A comparative study of Constraint Programming solvers for the MDVRPTW

João Marinho[1 [201905952]] and Rodrigo Tuna[1 [201904967]]

University of Porto, FEUP

Abstract. The Vehicle Routing Problem (VRP) is an important problem in the fields of logistics and transportation [8], known for its NP-hard complexity. Constraint Programming (CP) has emerged as a valuable approach for solving NP-hard problems, including the VRP. We compare the performance and effectiveness of two solvers, Prolog and OR-Tools, on three problem variants falling under the VRP umbrella. We test our approaches on a standard dataset, which shed light on the exceptional performance and improved quality of OR-Tools versus the struggling Prolog.

Keywords: Vehicle Routing Problem · Constraint Programming.

1 Introduction

The Vehicle Routing Problem (VRP) is an important problem in the fields of logistics and transportation [8]. The problem has several practical applications and real-world value, as most of the transportation of goods is done by road [2]. The problem has NP-hard complexity and as such it is difficult to develop exact algorithms to solve it. Most methods to solve the VRP are approximate or heuristic/meta-heuristic.

Constraint Programming (CP) has emerged as a valuable approach for solving NP-hard problems [5], including the VRP. In our work, we compare the performance and effectiveness of two different solvers, Prolog and OR-Tools, on three problem variants falling under the VRP umbrella: the Multi-Depot VRP, the VRP with Time-Windows, and the main focus of our work the MDVRPTW.

We test our approaches on a dataset used by previous researchers that contain problems with a variety of sizes. We assess results on the solvers separately, considering which parameters work best in which problem. We also analyzed how the time taken to solve grows with the dimension of the problem. We compare both of the solvers in terms of the cost of the final solution and the time taken to reach it.

Results indicate better suitability of Or-Tools for vehicle routing problems as the search space is too big for the Prolog solver. The heuristics employed by Or-Tools provide a big reduction of search space and, even though it does not guarantee an optimal solution, it provides an acceptable solution in a reasonable time.

This paper covers the problem definition, related work, constraint programming model, implementation, experimental setup, and finally the results.

2 Problem Definition

In this section, we provide a comprehensive problem definition, outlining the key characteristics and objectives of the routing problem as follows:

Objective Function: The objective of the routing problem is to find an optimal solution that minimizes the cost function. The cost function is represented as the sum of the costs associated with each vehicle's route and its specific meaning depends on the application.

Vehicle Capacity: Each vehicle in the routing problem has a limited capacity to carry goods. It is crucial to ensure that the total demand for each route does not exceed the capacity of the assigned vehicle.

Route Duration: Each vehicle is subject to a limited duration within which it must complete its route. The duration encompasses both the travel time between locations and the service times for each customer. For the Vehicle Routing Problem with Time Windows, the vehicles are only expected to complete their routes within the specified time bounds, without explicitly considering route durations.

Homogeneous Fleet: The fleet of vehicles available for routing is assumed to be homogeneous, implying that all vehicles possess similar characteristics, i.e. they have comparable capacities, travel speeds, and other relevant attributes.

Time Windows: Specific time windows are associated with each customer or location that must be visited. It is essential for the vehicles to adhere to these time constraints and complete their deliveries within the specified time intervals. Naturally, for the Multi-Depot Vehicle Routing Problem, these time windows do not apply.

No Congestion or Delays: The problem assumes that there is no traffic congestion or delays encountered during the routes. It is assumed that the vehicles can travel at a constant speed of 2 units of distance per time unit. As a result, the calculated distances between locations are reduced by half to reflect this assumed travel speed.

A formal description of the problem is close to the proposed Constraint Programming model, section 4. The main difference is that the routes are given by a binary decision variable instead of a predecessor/successor list. For this reason, we will not present it. For a formalization of the MDVRPTW, the reader is referred to [11].

3 Related Work

In this section, we provide context about the problem that we are solving and other problems under the VRP umbrella. We then discuss the method used in solving the problem.

3.1 Vehicle Routing Problem

The Vehicle Routing Problem (VRP) is a classical optimization problem that seeks to find the optimal set of routes for a fleet of vehicles to visit a set of locations while minimizing the distance traveled. The VRP first appeared as a generalization of the Travelling Salesman Problem [3] and has been studied for over 50 years [7]. Several variants of the VRP problem exist, originated by adding new constraints or by introducing generalization, for an extensive list of VRP problems the reader is referred to [12]. For this work, we will focus on combining two variants of the problem, the multiple depots and the time windows arriving at the Multi-Depot Vehicle Routing Problem (MDVRPTW).

Several methods have been employed to solve the VRP, which can be divided into exact and approximate optimization methods [2]. Exact methods include Branch-and-X methods, Constraint Programming, and IDA*. However, these methods do not find the optimal solution in reasonable time. For this reason, the most recent effort has been put into approximate methods such as Approximation Algorithms and Heuristics. [7] For solving the MDVRPTW examples found in the literature include Meta-Heuristics [11, 6], and Approximation Algorithms [10].

3.2 Constraint Programming

A programming paradigm known as Constraint Programming (CP) uses constraints to specify how variables relate to one another [5]. It differs from other programming paradigms in that the attributes of the problem are expressed rather than the procedures to be taken to solve it. A CP model is composed of three components: variables, the domains in which they exist, and constraints linking each variable to the others. Constraint propagation is the main method used in CP to solve a problem. It functions by narrowing the scope of the variables, strengthening the constraints, or adding new ones. As a result, the search space is reduced, which makes the problem simpler for search algorithms to solve.

4 Constraint Programming Model

We have a set of clients C and a set of depots D. We define cost $Cost(i, j)$ and time functions $Time(i, j)$ for all $(i, j) \in D \cup C$. For all clients $i \in C$, we define a demand $Demand(i)$, a service time $ServTime(i)$, and a service window $[Open(i), Close(i)]$. The time windows of the depots are equal between all depots and the closing time serves as the schedule horizon. We have a homogeneous fleet as such we just need to consider the number of vehicles per depot V and the maximum load of a vehicle Q. We create set D' with a copy of each depot per each vehicle it has $d_{i,k} \in D',\ \forall i \in D, \forall k, 0 < k \leq V$.

We can formulate the MDVRPTW as a constraint satisfaction problem by creating domain variables that represent the next node in the route after node i (eq. (1)), the arrival time at node i (eq. (2)), the load of the route at node i (eq. (3)), the waiting time at node i (eq. (4)).

Constraint (eq. (5)) prohibits inter-depot routing, constraint (eq. (6)) guarantees each node is visited only once, constraints (eqs. (7) and (9)) guarantee time correctness, constraints (eqs. (8) and (10)) guarantee load correctness, constraints (eqs. (11) and (12)) implement the time windows.

The goal of the problem is to minimize (eq. (13)).

$$Next_i \in D' \cup C,\ \forall i \in D' \cup C \tag{1}$$

$$A_i \in [0, T] \tag{2}$$

$$L_i \in [0, Q] \tag{3}$$

$$W_i \in [0, T] \tag{4}$$

$$Next_i \in \{i\} \cup C,\ \forall i \in D' \tag{5}$$

$$AllDifferent(Next_i | i \in D' \cup C) \tag{6}$$

$$A_{Next_i} = Time(i, Next_i),\ \forall i \in D' \tag{7}$$

$$L_{Next_i} = DemandNext_i,\ \forall i \in D' \tag{8}$$

$$A_{Next_i} = A_i + W_i + ServTime(i) + Time(i, Next_i),\ \forall i \in C \tag{9}$$

$$L_{Next_i} = L_i + Demand_{Next_i},\ \forall i \in C \tag{10}$$

$$A_i + W_i \geq Open(i) \tag{11}$$

$$A_i \leq Close(i) \tag{12}$$

$$min \sum_{i \in D' \cup C} Cost(i, Next_i) \tag{13}$$

5 Implementation

5.1 Or-Tools

In order to implement the problem in Google Or-Tools [9] we resort to using Or-tool's Vehicle Routing Interface in Python [4]. Unlike traditional CP solvers, the interface hides many details and as such constraints are not placed directly on the solver. We add two dimensions to the solver:

Time: Time serves as our cost function, time between two nodes is calculated by the Euclidean distance over the velocity which was defined at 2. As we are dealing with service times, the service time of the node we are leaving from is added to the cost. As all nodes are visited once and only once, this makes a constant when summing the costs of all roots and it does not affect the solver. The sum of all service times is subtracted after the collection of the solution. The maximum time allowed is the closing window of the depots in the TW setting and the maximum route duration in the MDVRP. Slack time is allowed and the callback is given by:

$$Time(i, j) + ServTime(i) \tag{14}$$

Load: As we are dealing with a capacitated problem, we add a dimension that prohibits the sum of the demands of nodes in a route to exceed a maximum vehicle capacity. The maximum load allowed is the maximum load of the vehicles, there is no slack allowed and the callback is given by:

$$Demand(i) \tag{15}$$

5.2 Prolog

Considering the Prolog implementation for the mentioned VRP problem variants, we must define a set of variables and constraints so that we fulfill the model requirements and achieve the best solution for our target.

Variables

Routes: The actual problem solution, is represented as a list of vehicle routes, where each index in the list corresponds to a node (starting from the depots to the customers), and the value at that index represents the path that the vehicle should take. This variable allows us to extract a *Materialized Routes* matrix, which will prove to be useful in the constraints portion. By using the *subcircuit* constraint over each vehicle route we impose a Hamiltonian path structure.

$$subcircuit(R_i), \forall i \in Routes \tag{16}$$

Leave Times: Variable with the same structure as the *Routes*. It represents the time at which a vehicle leaves each node index. Depots and customers that are not visited by a particular vehicle have a value of 0 in the respective index of the vehicle list. The leave times play a crucial role in enforcing time window constraints, as each node must reach a customer before its associated time window closes.

$$Next\_Start\_Work\# = max(Arrive\_Time, Open\_Time) \tag{17}$$

$$Close\_Time\# >= Arrive\_Time \tag{18}$$

$$Next\_Leave\_Time\# = Next\_Start\_Work + Service\_Time \tag{19}$$

Total Route Times: It is a list containing the route times of each vehicle. By summing all the values in this list and subtracting the sum of service times, we obtain the total time for the routing solution, in other words, the cost which we are trying to minimize.

$$labeling([Options, minimize(Total\_Time)], Vars) \tag{20}$$

Constraints

Depot: Routes are ordered by depot, ensuring that each vehicle starts from its assigned depot. Additionally, within each depot, the *lex_chain* constraint can be applied to order the routes, which further helps in eliminating symmetric solutions.

$$lex\_chain(R\_Set_i), \forall i \in Depot\ Route\ Sets \tag{21}$$

Customer: Customers must be visited exactly once. This constraint is achieved by transposing the materialized routes matrix and applying a sum constraint to only the customers' portion. This ensures that no customer is skipped or visited multiple times by any vehicle.

$$sum(Materialized\_Customer_i, \#=, 1), \forall i \in Routes \tag{22}$$

Demand: The demand constraint is imposed by taking the scalar product of the node demands with the materialized route and ensuring that the result is less than or equal to the vehicle's capacity.

$$scalar\_product(Demands, Materialized\_Route_i, \#=<, Max\_Demand), \forall i \in Routes \tag{23}$$

Time Window: Each vehicle must reach its associated customer before the corresponding time window closes, and wait in case the time window is not open. By incorporating appropriate constraints based on the leave times, the solution ensures adherence to time constraints.

Total Route Time: The total route time for each vehicle is restricted to be less than or equal to a maximum route duration. This constraint ensures that the vehicles complete their routes within the specified duration.

$$Total\_Route\_Time_i \#=< Max\_Route\_Time, \forall i \in Routes \tag{24}$$

6 Experimental Setup

6.1 Datasets

In this subsection, we provide an overview of the datasets [1] used in our project, along with the parameters and structure of the dataset files.

We have utilized separate datasets for each problem variant, and each dataset consists of 10 problems with increasing complexity. The number of nodes and vehicles varies depending on the problem variant. Specifically, for the MDVRP (Multi-Depot Vehicle Routing Problem), the dataset ranges from 48 nodes up to 288, while for the MDVRPTW (Multi-Depot Vehicle Routing Problem with Time Windows), it ranges from 20 nodes up to 216. In the case of VRPTW (Vehicle Routing Problem with Time Windows), most of the datasets have a fixed size of 100 nodes.

The datasets are provided in plain text files, and they adhere to a specific structure as follows:

The first line: This line contains information about the type of problem (mdvrp, tw, etc.), the number of vehicles per depot, the total number of customers, and the number of depots.

The next (number of depot) lines: These lines contain details specific to each depot. The details include the maximum route duration and maximum vehicle load for each vehicle associated with the departing depot. It is worth noting that the datasets assume the same values for each depot, treating them as equivalent.

The final (number of customer and depots) lines: Each line corresponds to the information related to each node, this includes the x and y coordinates, the service times and demands. For practical parsing purposes, the dataset assigns a value of 0 to the service time and demand of each depot location.

6.2 Runs

For the experiments 10 files for each problem in the dataset were selected,

OR-Tools Or-Tools starts by computing a heuristic solution and then uses a local search method or a meta-heuristic method to improve on the solution [4]. We performed several runs by changing the initial solution for each problem. We refrained from using meta-heuristics as it would be outside the scope of our work.

The used parameters were:

- PATH_CHEAPEST_ARC - Greedily chooses the cheapest path.
- PATH_MOST_CONSTRAINED_ARC - Chooses the most constrained arc.
- LOCAL_CHEAPEST_INSERTION - Inserts node in the cheapest position.
- AUTOMATIC - The solver chooses the first solution

The time limit was set to limit the search to 20 minutes. All solutions that the solver found were collected, including the Wall Time at which they were collected.

Prolog We aimed to explore various search strategies to find the most effective combination of *Variable Ordering*, *Value Selection*, and *Value Ordering* options. However, considering the extensive number of possible variations, 132 in total, and the time constraints, we had to streamline our approach.

We narrowed down the variations to only 8, with two possible values for each mentioned option.

For *Variable Ordering*, we selected the standard *leftmost* option, which is the default, and the *ffc* (first-fail, smallest domain) option, which chooses the variable with the smallest domain first.

Regarding *Value Selection*, we once again chose the default *step* option and the *bisect* option, which uses a domain splitting strategy.

For *Value Ordering*, we opted for the ascending order, *up*, and descending order, *down*, options.

To run the solver, we needed to determine the time limits for each problem per combination. Considering that we had a total of 30 problems, we allocated time limits of 30 seconds, 1 minute, 5 minutes, 10 minutes and 20 minutes resulting in a maximum of 36.5 minutes per problem per combination for the worst-case scenario.

With the experimental parameters set, we proceeded to execute the solver and analyze the obtained results.

7 Results

7.1 Or-Tools

In the OR-Tools implementation, we observed remarkable performance in terms of solution time and solution improvement. The solver exhibited impressive efficiency in reaching solutions quickly, even for complex problem instances. However, it is worth noting that OR-Tools utilizes different heuristics to minimize the cost function. In some cases, the solver stopped improving a solution, considering it to be optimal according to its specific heuristics. Nevertheless, it is important to acknowledge that there might be alternative heuristics that could potentially yield better solutions. In terms of cost minimization, the results obtained from the OR-Tools implementation were highly promising. The solver consistently produced solutions with minimized costs, indicating its effectiveness in finding near-optimal or optimal solutions for the given problem instances. The combination of fast solution times and impressive cost-minimization capabilities makes OR-Tools a powerful tool for tackling the VRP variants. The results obtained from the OR-Tools implementation highlight its effectiveness and potential for solving complex routing problems efficiently. The costs and times obtained from all the runs can be seen in tables 1 to 3. Or-Tools only managed to find a solution for 3 files of the VRPTW, as such it will be excluded from our analysis.

Parameters When comparing the parameters per problem, we define that one parameter is superior to the other if its cost is lower than the other. In the case of ties, we choose the one that arrived at that solution faster. For the MDVRP problem, the parameter with the most wins is PATH_CHEAPEST_ARC, and for the MDVRPTW is LOCAL_CHEAPEST_INSERTION. We can also see that the LOCAL_CHEAPEST_INSERTION parameter usually leads to a shorter solve time independent of whether it is better than other parameters or not. It can also be noted that in most cases the curve of the parameter AUTOMATIC and LOCAL_CHEAPEST_INSERTION are superimposed, as seen in Figure 1(a). The only counter-example is seen in Figure 1(b).

Complexity Study In order to test how the time to solve evolves with the complexity, we plot the time taken to find the best solution against several measures of the problem dimension: Number of Nodes, Figure 2, Average Nodes

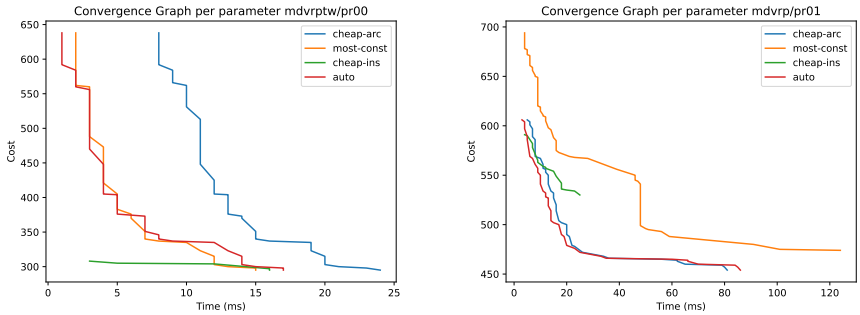

Fig. 1: Convergence graphs for problem pr00 of MDVRPTW on the left and problem pr01 of MDVRP on the left

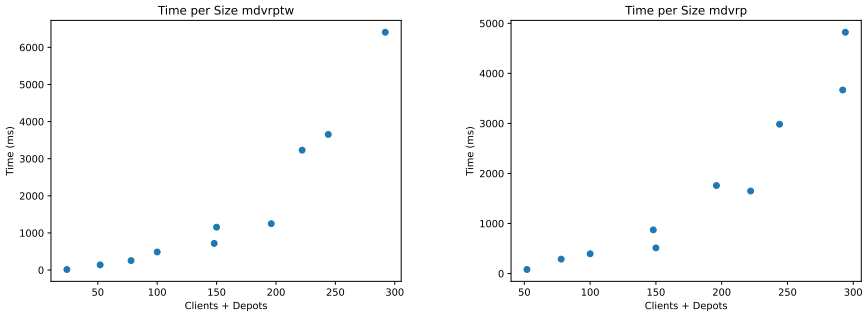

Fig. 2: Time evolution with number of Nodes for MDVRPTW(left) and MDVRP(right)

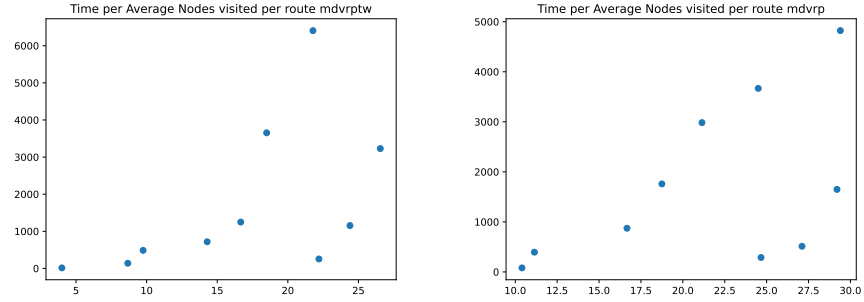

Fig. 3: Time evolution with Average Nodes visited per Route for MDVRPTW(left) and MDVRP(right)

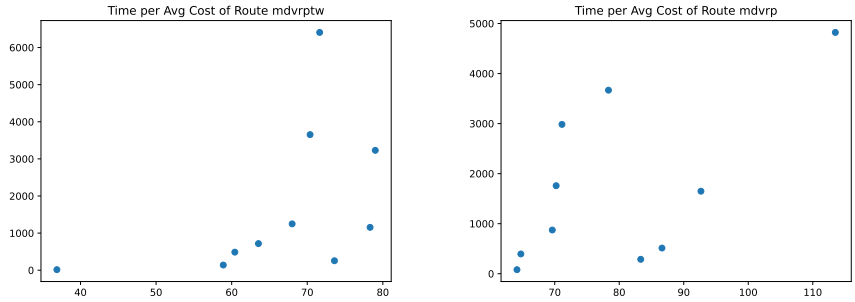

Fig. 4: Time evolution with Average Cost of Final Route for MDVRPTW(left) and MDVRP(right)

visited per Route, Figure 3 and Average Cost of Final Route, Figure 4. These plots are obtained using the best parameter choice for each file.

The measure that best relates to the time taken to solve is the Number of Nodes in the problem. As with the others, there seem to be two separate curves.

7.2 Prolog

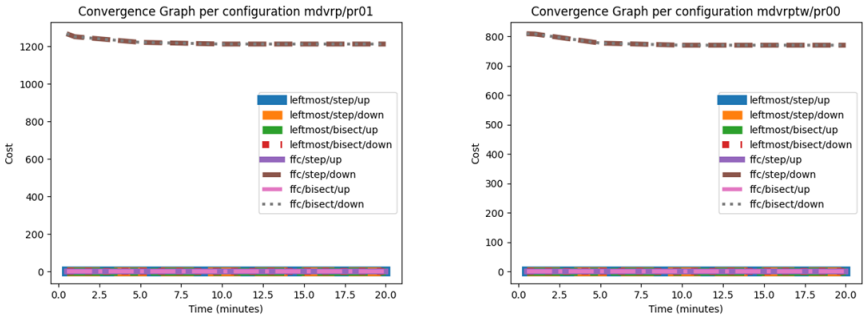

Fig. 5: Prolog results with different parameters for different problems

It became evident that Prolog is not the most suitable solver for solving this type of routing/optimization problems. Despite our efforts to mitigate symmetric solutions and improve efficiency, the search for viable problem solutions consumed a significant amount of time.

As illustrated by the graphs, Figure 5, most configurations resulted in a cost of 0, indicating that no solution was found within the given time limit. Only two specific problem instances yielded solutions in the Prolog implementation.

Notably, the configurations *ffc/step/down* and *ffc/bisect/down* demonstrated the ability to reach solutions relatively early in the search process. However, it is important to note that the solution costs stagnated shortly after reaching the penultimate time limit, 10 minutes.

These results highlight the challenges and limitations of using Prolog as a solver for complex routing problems, it not only struggles to reach feasible solutions but also to further optimize the achieved ones. While some configurations showed better results, the overall performance and efficiency of the Prolog implementation were suboptimal.

7.3 Comparison

Comparison between these solvers is unfair: Prolog tries to find an optimal solution whereas Or-Tools utilizes heuristics to find an acceptable solution. Nonetheless, in problems where Prolog managed to find at least one solution (6%), the Or-tools solution was on average 2.64 times better and it was reached 23,703.70 times faster than Prolog.

8 Conclusion

In conclusion, the objective of this project was to implement a constraint-based model in both Prolog and OR-Tools to solve three variants of the Vehicle Routing Problem. For each variant, we had access to specific datasets that allowed us to analyze and draw conclusions regarding the results obtained from each implementation.

It became evident that the Prolog implementation did not yield the expected results for this type of problem. Despite our best efforts and attempts to optimize the solution process, the search for viable solutions proved to be challenging and time-consuming.

On the other hand, the OR-Tools implementation demonstrated better performance and produced more favorable results. However, it is important to note that comparing the Prolog and OR-Tools implementations may not be entirely fair, as they employ different methodologies and heuristics. OR Tools utilizes a heuristic first solution and uses local search to improve it, whereas Prolog somewhat blindly searches the entire domain of the variables.

In summary, the project provided valuable insights into the limitations and capabilities of both Prolog and OR-Tools in solving the Vehicle Routing Problem variants. The experience gained from this project lays the foundation for future improvements and exploration of alternative approaches to tackle similar routing problems.

References

1. Problem datasets, http://neumann.hec.ca/chairedistributique/data/, last accessed on May 9, 2023.

2. Caceres-Cruz, J., Arias, P., Guimarans, D., Riera, D., Juan, A.A.: Rich vehicle routing problem: Survey. ACM Comput. Surv. **47**(2) (dec 2014). https://doi.org/10.1145/2666003, https://doi.org/10.1145/2666003
3. Dantzig, G.B., Ramser, J.H.: The truck dispatching problem. Management Science **6**(1), 80–91 (1959), http://www.jstor.org/stable/2627477
4. Didier, F., Perron, L., Mohajeri, S., Gay, S.A., Cuvelier, T., Furnon, V.: Or-tools' vehicle routing solver: a generic constraint-programming solver with heuristic search for routing problems (2023)
5. Hentenryck, P.V.: Constraint Satisfaction in Logic Programming. MIT Press (1989)
6. Hong, L., Xu, M.: A model of mdvrptw with fuzzy travel time and time-dependent and its solution. In: 2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery. vol. 3, pp. 473–478 (2008). https://doi.org/10.1109/FSKD.2008.77
7. Laporte, G.: Fifty years of vehicle routing. Transportation Science **43**, 408–416 (11 2009). https://doi.org/10.1287/trsc.1090.0301
8. Patel, R.: Your 101 on vehicle routing problem (Jun 2023), https://www.upperinc.com/blog/vehicle-routing-problem/
9. Perron, L., Furnon, V.: Or-tools, https://developers.google.com/optimization/
10. Tansini, L., Viera, O.: New measures of proximity for the assignment algorithms in the mdvrptw. Journal of the Operational Research Society **57**(3), 241–249 (2006). https://doi.org/10.1057/palgrave.jors.2601979, https://doi.org/10.1057/palgrave.jors.2601979
11. Tie-jun, W., Kai-jun, W.: Adaptive particle swarm optimization based on population entropy for mdvrptw. In: Proceedings of 2012 2nd International Conference on Computer Science and Network Technology. pp. 753–756 (2012). https://doi.org/10.1109/ICCSNT.2012.6526042
12. Toth, P., Vigo, D.: The Vehicle Routing Problem. Society for Industrial and Applied Mathematics (2002). https://doi.org/10.1137/1.9780898718515, https://epubs.siam.org/doi/abs/10.1137/1.9780898718515

9 Annexes

Table 1: Results of MDVRP runs in Or-Tools

File	Parameter	Cost	Time(ms)
pr01	cheap-arc	454	81
	most-const	474	124
	cheap-ins	530	25
	auto	454	86
pr02	cheap-arc	693	395
	most-const	735	521
	cheap-ins	738	314
	auto	693	377
pr03	cheap-arc	1002	1333
	most-const	1000	873
	cheap-ins	1006	1277
	auto	1002	1351
pr04	cheap-arc	1114	1799
	most-const	1203	1325
	cheap-ins	1286	2435
	auto	1114	1759
pr05	cheap-arc	1283	2984
	most-const	1344	2432
	cheap-ins	1536	2791
	auto	1283	2998
pr06	cheap-arc	1554	3668
	most-const	1597	2947
	cheap-ins	1812	4519
	auto	1554	3751
pr07	cheap-arc	556	293
	most-const	677	166
	cheap-ins	552	123
	auto	556	288
pr08	cheap-arc	1038	1091
	most-const	944	1053
	cheap-ins	940	513
	auto	1038	1083
pr09	cheap-arc	1264	1649
	most-const	1258	2781
	cheap-ins	1358	2836
	auto	1264	1690
pr10	cheap-arc	1707	4822
	most-const	1821	4400
	cheap-ins	1927	4197
	auto	1707	4826

Table 2: Results of MDVRPTW runs in Or-Tools

File	Parameter	Cost	Time(ms)
pr00	cheap-arc	295	24
	most-const	295	15
	cheap-ins	295	16
	auto	295	17
pr01	cheap-arc	574	126
	most-const	544	140
	cheap-ins	550	128
	auto	574	129
pr02	cheap-arc	989	623
	most-const	979	545
	cheap-ins	948	487
	auto	989	591
pr03	cheap-arc	1296	1021
	most-const	1468	1447
	cheap-ins	1253	718
	auto	1296	976
pr04	cheap-arc	1599	2157
	most-const	1529	3614
	cheap-ins	1472	1249
	auto	1599	2146
pr05	cheap-arc	1810	3922
	most-const	1689	3654
	cheap-ins	1839	3415
	auto	1810	3932
pr06	cheap-arc	2092	6253
	most-const	2064	4150
	cheap-ins	2006	6407
	auto	2092	6281
pr07	cheap-arc	736	290
	most-const	768	534
	cheap-ins	725	255
	auto	736	284
pr08	cheap-arc	1060	1156
	most-const	1105	1556
	cheap-ins	1166	641
	auto	1060	1155
pr09	cheap-arc	1525	3230
	most-const	1593	3365
	cheap-ins	1680	1737
	auto	1525	3269

Table 3: Results of VRPTW runs in Or-Tools

File	Parameter	Cost	Time(ms)
c100	cheap-arc	-	-
	most-const	-	-
	cheap-ins	190	24
	auto	-	-
c101	cheap-arc	-	-
	most-const	-	-
	cheap-ins	-	-
	auto	-	-
c102	cheap-arc	-	-
	most-const	-	-
	cheap-ins	-	-
	auto	-	-
c103	cheap-arc	-	-
	most-const	-	-
	cheap-ins	532	371
	auto	-	-
c104	cheap-arc	-	-
	most-const	-	-
	cheap-ins	541	409
	auto	-	-
c105	cheap-arc	-	-
	most-const	-	-
	cheap-ins	-	-
	auto	-	-
c106	cheap-arc	-	-
	most-const	-	-
	cheap-ins	-	-
	auto	-	-
c107	cheap-arc	-	-
	most-const	-	-
	cheap-ins	-	-
	auto	-	-
c108	cheap-arc	-	-
	most-const	-	-
	cheap-ins	-	-
	auto	-	-
c109	cheap-arc	-	-
	most-const	-	-
	cheap-ins	-	-
	auto	-	-