ISSN 0005-1179 (print), ISSN 1608-3032 (online), Automation and Remote Control, 2024, Vol. 85, No. 3, pp. 337–349.
© The Author(s), 2024 published by Trapeznikov Institute of Control Sciences, Russian Academy of Sciences, 2024.
Russian Text © The Author(s), 2024, published in Avtomatika i Telemekhanika, 2024, No. 3, pp. 101–118.

TOPICAL ISSUE

Smart Routes: A System for Development and Comparison of Algorithms for Solving Vehicle Routing Problems with Realistic Constraints

A. G. Soroka[*,a], **G. V. Mikhelson**[*,b], **A. V. Mescheryakov**[*,**,c], **and S. V. Gerasimov**[*,d]

[*] *Moscow State University, Moscow, Russia*
[**] *Space Research Institute, Russian Academy of Sciences, Moscow, Russia*
e-mail: [a] *andrew.soroka@student.msu.ru,* [b] *mikhelson.g@gmail.com,*
[c] *mesch@cosmos.ru,* [d] *gerasimov@cs.msu.ru*

Received July 8, 2023
Revised October 10, 2023
Accepted January 20, 2024

Abstract—The problem of route optimization with realistic constraints is becoming extremely relevant in the face of global urban population growth. While we are aware of approaches that theoretically provide an exact optimal solution, their application becomes challenging as the problem size increases because of exponential complexity. We investigate the Capacitated Vehicle Routing Problem with Time Windows (CVRPTW) and compare solutions obtaining by exact solver SCIP [1] with heuristic algorithms such as LKH, 2-OPT, 3-OPT [2], the OR-Tools framework [3], and the deep learning model JAMPR [4]. We demonstrate that for problem of size 50 deep learning and classical heuristic solutions became close to SCIP exact solution but requires less time. Additionally for problems with size 100, SCIP exact methods ~ 13 times slower that neural and classical heuristics with the same route cost and on $\sim 50\%$ worse for the first feasible solution on the same time. To conduct experiments, we developed the Smart Routes platform for solving route optimization problems, which includes exact, heuristic, and deep learning models, and facilitates convenient integration of custom algorithms and datasets.

Keywords: CVRPTW, Vehicle Routing Platform, Heuristics, Exact Solution, Reinforcement Learning

DOI: 10.31857/S0005117924030096

1. INTRODUCTION

The Vehicle Routing Problem (***VRP***) is a class of transport logistics problems that aims to reduce the cost of transport resources, route expenses, and cargo delivery time for a group of customers. In real-time scenarios, optimizing routes while considering various constraints is an issue for most companies. As the number of cities and customers increases, it becomes necessary to develop a solution that optimally utilizes allocated resources while maintaining service quality. A shorter route enables faster delivery for customers and allows more time to deliver goods to other customers.

In logistic problems, the *VRP* is often considered with various constraints that need to be taken into account when constructing the routes. The most popular ones are constraints on *customer time windows and service times* (TW) and *vehicle capacities* (C). Currently, there are significant challenges in developing and modifying algorithms capable of solving problems with increasing dimensions while striking a balance between solution quality and search time based on the number of imposed constraints. Additionally, there is no unified platform that allows users not only to solve

problems using built-in algorithms and experiment with them but also to add their own algorithms, data reading and processing methods, etc., without changing the system's architecture.

The contribution of this paper is the following:

— A platform for route problems optimization;

— A comparison of the effectiveness of common route optimization methods.

Our main focus is on the development of an innovative system called Smart Routes and an in-depth comparative analysis of different methodologies in the context of problematic constraints. We have studied heuristics, deep reinforcement network and exact method (SCIP), taking into account the limitations discussed in problems of different dimensions. Our study concludes with valuable information about the efficiency and quality of these solutions, which are primarily assessed using two main metrics: the solution optimization time and the final route cost determined by the objective function.

This paper reveals the architecture of our Smart Routes system designed to optimize route problems. The system seamlessly integrates exact and heuristic approaches, as well as deep reinforcement network methodologies, designed to solve the vehicle routing problem (VRP) in its various forms and modifications. So, it is useful for both experienced transport logistics experts and those less experienced in the field, providing a one-stop platform for testing new ideas. Consequently, this study not only delves into the ongoing search for the most efficient algorithmic approach to solving the Vehicle Routing Problem with Time Windows (CVRPTW) — a subcategory of VRP with popular and practical limitations—but also introduces an innovative tool ready to solve this problem comprehensively and efficient.

The article is structured as follows: Section 2 provides a review of recent literature containing both classical and deep reinforcement approach to route optimization. Sections 3 and 4 discuss the models considered and the developed Smart Routes system, respectively. Section 5 describes the data on which the experiments were conducted. In Section 6, we present the obtained results, and Section 7 contains the conclusions drawn from the conducted study.

2. RELATED WORK

This section considers three groups of algorithms capable of solving the problem. We note that although the problem has a rich history, there are many studies comparing different approaches to solving the route optimization problem. Most papers tend to suffer from a lack of exact approaches in comparison and either limited size of the problems considered, sometimes lack of constraints. We try to cover all the requirements in order to leave a complete picture of the applicability of classical approaches.

2.1. Heuristic Algorithms

The first group of algorithms is known as ***heuristic algorithms***, which are commonly used for such problems. Within this group, two distinct classes can be identified: *constructive heuristics* and *metaheuristics*.

Constructive heuristics are algorithms that iteratively build feasible solutions by adding components one at a time, such as the *nearest neighbor heuristic*, *insertion heuristic*, and *sweep heuristic*. While they do not guarantee optimal solutions, they can quickly generate high-quality solutions.

Metaheuristic algorithms are more general optimization methods that explore the solution space using randomization and heuristics, such as *simulated annealing*, *genetic algorithms*, and *ant colony optimization*. They may find better solutions than constructive heuristics but often require more computational resources.

The choice of these algorithms depends on the specific task and its application. Constructive heuristics are commonly used in practice due to their speed and efficiency in quickly obtaining high-quality solutions. On the other hand, metaheuristic approaches are more flexible and can be applied to a wider range of optimization problems, however with higher computational costs. Constructive heuristics are faster to implement and can provide effective solutions for certain problems.

2.2. Exact Algorithms

The next group consists of **exact algorithms**, including linear programming algorithms. These algorithms deal with problems in which the objective and constraint functions are linear. Linear programming problems have been extensively studied, and their solution properties are well-known.

Linear programming problem (***LP***) is an optimization problem expressed in standard form as follows:

$$\max c^{\mathrm{T}} x \mid Ax \leqslant b, x \geqslant 0. \tag{1}$$

Here, $A \in R^{m,n}$ represents the technological matrix, $b \in R^m$ is the resource vector, $c \in R^n$ is the price vector, and $x \in R^n$ is the unknown vector. If several or all variables in the vector $\boldsymbol{x}$ are integers, the problem is referred to as *MILP* (***Mixed Integer Linear Programming***).

There are several algorithms that can solve *MILP* problems:

1) *Branch&Bound* [5] is a widely used algorithm for solving MILP problems. It divides the problem into subtasks called nodes and solves each node using linear programming. The algorithm branches the node into subnodes by adding constraints until an integer solution is found or infeasibility is proven.
2) *Cutting Plane* [6] iteratively adds valid linear inequalities, called cuts, to exclude non-integer solutions in a problem formulation. Cuts are generated by solving the *LP*-relaxation of the problem. The algorithm continues until an integer solution is found or infeasibility is proven.
3) *Branch&Cut* [7] is a combination of the *Branch&Bound* and *Cutting Plane* algorithms. It generates cuts to eliminate non-integer solutions and branches nodes to create subnodes. This algorithm is typically more efficient than using Branch&Bound or Cutting Plane separately.

In general, the choice of algorithm depends on the specific problem and the requirements for solution quality and search time. Some *MILP* solvers, such as *CPLEX* [8], *SCIP* [1], and *Gurobi* [9], implement several of these algorithms and automatically select the most appropriate one for a given problem instance.

The main challenge with linear programming problems lies in their high dimensionality, often involving thousands of variables and constraints. The memory usage and solution time can increase exponentially as integer variables are added. Therefore, heuristics have been developed to find approximate optimal solutions in less time. For complex problems, heuristic approaches often provide the best trade-off between solution quality and computational time.

2.3. Deep Learning and Reinforcement Learning Algorithms

The first deep learning model proposed for solving VRP was introduced by Nazari et al. [10], who adapted the Pointer Network (PtrNet) from Vinyals et al. [11] to work with CVRP. Nazari et al. [10] completely abandoned the original RNN encoder part of the model and replaced it with a linear layer with shared parameters. A more recent algorithm, AM (attention model) by Kool et al. [12], replaced this architecture with an adapted transformer model using attention [13]. A direct improvement to this model is the JAMPR approach proposed by Falkner et al. [4], where the authors added additional fully connected networks for the current path and truck positions. This addition allowed the algorithm to successfully solve the *CVRPTW* problem. Chen and Tian [14]

propose an RL-based approach that iteratively selects a region on the graph and then selects and applies established local heuristics. This approach was further enhanced by the destruction operator introduced by Lu et al. [15]. The latest attempt to use deep learning for partitioning a set of points into subproblems and solving them using a black-box solver was proposed by Li et al. [16]. The authors presented two approaches: regression-based prediction of potential improvement in the final cost and classification for the best subproblem. By reducing the dimensionality and utilizing classical metaheuristic approaches in each subproblem, the authors were able to achieve good results on high-dimensional problems (over 1000 points). We have presented and explored in detail the applicability of our deep reinforcement approach to solve the route optimization problem with realistic constraints, showing how a fast suboptimal solution can be obtained using learned neural heuristics in the latest article [17].

3. ALGORITHMS AND MODELS

3.1. Classic Approaches

While considering classical heuristic approaches, the decision was made to focus on a group of algorithms known as *local search*. These algorithms provide a good balance between solution quality and search efficiency in real-world problems and also serve as fundamental components for metaheuristic and genetic algorithms [18].

One of the best classical heuristic approaches that was chosen as the main heuristic algorithm is the *Lin-Kernighan heuristic* [2]. This algorithm belongs to the class of local optimization algorithms. It operates by performing exchanges or moves, referred to as *opt*, that transform one route into another. Starting from an initial feasible route, the algorithm recurrently performs exchanges that reduce the length of the current route until a route is reached where no further exchange can lead to improvement. This process can be repeated multiple times from initially generated routes in a randomized manner. The algorithm assumes the existence of an initial route partitioning, and then iteratively improves upon the existing approximation. The improvement method involves exchanging vertices within each sub-route individually.

SCIP [1], an open-source software, is a powerful optimization tool specifically designed for solving mixed-integer programming problems [19, 20]. It is well-suited for addressing complex optimization challenges encountered in logistics, planning, and production scheduling. SCIP employs a range of methods and algorithms, including Branch&Bound, Cutting Plane methods, Constraint Propagation, Heuristics, Decomposition methods, and Integer Programming. These techniques involve dividing the problem, adding constraints, utilizing inference, applying rules of thumb, and solving integer programming problems to find optimal solutions. In summary, SCIP is a comprehensive optimization software that utilizes diverse algorithms and methods to effectively and accurately solve complex optimization problems.

In addition to SCIP, we selected OR-Tools (Operations Research Tools) [3], a highly regarded framework renowned for its effectiveness in solving VRP (Vehicle Routing Problem) challenges. OR-Tools is a versatile software designed to tackle combinatorial optimization problems, equation and inequality solving, and scheduling and routing problems. It offers a diverse array of optimization methods and algorithms, including *Linear Programming (LP) methods*, *Integer Programming (IP) methods*, *Discrete Optimization methods*, and *Equation and Inequality solving methods*. Notably, OR-Tools excels in Routing and Scheduling, providing algorithms for solving various routing problems such as the traveling salesman problem and vehicle routing problem. These methods encompass both heuristic approaches and exact techniques like local search and metaheuristic algorithms. With its flexibility and comprehensive range of methods, OR-Tools proves to be a powerful tool for addressing optimization problems across domains such as logistics, planning, and transportation.

3.2. Deep Reinforcement Learning Models

We modified the JAMPR model [4] and use as our deep reinforcement approach [17]. JAMPR model is a modification of the attention model (AM) [12] by Kool et al., which utilizes an attention-based encoder-decoder architecture [13]. Both models consider the route optimization problem as a sequential decision-making problem, modeled as a Markov decision process and solved using reinforcement learning. The problem solution is built incrementally by creating routes one node at a time. The current solution, route, and unvisited nodes are interpreted as the state, and the index of all unvisited nodes available for addition to the current route is treated as the actions.

We added to the JAMPR approach trainable matrices (M+ block from 1, separate for each constraint. We apply them to the result of the decoder, thus modifying the policy(π_θ). The final result of the network operation is a modified policy of size (k*, n*). The optimization process consists of minimizing the cost of the route, taking into account missed clients (soft setting):

$$Q = \sum_{j=1}^{m} \sum_{i=1}^{n} c_{ij}\, x_{ij} + \lambda \sum_{i=1}^{n} z_i, \tag{2}$$

where clients i, $i = 1, \ldots, N$, vehicles j, $j = 1, \ldots, M$, c_{ij} — cost of route for j vehicle to i client and z_i — binary parameter if we missed client.

Workflow of model: the encoder receives the features x_i of each node i (coordinates, cargo weights, time windows, etc.) and encodes them into a hidden linear vector $\tilde{x_i} \in R^{d_{emb}}$ of size d_{emb}. Then, the decoder model computes the attention query for each $\tilde{x_i}$ with respect to a specific context $C^{(t)}$ at decoding step t, in order to obtain scores for all nodes that can be added to the current route. Here, the context incorporates implicit graph embedding of the problem and additional problem-specific information such as the depot node index, the last node added to the current route, and the remaining capacity. The obtained scores are then either used in a greedy selection procedure, where the node with the highest score is always chosen, or transformed using softmax into a distribution used for sampling. In general, the encoder-decoder model represents a policy $\pi(i^{(t+1)}|C^{(t)}, x; \theta)$ with trainable parameters θ. You can see full $JAMPR^*$ architecture from original paper [4] on Fig. 1.

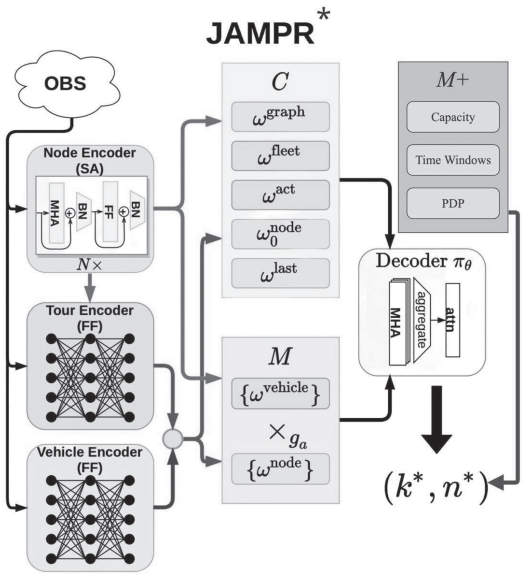


Fig. 1. The JAMPR deep RL architecture modified to solve CPDPTW problems. Added block $M+$ of learnable masks applied to output of decoder (policy π_θ).

4. SYSTEM *SMART ROUTES*

4.1. Requirements

System should meet the following functional **basic requirements**:

— **Solving VRP with popular constraints.** This involves solving VRP with common and important constraints such as vehicle capacity and time windows for delivery. As the system expands, new constraints will be added.

— **Solving large-scale problems (∼1000 points).** Solving large-scale VRP presents challenges as the number of customers to be visited increases, making it difficult for algorithms to find solutions and leading to longer search times.

— **Using real-world data.** While our work compares algorithms using artificially generated data, the proposed system also includes a feature for processing real-world data in a specific format.

— **Conducting experiments with classical heuristic, exact, and deep reinforcement network approaches.** Problem of identification of the algorithm type that is best suited for solving VRP remains relevant Therefore, our system provides the opportunity to experiment with algorithms from different classes.

— **Interactive usage of the system.** To facilitate the use of built-in algorithms, an intuitive interface is provided for adjusting parameters and interacting with the system.

— **Visual interpretation of solutions.** Implementing algorithms can sometimes make it difficult to determine the correctness and optimality of solutions. Visualizing the solution simplifies this verification process by providing a representation of the displayed route.

— **Comparison of quality indicators.** When comparing multiple algorithms, it is useful to analyze the differences in the quality of the obtained solutions. Graphs demonstrating the decrease or increase in route cost over a certain period of time are particularly useful and convenient for conducting these experiments.

4.2. Architecture

The system architecture is shown on Fig. 2. Let us describe the system's main flow:

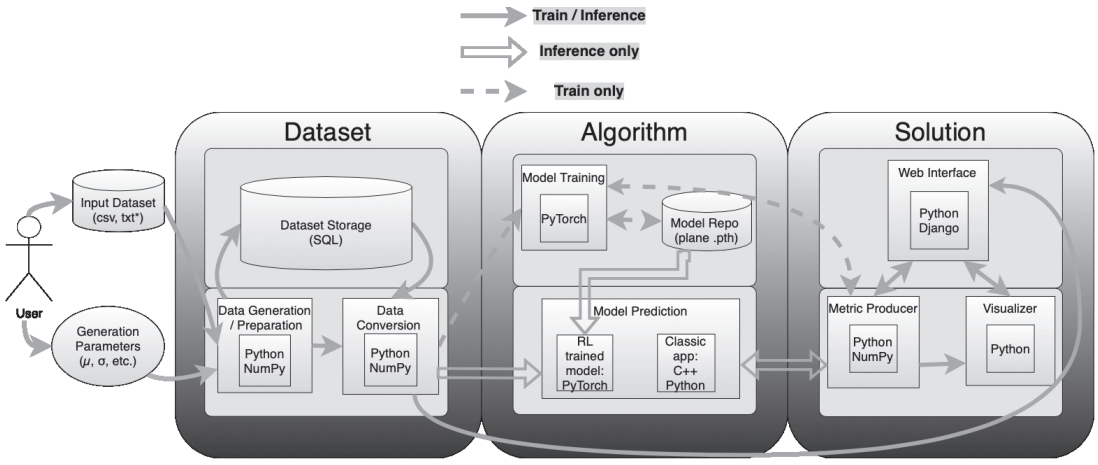


Fig. 2. Smart Routes System architecture. Three main blocks: 1) Dataset module is an entry point for all data manipulations, also stores prepared datasets and allows loading third-party data in a specified format; 2) Algorithm module responsible for train and evaluate all available algorithms on prepaired datasets. There is possibility to add user algorithm via special interfaces; 3) Solution module produces metrics, visualizes them and offers web interface for communication with system.

The user installs the system, strictly following the documentation. Upon launching it, the user is redirected to a web browser where a web page is displayed, prompting them to enter the necessary parameters.

1) **Data and Parameters.**

   (a) The user selects the problem they intend to solve: *CVRP* (Capacitated Vehicle Routing Problem), *VRPTW* (Vehicle Routing Problem with Time Windows), *CVRPTW* (Capacitated Vehicle Routing Problem with Time Windows);
   (b) Name of the solving algorithm is chosen;
   (c) A time limit is set for solving one task;
   (d) The option to solve tasks sequentially or in parallel on the available number of processors is chosen;
   (e) A dataset is loaded in a specific format.
   (f) Parameters for generating datasets are specified if needed: 1) μ — mean value; 2) δ — variance; 3) $capacity_{\min}$ — minimum consumer demand value; 4) $capacity_{\max}$ — maximum consumer demand value; 5) $n_{samples}$ — number of tasks; 6) $n_{customer}$ — number of customers in each task; 7) $service_{window}$ — right time window for the depot; 8) $service_{duration}$ — customer service time.
2) **Dataset.** The user-defined parameters and loaded data are passed to the *Data Generation/Preparation* block, where the data is either artificially generated based on the user's specified parameters or stored in variables. The user datasets are then saved in a database or immediately passed to the *Data Conversion* block, where the data is transformed by converting it to the required types.
3) **Algorithm.** This block receives the preprocessed data and user-specified parameters. The system determines which class of problems the selected algorithm belongs to: *classic_heuristics*, *exact_methods*, *neural_methods*.

   As the result, for each submitted task, the user obtains the final route, i.e., a list of customers and its cost.
4) **Solution.** The results obtained from the previous block are passed to the *Metric Producer* block, which generates an overall graph with metrics for the user-selected algorithms, demonstrating the decrease in route cost within the specified time limit. The results are also passed to the *Visualizer* block, which visualizes the complete graphical route based on the obtained list of customers.

Upon completion, the system displays two generated graphs, the final route cost, and the list of customers in the order of their visitation on the initial web page.

4.3. Advantages

Our platform has a key feature that allows users to add their own algorithms and models, giving it a significant advantage over other frameworks because:

1) Understanding the framework's toolkit can take a lot of time. For example, implementing the CVRPTW model using SCIP software required a considerable amount of time to understand how to correctly implement constraints, define the model architecture, and preprocess input data.
2) Software packages such as OR-Tools, SCIP, Gurobi, CPLEX only allow building models that act as exact methods. As the dimensions of the problem instances increase, the solution search time also grows, which can become critical.

In our system, users can integrate their algorithms by inheriting from base classes and implementing the necessary methods. The built-in algorithms can be applied to all types of approaches capable of solving VRP.

In addition our system also includes two convenient features. The first feature is result visualization, which is essential for evaluating the performance and suitability of the developed approach for the problem at hand. It automatically visualizes the final route within a specified time frame, allowing users to assess the solution. The second important feature is the API, which allows users to easily utilize the built-in approaches without integrating their own algorithms. By launching the server, users can upload a dataset, select an approach, and input additional parameters through a browser page. They will then receive the overall route cost, the route itself, and its interpretation.

Thus, these platform features simplify the development, testing, and experimentation of VRP solution approaches and demonstrate a significant advantage over existing software.

4.4. Comparison of Smart Routes and Classical Frameworks

Currently, there are some frameworks that provide tools for solving the Vehicle Routing Problem (VRP) with various constraints. The most popular ones include OR-Tools, Gurobi, and SCIP.

The table below shows the types of algorithms supported by these frameworks, denoted by + and — :

Algorithm Type	Framework			
	SCIP	Gurobi	OR-Tools	Smart Routes
Exact Approaches	+	+	+	+
Heuristic Approaches	+	+	+	+
Deep Reinforcement Approaches	—	—	—	+
Visualization of Results	—	—	—	+
Adding Custom Algorithms	—	—	—	+

It is important to note that OR-Tools and SCIP may not provide the same level of performance and scalability as Gurobi when solving very large-scale optimization problems.

Our proposed system differs from the aforementioned software in several aspects. In addition to covering all types of algorithms for solving the VRP, Smart Routes allows for the integration of custom models, showcasing its flexibility without compromising the overall architecture. We have implemented visualization features to create performance charts and visualize the final routes. Furthermore, during the system installation, OR-Tools and SCIP are automatically installed, making it easier to conduct experiments comparing user models with models implemented using the toolsets provided by these frameworks. There is no need to develop models or algorithms for solving custom logistics problems, as our system already offers built-in implementations. Moreover, our system provides a user-friendly interface that simplifies the workflow and makes it a convenient tool for users.

5. DATA

The algorithms considered in this research aimed to solve the *CVRPTW* problem. The interest is to observe the behavior of these approaches when multiple constraints are present simultaneously. In particular, the problem was solved in the SOFT settings, which allow skipping points that cannot be visited. In such cases, these points are excluded from the final route and incur a penalty, represented by an increase in the *missed_nodes* variable in the cost formula.

For the *CVRPTW* problem, suitable instances were selected from the well-known reference set of *Solomon* based on the R201 statistics [21]. The truck volumes were given as $Q_{50} = 750$ and $Q_{100} = 1000$ for problem sizes of 50 and 100, respectively. The total time horizon is defined as $[a_0, b_0]$, where $a_0 = 0$ for all examples, and the right boundary b_0 varies depending on the problem

size, specifically 1000 for 50 and 100 points. Additionally, the service duration h_i for each point is uniformly set to 10.

6. EXPERIMENTS

Artificial datasets were generated for the experiments, consisting of 100 examples with dimensions of 50 and 100 points. Each problem was assigned a specific time limit for solving using the algorithms: 100 seconds for 50-point problems and 200 seconds for 100-point problems. Since exact methods involve a complete search of possible solutions, they required 10 times more time for each problem to obtain any solution, even if it is not optimal. Therefore, the exact method was given 1000 and 2000 seconds for 50-point and 100-point problems, respectively.

The experiments were conducted using the developed system Smart Routes, which simplified the process of setting parameters when running different algorithms and tasks. To save time during the experiments, the tasks were parallelized on all available processors of the machine.

All experiments were performed on a server with NVIDIA Tesla A40 GPU and Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz with 16 virtual cores.

6.1. Main Results

The main experimental results are presented on the Fig. 3 with each algorithm represented as a point. Straight lines of different colors connect the points, showing that the overall cost of routes increases as the problem dimensions increase. The results obtained from SCIP are connected by an orange line, with blue dashes indicating the potential geometric position of the SCIP results. The x-axis represents the average solution time for a single task, while the y-axis represents the average route cost for a single task.

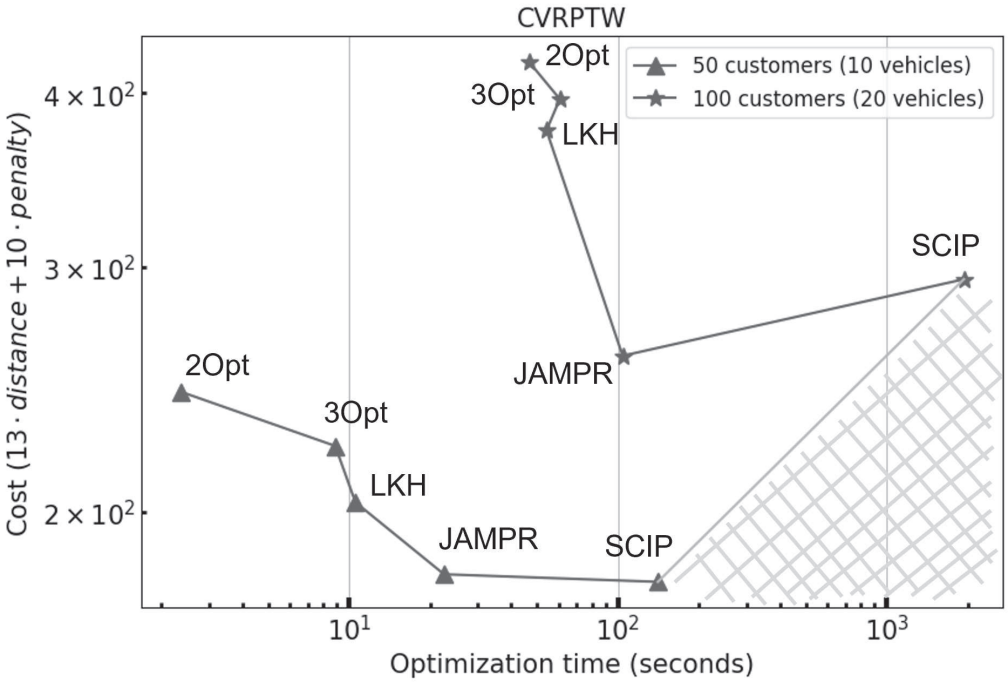


Fig. 3. Summary result of classical approaches, where each dot denotes the result of a specific algorithm on a given problem size. The horizontal line indicates the time required for the algorithm to achieve the best cost of the path, the vertical line indicates the best cost achievable by the algorithm. The orange line shows the applicability limit of exact approaches: on the left there is not enough time for the algorithm to provide a solution, the zone on the right is an area favorable for choosing exact approaches.

Based on the obtained results, the following conclusions can be drawn:

1) Increasing the value of λ in the LKH algorithm (λ Opt) improves the quality of solutions but increases the search time.
2) As the problem dimension and number of vehicles increase, SCIP solution time significantly increases. Doubling the problem dimension led to a roughly 14-fold increase in SCIP solution time. In contrast, the JAMPR neural network algorithm performs better than exact approaches in terms of solution quality and search time for problem instances with a dimension of 100 points. This suggests that exact methods have limited applicability for solving problems with dimensions of 100 points or more.

The following two chapters provide a more detailed description of the conducted experiments on problem instances with dimensions of 50 and 100 points.

6.2. Results for 50-Point Instances

For the diagram with GAP metrics, we calculated the percentage deviation of the algorithms at each point compared to the final best result achieved within 1000 seconds of optimization (SCIP). The overall metric represents the average value among all solved problem instances.

Figure 4 presents the graphs showing the change in the quality of the final route costs depending on the optimization time. On the plot, it can be observed that for small 50-point instances, both classical heuristics and deep reinforcement approaches quickly provide suboptimal solutions. The results of 2-Opt, 3-Opt, LKH, OR-Tools, and JAMPR reach a plateau, but in the first few seconds of optimization, the initial greedy solutions are provided by the classical heuristics, with LKH gradually approaching the results of JAMPR and OR-Tools. However, after about 10 seconds, SCIP solution becomes inferior to LKH, OR-Tools, and JAMPR, but after 100 seconds, SCIP surpasses them. Although exact methods provide the best solution, they require significantly more time to outperform LKH, JAMPR, and OR-Tools in terms of solution quality, with the final gap between the deep reinforcement model and SCIP being less than 5%.

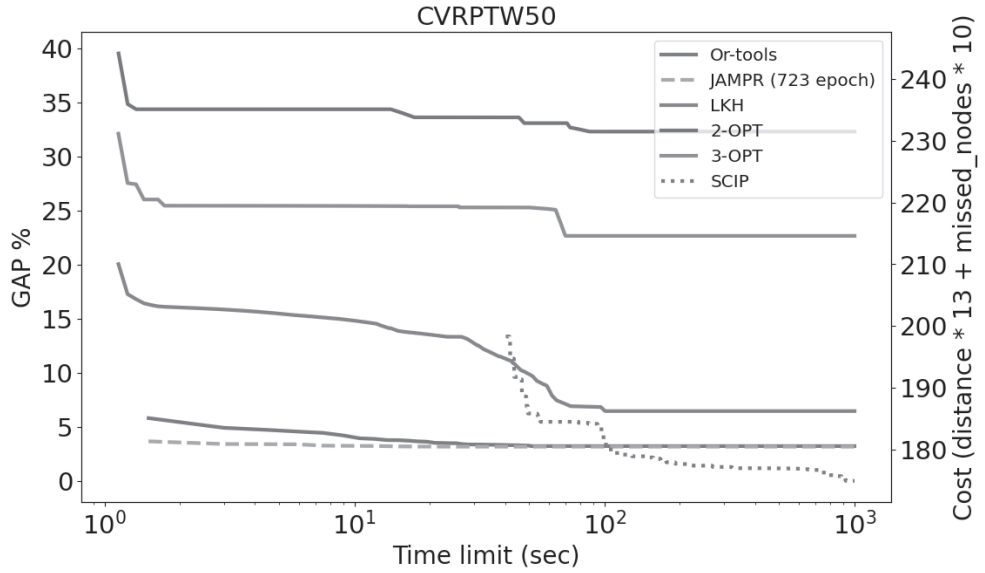


Fig. 4. Performance of the considered models on the CVRPTW with 50 points. Solid lines represent the results of approximate heuristic approaches. Dotted line represents the results of SCIP — the best cost achieved with the longest time. Dashed line represents JAMPR after 700 epochs of training — a fast suboptimal solution with subsequent asymptotic approximation to the heuristic. The figure has two scales: total cost (on the right) and the gap compared to the final best solution (on the left).

Thus, for 50-point instances, although exact approaches demonstrate the best solution, they are outperformed by classical heuristics and deep reinforcement approaches in terms of time by an order of magnitude. It is important to note that neural networks and classical heuristic approaches can provide a suboptimal solution to the CVRPTW within the first few seconds.

6.3. Results for 100-Point Instances

Figure 5 illustrates the change in route cost quality depending on the optimization time and the chosen algorithm for 100-point instances. As with the 50-point instances, we calculated the percentage cost for each algorithm compared to the best result shown by SCIP within the maximum optimization time (in this case, 200 seconds for classical heuristics and deep reinforcement approach, and 2000 seconds for SCIP).

From the graph, it is evident that classical heuristic approaches provide a faster suboptimal solution compared to other algorithms. However, they lag behind JAMPR and OR-Tools in terms of solution quality. Although JAMPR reaches a plateau around the 100th second, its initial greedy solution outperforms LKH with a GAP that is approximately 10% better, and the final result surpasses LKH GAP by around 50%. As the problem dimensionality doubles, there is a noticeable increasing GAP between the initial fast solutions in the deep reinforcement model and the heuristics, as depicted in Figs. 4 and 5.

Figure 5 also shows that SCIP required more than 900 seconds to obtain the first solution, which is approximately 13 times longer than it took for JAMPR and OR-Tools to achieve similar results when reaching a plateau. It is worth noting that within the allocated time, SCIP was unable to propose an optimal solution. Moreover, the first proposed solution is about 50% more costly than the heuristic solutions provided by OR-Tools and JAMPR. This indicates that as the problem dimensionality increases, exact approaches will become less and less applicable for solving them due to exponentially increasing time.

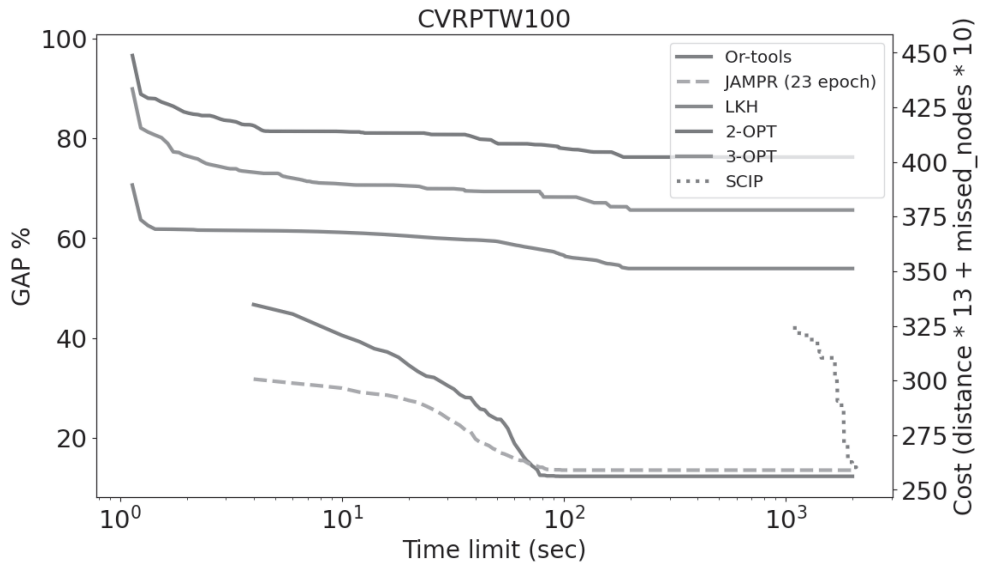


Fig. 5. Performance of the considered models on the CVRPTW with 100 points. Solid lines represent the results of approximate heuristic approaches. Dotted line represents the results of SCIP. Dashed line represents JAMPR after 23 epochs of training - a fast suboptimal solution with subsequent asymptotic approximation to the heuristic. Despite the considerable optimization time, SCIP failed to achieve the accuracy of JAMPR. The figure has two scales: total cost (on the right) and the gap compared to the final best solution (on the left).

Thus, in the case of 100-point instances, deep reinforcement approaches can provide a fast suboptimal solution that outperforms classical heuristics in terms of quality due to the longer training time and outperforms exact approaches in terms of time required to find a solution.

7. CONCLUSIONS

The problem of route optimization with realistic constraints is becoming extremely relevant in the face of global urban population growth. While we are aware of approaches that theoretically provide an exact optimal solution, their application becomes challenging as the problem size increases because of exponential complexity. We investigate the Capacitated Vehicle Routing Problem with Time Windows (CVRPTW) and compare solutions obtaining by exact solver SCIP [1] with heuristic algorithms such as LKH, 2-OPT, 3-OPT [2], the OR-Tools framework [3], and the deep learning model JAMPR [4].

All the metrics presented in the article were obtained using the Smart Routes platform. This system has the capability to solve the route optimization problem using various approaches, which facilitates research and result comparison.

Based on the experimental results, the following conclusions can be drawn:

— For problem instances with 50 points, both classical heuristic approaches and reinforcement learning approaches demonstrate their effectiveness by providing fast suboptimal solutions. The overall results are close to the results obtained by the exact method (SCIP) with GAP less than 5%. This indicates that for problem instances with 50 points, classical heuristic approaches and neural network-based approaches can offer a good trade-off between solution quality and search time.

— For problem instances with 100 points, exact methods required 13 more time to find the initial solution. They were unable to provide an optimal solution within the given time, resulting in up to 50% higher route costs during optimization. Classical heuristic approaches provide fast suboptimal solutions but start to lag behind more advanced methods such as JAMPR and OR-Tools in terms of solution quality. These findings suggest the complete infeasibility of exact methods for solving the vehicle routing optimization problem with 100 (or more) points.

We believe that the developed platform is an important component for future research in the field of route optimization with diverse constraints. Access to the source code of the system can be obtained by communicating with the authors of the article.

REFERENCES

1. Bestuzheva, K., Besançon, M., Wei-Kun Chen, et al., , The SCIP Optimization Suite 8.0, *Technical Report, Optimization Online*, 2021. `http://www.optimization-online.org/DB_HTML/2021/12/8728.html`

2. Helsgaun, K., An Effective Implementation of the Lin–Kernighan Traveling Salesman Heuristic, *European Journal of Operational Research*, Elsevier Publ., 2000, vol. 126, no. 1, pp. 106–130.

3. Perron, L., Operations Research and Constraint Programming at Google, *International Conference on Principles and Practice of Constraint Programming*, Springer, 2011, pp. 2–2.

4. Falkner, J.K. and Schmidt-Thieme, L., Learning to Solve Vehicle Routing Problems with Time Windows Through Joint Attention, *arXiv preprint arXiv:2006.09100*, 2020.

5. Laporte, G. and Nobert, Y., A Branch and Bound Algorithm for the Capacitated Vehicle Routing Problem, *Operations-Research-Spektrum*, Springer, 1983, vol. 5, pp. 77–85.

6. Cook, W. and Rich, J.L., A Parallel Cutting-Plane Algorithm for the Vehicle Routing Problem with Time Windows, *Technical report, Computational and Applied Mathematics*, Rice University, 1999. (http://www.math.uwaterloo.ca/ bico/papers/vrptw.pdf)

7. Augerat, Ph., Naddef, D., Belenguer, J.M., et al., Computational Results with a Branch and Cut Code for the Capacitated Vehicle Routing Problem, *Research report RR949-M*, ARTEMIS-IMAG, France, 1995.

8. IBM ILOG Cplex V12. 1: User's Manual for CPLEX, *International Business Machines Corporation*, 2009, vol. 46, no. 53, p. 157.

9. *Gurobi Optimization, LLC*, Gurobi Optimizer Reference Manual, 2023. `https://www.gurobi.com`

10. Nazari, M., Oroojlooy, A., Snyder, L., and Takác, M., Reinforcement Learning for Solving the Vehicle Routing Problem, *Advances in Neural Information Processing Systems*, 2018, vol. 31.

11. Vinyals, O., Fortunato, M., and Jaitly, N., Pointer Networks, *Advances in Neural Information Processing Systems*, 2015, vol. 28.

12. Kool, W., Van Hoof, H., and Welling, M., Attention, Learn to Solve Routing Problems!, 2018, *arXiv preprint arXiv:1803.08475*.

13. Vaswani, A., Shazeer, N., Parmar, N., et al., Attention is All You Need, *Advances in Neural Information Processing Systems*, 2017, vol. 30.

14. Xinyun Chen, Yuandong Tian, Learning to Perform Local Rewriting for Combinatorial Optimization, *Advances in Neural Information Processing Systems*, 2019, vol. 32.

15. Hao Lu, Xingwen Zhang, Shuang Yang, A Learning-Based Iterative Method for Solving Vehicle Routing Problems, *International conference on learning representations*, 2019.

16. Sirui Li, Zhongxia Yan, Cathy Wu, Learning to Delegate for Large-Scale Vehicle Routing, *Advances in Neural Information Processing Systems*, 2021, vol. 34.

17. Soroka, A.G., Meshcheryakov, A.V., and Gerasimov S.V., Deep Reinforcement Learning for the Capacitated Pickup and Delivery Problem with Time Windows, *Pattern Recognition and Image Analysis*, Springer, 2023, vol. 33, no. 2, pp. 169–178.

18. Groër, Ch., Golden, B., and Wasil, E., A Library of Local Search Heuristics for the Vehicle Routing Problem, *Mathematical Programming Computation*, 2010, vol. 2, pp. 79–101.

19. Çetinkaya, C., Karaoglan, I., and Gökcen, H., Two-Stage Vehicle Routing Problem with Arc Time Windows: A Mixed Integer Programming Formulation and a Heuristic Approach, *European Journal of Operational Research*, Elsevier, 2013, vol. 230, no. 3, pp. 539–550.

20. Tahernejad, S., Ralphs, Ted K., and DeNegre, Scott T., A Branch-and-Cut Algorithm for Mixed Integer Bilevel Linear Optimization Problems and Its Implementation, *Mathematical Programming Computation*, Springer, 2020, vol. 12, pp. 529–568.

21. Solomon, M.M., Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints, *Operations Research, Informs.*, 1987, vol. 35, no. 2, pp. 254–265.

This paper was recommended for publication by A.A. Galyaev, a member of the Editorial Board