S©MR

СИБИРСКИЕ ЭЛЕКТРОННЫЕ
МАТЕМАТИЧЕСКИЕ ИЗВЕСТИЯ
Siberian Electronic Mathematical Reports
http://semr.math.nsc.ru
ISSN 1813-3304

Vol. 20, No. 2, pp. 144–145 (2023)
https://doi.org/10.33048/semi.2023.20.???

УДК 519.8
MSC 90B06

HYBRID MEMETIC ALGORITHM FOR THE PICKUP AND DELIVERY PROBLEM WITH TIME WINDOWS

A.D. YUSKOV, **I.N. KULACHENKO**, AND **Y.A. KOCHETOV**

Communicated by P.P. Petrov

Abstract: We consider the classical Pickup and Delivery Problem with Time Windows (PDPTW). To tackle the problem, we present a heuristic scheme that is based on a memetic algorithm. The algorithm uses EAX and SREX crossovers, a large neighborhood search and a local search with multiple neighborhoods as an improvement step, and a special population management mechanism. Additionally, we implement several auxiliary procedures to diversify the search and also an iterated local search algorithm that helps the memetic algorithm. We compare the scheme to the best-known solutions for two benchmark libraries with up to 5000 clients and 148 vehicles. The proposed algorithm demonstrates good final results and is able to improve best-known solutions for 53 instances.

Keywords: VRP, ruin and recreate, LNS, LS, ILS.

1 Introduction

The Vehicle Routing Problem (VRP) is an NP-hard combinatorial optimization problem that arises from real-world transportation problems [27]. The goal is to find the best routes for serving clients for multiple vehicles.

Yuskov A.D., Memetic algorithm for the pickup-deliver problem.

The work is carried out within the framework of the state contract of the Sobolev Institute of Mathematics (project no. FWNF-2022-0019).
Received January, 1, 2023, Published December, 31, 2023.

Different variations of the Vehicle Routing Problem have been studied for a long time until now. The Pickup and Delivery Problem with Time Windows (PDPTW) [8] is one of the classical formulations of the general Vehicle Routing Problems. It is a combination of the Capacitated Vehicle Routing Problem [7], the Pickup and Delivery Problem [25] and the Vehicle Routing Problem with Time Windows [26].

For this variant of the problem there are several Mixed Integer Linear Programming formulations [8, 9]. However, solving these models is only useful for small instances. Other exact approaches include branch-and-price solutions [24], branch-and-cut algorithms [21] and branch-and-cut-and-price schemes [20, 1, 10]. These approaches are able to solve instances with sizes up to 200 nodes and some tightly constrained instances with 1000 nodes. Also, authors of [19] proposed a generic exact solver for different VRP problems. Recently, Vadseth et al. [28] proposed an algorithm for improving existing solutions.

The most popular way of dealing with the problem is a metaheuristic approach. One of the first algorithms was proposed in [17]. The authors developed a Reactive Tabu Search for the PDPTW and solved some small instances up to 100 nodes. Later, Li and Lim [13] proposed a set of instances and found some solutions for them using a Tabu-embedded Simulated Annealing metaheuristic. The proposed set contained 354 instances with up to 1000 nodes. Other algorithms include genetic algorithms [18], Large Neighborhood Search [3], Adaptive LNS [22]. Current state-of-the-art algorithms include Guided Ejection Search (GES) [16] and Adaptive Guided Ejection Search (AGES) [6]. Later, this approach was improved in [23] to combine multiple optimization techniques. Beling et al. [2] proposed an idea of exploration of highly infeasible solutions in an evolutionary scheme to find better solutions in terms of the objective function. Also, recently there was proposed a decomposition-based approach for large-scale instances [11].

In this paper we propose an optimization scheme for PDPTW based on a memetic algorithm. The memetic algorithm uses a tournament selection, an *edge assembly crossover* (EAX) and a *selective route exchange crossover* (SREX) crossovers, and a special population management mechanism that aims to diversify the population. We use a large neighborhood search and a local search with multiple neighborhoods to improve child solutions. Furthermore, we implemented a soft restart procedure, intensification and diversification procedures and an auxiliary iterated local search algorithm to help the memetic algorithm find better solutions. We compare the proposed scheme to the best-known solutions for two benchmark datasets. The algorithm demonstrates good final results and improves best-known solutions for 53 instances.

The rest of the paper is organized as follows: In section 2 we describe the problem statement in more detail. In section 3 we describe our algorithmic approach. In particular, in section 3.1 we present the population management in the memetic algorithm, section 3.2 describes the large neighborhood

search algorithm, section 3.3 presents the local search structure, section 3.4 tells about some auxiliary procedures, and in section 3.5 we describe the iterated local search algorithm. In section 4 we present the results of computational experiments. Finally, section 5 concludes the article.

2 Problem statement

We consider a pickup-delivery vehicle routing problem with time windows. In this problem, we need to create routes for several vehicles to move goods between the clients. We are given the set of clients. We know the traveling distance c_{ij} between each pair of nodes (i, j). One node is a depot. All vehicles must start and end their route at the depot. Each client is either a pickup or delivery client. If a vehicle visits a pickup client, then it must also visit the corresponding delivery client later. Each client must be visited by any of the vehicles exactly once. Each pickup client p has positive demand q_p. The demand of each delivery d client equals the negative demand of the corresponding pickup client: $q_d = -q_p$. If a vehicle visits a client, then its load is increased by the client's demand. The load of any vehicle must not exceed its capacity C during the trip. Each client i has an associated earliest visit time (time window start) t_i^s, latest visit time (time window end) t_i^e and service time s_i. A vehicle can start serving a client only during the time period between the earliest and the latest times. The objective function is hierarchical. Firstly, we need to minimize the number of used vehicles, and the secondary objective is the total traveling distance.

A solution for this problem can be encoded as a set of routes where each route is a sequence of visited clients. Suppose we have a route $R = (i_1, i_2, \ldots, i_k)$. This route needs to satisfy pickup-delivery precedence constraints. Then the traveling distance $\text{distance}(R) = c_{0i_1} + \sum_{j=1}^{k-1} c_{i_j i_{j+1}} + c_{k0}$. The load of the vehicle after visiting each client is calculated recursively: $l_0 = 0$, $l_j = l_{j-1} + q_{i_j}$ $\forall j = 1, \ldots, k$. The capacity violation is calculated as follows: $\text{overload}(R) = \sum_{j=1}^{k} \max(l_j - C, 0)$. The visit times for each client are also computed recursively. We assume that if the vehicle arrives at a client before his earliest visit time, it waits until this time point. Let a_j be the arrival time and let b_j be the departure time for a client j. Then: $b_0 = 0, a_j = b_{j-1} + c_{i_{j-1} i_j}, b_j = \max\left(a_j, t_{i_j}^s\right) + s_{i_j}$. The time window violation $\text{lateness}(R) = \sum_{j=1}^{k} \max\left(a_j - t_{i_j}^e, 0\right)$. We call solution $S = \{R_1, \ldots, R_m\}$ a *semi-feasible* if each client is visited exactly once and each route satisfies pickup-delivery precedence constraints. Our goal is to find a solution that is

semi-feasible and also:

$$\min m \tag{1}$$

$$\min \sum_{i=1}^{m} \text{distance}\,(R_i) \tag{2}$$

$$\text{overload}\,(R_i) = 0 \quad \forall i = 1, \ldots, m \tag{3}$$

$$\text{lateness}\,(R_i) = 0 \quad \forall i = 1, \ldots, m \tag{4}$$

The first objective is the number of vehicles, and the second one is the total traveling distance. The objectives should be optimized lexicographically. That means that the solution that has fewer vehicles is considered better no matter the traveling distance. The constraints state that each load of each vehicle must not exceed its capacity, and each client must be visited within his time window.

3 Algorithmic approach

To tackle this problem, we use the so-called *memetic algorithm*. It is a hybridization of a genetic algorithm with local search methods [14]. It maintains a population of solutions. At each iteration, it selects parents from the population. Then it applies a crossover operator to the selected parents, producing some offspring. Then it tries to improve each offspring by running a local improvement procedure. The procedure uses a *large neighborhood search algorithm* (LNS) and then applies a *local search* with several neighborhoods (LS). If the obtained solutions are good enough, then they are added to the population and replace parents. The quality of solutions is defined in terms of a penalized objective function that includes total traveling distance and penalties for constraint violation:

$$\text{cost}(S) = \text{distance}(S) + \mu \cdot \text{overload}(S) + \nu \cdot \text{lateness}(S).$$

Here $\mu \in \mathbb{R}^+$ and $\nu \in \mathbb{R}^+$ are penalty coefficients. They are set by the algorithm.

Algorithm 1: Memetic algorithm scheme

```
1 Generate an initial population;
2 while the stop criterion is not met do
3     Select parents;
4     Apply a crossover and obtain children;
5     Apply a local improvement procedute to each child;
6     Modify the population;
7 return the best solution found
```

3.1. Population Management. The algorithm uses tournament selection to choose parent pairs. We select multiple parent pairs at each iteration. This is done for parallelization reasons: the algorithm is able to generate multiple children and improve them in a parallel manner. Then the algorithm applies *edge assembly crossover* (EAX) [15] or *selective route exchange crossover* (SREX) [4] crossover. The choice is random, and the SREX is selected with the probability 0.1. The SREX creates children by replacing some subset of the routes in the first parent with another subset of routes from the second parent. The replacement sets are chosen by a randomized procedure that considers the longest common subsequence of arcs between the chosen sets. The EAX constructs a child by finding and replacing so-called AB-cycles in the parent. The AB-cycle is a cycle in the graph combined of both parent solutions such that edges from different solutions are linked alternately.

We implement a population management mechanism proposed by Vidal et al. [29]. The mechanism introduces so-called "biased fitness" that is computed considering the objective function of the solution and its distance to other solutions in the population. Let $\text{costRank}(S)$ be the rank of the solution in the population according to its penalized objective function and let $\text{diversityRank}(S)$ be the rank of this solution according to its mean broken-edge distance to the 5 closest solutions in the population. Also, let K be the current population size and let nElite be a parameter of the algorithm describing the number of elite individuals one desires to survive to the next generation. Then the biased fitness function can be calculated using the following expression:

$$\text{fitness}(S) = \text{constRank}(S) + \left(1 - \frac{\text{nElite}}{K}\right) \text{diversityRank}(S) \qquad (5)$$

The algorithm also has two parameters, $K_{\min}$ and $K_{\max}$, corresponding to minimal and maximal population sizes accordingly. At each iteration of the memetic algorithm, all child solutions are always added to the population. If the population size reaches $K_{\max}$ it is truncated to the size $K_{\min}$ by discarding solutions with the worst biased fitness function (5). During the experiments we used $K_{\min} = 50$ and $K_{\max} = 150$.

Experiments showed that this population management scheme is able to keep the mean broken-edge distance between solutions in the population at a level around 30 % edges.

3.2. Large Neighborhood Search.

3.2.1. Removal and insertion operators. The LNS algorithm implements *ruin-and-recreate* strategy and uses several different operators for removal and insertion of clients. Removal operators include:

- *Random removal* that removes requests randomly with uniform distribution.

- *Worst removal* that greedily one-by-one removes request so that penalized objective function improvement maximized.
- *String removal* that selects first node at random and then removes consecutive span of nodes around first one with the corresponding paired nodes.
- *Shaw removal* that selects requests that are similar according to a special measure considering distance and similarities of demands and time windows.

All insertion operators insert one request at a time. For each request k we define the insertion cost $c_k = \text{cost}(S') - \text{cost}(S)$ where S' is obtained from S by inserting the pickup and delivery nodes of the request into their best positions according to the penalized objective function. The selection strategies include:

- *Best insertion* that selects the current best insertion option among all requests and all positions in all routes.
- *Greedy insertion* that shuffles all requests in random order and inserts them in that random order. It select for each request its best insertion position.
- *Regret insertion* calculates for each request i insertion cost c_k^1 to its best route and cost c_k^2 to its second best route. The heuristic selects request that maximizes the difference between these costs: $c_k^2 - c_k^1$.

These removal and insertion operators are described with more detail in the work of Ropke and Pisinger [22].

3.2.2. Adaptive operator selection. The choice of the operator is random at each iteration. The probabilities of each alternative can be adjusted according to the current efficiency of the corresponding operator to use the most effective operators more often. We propose the following adaptive scheme, which is inspired by the ant colony algorithm [5]. The algorithm keeps a vector of weights $[w_1, \ldots, w_k]$, which correspond to each operator. Operators are chosen with probabilities proportional to the weights. After any of the operators is used, its weight is updated by the following scheme:

(1) If an improving solution is obtained, the weight is increased by α ($w_i' = w_i + \alpha$), otherwise it is not changed ($w_i' = w_i$). We used $\alpha = 1$ for the experiments.
(2) The weight is always multiplied by some value β that is between 0 and 1 ($w_i'' = \beta w_i'$). For the experiments we used $\beta = 0.999$.

The first step of the procedure rewards the operator for finding improving solutions. And the second step does not allow the weights to grow indefinitely.

3.2.3. Simulated annealing acceptance. During the search, it is sometimes beneficial to move to a solution that is worse than the current one to explore a new region in the search space. So, we used an acceptance

criterion that takes inspiration from the well-known simulated annealing algorithm [12]. If the new solution has a better penalized objective function than the current one, then the new solution is always accepted. But if the new solution is worse than the current one, it can also be accepted with the probability $\exp\left(\frac{\text{cost}(S_{\text{old}})-\text{cost}(S_{\text{new}})}{T}\right)$, where T is a current "temperature" value. Initially, it equals 5 % of the objective value of the solution. Then it is decreased during the run by multiplying by some value that is less than 1. This value is chosen in such a way that by the end of the run, the probability of accepting a worsening solution equals 0.01.

3.3. Local Search. In addition to the large neighborhood search algorithm, we also implemented a simple local search algorithm that utilizes several different neighborhoods to obtain a locally optimal solution. It uses the following neighborhoods:

(1) *Move*: move a single pickup-delivery pair to a new position in the same or in the different route.
(2) *Swap*: swap 2 pickup-delivery pairs from different routes.
(3) *Swap**: swap the routes of 2 pickup-delivery pairs. Each pair is inserted in the best position in the new route.
(4) *Or-opt*: move a continuous segment in a route to a new position in the same route. We consider only moves that do not violate the pickup-delivery precedence constraints.
(5) *Cross-exchange*: swap continuous segments between 2 routes.

The cross-exchange swap may lead to the infeasibility of a route due to broken pickup-delivery pairs. To mitigate this issue, we also move all paired nodes to the ones that are in the segment. For example, if the segment contains a pickup node, but the corresponding delivery node is not in the segment, then we also move the delivery node to the second route. For these additional nodes we search for the best insert positions that keep the relative order of the nodes.

The swap* and cross-exchange neighborhoods are quite big, and their exploration takes significant time. So we only try to move nodes and segments next to the geometrically close nodes in the remaining route similarly to the way it is done in [30]. Other neighborhoods are fully explored.

These neighborhoods are tried consequently in the order of increasing size. If an improving solution is found in any of the neighborhoods, then the sequence starts from the beginning for the next try. If none of the neighborhoods contain an improving solution, the algorithm stops.

3.4. Auxiliary procedures. The algorithm adapts the penalties for violations depending on the violations in the current population. If the best solution in the population according to the penalized objective function has capacity or time window violations, then the corresponding penalty coefficient is increased:

$$\mu \leftarrow 1.1\mu \text{ or } \nu \leftarrow 1.1\nu.$$

Otherwise, the corresponding coefficient is decreased:

$$\mu \leftarrow 0.9\mu \text{ or } \nu \leftarrow 0.9\nu.$$

To accelerate the evaluation of neighbors during the insertion in LNS and during the exploration of neighborhoods in LS, we used a scheme proposed in [30]. Time windows are relaxed. Upon a late arrival to a customer, we pay for a "time warp" and set the current time to the end of the window. This choice of relaxation is motivated by the availability of efficient penalty evaluation procedures within neighborhood searches. We need to calculate several properties for each subsequence of the initial route for this scheme. Then these properties can be recalculated for a concatenation of sequences in constant time.

If the memetic algorithm cannot find new best solutions for a long time, then a special intensification procedure is launched for each child. This procedure uses the same LNS and LS algorithms but with more aggressive parameters: increased number of iterations and increased number of neighbors. Also, if any of the local improvement algorithms finds an improving solution, then the whole sequence is repeated again.

If the algorithm still cannot find an improving solution, then we reset penalties to their default values. This is done to move to an infeasible area to explore solutions in a different region.

For the case if the population still converges to similar solutions, we propose a way to use a second population. This second population is initialized at the start of the algorithm. But it remains unchanged during the normal operation. If the mean distance between solutions in the main population drops below 8 % of the maximum possible, then the populations are switched, and the algorithm starts to optimize the second population from scratch. Next time the population degrades, we move half of the solutions from the first population to the second and half of the solutions from the second population to the first. When the population converges once more, we switch the populations again and so on. If this switching occurs too frequently, then one of the populations is discarded and regenerated from scratch. This allows the algorithm to keep optimizing even in case it converged to a deep local optimum. The final scheme of the memetic algorithm is presented in algorithm 2.

3.5. Iterated local search. Additionally to the memetic algorithm, we implemented an *iterated local search* algorithm (ILS). It is built from the same components as the memetic algorithm: a local improvement sequence and an acceptance criterion.

It perturbs a solution by performing several random moves. Then it tries to improve the solution by applying the same sequence of algorithms as the memetic scheme uses. If the new solution is good enough according to the acceptance criterion (for example, the simulated annealing criterion from section 3.2.3), then the current solution is replaced by the new one.

Algorithm 2: Final scheme fo the memetic algorithm

```
Generate 2 initial populations;
while the stop criterion is not met do
    Use a tournament selection to choose parents;
    forall parent pairs do
        Apply the EAX or SREX crossover and obtain children;
    forall children in parallel do
        Apply LNS;
        Apply the local search;
    Modify the population using the truncation scheme;
    if all solutions in population are similar then
        Switch populations or migrate solutions;
    if populations are switched too frequently then
        Regenerate one of the populations;
return the best solution found
```

Algorithm 3: Iterated local search algorithm

```
while the stop criterion is not met do
    Perturb the solution;
    Run the improvement sequence;
    if solution is good enough then
        Replace the current solution;
return the best solution found
```

This algorithm serves two purposes:

(1) Intensify the search by improving solutions generated by the memetic algorithm. Every time the memetic algorithm finds a new best solution, this solution is sent to the ILS for asynchronous improvement. After the ILS execution the resulting solution is returned to the population of the memetic algorithm.
(2) Diversify the search by generating new solutions from scratch. ILS periodically generates new solutions and tries to improve them. The resulting solutions are sent to the population of the memetic algorithm.

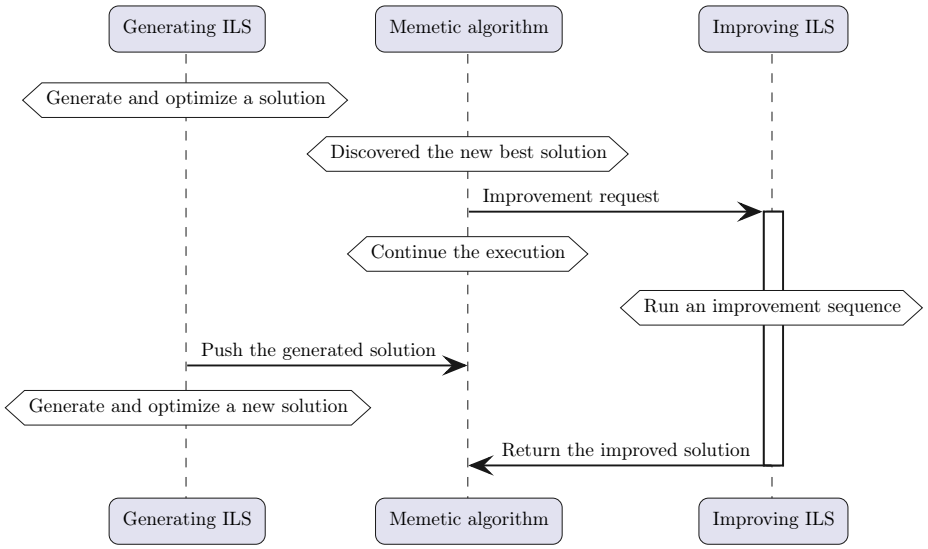

FIGURE 1. Scheme of the communications between the memetic algorithm and ILS

4 Results

In this section we present the experimental results of the proposed scheme. We tested the algorithm using two benchmark data sets: the SINTEF benchmark library[1] and an instance set proposed by Sartori and Buriol [23][2]. We set the number of routes to the same value as in the best-known solutions for each instance. The time limit was as follows: 5 minutes for 100 client instances, 15 minutes for instances with 200 clients, 1 hour for 400-client instances, 4 hours for 600-client instances, 12 hours for 800-client instances, 96 hours for instances with 1000 to 2500 clients and 168 hours for all bigger instances.

To understand the contribution of different components of the algorithm, we performed several runs with disabled sub-algorithms. For each execution type, the algorithm was run 4 times. The table 1 shows the mean values of the objective functions for several instances.

We can see that disabling any of the components leads to worse final solutions. The large neighborhood search has the most impact on the solution quality. Disabling iterated local search affects the scheme in the least way.

The table 2 shows some objective function values of the best-found solutions for the SINTEF library. The results are compared to the best-known solution by the time of the experiments.

[1]https://www.sintef.no/projectweb/top/pdptw
[2]https://github.com/cssartori/pdptw-instances

Table 1. Contribution of algorithm components

Instance	Full scheme	Without LNS	Without LS	Without ILS
LRC2_2_4	2847.3	2909.0	2850.5	2848.2
LRC2_4_4	5314.5	5367.2	5328.0	5327.2
LR2_4_5	10 016.8	11 217.3	10 387.5	10 116.8
LR2_6_4	10 885.0	13 005.6	11 068.2	10 994.3
bar-n200-2	2077.0	2087.0	2082.0	2083.0
nyc-n200-3	1021.0	1067.5	1032.3	1031.0
bar-n400-3	2513.3	2545.2	2532.5	2519.6
nyc-n400-3	1811.3	1811.3	1821.0	1812.0
bar-n600-5	2569.5	2649.5	2580.5	2577.0

Table 2: Best found solutions for the SINTEF dataset

Instance	Node count	Vehicle count	Distance	BKS	Gap, %
LR2_4_1	402	8	9726.88	9726.88	0.0
LR2_4_2	402	7	9405.4	9405.4	0.0
LR2_4_6	402	5	9380.07	8946.91	4.84
LR2_4_10	404	5	8093.95	7596.62	6.55
LR2_4_4	404	4	6205.34	6201.84	0.06
LR2_4_5	404	6	9943.98	9894.46	0.5
LR2_4_8	404	4	5260.42	5260.42	0.0
LR2_4_9	406	6	7930.55	7926.07	0.06
LR2_4_7	406	4	8070.86	7993.16	0.97
LRC2_4_4	406	5	5277.72	5238.77	0.74
LRC1_4_4	414	19	5803.31	5803.31	0.0
LR2_6_4	602	6	10682.5	10639.08	0.41
LRC2_6_4	604	7	9833.93	9833.93	0.0
LR2_6_8	606	4	12576.0	12341.9	1.9
LC2_6_4	612	17	7894.28	7860.38	0.43
LC1_6_4	628	47	13348.5	13300.55	0.36
LR2_8_8	806	5	18692.1	18327.23	1.99
LR2_8_3	808	9	26939.5	26391.07	2.08
LR2_8_7	808	7	26295.3	25502.39	3.11
LR1_8_3	836	44	29408.7	29223.1	0.64
LR2_10_2	1008	14	54716.8	51357.3	6.54
LR2_10_4	1008	8	26838.8	26595.39	0.92
LR2_10_5	1008	13	58811.2	54951.12	7.02
LR2_10_8	1008	6	27621.6	26742.31	3.29
LR2_10_9	1008	12	53479.4	50781.83	5.31
LR2_10_6	1012	11	49097.3	46253.37	6.15
LC2_10_2	1016	30	21035.3	20764.09	1.31

Instance	Node count	Vehicle count	Distance	BKS	Gap, %
LRC1_10_4	1042	40	27315.8	27211.87	0.38
LR1_10_5	1048	58	61769.5	59053.68	4.6

The results show that the scheme is able to find good solutions that are close to or even equal to the best-known ones. However, this library is explored well by many great algorithms, so the best-known solutions are hard to find.

Also, we tested the algorithm at a newer dataset of Sartori and Burio. The table 3 presents the part of the results for those instances[3]. For instances with 400 clients and less, only results with a non-zero gap are shown.

Table 3: Best found solutions for the Sartori and Burio dataset

Instance	Node count	Vehicle count	Distance	BKS	Gap, %
ber-n100-2	100	6	1489.00	1484.00	0.34
nyc-n100-3	100	3	490.00	492.00	-0.41
bar-n200-1	200	22	1819.00	1828.00	-0.49
bar-n200-2	200	23	2073.00	2072.00	0.05
bar-n200-3	200	8	1567.00	1569.00	-0.13
bar-n200-4	200	13	834.00	832.00	0.24
bar-n200-5	200	5	844.00	842.00	0.24
bar-n200-6	200	9	849.00	842.00	0.83
bar-n200-7	200	11	1866.00	1863.00	0.16
ber-n200-1	200	27	3203.00	3197.00	0.19
ber-n200-2	200	12	3254.00	3228.00	0.81
ber-n200-5	200	27	3954.00	3944.00	0.25
nyc-n200-1	200	7	903.00	935.00	-3.42
nyc-n200-2	200	8	1101.00	1102.00	-0.09
nyc-n200-3	200	7	980.00	1017.00	-3.64
nyc-n200-4	200	4	1025.00	1030.00	-0.49
nyc-n200-5	200	5	1181.00	1189.00	-0.67
poa-n200-1	200	25	2426.00	2433.00	-0.29
poa-n200-2	200	12	2417.00	2436.00	-0.78
poa-n200-5	200	15	2320.00	2321.00	-0.04
poa-n200-6	200	27	3140.00	3143.00	-0.10
poa-n200-7	200	10	2581.00	2550.00	1.22
bar-n400-1	400	32	3049.00	3061.00	-0.39
bar-n400-2	400	30	2697.00	2724.00	-0.99
bar-n400-3	400	11	2485.00	2499.00	-0.56
bar-n400-4	400	17	1761.00	1768.00	-0.40

[3]The objectives listed here are as at 01.04.2025.

Instance	Node count	Vehicle count	Distance	BKS	Gap, %
bar-n400-5	400	41	3353.00	3349.00	0.12
bar-n400-6	400	21	2896.00	2886.00	0.35
bar-n400-7	400	11	3056.00	2987.00	2.31
ber-n400-1	400	34	5567.00	5569.00	-0.04
ber-n400-2	400	33	5538.00	5494.00	0.80
ber-n400-4	400	19	2173.00	2216.00	-1.94
ber-n400-5	400	26	5745.00	5918.00	-2.92
ber-n400-6	400	19	6419.00	6280.00	2.21
ber-n400-7	400	20	6477.00	6501.00	-0.37
nyc-n400-1	400	13	1893.00	1898.00	-0.26
nyc-n400-2	400	14	1977.00	1974.00	0.15
nyc-n400-3	400	7	1790.00	1826.00	-1.97
nyc-n400-4	400	7	1942.00	1964.00	-1.12
poa-n400-1	400	24	4457.00	4554.00	-2.13
poa-n400-2	400	41	3064.00	3073.00	-0.29
poa-n400-4	400	19	2108.00	2147.00	-1.82
poa-n400-5	400	14	2280.00	2299.00	-0.83
poa-n400-6	400	41	5426.00	5567.00	-2.53
bar-n600-1	600	43	3669.00	3680.00	-0.30
bar-n600-3	600	22	3860.00	3852.00	0.21
bar-n600-4	600	53	2775.00	2776.00	-0.04
bar-n600-5	600	13	2548.00	2568.00	-0.78
ber-n600-1	600	47	7641.00	7486.00	2.07
ber-n600-4	600	75	11016.00	11124.00	-0.97
nyc-n600-1	600	20	2968.00	2940.00	0.95
nyc-n600-5	600	10	2882.00	2852.00	1.05
poa-n600-2	600	25	5202.00	5358.00	-2.91
poa-n600-5	600	19	2515.00	2547.00	-1.26
poa-n600-6	600	76	7851.00	7875.00	-0.30
bar-n800-3	800	22	5727.00	5872.00	-2.47
bar-n800-7	800	30	5498.00	5554.00	-1.01
ber-n800-3	800	17	3663.00	3639.00	0.66
ber-n800-4	800	105	16066.00	16205.00	-0.86
nyc-n800-2	800	26	3609.00	3850.00	-6.26
poa-n800-1	800	58	9193.00	9213.00	-0.22
bar-n1000-1	1000	51	7795.00	7810.00	-0.19
bar-n1000-3	1000	88	4727.00	4781.00	-1.13
ber-n1000-6	1000	148	18472.00	18990.00	-2.73
ber-n1000-7	1000	71	16745.00	17542.00	-4.54
nyc-n1000-1	1000	27	3986.00	3990.00	-0.10
poa-n1000-3	1000	68	5494.00	5539.00	-0.81
bar-n1500-1	1500	73	9125.00	9501.00	-3.96
ber-n1500-3	1500	69	8804.00	9053.00	-2.75

Instance	Node count	Vehicle count	Distance	BKS	Gap, %
nyc-n1500-3	1500	42	6387.00	6347.00	0.63
bar-n2000-4	2000	71	11543.00	11853.00	-2.62
nyc-n2000-4	2000	26	7239.00	7710.00	-6.11
bar-n2500-4	2500	62	15237.00	16607.00	-8.25
nyc-n3000-1	3000	72	10946.00	10918.00	0.26
nyc-n3000-3	3000	39	12506.00	12467.00	0.31
bar-n4000-3	4000	97	23746.00	24662.00	-3.71
nyc-n5000-3	5000	65	17560.00	20883.00	-15.91

This benchmark is newer and not as well explored as the SINTEF. We were able to improve the best-known solutions for some of these instances.

5 Conclusion

In this paper we consider a classical Pickup and Delivery Problem with Time Windows (PDPTW). We propose a new scheme based on a memetic algorithm for the Pickup and Delivery Problem with Time Windows. The scheme selects parents from the population using tournament selection. Then it applies EAX and SREX crossovers. After that, child solutions are improved using a large neighborhood search and a local search. The LNS algorithm uses an adaptive scheme to select the best removal and insertion operators and a simulated-annealing-like acceptance criterion. The LS algorithm uses 5 different neighborhoods. The memetic algorithm uses a special population management strategy to keep only diverse solutions in the population. Also, it performs several actions to further diversify the search process: using another population, intensification and diversification procedures. Also, we implemented an Iterated Local Search algorithm to help the memetic algorithm. It is used for two purposes: generating new solutions and improving newly found best solutions. The whole scheme is parallelized and can be distributed to several computational nodes.

The proposed scheme demonstrates good results for many instances. It is able to find the best-known or near-best-known solutions for many instances. We were able to improve the best-known solutions for 53 instances.

References

[1] R. Baldacci, E. Bartolini, and A. Mingozzi. *An exact algorithm for the pickup and delivery problem with time windows.* Operations Research, 59(2):414–426, 2011.

[2] P. Beling, P. Cybula, A. Jaszkiewicz, P. Pełka, M. Rogalski, and P. Sielski. *Deep infeasibility exploration method for vehicle routing problems.* In L. Pérez Cáceres and S. Verel, editors, *Evolutionary Computation in Combinatorial Optimization*, pages 62–78, Cham, 2022. Springer International Publishing.

[3] R. Bent and P. V. Hentenryck. *A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows.* Computers & Operations Research, 33(4):875–893, 2006. Part Special Issue: Optimization Days 2003.

[4] M. Blocho and J. Nalepa. *Lcs-based selective route exchange crossover for the pickup and delivery problem with time windows.* In B. Hu and M. López-Ibáñez, editors, Evolutionary Computation in Combinatorial Optimization, pages 124–140, Cham, 2017. Springer International Publishing.

[5] C. Blum. *Ant colony optimization: Introduction and recent trends.* Physics of Life Reviews, 2(4):353–373, 2005.

[6] T. Curtois, D. Landa-Silva, Y. Qu, and W. Laesanklang. *Large neighbourhood search with adaptive guided ejection search for the pickup and delivery problem with time windows.* EURO Journal on Transportation and Logistics, 7(2):151–192, 2018.

[7] G. B. Dantzig and J. H. Ramser. *The truck dispatching problem.* Management Science, 6(1):80–91, 1959.

[8] Y. Dumas, J. Desrosiers, and F. Soumis. *The pickup and delivery problem with time windows.* European Journal of Operational Research, 54(1):7–22, 1991.

[9] M. G. S. Furtado, P. Munari, and R. Morabito. *Pickup and delivery problem with time windows: A new compact two-index formulation.* Operations Research Letters, 45(4):334–341, 2017.

[10] T. Gschwind, S. Irnich, A.-K. Rothenbächer, and C. Tilk. *Bidirectional labeling in column-generation algorithms for pickup-and-delivery problems.* European Journal of Operational Research, 266(2):521–530, 2018.

[11] G. Hiermann and M. Schiffer. *A decomposition-based approach for large-scale pickup and delivery problems*, 2024.

[12] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. *Optimization by simulated annealing.* Science, 220(4598):671–680, 1983.

[13] H. Li and A. Lim. *A metaheuristic for the pickup and delivery problem with time windows.* In Proceedings 13th IEEE International Conference on Tools with Artificial Intelligence. ICTAI 2001, pages 160–167, Nov 2001.

[14] P. Moscato and C. Cotta. *A Modern Introduction to Memetic Algorithms*, pages 141–183. Springer US, Boston, MA, 2010.

[15] Y. Nagata, O. Bräysy, and W. Dullaert. *A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows.* Computers & Operations Research, 37(4):724–737, 2010.

[16] Y. Nagata and S. Kobayashi. *Guided ejection search for the pickup and delivery problem with time windows.* In P. Cowling and P. Merz, editors, Evolutionary Computation in Combinatorial Optimization, pages 202–213, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[17] W. P. Nanry and J. Wesley Barnes. *Solving the pickup and delivery problem with time windows using reactive tabu search.* Transportation Research Part B: Methodological, 34(2):107–121, 2000.

[18] G. Pankratz. *A grouping genetic algorithm for the pickup and delivery problem with time windows.* OR Spectrum, 27(1):21–41, Jan 2005.

[19] A. Pessoa, R. Sadykov, E. Uchoa, and F. Vanderbeck. *A generic exact solver for vehicle routing and related problems.* In A. Lodi and V. Nagarajan, editors, Integer Programming and Combinatorial Optimization, pages 354–369, Cham, 2019. Springer International Publishing.

[20] S. Ropke and J.-F. Cordeau. *Branch and cut and price for the pickup and delivery problem with time windows.* Transportation Science, 43(3):267–286, 2009.

[21] S. Ropke, J.-F. Cordeau, and G. Laporte. *Models and branch-and-cut algorithms for pickup and delivery problems with time windows.* Networks, 49(4):258–272, 2007.

[22] S. Ropke and D. Pisinger. *An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows.* Transportation Science, 40(4):455–472, 2006.

[23] C. S. Sartori and L. S. Buriol. *A study on the pickup and delivery problem with time windows: Matheuristics and new instances.* Computers & Operations Research, 124:105065, 2020.
[24] M. Savelsbergh and M. Sol. *Drive: Dynamic routing of independent vehicles.* Operations Research, 46(4):474–490, 1998.
[25] M. W. P. Savelsbergh and M. Sol. *The general pickup and delivery problem.* Transportation Science, 29(1):17–29, 1995.
[26] L. Schrage. *Formulation and structure of more complex/realistic routing and scheduling problems.* Networks, 11(2):229–232, 1981.
[27] P. Toth and D. Vigo. *Vehicle Routing*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2014.
[28] S. T. Vadseth, H. Andersson, J.-F. Cordeau, and M. Stålhane. *Mixed integer programs to improve solutions of vehicle routing problems with intra-route constraints*, 2023. Preprint available at AXIOM Research Project.
[29] T. Vidal, T. G. Crainic, M. Gendreau, N. Lahrichi, and W. Rei. *A hybrid genetic algorithm for multidepot and periodic vehicle routing problems.* Operations Research, 60(3):611–624, 2012.
[30] T. Vidal, T. G. Crainic, M. Gendreau, and C. Prins. *A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows.* Computers & Operations Research, 40(1):475–489, 2013.

Alexander Dmitrievich Yuskov
Novosibirsk State University,
Pirogova, 1,
630090, Novosibirsk, Russia
Email address: a.yuskov@g.nsu.ru

Igor Nikolaevich Kulachenko
Sobolev Institute of Mathematics,
pr. Koptyuga, 4,
630090, Novosibirsk, Russia
Email address: ink@math.nsc.ru

Yury Andreevich Kochetov
Sobolev Institute of Mathematics,
pr. Koptyuga, 4,
630090, Novosibirsk, Russia
Email address: jkochet@math.nsc.ru