AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ

Praca dyplomowa

Fleet Structure Optimization in Vehicle Routing Problems
Optymalizacja struktury floty w problemach marszrutyzacji

Autor: *Bohdan Forostianyi*
Kierunek studiów: *Informatyka*
Opiekun pracy: *dr inż. Krzysztof Kluza*

Kraków, 2022

Serdecznie dziękuję mgr inż. Mateuszowi Ślażyńskiemu za czas i wysiłek,który wkłada w prowadzenie dydaktyki na naszym Wydziale. Jego starania zawsze budziły i budzą we mnie chęć do poznawania nowych rzeczy i rozwoju. Poza tym chciałbym podziękować mu za propozycję tematu oraz za pomoc przy pracy nad projektem dyplomowym. Dziękuję również dr inż Krzysztofu Kluzie, za objęcie pracy opieką. Osobne podziękowania należą też do Alicji Krzemińskiej, Jakuba Turaja, Pawła Renca i Tomasza Pęcaka – kolegów i przyjaciół z kół naukowych, którzy zawsze byli dla mnie wsparciem i pomocą.

Contents

1. Introduction

1.1. Related Problems

Vehicle Routing Problems (abbreviated as VRP) and their numerous variants are common challenges in logistics. Defined more than 40 years ago these problems consist in designing an optimal set of routes to serve all necessary points given a predefined fleet of vehicles and problem constraints. The interest in VRP is motivated by its practical relevance and complexity.

One of the reasons behind mentioned complexity is a number of related tasks, that are going beyond the scope of the classical problem formulation, such a necessity for initial fleet selection. This problem is called is called Fleet Structure Optimization Problem and arises in many areas such as:

- supply chain management for companies that use services of external carriers instead of maintaining their vehicle fleets. Such companies have to sign contracts with carriers taking under consideration available budget and existing demands.

- emergency medical fleet allocation, where the key factor in the performance of an EMS is the speed at which emergency vehicles can respond to incidents. It is vital to increase the survival rate among patients that at all times emergency vehicles are located so as to ensure adequate coverage and rapid response times [1].

- humanitarian operations, where the goal is to determine the appropriate number of vehicles to support operations in a country. Humanitarian relief and development activities mostly take place in areas that are difficult to reach, because road infrastructure is poorly developed or has been destroyed. Because of mission importance and high costs, humanitarian organizations need to carefully manage their vehicle fleets [2].

This thesis addresses a Fleet Structure Optimization problem for a case of supply chain management, outlined in the above list. There are a number of approaches to solving this type of problem including, but not limited to linear regression models [2], multi phase optimization algorithms [3], optimization algorithms with underlying numerical stochastic algorithms [4].

1.2. Thesis Outline

The goal is to purpose and test a new approach based on reformulating a problem as a Rich VRP instance with heterogeneous fleet and time and capacity constraints. In chapter 2 the theory needed for purposed method understanding is outlined, chapter 3 describes details and requirements of a reference problem used for testing, in chapter 4 approach and baseline models are presented. In chapter 5 experiments are done and their results are discussed. Chapter 6 summarizes the work done and future improvements are considered.

2. Background

2.1. Vehicle Routing Problem

Vehicle Routing Problem, or VRP for short, is a common optimization task where one has a set of vehicles, often called a fleet and a set of points that should be visited given this fleet. Despite such an easy formulation, the problem belongs to the NP-hard complexity class and is an active research topic in Operational Research and Logistics [5, 6]. The goal of this section is to introduce notation used in the problem and discuss existing VRP variations.

A classical VRP instance is as an undirected graph $G(N, A)$, where $N = \{0, 1, \ldots, n\}$ is a set of existing **nodes** and A is a set of arcs representing all possible routes between these nodes. Normally, there always exists a **point of interest** $p_i \in P$, that should be visited multiple times. These visits are represented by nodes $A = \{n \mid n \in N\}$. An example point of interest, common in most VRP problems, is depot – a point where route of every vehicle $v_i \in V$ starts and ends.

Each vehicle v_i could be associated with a route $r_i \in R$ which is defined as a set of nodes $\{n_j \mid n_j \in N \wedge visited\_by(n_j, v_i)\}$ where function $visited\_by$ is defined as $f : N \times V \rightarrow \{true, false\}$ and returns $true$ in case if node n_j was visited by vehicle v_i and $false$ otherwise.

The goal of a VRP is to find a set R, called **assignment**, which will have the smallest value of an objective function $O : R \rightarrow \mathbb{R}$. Classically the distance travelled by all vehicles during theirs routes or total usage cost are set as an objective function to minimise.

There are many variations of VRP, each adding some complications to the model introduced above. The most common are described in the 2.1.

Abbreviation	Full name	Modifications	References
CVRP	Capacitated VRP	Adds capacities to the vehicles that constrain how many goods could be transported by a vehicle. The objective is to satisfy demands from all nodes appearing in the problem.	[7, 8]

VRPTW	VRP with Time Windows	Adds time windows to each node defining when this node should be visited by a vehicle.	[8, 9, 10, 11, 12]
HVRP	VRP with heterogeneous fleet	Groups vehicles into types according their various features such as travel speed or capacity. Often appears in the combination with one of the above types.	[8, 13, 14]
MVRP	Multidepot VRP	Adds multiple points of interest where vehicle could start or end its route.	[8, 15, 16]
DVRP	Dynamic VRP	Number of nodes can be changed during a plan execution phase.	[17, 18, 19, 20]
PDVRP	Pickup and Delivery VRP	Variation of CVRP, where vehicles could collect goods and drop not only in depots, but also in nodes during their route.	[10, 21, 22]
OVRP	Open VRP	Variation of a VRP where vehicles are not obliged to return to the depots.	[23, 12]
RVRP	Rich VRP	Rich VRP is a common name for the real-world instances of VRP, where multiple constraints from introduced previously are combined.	[8, 24, 25]

Table 2.1. Most common VRP types

2.2. Approaches for Solving VRP

There are many different approaches to solve optimization problems from VRP family, but this work refers only three of them: constraint programming, mathematical programming and a stochastic search guided by some heuristic strategy.

First two methods share a similar idea of describing a problem with the constraints and later solving it with an appropriate solver. The main difference is a way in which solvers works, which have a direct influence on available constraints and optimization speed.

In case of Constraint Programming solving is done by performing search with a constraint propagation mechanism over a graph of possible states. This approach is described in details in

subsection 2.3.2 and its main benefit is a wide range of non linear relationships that could be added to the model [26].

In mathematical programming instead of constraint propagation a sequence of math operations is performed. It makes this approach faster than constraint programming, but requires more effort from the modeler because all constraints should be modeled as a system of linear equations and inequalities [27].

Third method is based on searching through a set of possible states, generated by a function defined by the modeler, until global minimum or time limit will be achieved. [28].

All approaches have their advantages and disadvantages and are widely used in industry. However, one of the most promising approaches is a hybrid approach that combines constraint programming and metaheursitic optimization, so it was also used in this thesis and is briefly described in the following section.

2.3. Constraint Programming with Metaheuristic Optimization

Solving a problem with this approach comes down to three tasks:

1. model a problem in the terms of existing constraints,
2. find an assignment, that satisfies defined constraints,
3. use a metaheuristic algorithm to optimize an initial solution.

This section guides reader through the solving process on the toy example of a Traveling Salesman Problem, which is similar to VRP problem, but limited to a single vehicle so works better for a demonstration. The objective of the problem is to find such a route through the nodes $1, 2, 3, 4, 5$ such that the distance travelled by vehicle starting from the node 0 and returning to the same node is minimal. Locations of points of interest that should be visited are shown on the figure 2.1

2.3.1. Modeling

In a modeling phase, the goal is to select decision variables and to define model constraints. Decision variables are variables that correspond to information which one who solves the problem wants to obtain. In the case of our toy problem, there will be 7 decision variables. They will correspond to the order in which nodes will be visited by vehicle so the domain of each variable will range from 0 to 7. Model constraints are tools used by modeler to provide an information

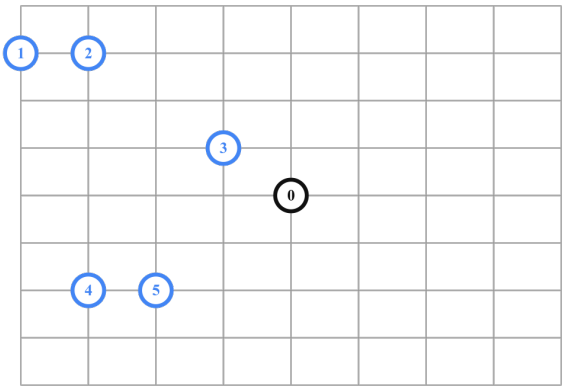

Fig. 2.1. Points to visit in a toy problem

about a problem to the solver in order to allow it to find a correct assignment. Crucial constraints in this case are constraints that define start and end nodes and constraint that require the vehicle to visit every node in a problem. An example model in MiniZinc [29] with this constraints defined is shown in the listing 2.1.

Listing 2.1. Toy problem model in MiniZinc

```
set of int: VISIT_ORDER = 0 .. 6;
array[0 .. 6] of var VISIT_ORDER: visits;
% 0 and 6 visits corresponds to a depot node
constraint visits[0] = 0;
constraint visits[6] = 6;
alldifferent(visits);
```

2.3.2. Feasible Assignment Search

The above model description is used by the solver to generate a state space – a graph with vertices corresponding to a variable assignments and edges corresponding to a mutations of some variable value. The goal of the search process is to build up a feasible solution, so it should validate every transition it makes. In order to achieve this, the search process consults a Constraint Store – an instance responsible for storing all constraints and variable domains. Steps done by the solver in each vertex could be represented with the following pseudocode.

Listing 2.2. Search phase

```
1 for transition in possible_transitions_from(assignment):
2     if constraint_store.is_transition_breaks_constraints(transition) :
3       continue
4     else:
```

```
5         assignment = make_transition(transition)
6         constraint_store.propagate_constraints(assignment)
```

As it can be seen in the listing, when transition is valid, solver makes it and calls a Constraint Store, which propagates other constraints based on the current assignment. This step is called Constraint Propagation and is the crucial part of every constraint solver, which forms the basis of the Constraint Programming computational paradigm. The goal of the phase is to reduce the size of a search space. Each constraint has its logic that, given a current state, tells which states would make this constraint unsatisfiable. From the high perspective it could be described with the following pseudocode.

Listing 2.3. Constraint propagation

```
1 for constraint in model:
2     constraint.propagate_from_state(current_state)
```

The search ends when the solver finds an assignment that not satisfies all constraints defined in the model. For the toy example we consider, possible ends is demonstrated on the figure 2.2.

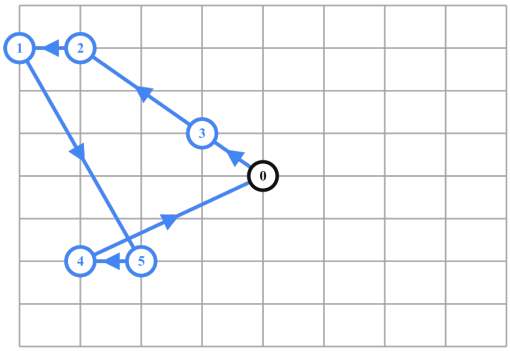

Fig. 2.2. Assignment after search described in 2.3.2

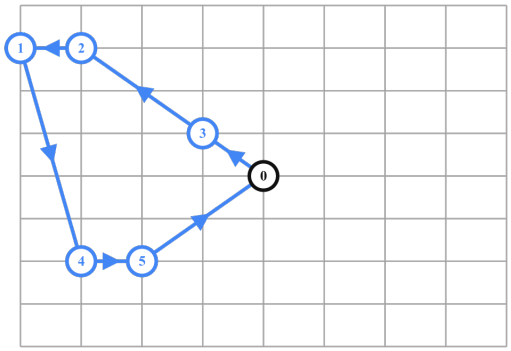

Fig. 2.3. Optimal assignment

2.3.3. Assignment Optimization

As it can be seen in the figure 2.2 solution is not optimal, because some of the routes are crossed. One of the possible ways of solving the issue is to use an optimization algorithm, such as Local Search. From a high-level perspective, the idea behind this algorithm is to perform small changes in the current assignment and accept those of them which are improve the solution. An example pseudocode is shown in the listing 2.4.

Listing 2.4. Hill climbing

```
for neighbour in generate_neighbourhood(assignment):
    if objective(neighbour) < objective(assignment):
      assignment = neighbour
    elif is_equlibrium():
      return optimal_assignment
```

This family of algorithms differs from the search algorithms used during the previous step in a way, that search algorithms have a defined goal that ends a search. Nevertheless, in the case of optimization process, it is harder to express it because the goal is to minimize an objective function, and one cannot tell, when the value is minimum, without checking all states in the search space.

The main reasons the main reason behind it, are shoulders and local optima that often appear in the objectives. Search process may get stuck in the local optimum, therefore ignoring the vast part of the search space. To avoid this situation, metaheuristic algorithms could be used. The employ various strategies guiding the search process to balance exploitation/exploration tradeoff.

While there exist many numerous metaheuristics [30] this thesis will focus on three of them, that integrate well with the Constraint Programming paradigm and are used in existing open source solvers (references to solvers are provided next to metaheuristics): Simulated Annealing [31, 32], Tabu Search [31, 33] and Guided Local Search [31].

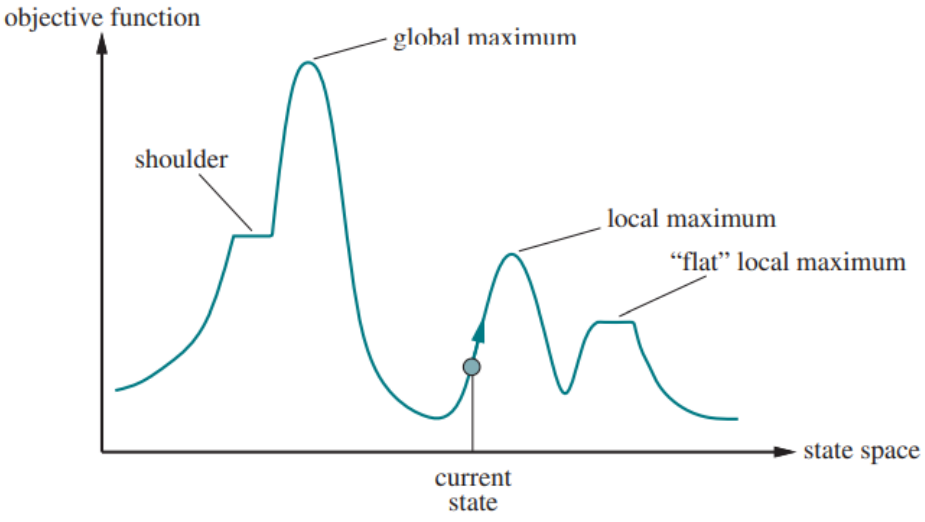

Fig. 2.4. A one-dimensional state-space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum [34]

2.3.3.1. Simulated Annealing

Simulated annealing is a randomized local search method, which can worsen its current state with some probability that is computed by the $schedule$ function. This approach allows the algorithm, to effectively escape local optima and move towards the best solution. It comes from an analogy with the physical process aimed to generate solids with low-energy states [30]. Simulated Annealing is an iterative algorithm, which on each iteration:

- computes objective function for state,
- calculates a difference in terms of the objective function between new neighbor and current state,
- always selects state if the state is improving otherwise the new state is selected with probability defined by $schedule$ function.

Pseudocode for the algorithm is provided in the listing 2.5. The main logic is performed in 6 and 14-16 lines. In 6 line, algorithm computes selection probability with a special function that depends on temperature, a parameter that initially have high value and declines over time. This function is called $schedule$ and returns high values if temperature is high and low values if temperature is low. In other words, algorithm has a property of selecting a worsening state with a high probability when the temperature is high and behave as a greedy algorithm when the temperature is low, which allows to escapes from local optima. To further decrease probability of getting stuck in the local optimum, the standard algorithm may be extended with reheating which increases a temperature and forces an exploration. Reheating is shown in lines 14-16 and it can be observed that it occures when an algorithm approaches its local optimum.

Listing 2.5. Simulated annealing pseudocode [35]

```
1  for neighbour in generate_neighbours_in_random_order(assignment):
2      improvement = calculate_improvement(assignment, neighbour)
3      if improvement > 0:
4          assignment = neighbour
5
6      elif exp(improvement / temperature) > random(0,1):
7          assignment = neighbour
8
9      if not is_equilibrium():
10         temperature = max(temperature * (cooling_step **
11                                 cooling_time), min_temperature)
12         cooling_time += 1
13      else:
```

```
14          best_find_assignment = assignment
15          temperature = initial_temperature
16          cooling_time = 1
17  return best_find_assignment
```

2.3.3.2. Tabu Search

Tabu search is a local search method, that applies a Tabu List as a metaheuristic to escape from local optimum. Tabu List is usually implemented as an in-memory data structure where n of previously visited states are stored [36]. All states stored in the Tabu List cannot be visited by solver. The main idea behind it is to force the search process to explore new states when it is stuck in a local optimum.

The size of the Tabu List may vary and heavily depends on a problem. It can be chosen experimentally or change dynamically during the search process [37]. Pseudocode for Tabu search is presented in the below listing:

Listing 2.6. Tabu Search [36]

```
1  curr_neighbour = assigment = initial_assigment
2  k = 1
3  while is_not_equilibrium():
4       neighbourhood = generate_neighbourhood(assigment, k)
5       curr_neighbour = select_best_neighbour(curr_neighbour, k)
6       if objective(curr_neighbour) < objective(assigment):
7         assigment = curr_neighbour
```

What differentiates this algorithm from others is function $generate\_neighbourhood$ that accepts state and step k for which it should generate neighborhood and is responsible for generating previously unvisited states based on the Tabu List. One of the most memory-efficient approaches is to store n last moves and to remove from the neighborhood all moves that can reverse them.

2.3.3.3. Guided Local Search

Guided local search, further referred to as GLS, is another metaheuristic that demonstrates a significant performance on several complex optimization problems and particularly on VRP [38].

Listing 2.7. Guided local search [38]

```
1  current_state = assignment = initial_assignment
2  # create a zero vector of size number_of_penalty_features
3  penalties = zeros(number_of_penalty_features)
4  while not is_equilibrium():
```

```
5       penalty_features = choose_penalty_features()
6       for feature_idx in penalty_features:
7          penalties[feature_idx] += 1
8       current_state = local_search(state, penalties)
9       if objective(current_state) < objective(assignment):
10           assignment = current_state
11  # final local search without penalites to get to true local minimum
12  assignment = local_search(assignment, zeros(number_of_penalty_features))
```

GLS dynamically modifies the objective function to penalize states belonging to the known local optima. This process requires a fine grained description of a state and its features from the modeller. Description will not be discussed as it is not the essense of the algorithm.

3. Problem Statement

3.1. Problem Outline

This thesis addresses problems of retailer companies, referenced later as Companies, maintaining physical stores and distribution centers located all over the country. Each distribution center is a specialized warehouse, where products are deposited before being redistributed by commercial vehicles to stores assigned to the center. Companies do not keep own fleet of vehicles and use services of the local carriers, referenced later as Carriers, to satisfy their needs. The business model looks in a way, that once a month each Company signs with the selected Carrier a contract, which obliges the Carrier to provide the Company with access to the specified number of vehicles for a fixed price.

3.2. Renting Details

Every Carrier offers multiple types of vehicles, which are different in their costs and capacities. With every rented vehicle a Carrier always provides a human driver. Work shift for a driver could not be longer than eight hours. Shift starts when the driver comes to a Company's distribution center and ends when the driver leaves it. Between every work shift there should be a break that lasts at least eight hours.

Apart from renting vehicles for a month on a contract base, as it was mentioned in section 3.1, Carriers also offer rents on demand. Vehicles rented in that way are referenced as spot vehicles and costs more on a per-day basis but are beneficial in case if a vehicle will not be used often.

3.3. Distribution Requirements Planning

Companies considered by this thesis perform distribution planning on a day to day basis. The planning is performed for every distribution center based on a data that was submitted

by stores assigned to the center during the previous day. Every store, that has any demands, submits a report where it specifies all goods requirements and a preferred time window for goods delivery.

3.4. Input Data

As it can be seen from sections 3.3 and 3.2 VRP that takes place in the problem is an instance of a Rich VRP with Capacity Constraints, Time Windows and Heterogeneous Fleet. It is an NP-hard problem, which means that with more data time required to find a solution grows exponentially. To decrease this unwelcome issue, the full planning period could be reduced into several commonly occurring patterns, called characteristic days. Pattern defines typical amount of visits, time windows and demands for the day.

An intuition behind this concept could be easily developed if consider demands that can have a grocery store in a month with a holiday. During month for such a store, there will be a similar demand for weekdays, another demand for weekends (because people spent more time at home) and a different demand for a holiday (because people often organise holiday dinners).

The method used to identify this pattern goes beyond the scope of this thesis and as such will not be discussed. For experiments characteristic days were created based on domain knowledge. Each of them corresponding to days with a high, low and normal demands.

3.5. Goal

Every Company has two conflicting goals: satisfy all logistic demands, which requires bigger fleet size and minimize fleet costs, which requires smaller. An approach used in this thesis to solve a problem is described in chapter 4 and is evaluated on data generated in accordance with section 3.4 from the Solomon Benchmark described in details in appendix B. Rental policy used for evaluation is demonstrated in table 3.1.

Vehicle type	Rental Cost per usage day		Cost per km	Capacity
	as regular	as spot		
Small	100	150	1	100
Medium	200	300	1	200
Large	300	500	2	300

Table 3.1. Vehicle offer

4. Method

This chapter describes a methodology used to address a Fleet Optimization Problem for conditions defined in 3.1. The proposed approach is based on reformulating a Fleet Structure Optimization Problem as a VRP problem and could be divided into two parts:

– predicting of the demands of goods for the contract period,

– finding an optimal fleet that satisfies predicted demands.

The thesis works with the second part, which relies on finding an optimal fleet that satisfies the VRP model described in 3.3 for predicted demands grouped into Characteristic Days desribed in 3.4.

An assumption is made, that the proposed approach will be able to predict fleet with the relative error between real and estimated fleet costs smaller than 10%. In order to get a better insight on an approach performance it will also be compared with two baseline models created in other computational paradigms and described in details in 4.2. Obtained results will be discussed in 5.3.

4.1. Proposed Model

As mentioned before the main idea proposed by this thesis is based on reformulating a Fleet Structure Optimization as a VRP problem. To reformulating a Fleet Structure Optimization as a VRP problem three things had to be done:

1. definition of the problem in terms of constraints from 3.1

2. implementation of logic for differentiation between regular and spot vehicles. Solver should be accounted that regular vehicles if added to the fleet once will also contribute to overall fleet costs in every other day in most,

3. selection of a metaheuristic algorithm that fits best to the defined problem constraints.

This section is split into two subsections, first describes how the first two points from the above list were addressed and the second describes the metaheuristic selection .

4.1.1. Model

The model was implemented in an open-source solver Google OR Tools [31] and the following constraints were added:

– each node could be visited only in the time window established for it,

– difference between vehicle cumulative variable [39] used to track satisfying demands and maximum vehicle capacity had to be less than node demand, which means a vehicle that visits node should have enough goods to satisfy this node demands,

– value of vehicle cumulative variable [39], used to track work shift duration, had to be less than eight hours,

– vehicle usage costs were calculated based on costs defined in table 3.1, and could not overcome some defined budget.

Also, all nodes in the problem were made optional but with a big penalty for dropping them, in order to speed up computations. This modification is possible because the problem objective is to find an optimal fleet and because planning was performed for characteristic days which are only approximations of VRP instances that will arise in practice.

In order to be able to differentiate spot and contract vehicles, a solver was adapted to accept on input multiple days instead of one, as usually done. To achieve this for each vehicle break nodes that corresponded to depot coordinates on a map (equal to the number of days in input). In the break node, every vehicle was able to reset values collected with cumulative constraints. Also, to avoid assure required breaks for drivers an interval constraint [40] was with a minimum duration of eight hours was added to every break node.

4.1.2. Metaheuristic Tuning

Google OR Tools [31] is a hybrid solver, which means that after feasible solution is found it performs an optimization phase. Optimization can be executed in a Greedy Descent mode or with one of metaheuristics algorithms: Simulated Annealing, Tabu Search and Guided Local Search (see 2 for details).

Solver performance strongly depends on selected approach, so in order to determine the most prominent one, for model used in this thesis multiple tests, described later in this section, were performed.

During tests the solver was faced with two different types of tasks:

– find an optimal route with a predefined fleet, for a different number of nodes in input,

– minimize used fleet given different initial configurations.

4.1.2.1. Optimal Route Test

In this test, metaheuristics were tested against different amounts of nodes in an input. An objective value was calculated as a sum of the costs of all used vehicles and the length of their routes. Results are shown in table 4.1. It can be noticed, that the Greedy Descent approach shows the best performance for 30 and 100 nodes, second-best metaheuristic after it is Tabu Search which outperforms all algorithms on 50 nodes demonstrating good results for tests on 30 and 100 nodes.

Nodes in model / Metaheuristic	30	50	100
Simulated annealing	26055	38306	114859
Tabu Search	25973	38320	114750
Guided Local Search	26020	39905	1114746
Greedy Descent	24493	39905	114699

Table 4.1. Results of Metaheuristics Testing on Different Day Amounts

4.1.2.2. Optimal Fleet Test

In this test metaheuristics were tested against two different initial setups:

– A – 20 medium vehicles

– B – 20 medium 10 small and 10 large vehicles

An objective value is calculated as a sum of the costs of all used vehicles. Results are shown in table 4.2. It can be noticed, that the Greedy Descent algorithm shows the best performance for configuration B and Tabu Search shows the best performance for configuration A.

4.1.2.3. Summary

Greedy Descent algorithm outperformed all metaheuristics in three of five test cases. Nevertheless, as a metaheuristic for the solver the Tabu Search was selected. The main reason for this decision is dominance which metaheuristic algorithms demonstrated over simple Local Search methods for other problems in this field.

Fleet configuration / Metaheuristic	A	B
Simulated annealing	399 650	433 240
Tabu Search	367 230	433 400
Guided Local Search	391 570	468 690
Greedy Descent	386 260	432 980

Table 4.2. Results of metaheuristics testing with different initial fleets. Columns correspond to configurations introduced in above list

4.2. Baseline Models

4.2.1. Linear Programming Model

This approach is based on modeling a problem as a system of linear inequalities and solving it with a MIP (Mixed Integer Programming) solver. The model applies to a relaxed version of an original formulation and considers only overall goods demand for the planning period, different types of vehicles, and different rental options, while omitting driver breaks and time windows for orders. Mathematical notation for model is as follows.

Let:

$T =$ all vehicle types that could be rented,

$A =$ amounts of rented vehicles for each vehicle type,

$V =$ available amounts of vehicle types: $A_i \leq V_i$,

$R =$ rental costs of vehicles,

$C =$ capacities of vehicles,

$N =$ duration of planing period,

$D =$ total goods demand for a planing period,

$j =$ number of vehicle types

$$min\ \Sigma_{i=0}^{j} A_i \cdot R_i \tag{4.1}$$

Objective 4.1 is to minimize total cost vehicle usage.

$$\Sigma_{i=0}^{j} A_i \cdot C_i \geq D \tag{4.2}$$

Constraint 4.2 states, that all amounts of items that could be transferred by vehicles should be at least as big as the total demand in the problem instance.

$$\forall_{i<j} A_i \leq V_i \tag{4.3}$$

Constraint 4.3 does not allow to use more vehicles of some type than it is possible to rent from the carrier.

4.2.2. SAT Model Enriched with Linear Constraints

This approach is similar to linear programming introduced above, but extends it with boolean constraints. This extension allows to add simple nonlinear details to the model and make it more detailed.

However even with this approach, it was hard to represent model constraints such as work shift duration or time windows. To imitate them for each day was set an upper bound limit on the number of nodes that vehicle could visit. In mathematical notation model can be introduced as follows.

Given:

$D =$ set of days in the model.,

For each day the following variables are defined and could be accessed with the notation $D_i[VariableName]$:

V = set of boolean variables indicating if the vehicle is used on the day D_i,

$V^r =$ subset of V, which contains only regular vehicles,

$C =$ set of vehicle capacities,

$H =$ set of vehicle rental costs,

$S =$ set of all constants indicating maximum amount of points that could be visited by vehicle,

$R =$ set of goods requests on day D_i,

$T =$ matrix of boolean variables with dimensions $N_R \times N_V$, which connects visit with a vehicle that served it.

$$min\ \Sigma_{i=0}\Sigma_{j=0} D_i \cdot V_j \cdot H_j \tag{4.4}$$

The objective 4.4 is to minimize the sum of all vehicle usage costs overall days.

$$\forall_{i,j}\Sigma_v D_i \cdot T_{jv} = 1 \tag{4.5}$$

Constraint 4.5 states that for every node could be visited by only one vehicle.

$$\forall i, v\Sigma_j D_i \cdot T_{jv} \cdot R_j \leq C_v \cdot V_v \tag{4.6}$$

Constraint 4.6 means that no vehicle could transfer more goods than its capacity.

$$\forall_{i,v}\Sigma_j D_i \cdot T_{jv} \leq S_v \tag{4.7}$$

Constraint 4.7 says that any vehicle could not visit more nodes during day than the defined upper bound limit.

$$\bigwedge_{i,j} D_i V_j^r \vee \bigwedge_{i,j} \neg D_i V_j^r \tag{4.8}$$

Constraint 4.8 assures that every regular vehicle should be either used at every day or not used at all.

5. Experiments and Results

5.1. Framework

To evaluate a proposed approach multiple experiments were performed for it and proposed baseline models(see 4.2). Each experiment can be described with two parameters:

1. benchmark instance name, from Solomon Benchmark [41], that was used to generate characteristic days,

2. characteristic day types in input. Type could be one of $Hard, Normal, Easy$ and it defines number of nodes in characteristic day,

and all of them are outlined in section 5.2. For each experiment configuration a fleet and its estimated cost were predicted with an approach proposed by this thesis and every of baselines models from section 4.2. After that, a the following simulation was performed:

1. thirty days (to emulate month contract period) were generated based on characteristic days defined in test configuration,

2. for every day from that days a planning with a VRP solver described in section 4.1 was executed. As an input solver received a predicted fleet of regular vehicles and nodes with requirements that had to be met. Solver was also able to use spot vehicles in case if predicted fleet will not be sufficient (see 3.2 for details on difference between spot and regular vehicles). Solver results were saved.

3. from saved results day usage costs for all used vehicles were summed and up and a relative deviation between real and estimated fleet costs were calculated with formula $(\frac{|real-predicted|}{predicted})$

5.2. Experiments

5.2.1. Characteristic Day Order

Motivation: see how order of characteristic days affects model prediction.

Grading criterion: predictions similarity. Problem is the same in every case, so a good model should not depend on order of characteristic days and results should be same for every configuration.

Characteristic days configurations:

– Benchmark c101, $Hard, Normal, Easy$

– Benchmark c101, $Easy, Normal, Hard$

Configuration / Model name	First	Second
Linear model	8%	11%
SAT model with linear constraints	1117%	1100%
VRP based model	13%	14%

Table 5.1. Overestimates of each model in percentage: $\frac{real\_cost - predicted\_cost}{real\_cost}$

Summary: all models are vulnerable to order of days in the input.

5.2.2. Number of Characteristic Days in Problem

Motivation: see how amount of days affects affects model prediction.

Grading criterion: prediction error and prediction time. Given a bigger number of characteristic days a good model should be able to predict a more accurate fleet.

Characteristic days configurations:

– Benchmark c101, $Easy, Normal, Hard$

– Benchmark c101, $Easy, Normal, Hard, Easy, Normal, Hard$

– Benchmark c101, $Easy, Normal, Hard, Easy, Normal, Hard, Easy, Normal, Hard$

Summary: in two of three tests, linear model showed better performance than two other models, but VRP based model outperformed it for bigger amount of input data.

Configuration / Model name	First	Second	Third
Linear model	8%	21%	34%
SAT model with linear constraints	1100%	1110%	1100%
VRP based model	14%	45%	33%

Table 5.2. Overestimates of each model in percentage: $\frac{real\_cost - predicted\_cost}{real\_cost}$

5.2.3. Characteristic Days Complexity

Motivation: see how amount of clients in characteristic day affects model prediction.

Grading criterion: prediction error and prediction time. Given a bigger number of clients model a good model should be able to predict a more accurate fleet.

Characteristic days configurations:

– Benchmark c51, $Easy, Normal, Hard$

– Benchmark c101, $Easy, Normal, Hard$

Configuration / Model name	First	Second
Linear model	0%	6%
SAT model with linear constraints	1200%	1100%
VRP based model	0%	14%

Table 5.3. Overestimates of each model in percentage: $\frac{real\_cost - predicted\_cost}{real\_cost}$

Summary: for small amount of nodes linear model and VRP based model showed the best performance, but for bigger amount of nodes linear model outperformed VRP model.

5.3. Results

Tests demonstrate, that the proposed approach is still not good enough to be used for Fleet Structure Optimization, as in most of tests its prediction errors were beyond the established

error bound of 10%. Also a model was significantly outperformed by a Linear Model in tests 5.2.1 and 5.2.3.

One of the possible reasons for this behavior is a big complexity of a search space caused by passing multiple days as an input to the solver. This belief is can be supported with results in test 5.2.3 with the maximum number of nodes was equal to fifty.

Another possible reason is the usage of an interval constraint [40] to model breaks between days. This constraint has a rigorous propagator, and as such could require more time to find an optimal solution than it was allocated for tests.

6. Conclusion

The author implemented and compared three models for Fleet Structure Optimization Problem based on different optimization paradigms. All models were created in an open-source library Google OR tools[31]. Implemented approaches were tested on several different configurations on data generated from Solomon Benchmark described in detail in appendix B. Tests were performed with a custom metric that indicated the difference between real and estimated costs. The main model suggested by this thesis outperformed other models in some tests, which proves that this approach is competitive. Nevertheless, the approach is still not production-ready because it prediction errors wasn't fit in defined error bounds, so future works would be done.

6.1. Future Work

Many different tests and experiments have been left for the future due to lack of time (i.e. the experiments with real data are usually very time-consuming, requiring even days to finish a single run). Future work concerns a deeper analysis of particular mechanisms, new proposals to try different methods and to make the software more generic and usable for others.

The Fleet Structure Optimization problem could also be addressed as an Algorithm Configuration problem, where the task is to find an optimal set of parameters for the Machine Learning model. In this case, a VRP solver could be treated as a black box for which parameters should be optimized. This approach could be tested and compared against approaches introduced in the thesis.

Before that special attention should be paid to test cases where the other models outperformed the proposed approach. Possible reasons could come up from the fact that data were clustered or from the fact that the VRP solver had not had enough time to plan an optimal vehicle assignment, so some improvements and more tests should be done.

Besides works mentioned above, work for prediction of demands and aggregation of predicted results into characteristic days should be done.

Bibliography

[1] Richard McCormack and Graham Coates. "A simulation model to enable the optimization of ambulance fleet allocation and base station location for increased patient survival". In: *European Journal of Operational Research* 247.1 (2015), pp. 294–309. ISSN: 0377-2217. DOI: *https://doi.org/10.1016/j.ejor.2015.05.040*.

[2] Nathan Kunz and Luk N. Van Wassenhove. "Fleet sizing for UNHCR country offices". In: *Journal of Operations Management* (2019).

[3] Alireza Rahimi Vahed et al. "Fleet-sizing for multi-depot and periodic vehicle routing problems using a modular heuristic algorithm". In: *Computers & Operations Research* 53 (Jan. 2015), 9–23. DOI: *10.1016/j.cor.2014.07.004*.

[4] George F List et al. "Robust optimization for fleet planning under uncertainty". In: *Transportation Research Part E: Logistics and Transportation Review* 39.3 (2003), pp. 209–227. ISSN: 1366-5545. DOI: *https://doi.org/10.1016/S1366-5545(02)00026-1*.

[5] Kris Braekers, Katrien Ramaekers, and Inneke Van Nieuwenhuyse. "The vehicle routing problem: State of the art classification and review". In: *Computers Industrial Engineering* 99 (2016), pp. 300–313. ISSN: 0360-8352. DOI: *https://doi.org/10.1016/j.cie.2015.12.007*.

[6] Bruce Golden, Saahitya Raghavan, and Edward Wasil. *The Vehicle Routing Problem: Latest Advances and New Challenges*. Vol. 43. Jan. 2008. ISBN: 978-0-387-77777-1. DOI: *10.1007/978-0-387-77778-8*.

[7] Luca Bertazzi and M.Grazia Speranza. "Inventory routing problems: An introduction". In: *EURO Journal on Transportation and Logistics* 1 (Dec. 2012). DOI: *10.1007/s13676-012-0016-7*.

[8] Sanne Wøhlk. "A Decade of Capacitated Arc Routing". In: *The Vehicle Routing Problem: Latest Advances and New Challenges*. Ed. by Bruce Golden, S. Raghavan, and Edward Wasil. Boston, MA: Springer US, 2008, pp. 29–48. ISBN: 978-0-387-77778-8. DOI: *10.1007/978-0-387-77778-8_2*.

[9] Daniela Favaretto, Elena Moretti, and Paola Pellegrini. "Ant colony system for a VRP with multiple time windows and multiple visits". In: *Journal of Interdisciplinary Mathematics* 10.2 (2007), pp. 263–284. DOI: *10.1080/09720502.2007.10700491*. eprint: *https://doi.org/10.1080/09720502.2007.10700491*.

[10] Yvan Dumas, Jacques Desrosiers, and François Soumis. "The pickup and delivery problem with time windows". In: *European Journal of Operational Research* 54.1 (1991), pp. 7–22. ISSN: 0377-2217. DOI: *https://doi.org/10.1016/0377-2217(91)90319-Q*.

[11] Aida Mauziah Benjamin and John E Beasley. "Metaheuristics with disposal facility positioning for the waste collection VRP with time windows". In: *Optimization Letters* 7.7 (2013), pp. 1433–1449.

[12] Zeynep Özyurt, Deniz Aksen, and Necati Aras. "Open Vehicle Routing Problem with Time Deadlines: Solution Methods and an Application". In: *Operations Research Proceedings 2005*. Ed. by Hans-Dietrich Haasis, Herbert Kopfer, and Jörn Schönberger. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 73–78. ISBN: 978-3-540-32539-0.

[13] Roberto Baldacci, Maria Battarra, and Daniele Vigo. "Routing a heterogeneous fleet of vehicles". In: *The vehicle routing problem: latest advances and new challenges*. Springer, 2008, pp. 3–27.

[14] Éric D Taillard. "A heuristic column generation method for the heterogeneous fleet VRP". In: *RAIRO-Operations Research* 33.1 (1999), pp. 1–14.

[15] William Ho et al. "A hybrid genetic algorithm for the multi-depot vehicle routing problem". In: *Engineering applications of artificial intelligence* 21.4 (2008), pp. 548–557.

[16] Henry CW Lau et al. "Application of genetic algorithms to solve the multidepot vehicle routing problem". In: *IEEE transactions on automation science and engineering* 7.2 (2009), pp. 383–392.

[17] Franklin T Hanshar and Beatrice M Ombuki-Berman. "Dynamic vehicle routing using genetic algorithms". In: *Applied Intelligence* 27.1 (2007), pp. 89–99.

[18] José Miguel De Magalhães and Jorge Pinho De Sousa. "Dynamic VRP in pharmaceutical distribution—a case study". In: *Central European Journal of Operations Research* 14.2 (2006), pp. 177–192.

[19] Roberto Montemanni et al. "Ant colony system for a dynamic vehicle routing problem". In: *Journal of combinatorial optimization* 10.4 (2005), pp. 327–343.

[20] Edyta Kucharska. “Dynamic Vehicle Routing Problem—Predictive and Unexpected Customer Availability”. In: *Symmetry* 11 (Apr. 2019), p. 546. DOI: *10.3390/sym11040546*.

[21] Jing Fan. “The Vehicle Routing Problem with Simultaneous Pickup and Delivery Based on Customer Satisfaction”. In: *Procedia Engineering* 15 (2011). CEIS 2011, pp. 5284–5289. ISSN: 1877-7058. DOI: *https://doi.org/10.1016/j.proeng.2011.08.979*.

[22] Guy Desaulniers et al. “VRP with Pickup and Delivery”. In: *The Vehicle Routing Problem*. 2002.

[23] Feiyue Li, Bruce Golden, and Edward Wasil. “The open vehicle routing problem: Algorithms, large-scale test problems, and computational results”. In: *Computers Operations Research* 34.10 (2007), pp. 2918–2930. ISSN: 0305-0548. DOI: *https://doi.org/10.1016/j.cor.2005.11.018*.

[24] Rajeev Kr. Goel and Sandhya Rani Bansal. “Chapter 5 - Hybrid algorithms for rich vehicle routing problems: a survey”. In: *Smart Delivery Systems*. Ed. by Jakub Nalepa. Intelligent Data-Centric Systems. Elsevier, 2020, pp. 157–184. ISBN: 978-0-12-815715-2. DOI: *https://doi.org/10.1016/B978-0-12-815715-2.00011-7*.

[25] Rahma Lahyani, Mahdi Khemakhem, and Frédéric Semet. “Rich vehicle routing problems: From a taxonomy to a definition”. In: *European Journal of Operational Research* 241.1 (2015), pp. 1–14. ISSN: 0377-2217. DOI: *https://doi.org/10.1016/j.ejor.2014.07.048*.

[26] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. USA: Elsevier Science Inc., 2006. ISBN: 0444527265.

[27] Hamdy A. Taha. *Operations Research: An Introduction (8th Edition)*. USA: Prentice-Hall, Inc., 2006. ISBN: 0131889230.

[28] El-Ghazali Talbi. *Metaheuristics: From Design to Implementation*. Wiley Publishing, 2009. ISBN: 0470278587.

[29] Nicholas Nethercote et al. “MiniZinc: Towards a Standard CP Modelling Language”. In: *Principles and Practice of Constraint Programming – CP 2007*. Ed. by Christian Bessière. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 529–543. ISBN: 978-3-540-74970-7.

[30] Michel Gendreau et al. “Metaheuristics for the Vehicle Routing Problem and Its Extensions: A Categorized Bibliography”. In: 2008.

[31] Laurent Perron and Vincent Furnon. *OR-Tools*. Version 9.1.9490. Google, July 19, 2019.

[32] Michael Marte. *Yuck*.

[33] Christophe Ponsard and Renaud De Landtsheer. "OscaR.cbls : an open source framework for constraint-based local search". In: Feb. 2013.

[34] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. USA: Prentice Hall Press, 2009. ISBN: 0136042597.

[35] P. J. M. Laarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Applications*. USA: Kluwer Academic Publishers, 1987. ISBN: 9027725136.

[36] F. Glover and E. Taillard. "A user's guide to tabu search". In: *Annals of Operations Research* (Sept. 1993), 1–28. DOI: *https://doi.org/10.1007/BF02078647*.

[37] Roberto Battiti and Giampietro Tecchiolli. "The reactive tabu search". In: *ORSA journal on computing* 6.2 (1994), pp. 126–140.

[38] Philip Kilby, Patrick Prosser, and Paul Shaw. "Guided Local Search for the Vehicle Routing Problem with Time Windows". In: *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Ed. by Stefan Voß et al. Boston, MA: Springer US, 1999, pp. 473–486. ISBN: 978-1-4615-5775-3. DOI: *10.1007/978-1-4615-5775-3_32*.

[39] Andreas Schutt et al. "Explaining the cumulative propagator". In: *Constraints* 16 (July 2011), pp. 250–282. DOI: *10.1007/s10601-010-9103-2*.

[40] Frédéric Benhamou. "Interval constraintsInterval Constraints". In: *Encyclopedia of Optimization*. Ed. by Christodoulos A. Floudas and Panos M. Pardalos. Boston, MA: Springer US, 2009, pp. 1733–1736. ISBN: 978-0-387-74759-0. DOI: *10.1007/978-0-387-74759-0_304*.

[41] Marius M. Solomon. "VRP with time windows for 100 customers". In: (). [Online; accessed 13-December-2021].

A. Results reporoduction

Hardware	
Processor	AMD Ryzen 5 4500U with Radeon Graphics 2.38 GHz
Installed RAM	16.0 GB (15.4 GB usable)
Operating System	
System type	64-bit operating system, x64-based processor
Name	Windows 10 Home
Version	20H2
OS build	19042.1415
Software	
Language	C#
Target Framework	.Net Core
SDK version	5.0.400
Language version	9.0
Google OR Tools [31] version	9.1.9490

Table A.1. System Configuration

All code for the project is available in free access under an MIT license on the repository. Tests were performed with the setup described in A.1.

To reproduce them described in thesis, download an archive fleet-structure-solver.zip from the latest release on the release page. Unzip it and execute script fleet-structure-solver.bat with the following command:

```
fleet-structure-solver  reproduce-thesis-fleet-structure-predictions
-p solomon_benchmarks
-m sat linear vrp
```

It will generate three folders corresponding to tests described in 5. Each folder will contain three subfolders with fleet structure predictions made by each solver and characteristic days

for which they were performed. To run a VRP simulation and see difference between predicted structure and a real one, run command

```
fleet-structure-solver vrp-simulate
-f PATH_TO_FLEET_STRUCTURE
-c PATH_TO_CHARACTERISTIC_DAYS
-m ts
```

where PATH_TO_FLEET_STRUCTURE and PATH_TO_CHARACTERISTIC_DAYS should refer to one of the files with predictions made in the following step.

For more details refer to a README attached to the repository.

B. Solomon benchmark

To generate points of interest and nodes used for model evaluation the c101 instance from the problem sets[41] created by Marius M. Solomon was used. This instance highlight several factors that affect the behavior of routing and scheduling algorithms. They are: geographical data; the number of customers serviced by a vehicle; percent of time-constrained customers; and tightness and positioning of the time windows. In this problem set the geographical data is clustered. Problem set has a short scheduling horizon and allow only a few customers per route (approximately 5 to 10). Points of interest are shown on the figure B.1. Demands and times are shown in table B.1 Another benchmark used for testing is referred as c50 and contains first 50 points from c101 instance. Points are shown on figure B.2.

CUSTOMER	XCOORD.	YCOORD.	DEMAND	FROM TIME	TO TIME	SERVICE TIME
1	40	50	0	0	123	0
2	45	68	10	912	967	90
3	45	70	30	825	870	90
4	42	66	10	65	146	90
5	42	68	10	727	782	90
6	42	65	10	15	67	90
7	40	69	20	621	702	90
8	40	66	20	170	225	90
9	38	68	20	255	324	90
10	38	70	10	534	605	90
11	35	66	10	357	410	90
12	35	69	10	448	505	90
13	25	85	20	652	721	90
14	22	75	30	30	92	90
15	22	85	10	567	620	90
16	20	80	40	384	429	90

17	20	85	40	475	528	90
18	18	75	20	99	148	90
19	15	75	20	179	254	90
20	15	80	10	278	345	90
21	30	50	10	10	73	90
22	30	52	20	914	965	90
23	28	52	20	812	883	90
24	28	55	10	732	777	90
25	25	50	10	65	144	90
26	25	52	40	169	224	90
27	25	55	10	622	701	90
28	23	52	10	261	316	90
29	23	55	20	546	593	90
30	20	50	10	358	405	90
31	20	55	10	449	504	90
32	10	35	20	200	237	90
33	10	40	30	31	100	90
34	8	40	40	87	158	90
35	8	45	20	751	816	90
36	5	35	10	283	344	90
37	5	45	10	665	716	90
38	2	40	20	383	434	90
39	0	40	30	479	522	90
40	0	45	20	567	624	90
41	35	30	10	264	321	90
42	35	32	10	166	235	90
43	33	32	20	68	149	90
44	33	35	10	16	80	90
45	32	30	10	359	412	90
46	30	30	10	541	600	90
47	30	32	30	448	509	90
48	30	35	10	054	1127	90
49	28	30	10	632	693	90
50	28	35	10	001	1066	90
51	26	32	10	815	880	90
52	25	30	10	725	786	90

53	25	35	10	912	969	90
54	44	5	20	286	347	90
55	42	10	40	186	257	90
56	42	15	10	95	158	90
57	40	5	30	385	436	90
58	40	15	40	35	87	90
59	38	5	30	471	534	90
60	38	15	10	651	740	90
61	35	5	20	562	629	90
62	50	30	10	531	610	90
63	50	35	20	262	317	90
64	50	40	50	171	218	90
65	48	30	10	632	693	90
66	48	40	10	76	129	90
67	47	35	10	826	875	90
68	47	40	10	12	77	90
69	45	30	10	734	777	90
70	45	35	10	916	969	90
71	95	30	30	387	456	90
72	95	35	20	293	360	90
73	53	30	10	450	505	90
74	92	30	10	478	551	90
75	53	35	50	353	412	90
76	45	65	20	997	1068	90
77	90	35	10	203	260	90
78	88	30	10	574	643	90
79	88	35	20	109	170	90
80	87	30	10	668	731	90
81	85	25	10	769	820	90
82	85	35	30	47	124	90
83	75	55	20	369	420	90
84	72	55	10	265	338	90
85	70	58	20	458	523	90
86	68	60	30	555	612	90
87	66	55	10	173	238	90
88	65	55	20	85	144	90

89	65	60	30	645	708	90
90	63	58	10	737	802	90
91	60	55	10	20	84	90
92	60	60	10	836	889	90
93	67	85	20	368	441	90
94	65	85	40	475	518	90
95	65	82	10	285	336	90
96	62	80	30	196	239	90
97	60	80	10	95	156	90
98	60	85	30	561	622	90
99	58	75	20	30	84	90
100	55	80	10	743	820	90
101	55	85	20	647	726	90

Table B.1. Demands and service times from C101 instance

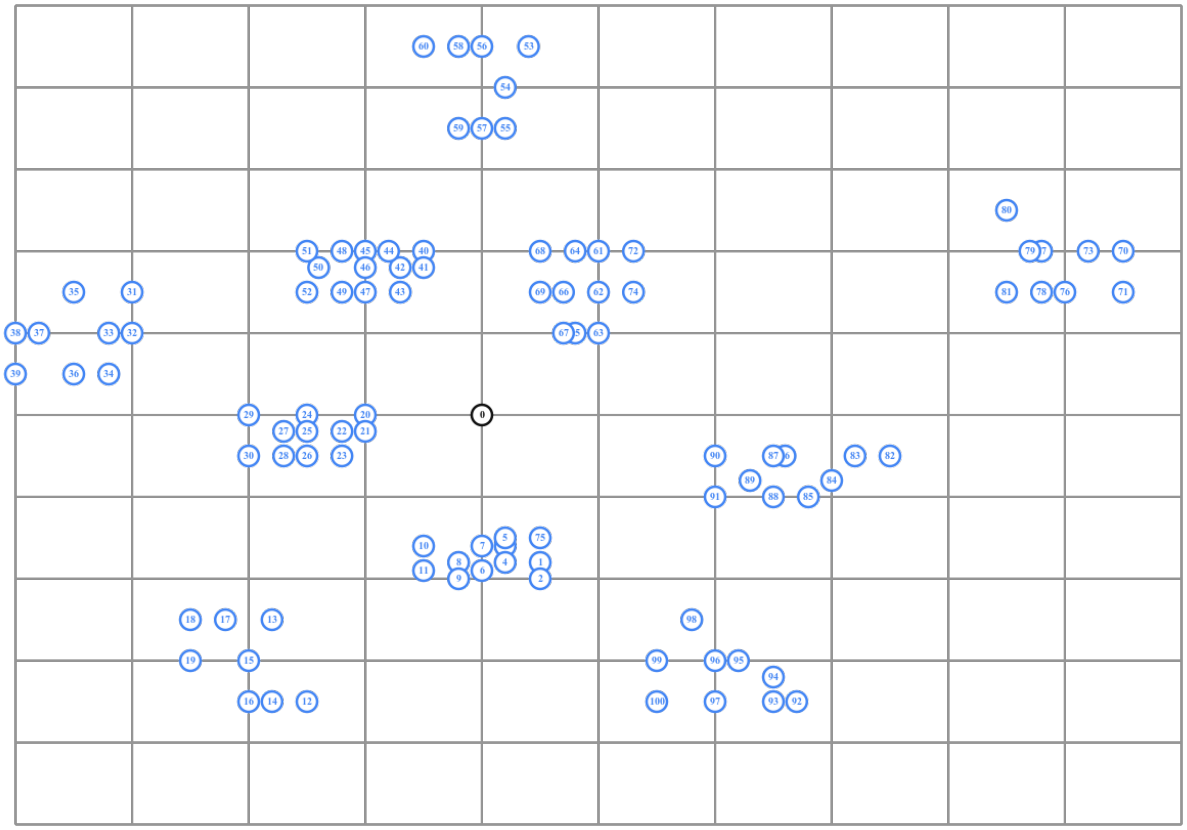

Fig. B.1. Points of interest in c101 instance

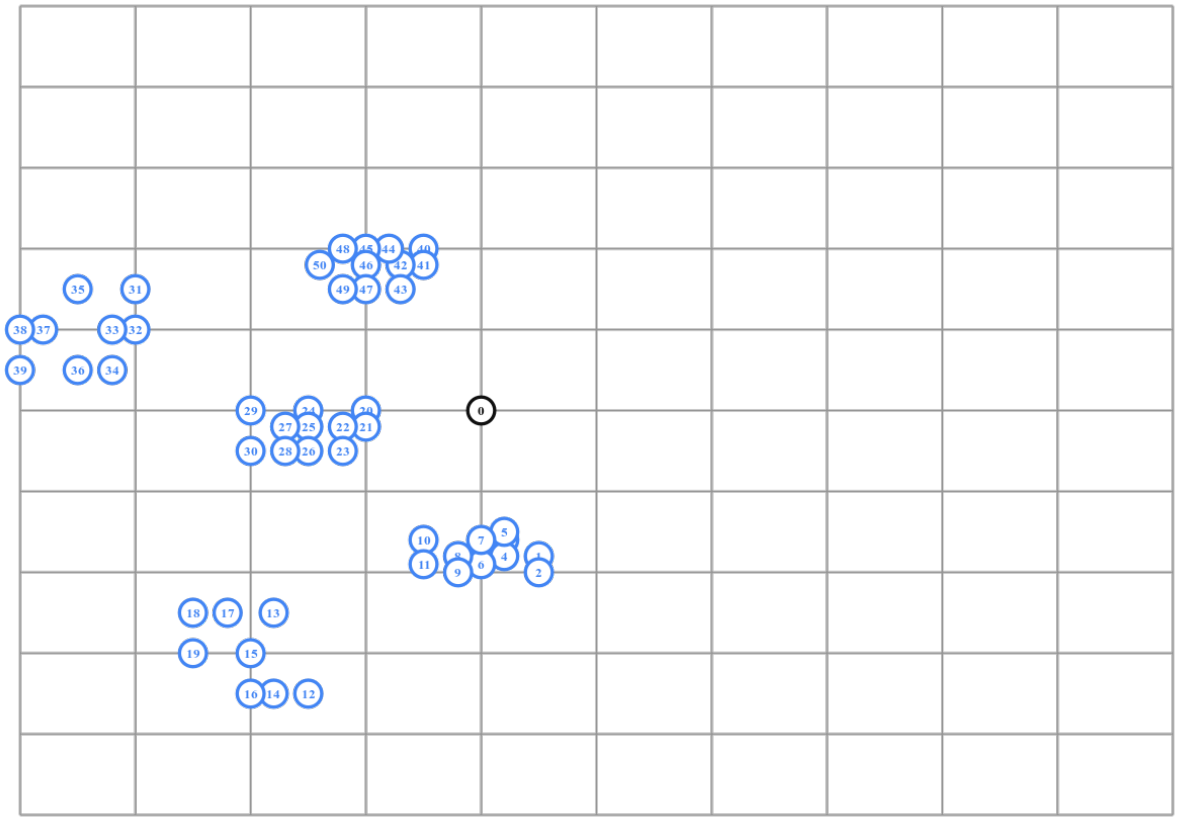

Fig. B.2. Points of interest in c50 instance