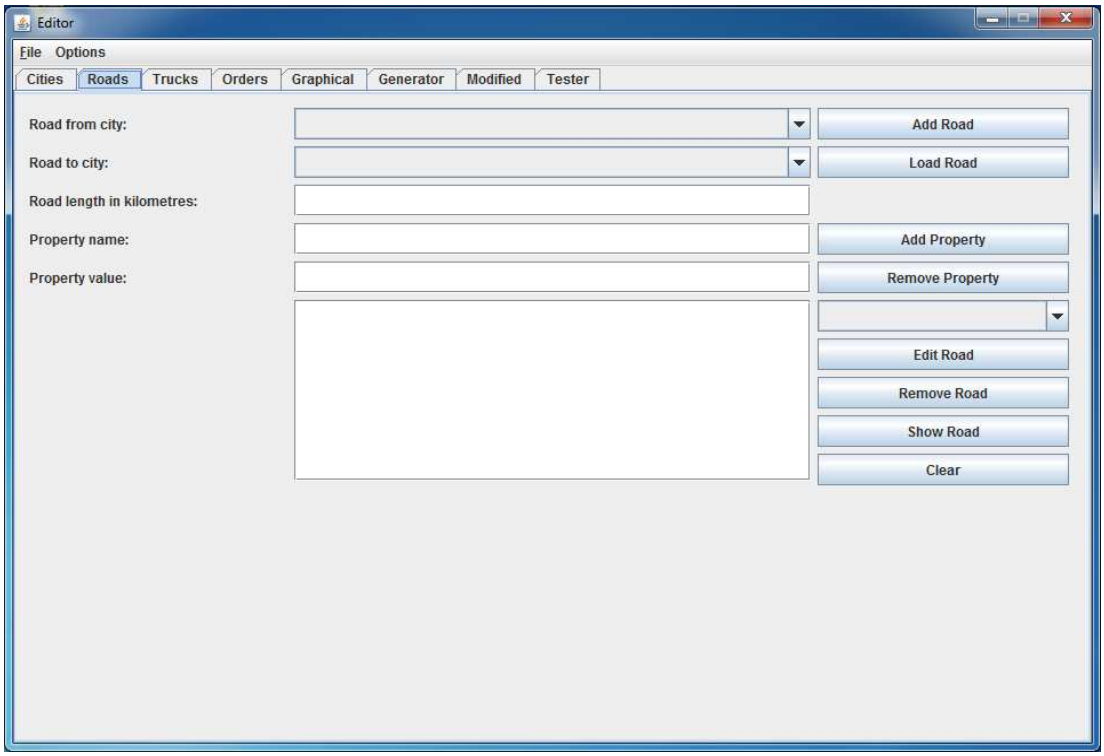


Figure 5.2: Road tab of Editor application

Truck Tab

The third tab that we describe is the **Truck Tab** (see Figure 5.3). In this tab we can create new trucks with the **Add Truck** button. Before we add a new truck to the current model instance we must specify five basic properties of the truck. These properties are location, speed, cost, penalty cost and capacity. In addition to **Add Truck** button the tab has the **Load Truck, Add Property, Remove Property, Edit Truck, Remove Truck, Show Truck** and **Clear** buttons.

In order to load the basic properties of the selected truck into the **Truck Tab** property fields, we need to click on the **Load Truck** button.

In order to add a special property to the selected truck instance, we need to click on the **Add Property** button. The same rules must be applied in case of the special property name as in case of the city name.

In order to remove a special property of the selected truck instance, we need to click on the **Remove Property** button.

In order to edit the basic properties of the selected truck instance, we need to choose the new location of the truck. We also need to fill in the truck speed text field, cost text field, penalty cost text field and the capacity text field. After that we need to click on the **Edit Truck** button.

In order to remove the truck from the TWVRP model instance, we need to click on the **Remove Truck** button.

In order to list every property of the selected truck, we need to click on the **Show Truck** button. After we clicked on this button the information is shown in the text area. We can clear the content of the text area with the **Clear** button.

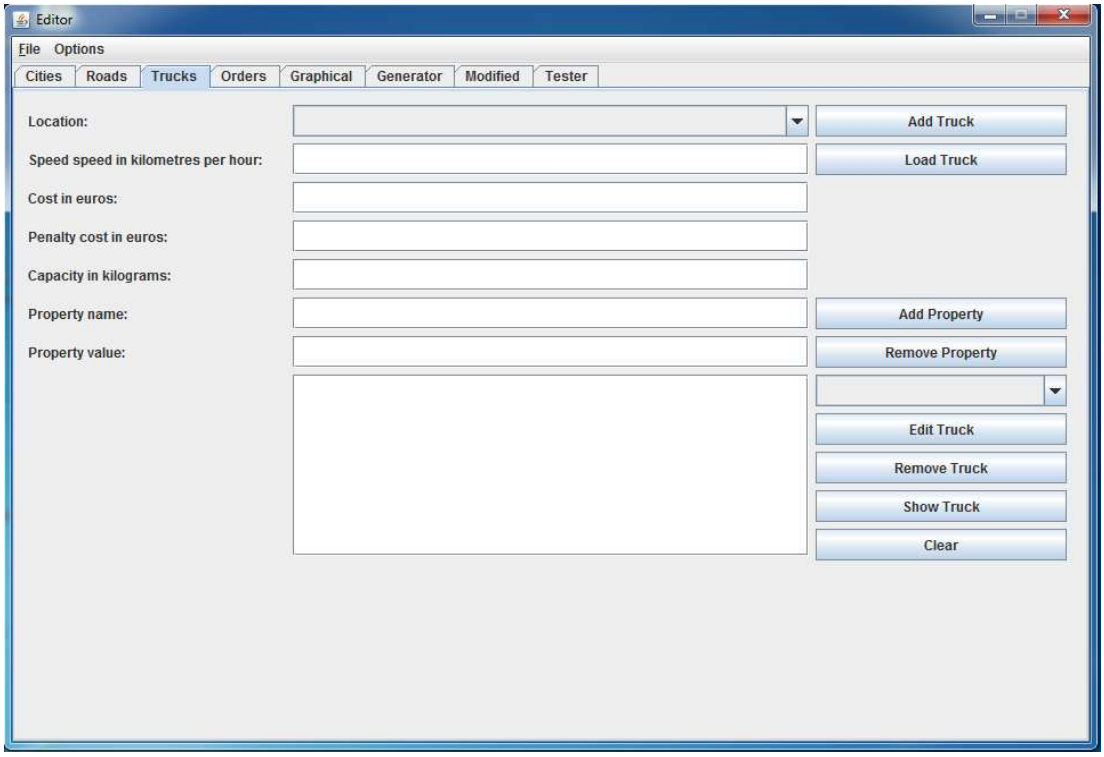


Figure 5.3: Truck tab of Editor application

Order Tab

The fourth tab that we describe is the **Order Tab** (see Figure 5.4). In this tab we can create new orders with the **Add Order** button. Before we add a new order to the current model instance we must specify six basic properties of the new order. These properties are the location of warehouse, the location of costumer, size, income, time window start and time window end. In addition to **Add Order** button the tab has the **Load Order, Date of Time Window Start, Date of Time Window End, Add Property, Remove Property, Edit Order, Remove Order, Show Order** and **Clear** button.

In order to load the basic properties of the selected order into the **Order Tab** property fields, we need to click on the **Load Order** button.

In order to set the date of time window start, we need to click on the **Date of time window start** button. After we clicked on this button a dialog box appears, where we can set the date of time window start. After we clicked on the **yes** option the application converts the selected date into milliseconds, converts the current date into milliseconds and subtracts the second from the first. After that the result of the subtraction is written to the time window start text field.

In order to set the date of time window end, we need to click on the **Date of time window end** button. After we clicked on this button a dialog box appears, where we can set the date of time window end. After we clicked on the **yes** option the application converts the selected date into milliseconds, converts the current date into milliseconds and subtracts the second from the first. After that the result of the subtraction is written to the time window end text field.

In order to add a special property to the selected order instance, we need to click on the **Add Property** button. The same rules must be applied in case of the special property name as in case of the city name.

In order to remove a special property of the selected order instance, we need to click on the **Remove Property** button.

In order to edit the basic properties of the selected order instance, we need to choose the new location of the order warehouse and costumer. We also need to fill in the order size text field, income text field, time window start text field and the time window end text field. After that we need to click on the **Edit Order** button.

In order to remove the order from the TWVRP model instance, we need to click on the **Remove Order** button.

In order to list every property of the selected order, we need to click on the **Show Order** button. After we clicked on this button the information is shown in the text area. We can clear the content of the text area with the **Clear** button.

Graphical Tab

The fifth tab that we describe is the **Graphical Tab** (see Figure 5.5). In this tab we can create the map of the current TWVRP model instance. In this tab the application enables to change the behavior of the mouse with the **Mouse Mode** menu item. We can choose from three different mouse modes. These modes are **Transforming, Picking** and **Editing**.

When the mouse mode is set to **Transforming** we can modify the shape of the graph. When we hold down the left **Shift** key and pull the mouse clockwise

the graph starts to rotate clockwise, otherwise it rotates anticlockwise. When we hold down the left **Ctrl** key and pull the mouse to right or left the application shears the shape of graph.

When the mouse mode is set to **Picking**, then the application enables to select one or more vertices in the graph. If we pick a vertex with the left mouse button in this mode and drag the mouse, then the application re-positions the vertex to the position where the mouse cursor is located. When we hold down the left **Ctrl** button and click on a vertex of the graph with the left mouse button, then the application re-positions the graph in such way that the vertex we clicked on is the closest vertex to the center of the **Graphical Tab**. When we hold down the left **Shift** key we can select more vertices simultaneously and re-position them.

When the mouse mode is set to **Editing**, then the application enables to edit the graph of the current TWVRP model instance. When we click on the tab with the left mouse button and the area where we clicked is empty, then the application creates a new vertex for the graph and adds a new city to the database of the application. If we click on a vertex with the left mouse button, drag the cursor of the mouse and stop on another vertex, then the application connects these two vertices with an edge. The default length of the created edge is 100, which we can edit in the **Road Tab**.

When we click on a vertex with the right mouse button a pop-up menu shows up. It shows the name of the city and also enables to delete the selected vertex. When we delete the vertex, the application also deletes the edges which connected the selected vertex to the other vertices, trucks that were located in the vertex and also orders. When we click on an edge with the right mouse button a pop-up menu shows up which shows the length of the edge and also enables to delete this edge.

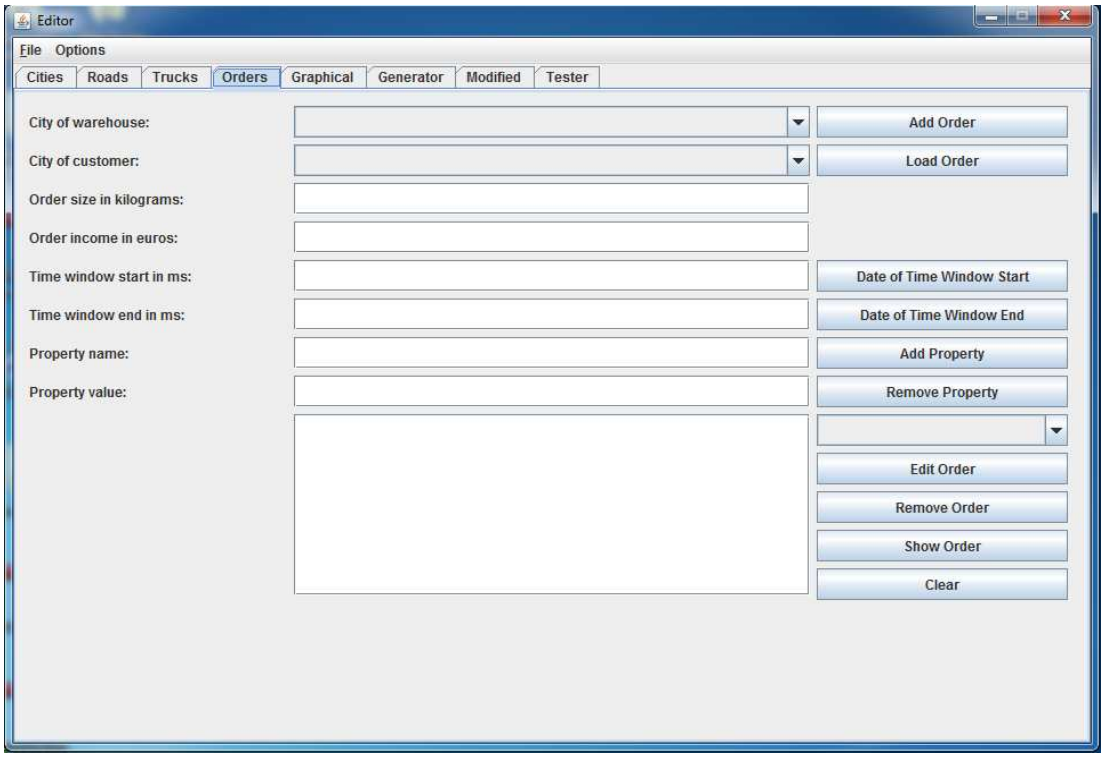


Figure 5.4: Order tab of Editor application

Generator Tab

The sixth tab that we describe is the **Generator Tab** (see Figure 5.6). This tab enables to generate random TWVRP model instances. In this tab we must define the minimum and maximum number of cities, roads, trucks and orders, that the generated TWVRP model instance will contain. We must also specify the basic property intervals of roads, trucks and orders. After we defined every interval, we need to click on the **Generate Model** button. After we clicked on this button a dialog appears, that warns the user that every data will be deleted before generation.

The application also enables to configure the spinners of **Generator Tab**. We can configure the spinner intervals with the **Configure Spinners from File** button. After we click on this button a window appears where we can search and select the text file, that contains the configuration of spinners. On figure 5.7 we can see an example of the configuration file.

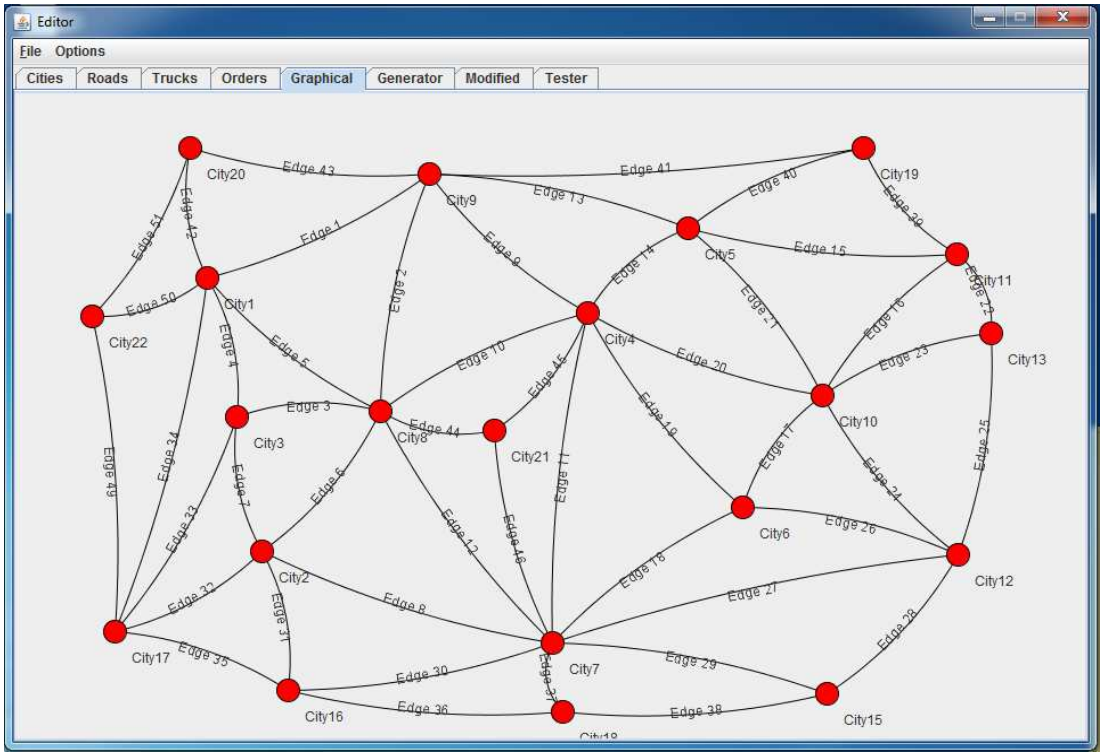


Figure 5.5: Graphical tab of Editor application

Modified Tab

The seventh tab that we describe is the **Modified Tab** (see Figure 5.8). This tab is a modification of the **Generator Tab**, it differs from **Generator Tab** in the definition of cities and roads. In this tab we can define the number of cities and roads with the score of the graph. We can type manually the score of the graph to the text field or read the score of the graph from a text file.

Tester Tab

The last tab that we describe is the **Tester tab** (see Figure 5.9). This tab is also a modification of the **Generator Tab**, it enables to create multiple TWVRP model instances. Each model instance is saved to its own text file

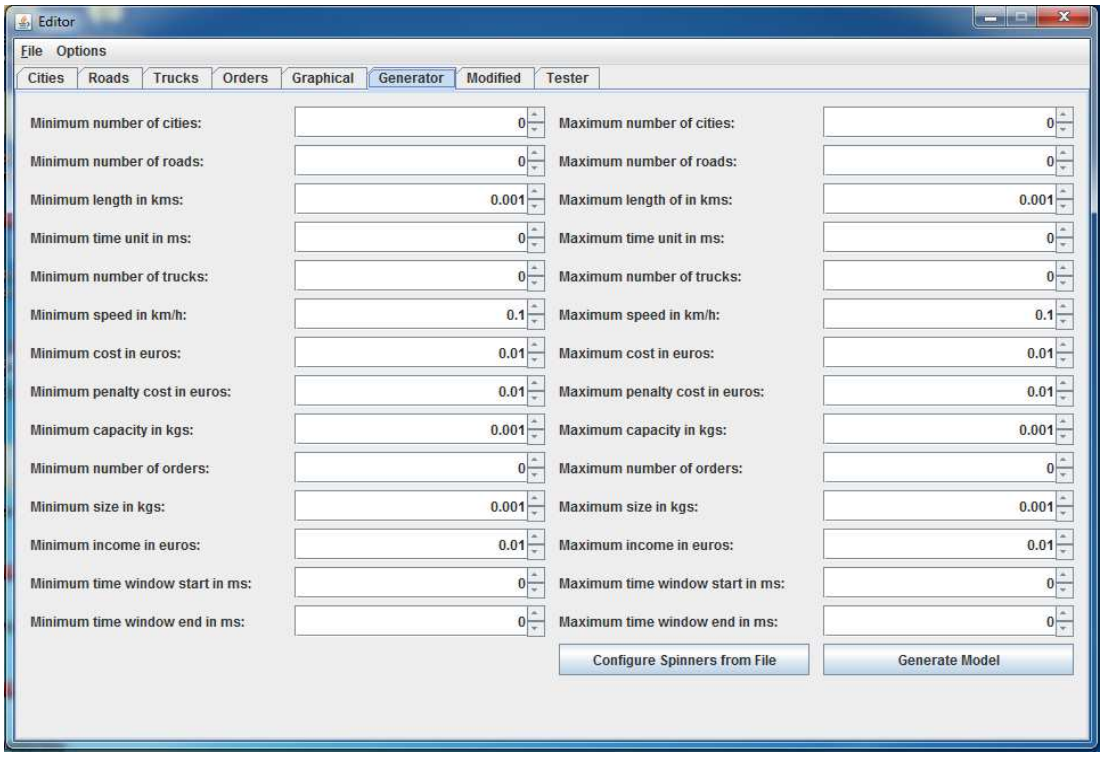


Figure 5.6: Generator tab of Editor application

```
0 200 // city number interval
0 20000 // road number interval
0.5 2000 0.5 // road length interval
0 200 // truck number interval
0.1 120 0.1 // truck speed interval
0.1 100 0.1 // truck cost interval
0.1 100 0.1 // truck penalty cost interval
2500 60000 0.01 // truck capacity interval
0 10000 // order number interval
0.2 30000 0.01 // order size interval
0.1 50000 0.01 // order income interval
0 1000000 10 // order time window start interval
0 2000000 15 // order time window end interval
0 200 // test number interval
0 2500 // time unit number interval
```

Figure 5.7: Generator spinner configuration file

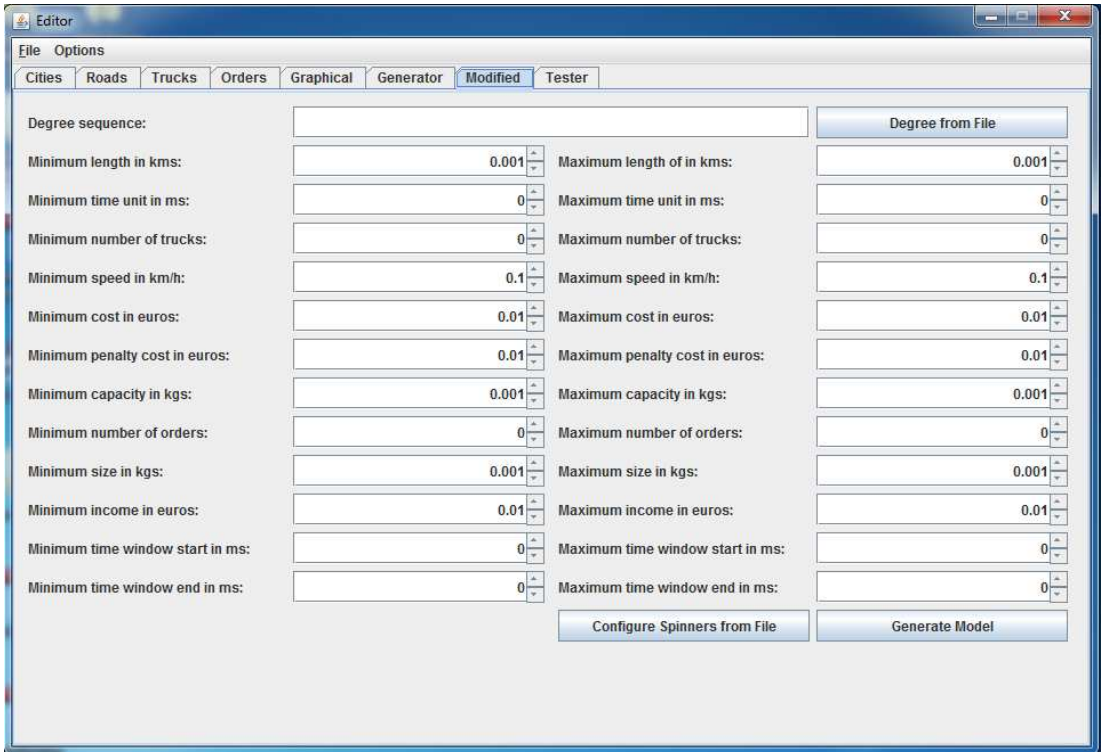


Figure 5.8: Modified tab of Editor application

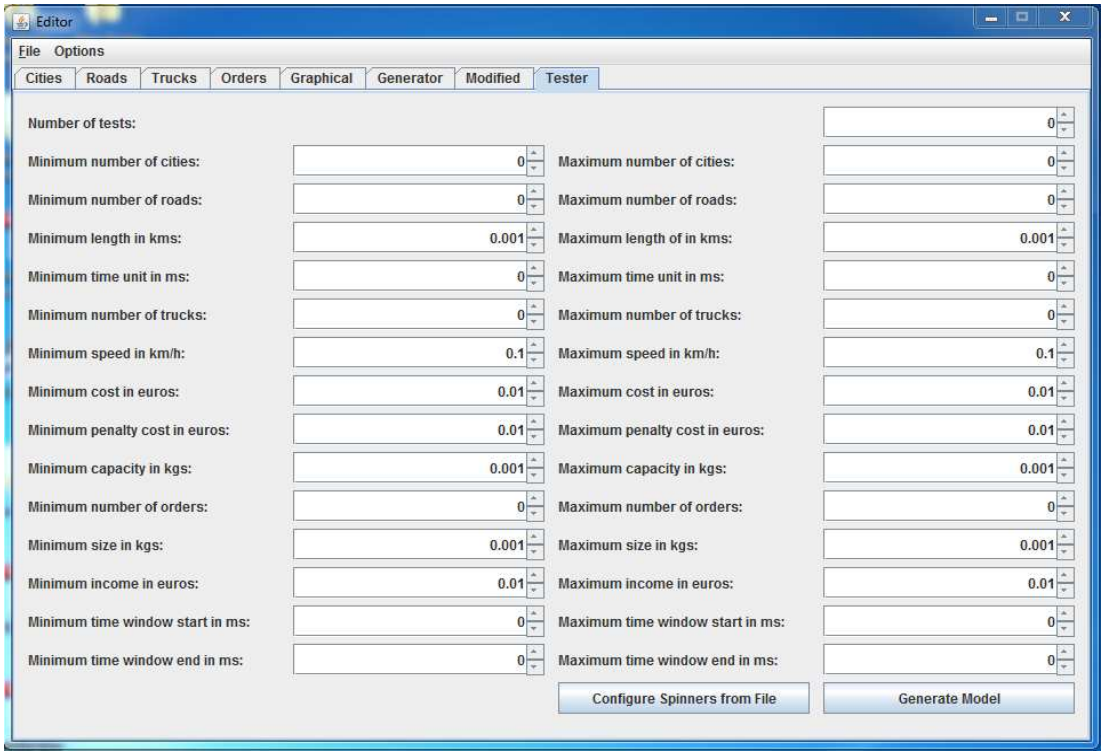


Figure 5.9: Tester tab of Editor application

5.2 Programmer Guide of Editor Application

Our **Editor** application consists of ten modules. These modules are **Main Module, I/O Module, City Module, Road Module, Truck Module, Order**

Module, Graphical Module, Generator Module, Modified Module and **Tester Module**. Before describing each of these modules we would like to introduce six classes (**Attribute, City, Road, Truck, Order, EditorDatabase**), that are essential for our **Editor** application. Each of these classes are from the **EditorDataStructures** package.

Essential classes of Editor application

The first class is the **Attribute** class. This class is the ancestor of the four modelling classes (City, Road, Truck and Order), which enable our program to create TWVRP model instances. Attribute class has two basic properties **identification number** and **map of properties**. The first property is represented by a unique positive integer number and the second property is represented by a hash map. This class enables to add or remove individual properties. On figure 5.10 we can see the inheritance of Attribute class.

The second class is the **City** class. This class is the descendant of the **Attribute** class and models the cities of the TWVRP model. In addition to the basic properties of the **Attribute** class this class has four other basic properties. The first property is the **city name**. This property is represented by a string and must be unique. The second property contains the **number** of **neighbor cities**. The third and fourth property represents the **position** of the city in the plane. These properties are represented by positive integer numbers.

The third class is the **Road** class. This class is the descendant of the **Attribute** class and models the roads of the TWVRP model. In addition to the basic properties of the **Attribute** class this class has three other basic properties. The first two basic properties are the **identification number of the cities** that the road connects. These two properties are represented by positive integer numbers. The last basic property is the **length** of road. This property is represented by a positive real number.

The fourth class is the **Truck** class. This class is the descendant of the **Attribute** class and models the trucks of the TWVRP model. In addition to the basic properties of the **Attribute** class this class has five other basic properties. The first basic property is the **identification number** of the city where the truck is located. This property is represented by a positive integer number. The other four basic attributes are **speed, cost, penalty cost** and **capacity**. Each of these properties is represented by a positive real number.

The fifth class is the **Order** class. This class is the descendant of the **Attribute** class and models the orders of the TWVRP model. In addition the basic properties of the **Attribute** class this class has six other basic properties. The first basic property is the **identification number** of the city, where the warehouse of the order is located. The second property is the **identification number** of the city, where the costumer of the order is located. The third property is the **size** of the order and the fourth property is the **income** of the order. These two properties are represented by positive real numbers. The last two properties represent the **time window** of the order. They are represented by positive integer numbers.

The last class that we describe is the **EditorDatabase** class. This class contains every city, road, truck and order of the TWVRP model instance. The class ensures fast insertion and search of objects. This class also enables to

calculate the shortest path between every city of the TWVRP model instance. The main task of this class is to ensure that the content of all module of the **Editor** application is in consistency.

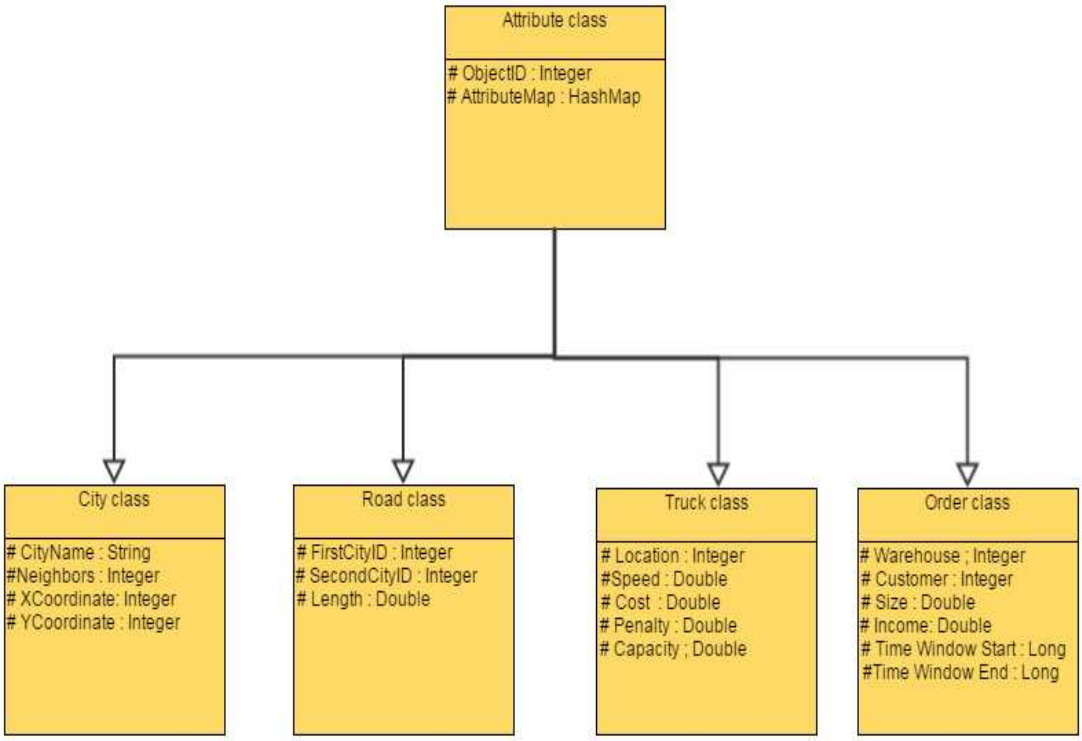


Figure 5.10: Inheritance of Attribute class

Main Module

The first module that we present is the **Main Module**. The duty of this module is to keep the **City tab, Road tab, Truck tab, Order tab** and **Graphical tab** content in consistency. This module is represented with the **EditorDatabase** class, which is located in the **EditorDataStructures** package.

I/O Module

The second module that we present is the **I/O module**. This module processes the events of **Open TWVRP** and **Save TWVRP** menu items. First we describe how this module processes the events of **Open TWVRP** and then the events of **Save TWVRP**

When we click on the **Open TWVRP** menu item the I/O module asks us to select a file that contains TWVRP model instance. After we selected the file the I/O module tests whether the selected file exists, is not a directory and is readable. After testing the I/O module starts to read the cities of the TWVRP model instance. On each city the module runs a test which tests the validity of the properties of city. If any of the properties is incorrect the module shows a warning message in a dialog box. Before inserting the module tests whether the currently processed city exists in database. If the city is not present in the

database then the module adds it to the database. On figure 5.11 we can see the example of a valid city instance in text format.

```
city identification number : 2
city name : City2
number of city neighbours : 4
x coordinate : 242
y coordinate : 388
number of properties : 0
```

Figure 5.11: City text format

When the I/O module finished the reading of city instances, it starts to read the roads from file. On each road the module runs a test which tests the validity of the properties of road. If any of the properties is incorrect the module shows a warning message in a dialog box. Before inserting the module tests whether the currently processed road exists in database. If the road is not present in database then the module adds it to the database. On figure 5.12 we can see the example of a valid road instance in text format.

```
road identification number : 11
road starting city identification number : 18
road ending city identification number : 17
road length : 100.0
number of properties : 2
name of property : Attr1
value of property : value
name of property : Attr2
value of property : value
```

Figure 5.12: Road text format

When the I/O module finished the reading of road instances, it starts to read the trucks from file. On each truck the module runs a test which tests the validity of the properties of truck. If any of the properties is incorrect the module shows a warning message in a dialog box. Before inserting the module tests whether the currently processed truck exists in database. If the truck is not present in database then the module adds it to the database. On figure 5.13 we can see the example of a valid truck instance in text format.

When the I/O module finished the reading of truck instances, it starts to read the orders from file. On each order the module runs a test which tests the validity of the properties of order. If any of the properties is incorrect the module shows a warning message in a dialog box. Before inserting the module tests whether the currently processed order exists in database. If the order is not present in database then the module adds it to the database. On figure 5.14 we can see the example of a valid order instance in text format.

```
truck identification number : 4
truck city identification number : 8
truck speed : 80.0
truck cost : 40.0
truck penalty cost : 40.0
truck capacity : 50000.0
number of properties : 0
```

Figure 5.13: Truck text format

```
order identification number : 3
order city from identification number : 3
order city to identification number : 1
order size : 30.0
order income : 4789.0
order date from : 200
order date to : 45000
number of properties : 0
```

Figure 5.14: Order text format

When we click on the **Save TWVRP** menu item the I/O module asks us to select a file where the TWVRP model instance will be saved. After we selected the file the I/O module tests whether the selected file exists, is not a directory and is readable. After testing the I/O module first writes the cities into the selected text files. When every city is written to the text file it continues with the roads, than truck and finally it writes the orders.

The main duty of this module is to load and save TWVRP model instances and keep the **City Tab, Road Tab, Truck Tab, Order Tab** and **Graphical Tab** content in consistency by calling update functions of **Main Module**. The implementation of this module is located in the **EditorMenu** package.

City Module

The third module that we present is the **City Module**. This module processes the events of **City Tab**. It tests the validity of user input. If the input is invalid or already exists in database, then the city module shows the appropriate warning message in a dialog box. The main duty of this module is to keep the **City Tab, Road Tab, Truck Tab, Order Tab** and **Graphical Tab** content in consistency by calling update functions of **Main Module**. The implementation of this module is located in the **CityModule** package.

Road Module

The fourth module that we present is the **Road Module**. This module processes the events of **Road Tab**. It tests the validity of user input. If the input is invalid or already exists in database, then the road module shows the appropriate warning message in a dialog box. The main duty of this module is to keep the **Road Tab** and **Graphical Tab** content in consistency by calling update functions of **Main Module**. The implementation of this module is located in the **RoadModule** package.

Truck Module

The fifth module that we present is the **Truck Module**. This module processes the events of **Truck Tab**. It tests the validity of user input. If the input is invalid or already exists in database, then the truck module shows the appropriate warning message in a dialog box. The main duty of this module is to keep the **Truck Tab** content in consistency by calling update functions of **Main Module**. The implementation of this module is located in the **TruckModule** package.

Order Module

The sixth module that we present is the **Order Module**. This module processes the events of **Order Tab**. It tests the validity of user input. If the input is invalid or already exists in database, then the order module shows the appropriate warning message in a dialog box. The main duty of this module is to keep the **Order Tab** content in consistency by calling update functions of **Main Module**. The implementation of this module is located in the **OrderModule** package.

Graphical Module

The seventh module that we present is the **Graphical Module**. This module processes the events of **Graphical Tab**. It tests the validity of the mouse events. In case the event is invalid the module ignores the event. The main duty of this module is to keep the **City Tab, Road Tab, Truck Tab, Order Tab** and **Graphical Tab** content in consistency by calling update functions of **Main Module**. The implementation of this module is located in the **MainModule** and **EditorMenu** packages.

Generator Module

The eighth module that we present is the **Generator Module**. This module processes the events of **Generator Tab**. It tests the validity of user input. If the input is not valid, then the generator module shows the appropriate warning message in a dialog box. The main duty of this module is to keep the **City Tab, Road Tab, Truck Tab, Order Tab** and **Graphical Tab** content in consistency by calling update functions of **Main Module**. The module also enables to re-configure the spinners of **Generator Tab**. The implementation of this module is located in the **GeneratorModule** package.

Modified Module

The ninth module that we present is the **Modified Module**. This module processes the events of **Modified Tab**. It tests the validity of user input. If the input is not valid, then the modified module shows the appropriate warning message in a dialog box. The main duty of this module is to keep the **City Tab, Road Tab, Truck Tab, Order Tab** and **Graphical Tab** content in consistency by calling update functions of **Main Module**. The module also enables to re-configure the spinners of **Modified Tab**. The implementation of this module is located in the **GeneratorModule** package.

Tester Module

The last module that we present is the **Tester Module**. This module processes the events of **Tester Tab**. It tests the validity of user input. If the input is not valid, then the tester module shows the appropriate warning message in a dialog box. The main duty of this module is to keep the **City Tab, Road Tab, Truck Tab, Order Tab** and **Graphical Tab** content in consistency by calling update functions of **Main Module**. The module also enables to re-configure the spinners of **Tester Tab**. The implementation of this module is located in the **GeneratorModule** package.

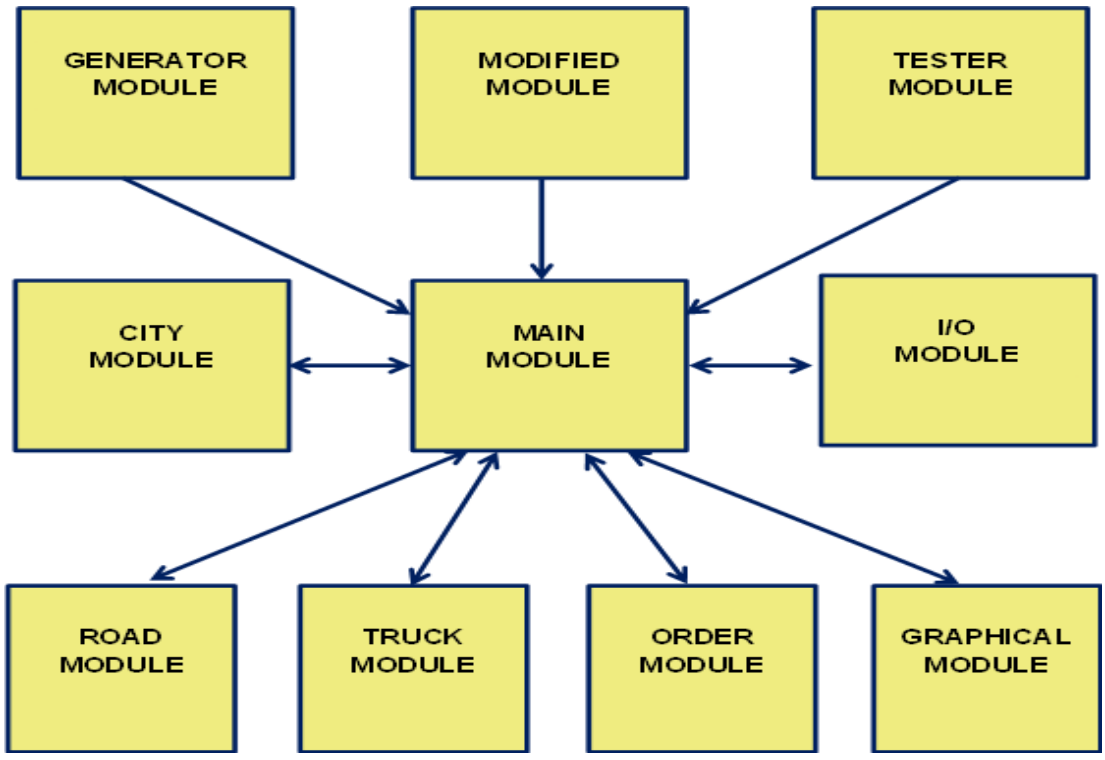


Figure 5.15: Communication among the ten modules of Editor application

Libraries

Editor application uses the libraries of the **Java Universal Network/Graph Framework** (JUNG). The framework enables to manipulate with graphs and has

rich functionality. Editor application only uses the basic functionality of JUNG to visualize, edit and reshape graphs. We build our application with JUNG 2.0.1. This is the last stable version of JUNG. We can find the libraries of JUNG on the CD in the **lib** directory.

5.3 User Guide of Simulator Application

Earlier we mentioned that the main reason why we created the Simulator application is to test our implemented algorithms on TWVRP model instances. Now we describe in detail how to work with the Simulator application. First we describe how to run the application and then we describe the menu items and tabs of the application.

Running Application Simulator

In order to run the Simulator application first we need to open a command line on our system. In the command line we need to find the location of the file **Simulator.jar**. After that we type the following command **java -jar Simulator.jar** and the application starts. The application always starts with an empty TWVRP model instance.

Menu Items of Simulator Application

The first menu item that we describe is the **Select Output Directory** menu item. This menu item enables to select a directory where the output files of the Simulator application will be saved.

The second menu item that we describe is the **Open TWVRP** menu item. This menu item enables to select a text file which contains a TWVRP model instance and load it to the Simulator application. Before we use this menu item it is mandatory to select an output directory for the application otherwise a warning message will appear in a dialog box.

The third menu item that we describe is the **Close** menu item. This menu item enables to close the Simulator application. Before the application is closed it asks whether we really would like to close the application. If the answer is **yes**, the application closes and the content of the application database will be lost.

The fourth menu item that we describe is the **Start Simulation** menu item. This menu item enables to run implemented algorithms on TWVRP model instances. When we click on this menu item a window appears where we can search and select the algorithm, which we would like to run on the currently loaded TWVRP model instance in the application.

The fifth menu item that we describe is the **Simulation Speed** menu item. This menu item enables to change the speed of simulation. The default speed of simulation is **0.01**. We can choose the speed of simulation from the interval [**0.001**, **10**].

The last menu item that we describe is the **Information** menu item. This menu item shows information about the Simulator application in a dialog box.

Tabs of Simulator Application

The application has five different tabs **Graphical Tab, Information Tab, Truck Information Tab, Order Properties Tab** and **Truck Properties Tab**. In these tabs we can measure the properties of simulation.

Graphical Tab

The first tab that we describe is the **Graphical Tab** (see Figure 5.16). This tab visualizes the movements of trucks on the network of the current TWVRP model instance.

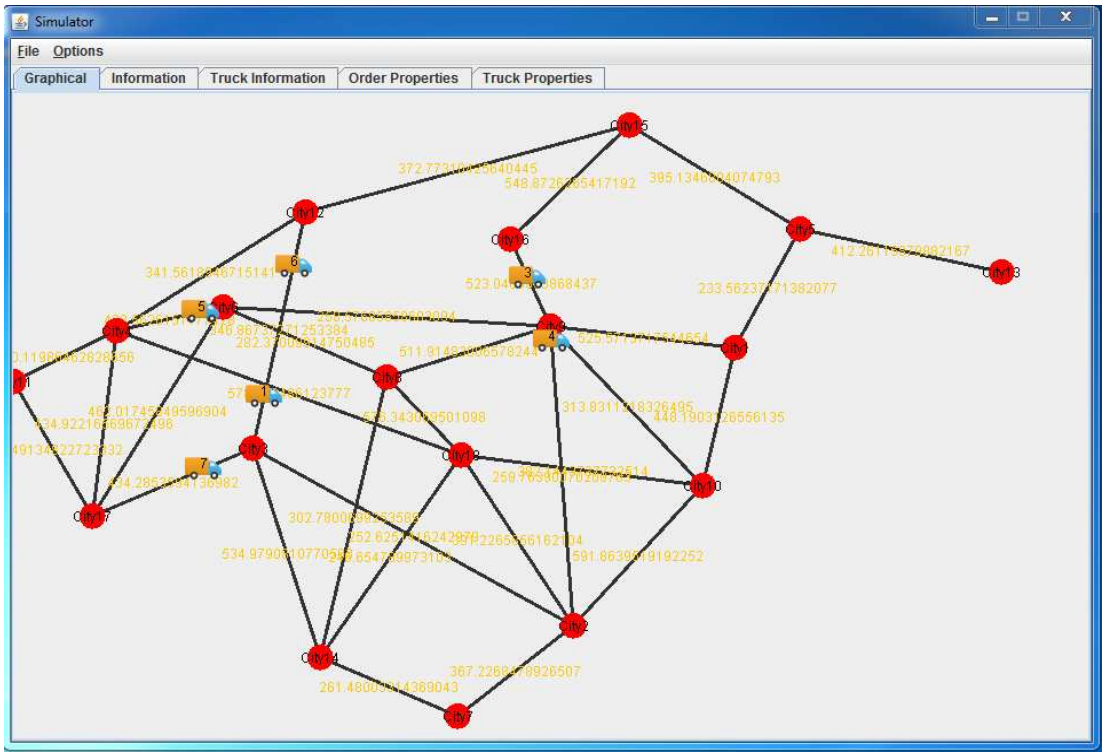


Figure 5.16: Graphical tab of Simulator application

Information Tab

The second tab that we describe is the **Information Tab** (see Figure 5.17). The tab displays the start date of testing, end date of testing, start date of simulation, end date of simulation, start date of algorithm and end date of algorithm. The tab also displays the running time of testing, simulation and algorithm in milliseconds. When a simulation is running we can read details about the movements of trucks in the text area of the **Information Tab**.

Truck Information Tab

The third tab that we describe is the **Truck Information Tab** (see Figure 5.18). For each truck this tab measures the distance that the truck travelled, time that the truck travelled, time that the truck waited, income that the truck earned, outcome that the truck spent, profit that the truck made and the number of

orders that the truck delivered. All this information is listed in a table. The last row of the table contains the sum of each column.

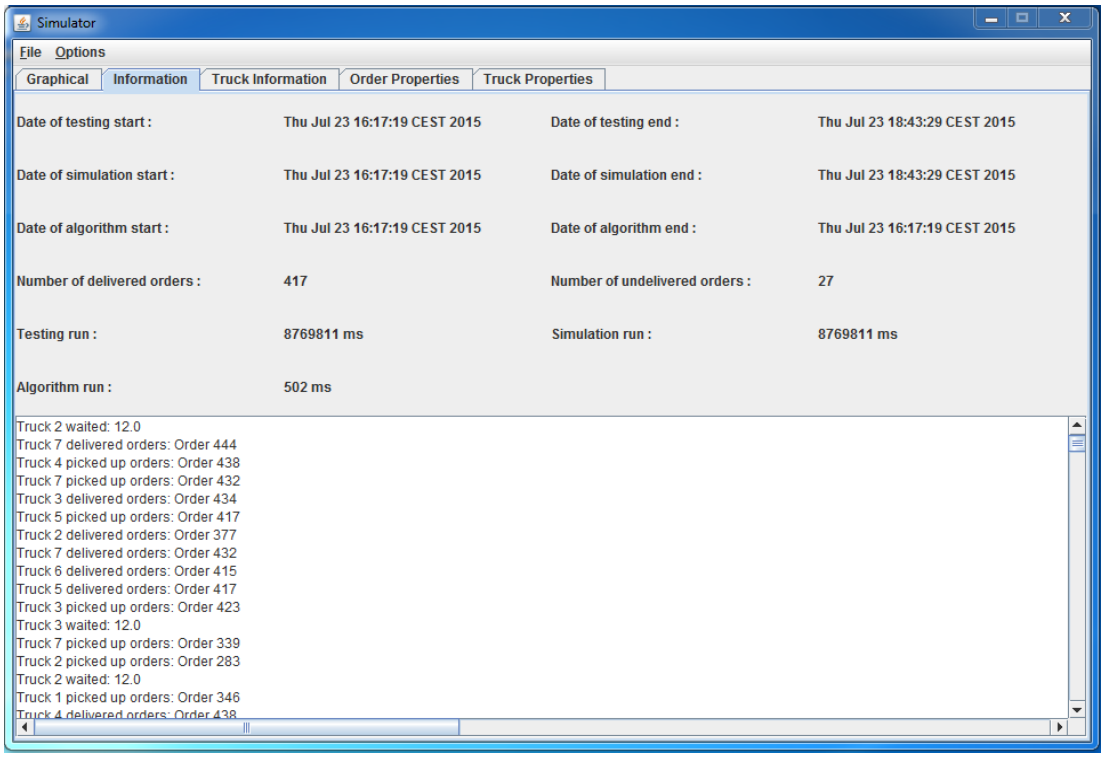
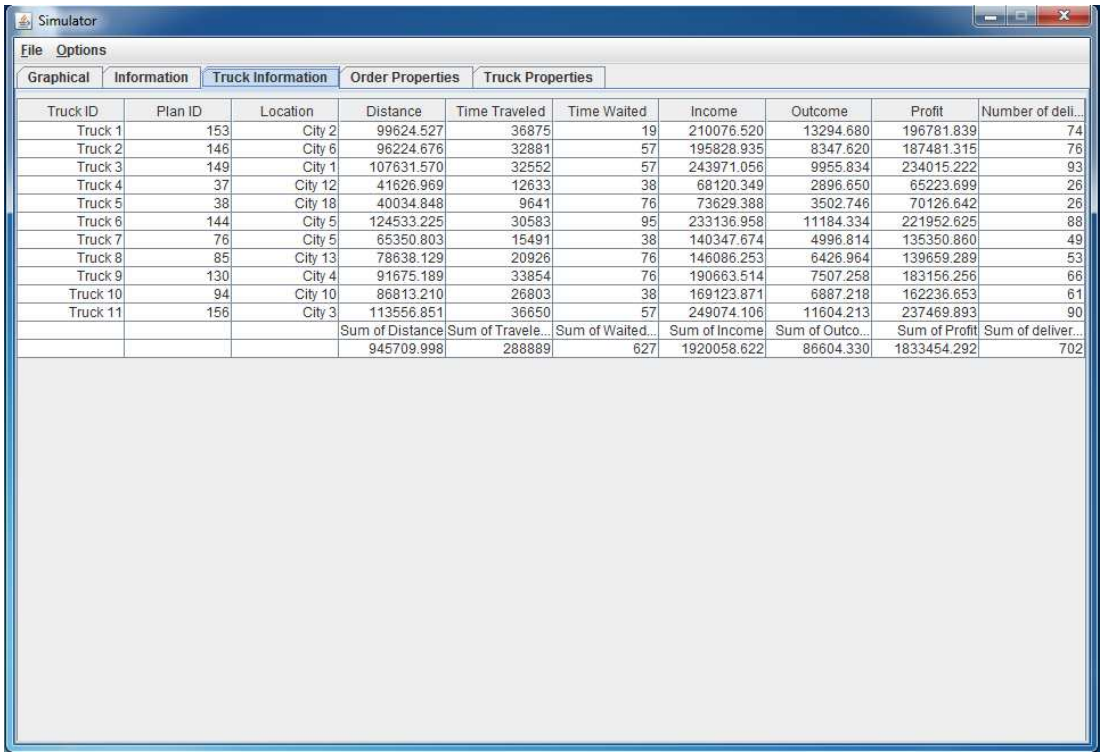
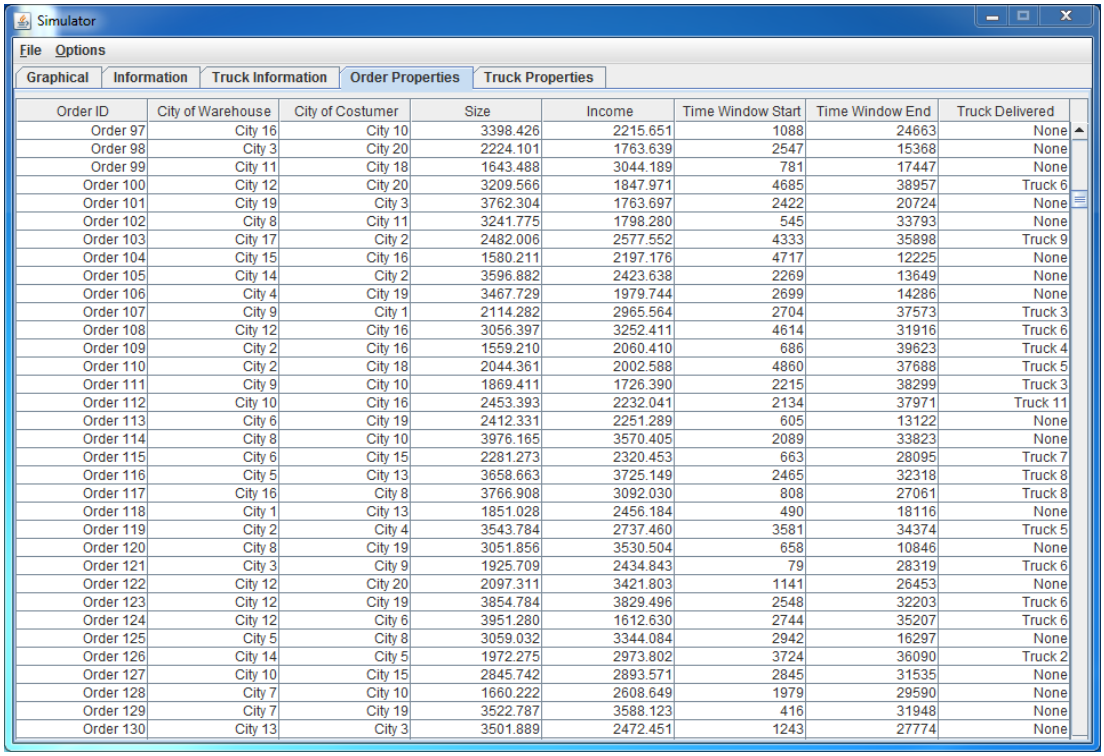


Figure 5.17: Information tab of Simulator application



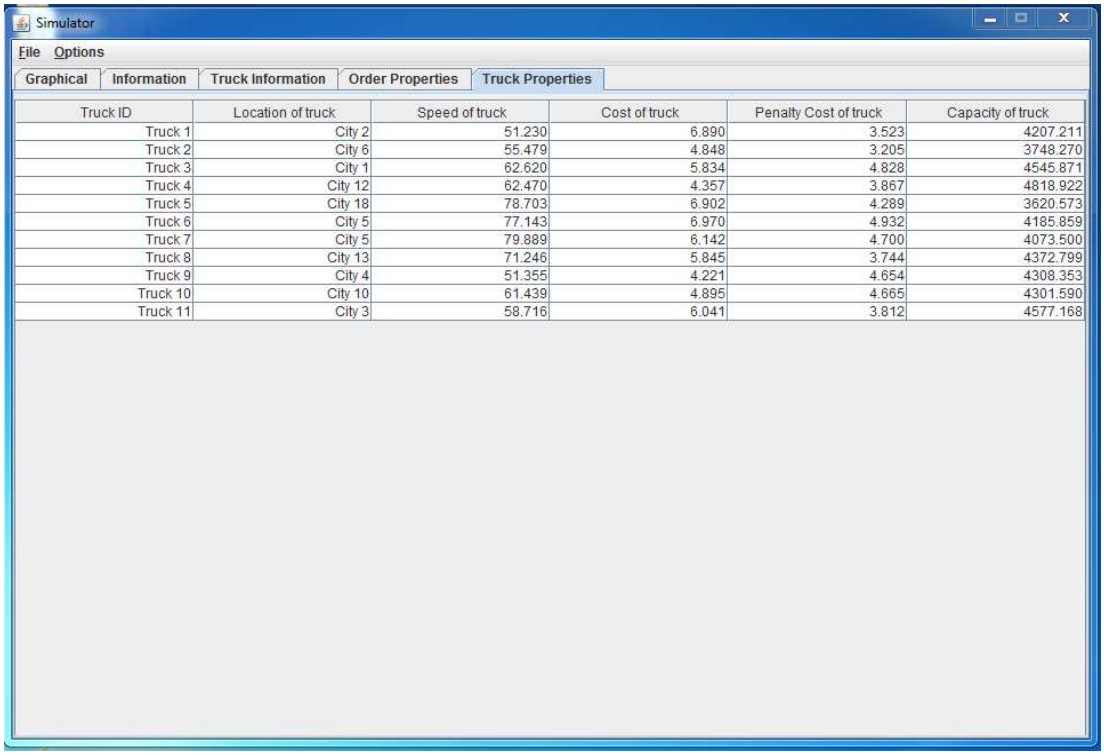
Truck ID	Plan ID	Location	Distance	Time Traveled	Time Waited	Income	Outcome	Profit	Number of deli...
Truck 1	153	City 2	99624.527	36875	19	210076.520	13294.680	196781.839	74
Truck 2	146	City 6	96224.676	32881	57	195828.935	8347.620	187481.315	76
Truck 3	149	City 1	107631.570	32552	57	243971.056	9955.834	234015.222	93
Truck 4	37	City 12	41626.969	12633	38	68120.349	2896.650	65223.699	26
Truck 5	38	City 18	40034.848	9641	76	73629.388	3502.746	70126.642	26
Truck 6	144	City 5	124533.225	30583	95	233136.958	11184.334	221952.625	88
Truck 7	76	City 5	65350.803	15491	38	140347.674	4996.814	135350.860	49
Truck 8	85	City 13	78638.129	20926	76	146086.253	6426.964	139659.289	53
Truck 9	130	City 4	91675.189	33854	76	190663.514	7507.258	183156.256	66
Truck 10	94	City 10	86813.210	26803	38	169123.871	6887.218	162236.653	61
Truck 11	156	City 3	113556.851	36650	57	249074.106	11604.213	237469.893	90
			Sum of Distance	Sum of Travele...	Sum of Waited...	Sum of Income	Sum of Outco...	Sum of Profit	Sum of deliver...
			945709.998	288889	627	1920058.622	86604.330	1833454.292	702

Figure 5.18: Truck information tab of Simulator application



Order ID	City of Warehouse	City of Costumer	Size	Income	Time Window Start	Time Window End	Truck Delivered
Order 97	City 16	City 10	3398.426	2215.651	1088	24663	None
Order 98	City 3	City 20	2224.101	1763.639	2547	15368	None
Order 99	City 11	City 18	1643.488	3044.189	781	17447	None
Order 100	City 12	City 20	3209.566	1847.971	4685	38957	Truck 6
Order 101	City 19	City 3	3762.304	1763.697	2422	20724	None
Order 102	City 8	City 11	3241.775	1798.280	545	33793	None
Order 103	City 17	City 2	2482.006	2577.552	4333	35898	Truck 9
Order 104	City 15	City 16	1580.211	2197.176	4717	12225	None
Order 105	City 14	City 2	3596.882	2423.638	2269	13649	None
Order 106	City 4	City 19	3467.729	1979.744	2699	14286	None
Order 107	City 9	City 1	2114.282	2965.564	2704	37573	Truck 3
Order 108	City 12	City 16	3056.397	3252.411	4614	31916	Truck 6
Order 109	City 2	City 16	1559.210	2060.410	686	39623	Truck 4
Order 110	City 2	City 18	2044.361	2002.588	4860	37688	Truck 5
Order 111	City 9	City 10	1869.411	1726.390	2215	38299	Truck 3
Order 112	City 10	City 16	2453.393	2232.041	2134	37971	Truck 11
Order 113	City 6	City 19	2412.331	2251.289	605	13122	None
Order 114	City 8	City 10	3976.165	3570.405	2089	33823	None
Order 115	City 6	City 15	2281.273	2320.453	663	28095	Truck 7
Order 116	City 5	City 13	3658.663	3725.149	2465	32318	Truck 8
Order 117	City 16	City 8	3766.908	3092.030	808	27061	Truck 8
Order 118	City 1	City 13	1851.028	2456.184	490	18116	None
Order 119	City 2	City 4	3543.784	2737.460	3581	34374	Truck 5
Order 120	City 8	City 19	3051.856	3530.504	658	10846	None
Order 121	City 3	City 9	1925.709	2434.843	79	28319	Truck 6
Order 122	City 12	City 20	2097.311	3421.803	1141	26453	None
Order 123	City 12	City 19	3854.784	3829.496	2548	32203	Truck 6
Order 124	City 12	City 6	3951.280	1612.630	2744	35207	Truck 6
Order 125	City 5	City 8	3059.032	3344.084	2942	16297	None
Order 126	City 14	City 5	1972.275	2973.802	3724	36090	Truck 2
Order 127	City 10	City 15	2845.742	2893.571	2845	31535	None
Order 128	City 7	City 10	1660.222	2608.649	1979	29590	None
Order 129	City 7	City 19	3522.787	3588.123	416	31948	None
Order 130	City 13	City 3	3501.889	2472.451	1243	27774	None

Figure 5.19: Order properties tab of Simulator application



Truck ID	Location of truck	Speed of truck	Cost of truck	Penalty Cost of truck	Capacity of truck
Truck 1	City 2	51.230	6.890	3.523	4207.211
Truck 2	City 6	55.479	4.848	3.205	3748.270
Truck 3	City 1	62.620	5.834	4.828	4545.871
Truck 4	City 12	62.470	4.357	3.867	4818.922
Truck 5	City 18	78.703	6.902	4.289	3620.573
Truck 6	City 5	77.143	6.970	4.932	4185.859
Truck 7	City 5	79.889	6.142	4.700	4073.500
Truck 8	City 13	71.246	5.845	3.744	4372.799
Truck 9	City 4	51.355	4.221	4.654	4308.353
Truck 10	City 10	61.439	4.895	4.665	4301.590
Truck 11	City 3	58.716	6.041	3.812	4577.168

Figure 5.20: Truck properties tab of Simulator application

Order Properties Tab

The fourth tab that we describe is the **Order Properties Tab** (see Figure 5.19). In this tab the application lists the basic properties of each order in a table. The

table also contains an extra column that shows for each order the truck that delivered the order.

Truck Properties Tab

The last tab that we describe is the **Truck Properties Tab** (see Figure 5.20). In this tab the application lists the basic properties of each truck in a table.

5.4 Programmer Guide of Simulator Application

Our Simulator application consists of six modules. These modules are **Main Module, Input Module, Computation Module, Graphical Module, Information Module** and **Output Module**.

Main Module

The first module that we present is the **Main Module**. The duty of this module is to keep the **Graphical Tab, Information Tab, Truck Information Tab, Order Properties Tab** and **Truck Properties Tab** content in consistency. This module is represented with the **SimulatorDatabase** class, which is located in the **SimulatorDataStructures** package.

Input Module

The second module that we present is the **Input Module**. This module processes the events of the **Open TWVRP** menu item. This module loads the TWVRP model instances to the database of the Simulator application. The duty of this module is to load TWVRP model instances and keep **Graphical Tab, Information Tab, Truck Information Tab, Order Properties Tab, Truck Properties Tab** and **Computation Module** content in consistency by calling the update functions of **Main Module**. The implementation of this module is located in the **SimulatorMenu** package.

Computation Module

The third module that we present is the **Computation Module**. This module processes the events of the **Start Simulation** menu item. After we have selected a scheduling algorithm the module tests whether the selected algorithm is a descendant of the **Algorithm** class and implements the methods of the **IAlgorithm** interface. In case testing was successful the module loads the class of the selected scheduling algorithm with the functions of the **Reflection API** (`https://docs.oracle.com/javase/tutorial/reflect/`) and creates two threads. The first thread is the **Algorithm** thread and the second thread is the **Simulator** thread. Between these two threads is a **Producer-Consumer** relationship. The **Algorithm** thread creates plans for each truck and the **Simulator** thread simulates the created plans.

The **Simulator** thread divides the plans created by the **Algorithm** thread into two groups. The first group contains plans that are currently simulated and the second group contains plans that are waiting for simulation. In each step of the simulation the **Simulator** thread moves a step forward in each plan of the first group. For each plan of the first group the **Simulator** thread creates a state, packs these states into a **Simulator event** and sends this event to the tabs of Simulator application. In case the plan of truck from the first group is finished, the **Simulator** thread removes it from the first group of plans and searches a new plan from the second group for the truck. If a new plan is found for the truck the thread removes this plan from the second group of plans and adds it to the first group of plans. The implementation of this module is located in the **Main** package.

Graphical Module

The fourth module that we present is the **Graphical Module**. This module processes the events of the **Computation Module**. When the module processes an event it recalculates the position of each truck in the **Graphical Tab** and repaints this tab. The implementation of this module is located in the **SimulatorTabs** package.

Information Module

The fifth module that we present is the **Information Module**. This module processes the events of the **Computation Module**. When the module processes an event it updates the table of **Truck Information Tab** and **Order Properties Tab**. The implementation of this module is located in the **SimulatorTabs** package.

Output Module

The last module that we present is the **Output Module**. The duty of this module is to save the content of the **Information Tab, Truck Information Tab, Order Properties Tab** and **Truck Properties Tab** to text file when the simulation has ended. The implementation of this module is located in the **SimulatorTabs** package. On figure 5.21 we can see the communication between the modules of the Simulator application.

Requirements of Scheduling Algorithm Plugins

When we would like to develop a new scheduling algorithm plugin, which we can add to the **Simulator** application, then our plugin needs to meet three requirements. The first requirement is that the class of the scheduling algorithm must be written in Java. The second requirement is that the class of the scheduling algorithm must extend the **Algorithm** class. The last requirement is that the class of the scheduling algorithm must implement the methods of the **IAlgorithm** interface. On figure 5.22 we can see the inheritance of Algorithm class.

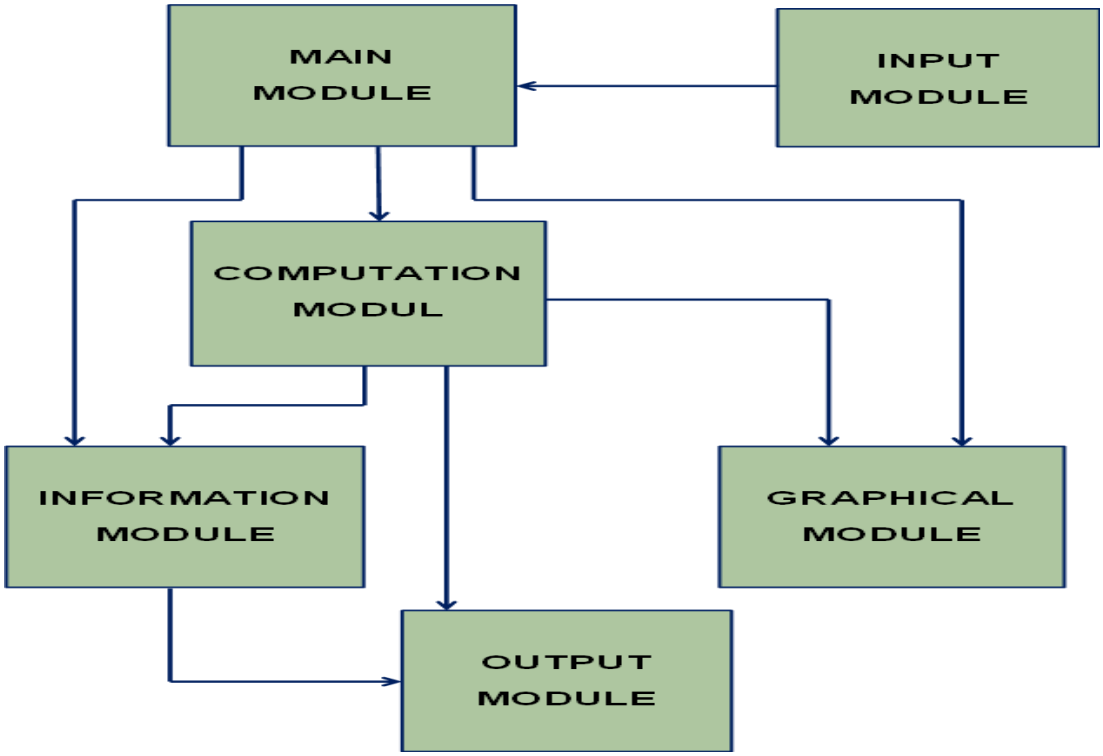


Figure 5.21: Communication among the six modules of Simulator application

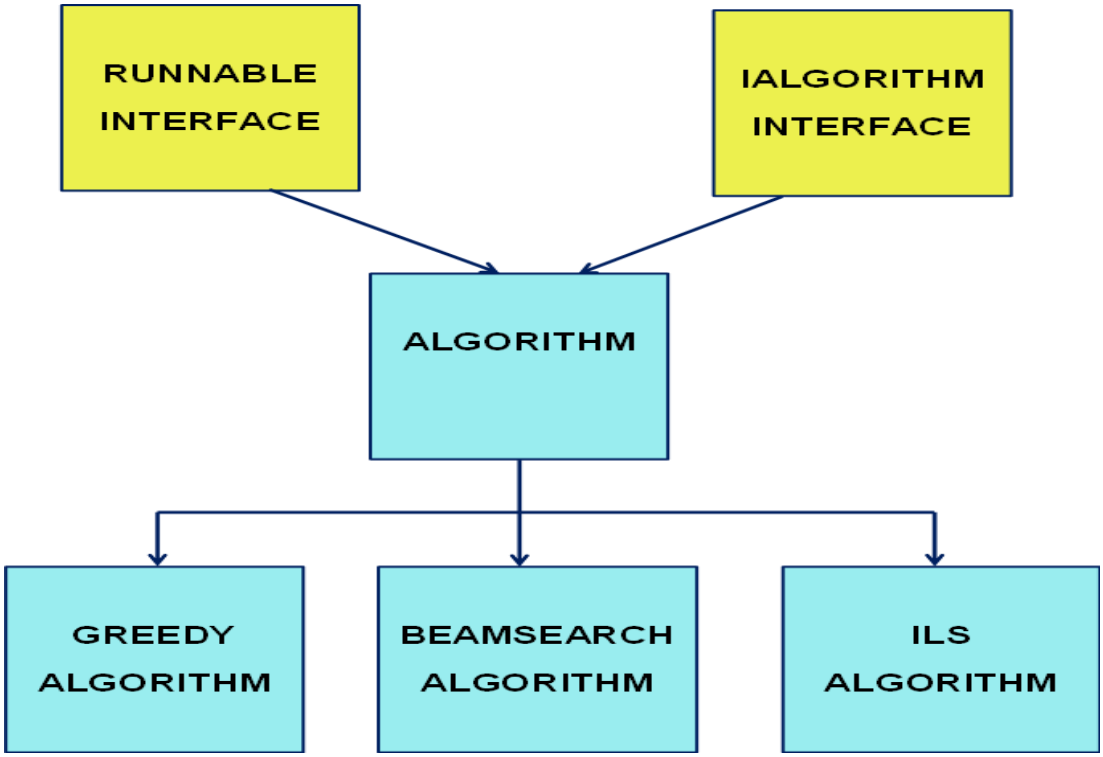


Figure 5.22: Inheritance of Algorithm class

6. Testing

In this chapter we describe how we tested the scheduling algorithms. We also formulate a hypothesis and test it with the Wilcoxon signed rank test [15].

Test Results

Before delving into the details of testing our program and its results, let us formulate the following hypothesis: *"The greedy algorithm is better then the Beam Search algorithm on TWVRP model instances, which have less than 900 orders."* In our context *"better"* means that the first algorithm attains larger profit than the second algorithm.

We tested the scheduling algorithms on ten different TWVRP model instances. Six of them are sparse graphs and the other four instances are dense graphs. The files of these tests can be found on the CD in **test/simulator** directory. We run the scheduling algorithms thrice on each TWVRP model instance. Each algorithm has two tables. The first table presents the results on sparse graphs and the second table presents the results on dense graphs.

Each of these six tables consists of seven columns. The first column contains the name of test files, the second column contains the distance that the trucks travelled, the third column contains the number of orders, the fourth column contains the number of delivered orders, the sixth column contains the running time of the simulation in milliseconds and the last column contains the running time of the algorithm in milliseconds.

File name	Distance	Profit	Orders	Delivered	Simulation	Algorithm
text1.txt	622834	306486	451	449	97149	941
text1.txt	628641	300774	451	442	89043	527
text1.txt	599781	287510	451	424	106016	449
text2.txt	682084	284228	444	426	106166	379
text2.txt	679658	282555	444	423	101702	404
text2.txt	683962	278040	444	414	101448	333
text3.txt	560509	230263	365	329	71550	280
text3.txt	551632	220907	365	318	77714	430
text3.txt	532473	222282	365	316	74404	267
text4.txt	889367	1193032	818	405	41143	229
text4.txt	887693	1185501	818	401	41459	206
text4.txt	859741	1179341	818	404	40877	207
text5.txt	872349	1312695	872	446	52203	241
text5.txt	890938	1380547	872	465	52639	258
text5.txt	849149	1246566	872	423	53636	252
text6.txt	857296	1127484	866	385	47587	203
text6.txt	913262	1211658	866	413	48322	215
text6.txt	890889	1250022	866	423	48095	423

Table 6.1: Testing results of the Beam Search algorithm on sparse graphs

File name	Distance	Profit	Orders	Delivered	Simulation	Algorithm
text7.txt	811223	1624414	1144	608	47724	331
text7.txt	789710	1586082	1144	595	47534	329
text7.txt	762316	1486230	1144	557	47289	339
text8.txt	941887	1942705	946	734	85758	493
text8.txt	894896	1852995	946	698	94586	482
text8.txt	856393	1793523	946	679	96558	392
text9.txt	945150	1855748	1121	706	87503	437
text9.txt	888954	1734735	1121	659	83062	331
text9.txt	1017132	1993336	1121	764	84666	390
text10.txt	740694	1426468	802	543	42420	351
text10.txt	768507	1491813	802	566	43656	437
text10.txt	728116	1380845	802	528	43564	311

Table 6.2: Testing results of the Beam Search algorithm on dense graphs

File name	Distance	Profit	Orders	Delivered	Simulation	Algorithm
text1.txt	646540	304346	451	451	65336	94
text1.txt	646540	304346	451	451	65239	10
text1.txt	646540	304346	451	451	65234	13
text2.txt	764279	299686	444	444	100915	10
text2.txt	764279	299686	444	444	100883	18
text2.txt	764279	299686	444	444	100876	11
text3.txt	657715	255487	365	365	68279	10
text3.txt	657715	255487	365	365	68276	12
text3.txt	657715	255487	365	365	68301	9
text4.txt	927705	1325633	818	442	42172	41
text4.txt	927705	1325633	818	442	42144	24
text4.txt	927705	1325633	818	442	42114	26
text5.txt	1009464	1559120	872	529	55532	40
text5.txt	1009464	1559120	872	529	55267	25
text5.txt	1009464	1559120	872	529	55308	16
text6.txt	1029734	1407217	866	479	48332	26
text6.txt	1029734	1407217	866	479	48276	19
text6.txt	1029734	1407217	866	479	48250	21

Table 6.3: Testing results of the Greedy algorithm on sparse graphs

File name	Distance	Profit	Orders	Delivered	Simulation	Algorithm
text7.txt	895033	1861068	1144	694	45736	29
text7.txt	895033	1861068	1144	694	45732	27
text7.txt	895033	1861068	1144	694	45732	34
text8.txt	1269356	2488134	946	946	91273	25
text8.txt	1269356	2488134	946	946	91104	19
text8.txt	1269356	2488134	946	946	91085	33
text9.txt	1258753	2440170	1121	939	84607	27
text9.txt	1258753	2440170	1121	939	84593	18
text9.txt	1258753	2440170	1121	939	84657	23
text10.txt	1000158	1921156	802	728	41924	21
text10.txt	1000158	1921156	802	728	41948	17
text10.txt	1000158	1921156	802	728	41946	16

Table 6.4: Testing results of the Greedy algorithm on dense graphs

File name	Distance	Profit	Orders	Delivered	Simulation	Algorithm
text1.txt	404786	314392	451	440	43474	5719
text1.txt	404786	314392	451	440	43092	5210
text1.txt	404786	314392	451	440	43657	5894
text2.txt	451812	308649	444	433	53118	5673
text2.txt	451812	308649	444	433	53081	5813
text2.txt	451812	308649	444	433	53004	5464
text3.txt	406945	266325	365	355	46726	3187
text3.txt	406945	266325	365	355	46595	2988
text3.txt	406945	266325	365	355	46869	3124
text4.txt	1070881	2435125	818	800	63269	29434
text4.txt	1070881	2435125	818	800	60192	21158
text4.txt	1070881	2435125	818	800	61775	23952
text5.txt	986446	2610378	872	853	66184	32066
text5.txt	986446	2610378	872	853	59732	20150
text5.txt	986446	2610378	872	853	57680	18183
text6.txt	1135110	2550504	866	838	69375	18016
text6.txt	1135110	2550504	866	838	74681	31051
text6.txt	1135110	2550504	866	838	70392	19126

Table 6.5: Testing results of the ILS algorithm on sparse graphs

We tested our hypothesis with the two-tailed Wilcoxon signed rank test, see table 6.7. From the table we can see that the sum of negative ranks is smaller than the sum of positive ranks. In this case the critical value W is equal to the absolute value of the sum of negative ranks. The critical value of W for $N = 21$ at $p \leq 0.01$ is 42. Therefore, the result is significant at $p \leq 0.01$. The Wilcoxon test has shown that we cannot reject our hypothesis.

File name	Distance	Profit	Orders	Delivered	Simulation	Algorithm
text7.txt	895829	3044799	1144	1117	81035	53280
text7.txt	895829	3044799	1144	1117	94721	71567
text7.txt	895829	3044799	1144	1117	85822	63068
text8.txt	731348	2475360	946	924	67055	27245
text8.txt	731348	2475360	946	924	70547	33429
text8.txt	731348	2475360	946	924	67416	27975
text9.txt	883306	2903540	1121	1089	87543	44684
text9.txt	883306	2903540	1121	1089	100926	63982
text9.txt	883306	2903540	1121	1089	100745	62672
text10.txt	626178	2072310	802	772	41019	23345
text10.txt	626178	2072310	802	772	33832	14057
text10.txt	626178	2072310	802	772	33954	13896

Table 6.6: Testing results of the ILS algorithm on dense graphs

Profit 1	Profit 2	Sign	Abs	Rank	Signed Rank
304346	306486	-1	2140	1	-1
304346	300774	1	3572	2	2
304346	287510	1	16836	4	4
299686	284228	1	15458	3	3
299686	282555	1	17131	5	5
299686	278040	1	21646	6	6
255487	230263	1	25224	7	7
255487	220907	1	34580	9	9
255487	222282	1	33205	8	8
1325633	1193032	1	132601	10	10
1325633	1185501	1	140132	11	11
1325633	1179341	1	146292	12	12
1559120	1312695	1	246425	16	16
1559120	1380547	1	178573	14	14
1559120	1246566	1	312554	18	18
1407217	1127484	1	279733	17	17
1407217	1211658	1	195559	15	15
1407217	1250022	1	157195	13	13
1921156	1426468	1	494688	20	20
1921156	1491813	1	429343	19	19
1921156	1380845	1	540311	21	21

Table 6.7: Wilcoxon signed rank test table of hypothesis

Conclusion

The goals of the thesis were to create a system which enables to run scheduling algorithms on TWVRP model instances, to measure the properties of scheduling algorithms and implement scheduling algorithms.

The first and second goal was achieved by creating the Simulator application. This application enables to run scheduling algorithms on TWVRP model instances. The application also enables to measure the running time of the algorithm, the running time of the simulation, the distances that that the trucks travelled, the time that the trucks travelled, the time that the trucks waited, the income that the trucks attained, the expenses that the trucks spent, the profit that the trucks made and the number of orders that the trucks delivered.

The third goal was achieved by implementing three scheduling algorithms. Each of them is from a different algorithm family. We tested these three algorithms on different TWVRP model instances.

We created the Editor application to simplify the creation and generation of TWVRP model instances.

6.1 Future Development

There are several possible ways to extend the project. First of all, more scheduling algorithms should be implemented and tested with the Simulator application.

Secondly the visualization of graphs should be improved in the Simulator application. The Simulator should enable editing of graphs in the Graphical Tab.

Another possible way to improve the project is to add visualization of algorithms to the Simulator application.

Bibliography

[1] M. Christopher, *Logistics & Supply Chain Managment*, Fourth Edition, Prentice Hall, 2011.

[2] M. R. Garey and D. S. Johnson. *Computers and Intractability A Guide to the Theory of NP-Completness*, W. H. Freeman & Co., New York, NY, USA, 1990.

[3] H. T. Cormen, E. CH. Leiserson, L R. Rivest and C. Stein, *Introduction to Algorithms*, Third Edition, The MIT Press Cambridge, Massachusetts London, England.

[4] F. Rossi, P. van Beek and T. Walsh, *Handbook of Constraint Programming.*

[5] M. Paolucci, *Vehicle Routing Problems*
http://www.discovery.dist.unige.it/didattica/LS/VRP.pdf

[6] J. Matoušek and J. Nešetřil , *Invitation to Discrete Mathematics*, Second Edition, Clarendon Press, 1998.

[7] M. O. Ball, T. L. Magnanti, C. L. Monma and G. L. Nemhauser, *Handbooks in Operations Research and Management Science, Vol. 8: Network*, Elsevier, Amsterdam, NL, 1995.

[8] R. Barták, *On-Line Guide to Constraint Programming*, http://kti.ms.mff.cuni.cz/ bartak/constraints/, 1998.

[9] A. Schrijver, *Theory of linear and integer programming*, John Wiley, 1986.

[10] M. Hifi, H. Akeb and A. Bouchakhchoukha, *A Beam Search Based Algorithm for the Capacitated Vehicle Routing Problem with Time Windows* Proceedings of the 2013 Federated Conference on Computer Science and Information Systems.

[11] P. Vansteenwegena, W. Souffriaua, G. V. Bergheb and D. Van Oudheusdena, *Iterated local search for the team orienteering problem with time windows* Computers & Operations Research, Elsevier, 2009.

[12] T. Fruchterman and E.Reingold, *Graph Drawing by Force-directed Placement*, John Wiley & Sons, 1991.

[13] T. Kamada and S. Kawai, *An algorithm for drawing general undirected graphs* Inform. Process. Lett., 1989.

[14] C. Walshaw, *A multilevel algorithm for force-directed graph drawing*, Journal of Graph Algorithms and Applications, 2003.

[15] F. Wilcoxon, *Individual Comparisons by Ranking Methods*, Biometrics Bulletin, Vol. 1, No. 6, 1945.

List of Abbreviations

VRP - Vehicle Routing Problem
LP - Linear Programming
CSP - Constraint Satisfaction Problem
AI - Artificial Intelligence
TSP - Travelling-Salesman Problem
NRP - Node Routing Problem
ARP - Arc Routing Problem
MTSP - Multiple Travelling-Salesman Problem
CVRP - Capacitated Vehicle Routing Problem
DCVRP - Distance Constrained Vehicle Routing Problem
VRPB - Vehicle Routing Problem with Backhaul
LC - Linehaul Customers
BC - Backhaul Customers
TWVRP - Vehicle Routing Problem with Time Windows
VRPPD - Vehicle Routing Problem with Pickup and Delivery
CP - Constraint Programming
ILP - Integer Linear Programming
ILS - Iterated Local Search
JRE - Java Runtime Environment
JUNG - Java Universal Network/Graph Framework

A. Content of the Attached CD

The attached CD contains the following data:

- proj/ netbeans project of Simulator and Editor application
- proj/Editor/dist contains Editor.jar
- proj/Simulator/dist contains Simulator.jar
- doc/ Javadoc generated for the Simulator and Editor application
- test/ test data for the Simulator and Editor application
- src/ source files of Simulator and Editor application
- lib/ contains libraries of JUNG
- thesis.pdf - this document

B. Editor Application Spinner Intervals

In the next table we present the default intervals of spinners in Generator Tab, Modified Tab and Tester Tab. The table contains four columns. The first column contains the names of intervals, the second column contains the minimum number of intervals, the third column contains the maximum value of intervals and the last column contains the value that increases or decreases the value of the spinner when we click on it.

Name of interval	**Min**	**Max**	**Step**
number of cities	0	100	1
number of roads	0	10000	1
road length	0.001	1000	0.001
number of trucks	0	100	1
truck speed	0.1	100	0.1
truck cost	0.01	100	0.01
truck penalty cost	0.01	100	0.01
truck capacity	0.001	50000	0.001
number of orders	0	10000	1
order size	0.001	50000	0.001
order income	0.01	1000000	0.01
order time window start	0	9223372036854775807	1
order time window end	0	9223372036854775807	1
time unit	0	9223372036854775807	1
test number	0	100	1

Table B.1: Spinner intervals