mathematics

MDPI

Article

Comparison of Genetic Operators for the Multiobjective Pickup and Delivery Problem

Connor Little *, Salimur Choudhury, Ting Hu and Kai Salomaa

School of Computing, Queen's University, Kingston, ON K7L 3N6, Canada
* Correspondence: connor.little@queensu.ca

Abstract: The pickup and delivery problem is a pertinent problem in our interconnected world. Being able to move goods and people efficiently can lead to decreases in costs, emissions, and time. In this work, we create a genetic algorithm to solve the multiobjective capacitated pickup and delivery problem, adapting commonly used benchmarks. The objective is to minimize total distance travelled and the number of vehicles utilized. Based on NSGA-II, we explore how different inter-route and intraroute mutations affect the final solution. We introduce 6 inter-route operations and 16 intraroute operations and calculate the hypervolume measured to directly compare their impact. We also introduce two different crossover operators that are specialized for this problem. Our methodology was able to find optimal results in 23% of the instances in the first benchmark and in most other instances, it was able to generate a Pareto front within at most one vehicle and +20% of the best-known distance. With multiple solutions, it allows users to choose the routes that best suit their needs.

Keywords: optimization; vehicle routing; genetic algorithm; local search; pickup and delivery

MSC: 90C59

Citation: Little, C.; Choudhury, S.; Hu, T.; Salomaa, K. Comparison of Genetic Operators for the Multiobjective Pickup and Delivery Problem. *Mathematics* **2022**, *10*, 4308. https://doi.org/10.3390/math10224308

Academic Editors: Abdellah Chehri and Francois Rivest

Received: 27 September 2022
Accepted: 10 November 2022
Published: 17 November 2022

1. Introduction

The pickup and delivery problem is a problem that has gained significant popularity since its inception. Much of its interest is due to strong applications in industry across several important problems such as supply chain routing, distribution, ride hailing, food delivery, etc. [1]. In recent years, in part due to the emergence of COVID-19, these problems have been brought to the forefront of the public consciousness. It has been harder to fulfill the demands of a global population. Supply chains have been hit particularly hard leading to a sharp decrease in the number of goods that have been shipped [2]. The need for robust and efficient solutions is more important than ever.

The pickup and delivery problem is a variation on vehicle routing [3]. Specifically, this paper addresses the multiobjective capacitated pickup and delivery problem (PDP) with time windows. Vehicle routing, in turn, is a generalization of the travelling salesman problem. Vehicle routing expands upon its predecessors by allowing multiple routes and multiple vehicles while still maintaining the goal of minimizing the total distance or time travelled. The pickup and delivery problem further expands on vehicle routing by adding precedence to pairs of nodes, pickup nodes and delivery nodes [3]. Each pickup node must be visited prior to the corresponding delivery node. The added precedence constrains the problem in unique ways such that many algorithms developed for regular vehicle routing must be altered. The added constraints of capacity and time windows further restrict operations on the solution.

The pickup delivery problem also has some additional constraints which are important to this version of the problem. All vehicles start and end at the same location called a depot. The fleet is assumed to have all the vehicles be the same and travel at uniform speeds to simplify the problem. It is assumed that no node is visited twice as the pickup and

delivery only needs to be completed once. If a vehicle must travel through a node to arrive at another node, this is simply ignored.

As pickup delivery with time windows is NP-hard [1], there have been many attempts to develop heuristics. One class of heuristics which have had success in solving the multiobjective PDP is genetic algorithms. With the introduction of NSGA-II (Non-dominated Sorting Genetic Algorithm II) [4], evolutionary algorithms have a strong framework for modelling this kind of problem. Standard techniques to approach multiobjective problems include assigning weights to different objective values or solving them one at a time. The latter essentially turns the problem into a sequential single-objective problem. NSGA-II introduces nondominated sorting which allows solutions to be ranked according to numerous objectives without specifying precedence or weights, by sorting solutions into dominating fronts. Further, a crowding distance allows the comparison between within each front while still avoiding the previous issues. This allows better solutions to propagate without placing bias or preference to any one solution.

The large majority of current research into the pickup and delivery problem is single objective. This is insufficient in practice in many cases. Real-world scenarios are dynamic with many factors such as profit, vehicle count, or greenhouse emissions being factors in determining routes. Even the visual nature of the routes can determine whether a solution will be utilized in practice [5]. The research that does explore multiobjective pickup and delivery often reduces it down into a weighted single-objective problem, adding bias and eliminating diverse solutions [6]. This area of research has also grown with the recent popularity in green vehicle routing. Green vehicle routing often attempts to minimize environmental impacts alongside reducing costs and distance, making it a prime candidate for multiobjective techniques.

To improve upon NSGA-II, mutation and crossover operators can be specified. There are many different mutation and crossover operators that can be utilized for travelling salesman problems and their variants [7].

This paper introduces several local search operators to quantify and compare them. There are two main classifications of operators that are covered: inter-route operations and intraroute operations. As multiple inter-route operations need to be included in order to cover the search space, an ablation study is utilized to explore each operator's effectiveness. Intraroute operations are compared directly.

The paper is structured as follows. Section 2 reviews the current state of works in the field of multiobjective pickup and delivery. Section 3 formally introduces the problem and supply a linear programming model for the problem. Section 4 introduces the genetic algorithm and explains its properties. Section 5 shows results and finally, Section 6 concludes with a discussion of the results and future directions.

2. Background and Related Work

The pickup and delivery problem is a well-researched problem within the literature. There are numerous different variations and distinctions, each presenting different challenges. A full taxonomy was constructed by Berbeglia et al. [8].

Being a combinatorial optimization problem, much of the research is looking into heuristics to speed up computation time. Multiobjective capacitated pickup and delivery with time windows (MOCPDPTW) is an extension of the vehicle routing problem. The highly constrained nature of the problem makes designing heuristics a unique challenge. One important paper for multiobjective problems is "A fast and elitist multiobjective genetic algorithm: NSGA-II" by Deb et al. [4]. This paper proposed an efficient framework for multiobjective evolutionary algorithms. Based on nondominated sorting, fronts were assembled. Fronts are collections of solutions based on how many other solutions are dominated. This allowed direct comparisons of groups of solutions.

Evolutionary algorithms such as genetic algorithms and memetic algorithms are common approaches to solve the multiobjective pickup and delivery problem. Bravo, Rojas, and Parada [9] focused on green vehicle routing, specifically on reducing pollution.

They introduced an MO-PRP (multiobjective pollution routing problem) model which considered customers serviced, distance travelled, and fuel consumption. This could involve introducing additional variables to capture each objective. Chami et al. [10] offered a hybrid algorithm that combined genetic algorithms with a local search to optimize distance and cost. They did not cover time windows in their formulation. Fatemi-Anaraki et al. [11] also offered a hybrid genetic algorithm which first clustered the nodes before creating an initial population using one genetic algorithm. After the population had been generated, NSGA-II was run to find the final solutions. Fatemi-Anaraki et al. [11] aimed to minimize greenhouse emissions and cost of travel. Their formulation also did not contain time windows. Garcia-Najera and Gutierrez-Andrade [12] attempted to solve the multiobjective capacitated pickup and delivery problem with time windows by designing their own evolutionary algorithm based on solution domination. Gong et al. [13] used a bee-inspired algorithm to solve the MOCPDPTW. Their framework combined NSGA-II with the bee-evolutionary-guided algorithm to minimize fuel consumption, waiting time, and distance. Again, their model abstained from considering time windows. Li, Sahoo, and Chiang [14] designed their evolutionary algorithm based on R2 indicators. Velasco et al. [15] formulated their problem assuming the vehicles would be helicopters with no need for time windows. They designed a genetic algorithm based on NSGA-II as well and improved upon it with local search operators. Wang and Chen [16] explored genetic algorithms with numerous different mutations in order to minimize vehicles and minimize travel distance. Finally, Zhu et al. [17] introduced a memetic algorithm with locality-sensitive-hashing local-search operators. They did not include time windows in their analysis.

There has been much exploration outside of the field of evolutionary computing as well. Grandinetti [18] solved the problem by an ϵ-constraint method. This involved iteratively solving constrained single objective functions to approximate the Pareto front. Ren et al. [19] designed a variable neighbourhood search algorithm. Their methodology generated the solutions, perturbed them with nonoptimal search operators, before improving them again in hopes of leaving local optima. Wang et al. [6] compared two different frameworks for MOCPDPTW: multiobjective local search (MOLS) and multiobjective memetic algorithms (MOMA). Zou, Li, and Li [20] used particle swarm optimization hybridized with a variable neighbourhood search.

The majority of the work done in the domain of vehicle routing and pickup and delivery has been conducted on single objective functions. This is slowly changing as the demand and the types of problems encountered change. The number of multiobjective papers has been increasing in recent years. Previous surveys mention very few instances of MO-PDP [21] but in the last 5 years, there has been papers by Chami et al. [10], Li et al. [14], Gong et al. [13], and Bravo et al. [9], amongst others. The consequence of single-objective work taking the majority of the attention is that many ideas have not been applied in this domain. Carrabs, Cordeau, and Laporte [22] worked on the single-objective version of the problem and introduced novel local search operators based on combining pickups and deliveries into single entities. In recent years, the exploration of multiobjective problems has become more prevalent. The increase in attention is driven in part by green vehicle routing. Green vehicle routing aims to not only reduce the distance and the number of vehicles but also to reduce the emissions the routes will produce.

One drawback of all of these papers is that most abstain from describing their methodology in full. The mutation and crossover operators are overlooked or implemented far more simply than they need to be. The work by Bravo et al. [9] did not mention which operators were chosen, making replication and derivation of their work difficult. Others, such as the work by Chami et al. [10], only tested one operator: the swap operator. This operator, while a classic genetic algorithm operator, does not take into account the structure of the problem. Our work aims to further help the creation and study of genetic algorithms by giving other researchers a jumping-off point when creating their algorithms. We aim to compare and contrast how different operators affect the final solutions so that a more intelligent algorithm design can be implemented.

3. Problem Definition

The capacitated pickup and delivery with time windows problem is built on a complete directed graph $G = \{N, A\}$. N is the set of nodes and is further broken down into $N = \{0\} \cup P \cup D$, where 0 is the depot, $P = \{1, 2, \ldots, n\}$ is the set of all pickups and $D = \{n+1, n+2, \ldots, 2n\}$ is the set of all deliveries. n is the number of requests. Pickup and delivery locations always come in pairs. Explicitly, for any pickup i, the corresponding delivery is $n + i$ with n being the number of pickup and delivery pairs. The depot defines the starting and ending location for all vehicles.

For each pickup or delivery node, there is additional information supplied. Given a node $i \in N$, there exists q_i, d_i, ETW_i, and LTW_i. q_i designates the demand at node i. For pickups, this represents the space needed in the vehicle to pick an item up and is positive, while for deliveries it is negative to represent the removal of an item from the vehicle. d_i is the service time at each node. This represents the amount of time it takes to perform a pickup or delivery. Finally, [ETW_i, LTW_i] are the early time window and late time window, respectively. This represents the time when a vehicle can visit and the service can be performed. Should a vehicle arrive prior to ETW_i, it has to wait, and should a vehicle arrive after LTW_i, the route is invalid.

We are also given a set K of vehicles. Each vehicle must keep track of how much it is carrying and how long it has been travelling. Let Q_i, k be the total capacity of a vehicle at a given node. As a vehicle traverses through the graph, the latter is updated to add q_i. For nodes a vehicle does not visit, this value is irrelevant. Let B_i, k be the time at which a vehicle has arrived at a given node. For each visited node, this should add the travel time and the service time of the node. Again, should a vehicle not visit a node, this value is irrelevant.

A is the set of all edges between the nodes $A = \{(i, j) | i, j \in N, i \neq j\}$. Each element (i, j) in A has an associated cost $C_{i,j}$. This typically represents distance or time to travel.

In addition to G, we also receive the max capacity of each vehicle Q. As this version of the problem has a homogeneous fleet, it is a constant. The max route time is implicitly supplied by the latest time that the depot may be visited. Again, we are assuming a homogeneous fleet, so all vehicles travel at the same speed and have the same capacity.

Let x_i, j, k be a decision variable to determine if a vehicle k travels from node i to node j. This is a multiobjective problem so there are two objective functions to minimize.

$$\min \sum_{i \in N, k \in K} x_{i,0,k}$$

Objective 1 aims to minimize the total number of vehicles used.

$$\min \sum_{i,j \in N, i \neq j} c_{i,j} * x_{i,j,k}$$

Objective 2 minimizes the total travel time that a vehicle takes. This does not take into account waiting time to not incentive idling.

With these objectives, the following constraints are added to construct a linear programming model:

$$\sum_{k \in K} \sum_{j \in N} x_{i,j,k} = 1 \; \forall i \; \in N \tag{1}$$

$$\sum_{j \in N} x_{i,j,k} - \sum_{j \in N} x_{n+i,j,k} = 0 \; \forall \, i \in P, k \in K \tag{2}$$

$$\sum_{j \in N} x_{0,j,k} = 0 \; \forall \, k \in K \tag{3}$$

$$\sum_{j \in N} x_{j,i,k} - \sum_{j \in N} x_{i,j,k} = 0 \; \forall \, i \in N, k \in K \tag{4}$$

$$\sum_{j \in N} x_{j,0,k} = 0 \; \forall k \in K \tag{5}$$

$$x_{i,j,k} * (Q_{i,k} + q_j) <= Q_{j,k} \; \forall \, i \in N, j \in N, k \in K \tag{6}$$

$$max[0, q_i <= Q_{i,k} <= min[Q, Q + q_i] \; \forall \, i \in N, k \in K \tag{7}$$

$$x_{i,j,k} * (B_{i,k} + c_{i,j} + d_j) <= B_{j,k} \; \forall \, i \in N, j \in N, k \in K \tag{8}$$

$$b_{i,k} + c_{i,n+i} + q_{n+i} <= b_{n+i,k} \; \forall \, i \in P, k \in K \tag{9}$$

$$ETW_i <= B_{i,k} <= LTW_i \; \forall \, i \in N, k \in K \tag{10}$$

$$x_{i,j,k} \in [0, 1] \tag{11}$$

Constraint 1 enforces that each node is visited once and only once across all vehicles. As the pickup delivery problem with time windows assumes a complete graph, it is assumed that any intermediate stops are irrelevant. Constraint 2 is to enforce that pickup and delivery pairs are in the same route. If a vehicle picks up a product, it must also be the one to deliver it. Constraints 3, 4, and 5 ensure that a subroute is consistent and both starts and ends at the depot. In other words, the vehicle must start and stop at the depot, while also making a cycle. Constraints 6, 7, and 8 guarantee that the routes always arrive within the allowed time window. Equations (9) and (10) guarantee that a vehicle always has a sufficient capacity for the route it is assigned to. Lastly, Constraint 11 enforces that the decision variable be a Boolean variable.

The above is a 3-index model of the pickup and delivery problem with time windows, constructed by [23]. With mixed-integer programming, the objectives are solved hierarchically. First, the minimum number of vehicles are found by solving the model with only one objective. The number of vehicles is then set constant by adding an equality constraint, and the model is rerun with the second objective function to find the minimum distance.

4. Genetic Algorithm

The motivation behind constructing a genetic algorithm heuristic is the size of the problem. MOCPDPTW is NP-hard [1], as it extends the vehicle routing problem (VRP) which is provably NP-hard. This makes finding solutions increasingly difficult as the size of the problem increases. Using Gurobi, the three-index model was unable to find solutions to 50 request instances within the time limit of an hour. The two-index model [24] was able to find solutions but they were worse than using simple construction heuristics such as the cheapest insertion method. Heuristics are required as they trade speed for solution quality. For unexplained notations and for those unfamiliar with evolutionary computing, the reader is referred to a review by Katoch et al. [25] or the introduction by Mitchell [26]. A survey on genetic algorithms with respect to capacitated vehicle routing is provided by Karakatič and Podgorelec [21].

4.1. Solution Representation

For this problem, we encoded a solution (chromosome in genetic algorithms) as an array of arrays. Each array in the outer array represented the route a vehicle would take. Each route was a permutation of nodes sampled from N. An example route can be seen in Figure 1. For each pickup, the corresponding delivery, x + n, appears after. Each route is implicitly known to start and end at the depot, so those nodes are added during the evaluation step.

3	4	2	(4 + n)	(3 + n)	(2 + n)	1	5	(1 + n)	(5 + n)

Figure 1. Example route with 5 pickups and 5 deliveries; n = 5.

4.2. Initial Population

The populations were initialized using the insertion heuristics as construction heuristics. First, we predicted an upper bound on the number of vehicles that were available to

cap the number of routes generated. Each route was seeded with a random request. This ensured that each individual would be different. Afterwards, a cheapest parallel insertion heuristic was used. This algorithm is explained in Algorithm 1. Given a solution, each request is inserted into each possible location and the cost is calculated. The solution with the cheapest cost is kept and the process is repeated with the next request. This inserts the request which minimizes the total route time. Once each request has been inserted, a 2-opt algorithm is run on each route to improve the initial solutions.

Algorithm 1 Parallel Insertion

Input: Insertion heuristic H, insertion operator I, local search operator O, number of routes K

Output: A feasible solution

```
routes ← List of K empty lists
Let requests be a set of pickup and delivery pairs
Initialize each route in routes with a randomly chosen request from requests
Remove each inserted request from requests
while Not all requests are inserted do
    newSolution ← None
    bestRequest ← None
    for each idx, route in routes do ▷ At the end newSolution contains the best request inserted in the best location
        Choose request with H
        Insert request into newRoute with I
        Improve newRoute with O
        if newRoute is feasible then
            if newSolution is None then ▷ If this is the first valid insertion
                newSolution ← routes
                newSolution[idx] ← newRoute
                bestRequest ← request
            else
                tempSolution ← routes
                tempSolution[idx] ← newRoute
                if tempSolution is better than newSolution then
                    newSolution ←tempSolution
                    bestRequest ← reqest
                end if
            end if
        end if
    end for
    if newSolution is None then ▷ The request cannot be inserted anywhere
        bestRequest ← random request from requests
        Append routes with new route containing bestRequest
    else
        routes = newSolution
    end if
    Remove bestRequest from requests
end while
return routes
```

4.3. Evaluation

Our genetic algorithm utilized NSGA-II to enable multiple objectives. The first objective was to minimize the total distance over all routes. This did not include waiting time or service time. Service time was constant across all nodes, so adding it did not change the solutions relative to each other. The waiting time was the time during which a vehicle was simply sitting idle. This could occur if a vehicle arrived prior to the earliest time window.

The second objective was to minimize the number of vehicles needed. This evaluation step was different from the linear programming version as nondominated sorting was used instead of hierarchical methods.

4.4. Selection

Selection followed the standard given by NSGA-II [4]. For parent selection, a binary tournament was employed with replacement. Parents were chosen iteratively until the number of parents was equal to two times the population. This allowed for the number of offspring to be equal to the population. The offspring were then generated by performing a crossover and a mutation operation before being added into the population. Selecting the next generation was done by sorting the combined offspring and prior population into fronts by which nodes they dominated/were dominated by. They were then sorted within each front by the crowding distance. The individuals were then chosen based hierarchically on their front, followed by their crowding distance until the new population was the same size as the old population. This framework is the same as the $(\mu + \lambda)$ framework [26].

4.5. Crossover

Due to the highly constrained nature of the problem, a specialized crossover function was used. The crossover function began by initializing an empty solution. Iteratively, a route was selected from each parent until no route was left in either parent. If that route contained only pickup and delivery pairs which had not been seen prior, the route was appended to the solution as is. If a route had a node which had already been included, that pickup delivery pair was removed, and the rest of the route was kept intact. The shortened route was then added to the solution. At the end, the routes added got smaller due to nodes being removed. As a final optimization step, all routes with 2 or fewer pickup and delivery nodes were removed from the solution, and the pickup and delivery pairs were extracted. These requests were then reinserted into the solution in a parallel fashion. The final solution was then returned. The intuition for allowing partial routes was that it did not separate requests that were often paired together. This crossover function was called route crossover with ejection. For an example of how this works, see Figure 2.

First Parent	2, 2+n, 3, 3+n	1, 1+n	5, 5+n, 4, 4+n	
Second Parent	2, 1, 2+n, 4, 4+n 1+n,	3, 3+n, 5, 5+n		
Offspring 1st Iteration, route 1 is taken	2, 2+n, 3, 3+n			
Offspring 2nd Iteration, route 2 is taken	2, 2+n, 3, 3+n	5, 5+n		
Offspring 3rd Iteration, route 3 is taken	2, 2+n, 3, 3+n	5, 5+n	4, 4+n	
Offspring Final Iteration, route 1 is taken	2, 2+n, 3, 3+n	5, 5+n	4, 4+n	1, 1+n

Figure 2. Example of crossover without ejection. The final offspring would then have all "small" routes removed and reinserted.

A second crossover operator was tested. Developed by Wang et al. [6] and Alvarenga and Mateus [27]. Wang et al. [6] and Alvarenga and Mateus [27] chose routes iteratively but only accepted routes that could be added in their entirety. Those that could not had all nodes set aside to be reinserted into the surviving routes.

In our trials it was found that the first crossover operator produced slightly better results, so in the results, we opted to use that one. A crossover rate of 1/5 was used. The crossover algorithm is described in Algorithm 2.

Algorithm 2 Crossover Operation

Input: Parent A, parent B **Output:** A single offspring C

```
offspring ← []
while A and B are not empty do
    Select a random subroute from A
    for each elem in subroute do
        if elem in offspring then
            Remove elem from subroute
        end if
    end for
    Append subroute to offspring
    Select a random subroute from B
    for each elem in subroute do
        if elem in offspring then
            Remove elem from subroute
        end if
    end for
    Append subroute to offspring
    for each subroute in offspring do
        if size of subroute <=2 then
            Reinsert all element into routes in offspring
        end if
    end for
end while
Return offspring
```

4.6. Mutation

Mutation was divided into two stages. The first step was to perform an inter-route operation. This moved nodes between each subroute. The second step was to perform intraroute optimizations. After a route was chosen, it searched an operational neighbourhood for an improved solution. The motivation behind exploring different mutation operators stemmed from the lack of diversity within the literature. Of those that employ intra-route operations, swap mutations are by far the most common mutation. Chami et al. [10], Gong et al. [13], Garcia-Najera and Gutierrez-Andrade [12], and Zhu et al [17] all used a variation of this operator.

4.6.1. Inter-Route Operations

There were six inter-route mutations applied.

- Mutation 1: single-pair relocation
  - Removes a single pickup and delivery pair from a random route and attempts to insert it into another route.
- Mutation 2: double-pair relocation
  - Randomly selects 2 routes and attempts to swap a pickup and delivery pair between them.
- Mutation 3: customer relocation
  - Randomly picks a route and attempts to add a random pickup and delivery pair.
- Mutation 4: best-customer relocation
  - Randomly picks a route and attempts to add the pickup and delivery pair according to a heuristic

- Mutation 5: route ejection
  - Selects a route and unassign all pickup and delivery pairs. Afterwards, it attempts to insert all of them.
- Mutation 6: route divide
  - Selects a route and creates 2 new routes out of the pickup and delivery pairs.

These 6 mutations were given an equal probability of occurring. Each of these mutations required an insertion operator. To make the most general insertion operators, we followed the algorithms as described in Algorithms 1 and 3. These inter-route operations were inspired by the works of Wang and Chen [16] and Yanik, Bozkaya, and Dekervenoael [28].

Algorithm 3 Sequential Insertion

Input: Insertion heuristic H, insertion operator I, local search operator O
Output: A feasible solution

```
routes ← []
Let requests be a set of pickup and delivery pairs
while True do
    newRoute ← []
    while Not all requests are inserted do
        tempRoute ← newRoute
        Choose request with H
        Insert request into newRoute with I
        Improve newRoute with O
        if newRoute is feasible then
            Remove request from requests
        else
            Append tempRoute to routes
            break
        end if
    end while
    if requests is empty then
        return routes
    end if
end while
```

For the sequential insertion, a heuristic, an insertion operator, and a local search operator were supplied. Starting with an empty solution, routes were built iteratively by choosing a request based on the insertion heuristic and the insertion operator. Once the optimal request had been inserted, the resulting route was improved by the local search operator until it could not be improved anymore. If at any point a new pickup and delivery pair could not be inserted, the route was added to the solution as is and a new route was started with the previously uninserted pair. These processes were repeated until all requests were inserted.

The parallel insertion heuristic worked very similarly. Given a starting number of vehicles k, k routes were initialized with a randomly chosen request. The optimal request across all routes were chosen via the insertion heuristic and operator. Only one route was improved at each iteration. From there, the route was improved with the local search operator and inserted into the solution. If a request could not be inserted into any route, a new route was appended much like the sequential variation.

4.6.2. Insertion Operators

Another consideration in the design was which insertion operators to use. As mentioned previously one can insert in both parallel and sequential fashions. There are also several heuristics to improve which requests get inserted and where to insert. Common

methods include the cheapest insertion, which inserts the request that minimizes the total distance, the furthest insertion, which maximizes the total distance, and the random insertion, which places a random request.

We found that choosing requests via the cheapest insertion method was the most useful method. Moreover, our trials found that the number of vehicles had little effect on the final result. As such, choosing a number of initial routes for the parallel insertion operator was not impactful. In the end, we chose to use the parallel construction heuristic for our initial populations. While both produced similar results, the parallel construction was able to create more varied populations and therefore had more diversity throughout.

4.6.3. Intraroute Operations

In addition to insertion operators, the routes are often further optimized with local search operators. Multiple local search operators are tested and explained in Table 1 below. Some were standard genetic algorithm operators such as the swap mutation while others were more problem-specific such as the blocked 2-opt operator. A blocked operation involves grouping the pickup and delivery pairs into single entities. The idea is to keep pickup and delivery pairs together. Sequentially going down a list, the nodes are added into a bin starting with a pickup and until the corresponding delivery node is reached. After the first bin has been filled, the route is restarted at the next pickup node and the process repeats until all pickup and delivery pairs are considered. All nodes between a pickup and delivery pair are included in the group and as such this results in multiple copies of some nodes. The original decoding by [22] was LIFO (last in, first out) and assumed that there would never be any overlap. To address this, only the first copy of a node was kept when converting back into a normal route. This can be further seen in Table 2. Operations were then performed on these groups instead of individual elements. The reasoning for this was to preserve precedence. If pickup and delivery pairs moved together, it was impossible for the precedence to be violated.

4.7. Datasets

When choosing a dataset, there is often many factors to consider. For nonstandard PDP, there is no consensus on a benchmark dataset, with most papers generating their own [1]. While this does allow data to be curated for any problem, there is the issue on how representative of real-world scenarios the synthesized data will be.

One dataset we elected to use comes from Sulzbach Sartori and Buriol [29]. This dataset is an open-source dataset based on geographical data from capital cities. It supplies several instances of varying node counts and incorporates real-world travel time to ensure that it is representative of actual data. We used 25 instances with 100 nodes. The input instances were labelled with the city they were based on, the number of nodes, and the instance number. In this case bar-n100-2 would be the second instance in Barcelona with 100 nodes.

In addition, we also used the well-known Li Lim [30] dataset. This is a commonly used benchmark dataset for the pickup and delivery problem. It uses Euclidean distances between points and hierarchically solves for the number of vehicles and then distance. Li Lim [30] distinguished their instances based on how the nodes were arranged. Lr instances were randomly distributed, lc instances were clustered, and lrc instances were partially distributed randomly.

Table 1. Descriptions of each local search operator.

Mutation	Description
Swap	Selects 2 nodes in a route and attempts to switch them.
Displacement	Selects a subroute of a fixed size within a route and translates it.
Insertion	Special case of displacement mutation with size 1.
Gaussian displacement	Selects a subroute of a randomly chosen size within a route and translates it.
K-Opt	Removes k edges and attempt to reconnect them.
Blocked K-Opt	Combines nodes into request blocks, removes k edges and attempts to reconnect them.
End request swap	Swaps the delivery of one request with the pickup of another later on.
Request Swap	Swaps a pickup and delivery with another pickup and delivery node within the same route.
Boundaries	Half of the time, performs a request swap and half of the time, performs an end request stop.

Table 2. Example of a route that has been blocked.

3, 4, 2, 4 + n, 3 + n	4, 2, 4 + n	2, 4 + n, 3 + n, 2 + n	1, 5, 1 + n	5, 1 + n, 5 + n

Both datasets can be trivially adapted into multiobjective instances. Moreover, both datasets come with the best-known solution which was treated as the optimal solution with respect to the lower bound and for calculating optimality. Optimality gaps were calculated through the equation

$$1 - (\text{Found solution}/\text{best known solution})$$

5. Results

All instances were run for a max time of 30 min or a max epoch of 300 on an i7-9750H CPU with 16 GB of RAM. A population of 50 was utilized. A time limit of 1800 s was applied, should the algorithm fail to terminate within that time. The total results for each instance can be found in Appendix A.

The max epochs of 300 was chosen arbitrarily such that the time limit was the more important factor. This allowed the algorithms to compare highly complex operations against very efficient operators without heavily biasing the results towards search techniques with larger neighbourhoods. Measuring epochs instead of time limits biases the algorithm towards complex and costly operations. Very rarely did instances hit the epoch limit as opposed to the time limit. In this instance, convergence meant that all genomes within the population had the same fitness value, essentially reducing the diversity to 0. Convergence is important as if the population is still very diverse, it means that the local optimum has not been reached yet, while if the diversity is 0, then no more learning can be done. Test runs were run in 10 min intervals on the first five instances of the Li Lim dataset in order to empirically choose these values. The 2-opt operator was chosen for these runs. These runs held all other parameters constant. For a time limit of 10, the algorithm did not converge at all, still having around 16 fronts on average. In four out of five cases the 20 min test run converged, while when given 30 min, all test cases converged. As the time got longer the

solutions got better. A final time limit of 30 min was chosen to allow a greater chance at convergence, especially given more complex operators. On the final run, with a 30 min time limit, the algorithm converged about 25% of the time.

The population size was chosen in much the same fashion. Using a grid search technique, population sizes of 25, 50, and 100 were tested. A time limit of 30 min was allotted. On the five test instances, a populations size of 50 was found to perform the best. A population size of 25 had the population converge very early on, preventing further learning from taking place. With a population size of 50, there was still some diversity within the population. A population size of 100 was far too large for the problem. Within 30 min, none of the five test cases had converged and the four instances had upwards of 60 fronts. Of the resulting best solutions, all five came from the test case where the population size was 50. With 25 and 50, the optimal solution was found twice.

The results from the Li Lim dataset can be seen in Table A1. This table lists the instance name, the best solution and our found solution. The solutions are in the form of number of vehicles, distance travelled. Our algorithm was able to find the optimal (best known) result in 13 out of 56 instances. In the rest of the trials, we were able to find results within one vehicle and within 10% in most cases for the distance. The worst result we achieved was on the instance lr205 in which our three-vehicle solution was only 27% within optimality in regard to distance. Our five-vehicle solution was within 13%, however. Four of our results were 20% or higher, while in forty-three instances our result was within 10% of optimality. Omitting the lr 200 instances, the average distance optimality gap was 4% as seen in Table 3. When nodes were randomly distributed, our algorithm performed the worst. The proposed algorithm performed the best when the nodes were clustered, in which 9 out of 17 were optimal. In three cases, our algorithm was able to reduce the total distance to below that of the best-known solution. Often, the genetic algorithm would converge to a local optimum and would then cease learning. The addition of the 4-opt mutation was able to help remove solutions from this pool on occasion.

Table 3. Summary on Li-Lim benchmark.

Instance	Distance Optimality Gap	Vehicle Optimality Gap
lc 100 instances	−0.0462	0.0416
lc 200 instances	0.02983	0.0
lr 100 instances	0.0402	0.0790
lr 200 instances	0.1364	0.2424
lrc 100 instances	0.0441	0.0658
lrc 200 instances	0.04779	0.1562

The results from the Sulzbach Sartori and Buriol's [29] dataset can be seen in Table A2, and summarized in Table 4. This dataset was more complex than the previous one, with our algorithm not always converging within a 30 min time limit. As such, our algorithm was not able to solve to optimality in any of the instances. Despite this, we were able to solve the problem in every case and produce solutions within one vehicle and an average of 2.97% of the optimal distance on average. Distancewise the worst solution was bar-n100-6 with an optimality gap of 11.81%. It was one of two total solutions with a gap larger than 10%. It is not surprising that we did not find that many optimal solutions within the specified time limit. Our algorithm had a lot of overhead dedicated to finding many feasible solutions as opposed to finding one optimal solution. Maintaining multiple unique genomes allow a greater diversity and more options for choosing a final solution.

Table 4. Summary on Sulzbach Sartori and Buriol's benchmark.

Instance	Distance Optimality Gap	Vehicle Optimality Gap
bar instances	0.0550	0.0694
ber instances	0.0292	0.1360
nyc instances	0.0388	0.2666
poa instances	0.0019	0.1170

A secondary study was conducted to determine which inter-route and intraroute operators would be most effective.

To choose which intraroute operations to utilize, a comparison was generated. For each operator five trials were run on five different instances. Each trial was run using identical parameters. Each run had a population of 50 and was run for 200 epochs. Initial populations were generated with a parallel construction and then solved to be 2-opt optimal. The crossover rate was 0.2 and the starting number of vehicles was chosen to be slightly higher than the known best solution, typically higher than four.

For each run, the total number of fronts and the number of unique solutions were measured to quantify the diversity of the population, as seen in Table 5. Table 6 measures the solution quality. The points at the Pareto front and the hypervolume were measured to compare the quality of the solutions. For each operator the z score of the hypervolume was also recorded. This allowed a direct measurement of how much better each operator was in comparison. Table 7 aggregates all of the trials.

Table 5. Local search operators effects on diversity of bar-n100-1 [29].

Local Search Operator	Final # of Fronts	Number of Unique Solutions
No-op	1	1
Swap mutation	3	19
Blocked swap mutation	2	6
Insertion mutation	3	6
Blocked insertion	1	3
Displacement mutation	3	13
Blocked displacement mutation	3	10
Gaussian displacement mutation	1	6
Blocked Gaussian displacement mutation	1	5
2-Opt	1	5
Blocked 2-opt	1	3
3-Opt	8	12
Blocked 3-opt	7	27
4-Opt	1	8
Blocked 4-opt	1	5
Boundary operators	2	15

Each operator with the exception of the 4-opt operator was run in a dynamic programming fashion, fully exhausting the neighbourhood to ensure the best move was made. For the 4-opt and blocked 4-opt operators, this was infeasible due to the size of the search space,

so a Monte Carlo framework was used. One hundred random neighbourhood moves were tested and the best one was used for this iteration of local search.

Table 6. Local search operators effects on solution quality of bar-n100-1 [29].

Local Search Operator	(Vehicle Count, Distance) in Front 0	Hypervolume	z Score
No-op	(7, 774)	126	−1.40152
Swap mutation	(6, 779), (7, 776)	245	0.59139
Blocked swap mutation	(6, 797), (7, 762)	241	0.5244
Insertion mutation	(6, 754)	292	1.37851
Blocked insertion mutation	(6, 797), (7,766)	237	0.45742
Displacement mutation	(6, 779), (7, 764)	257	0.79236
Blocked displacement mutation	(7, 783)	117	−1.55225
Gaussian displacement mutation	(6, 837), (7, 815)	148	−1.03308
Blocked Gaussian displacement mutation	(6, 782), (7, 771)	247	0.62489
2-Opt	(6, 768)	264	0.90959
Blocked 2-opt	(6, 776), (7, 778)	248	0.64164
3-Opt	(7, 798)	102	−1.80345
Blocked 3-opt	(6, 794), (7, 769)	237	0.45742
4-Opt	(6, 790), (7, 787)	223	0.22296
Blocked 4-opt	(7, 770)	130	−1.33453
Boundaries operators	(6, 786), (7, 773), (8, 771)	241	0.5244

Reference Point 8, 900.

The results indicated that the 2-opt operator was the best move by a decently large margin. The 3-opt operator move was too slow to test exhaustively and so failed to converge like the other trials. The 4-opt operator had the largest variance of any operator. On some runs it found the best solution and on some it found the worst. The standard array of mutation operators performed adequately but were not able to compare to more specialized operators.

As for diversity, out of those that converged, the 4-opt operator had the best diversity within each front, with an average of eight unique individuals per front. Insertion mutation had the worst diversity averaging only 1.8 fronts and 5.6 individuals.

To address both diversity and solution quality, a combination of 4-opt and 2-opt was used in the final model.

To assess the effectiveness of each intraroute operator an ablation study was conducted. There were seven different scenarios run over five different instances. All runs held all parameters constant aside from the inter-route operations. They were run for 200 epochs with a population of 50. The initial population was created with parallel insertion and the 2-opt operator, with the intraroute operation also using the 2-opt operator. The 2-opt operator is a standard local search operator for variants on the travelling salesman problem. It involves selecting two edges and swapping them, effectively generating two new routes.

In each scenario, a single inter-route operation was removed to assess its effect on the final solution. Results are in Tables 8–10. Figure 3 shows the average hypervolume value over time and Figure 4 shows how minimum distance is affected. The hypervolume is a special measurement that calculates the area of the solution space and a reference point that is larger in magnitude than any given point in all dimensions. This area allows a direct comparison of the solutions generated. It was first introduced by Ziztler and Theile [31] in 1999. The main benefit of this measure is that it makes no assumptions on any knowledge about the Pareto front, which the other measures require.

Table 7. Local search operators summary statistics.

Local Search Operator	Mean Z Score	Mean # of Fronts	Mean # of Unique Solutions
No-op	−0.51334	2.4	6.4
Swap mutation	−0.043814	2.6	13.2
Blocked swap mutation	0.97958	3.4	12.6
Insertion mutation	0.33991	1.8	5.8
Blocked insertion mutation	0.32763	1.6	8.2
Displacement mutation	0.3491	2.2	8.2
Blocked displacement mutation	−0.31909	2.8	12.6
Gaussian displacement mutation	−0.179226	1.8	8.4
Blocked Gaussian displacement mutation	−0.16168	2.4	10.8
2-Opt	0.87708	2	8
Blocked 2-opt	0.118585	3.6	11
3-Opt	−1.22385	6.6	21.4
Blocked 3-opt	0.17869	4.8	16.6
4-Opt	0.16474	1.6	12.8
Blocked 4-opt	−0.80439	3	14.6
Boundaries operators	0.044518	3.8	18.6

The experiment tested how much each pickup and delivery operation affected the end result. For each operator, the genetic algorithm was run with all other parameters fixed. The only difference was the inclusion of each operator. Each run was executed with a population of 40, over 300 epochs. All other parameters were held constant.

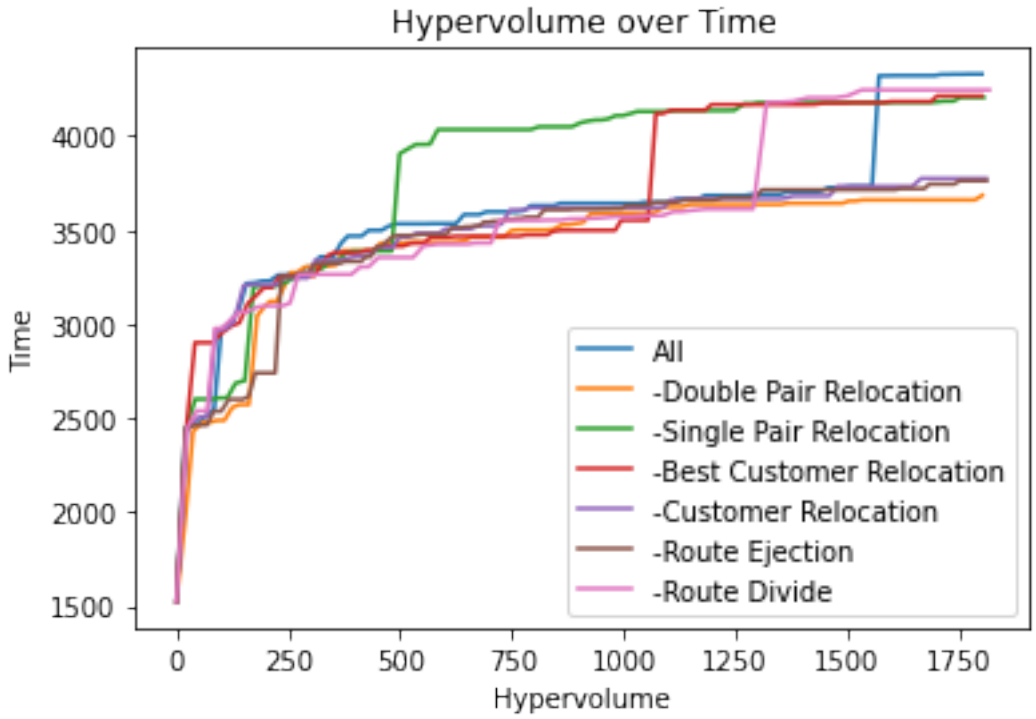

Figure 3. Hypervolume after removing each inter-route operator.

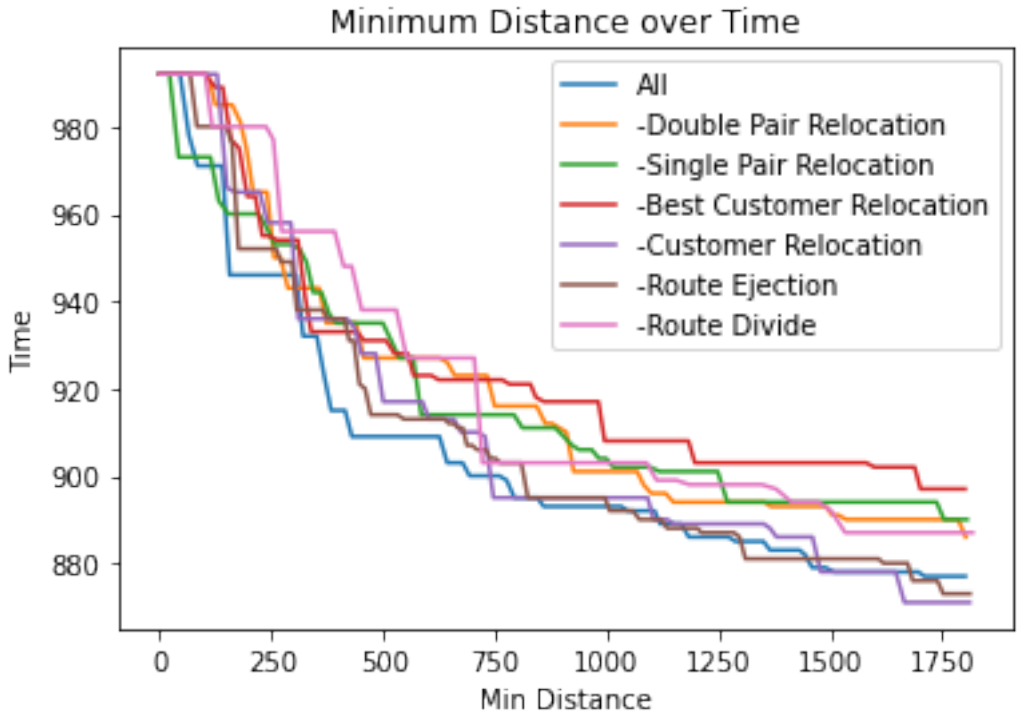

Figure 4. Minimum distance after removing each inter-route operator.

Table 8. Inter-route operation effects on diversity of bar-n100-5 [29].

Inter-Route Operator	Final # of Fronts	Number of Unique Solutions
All	1.8	5
Double-pair relocation	1	4
Single-pair relocation	3	28
Best-customer relocation	3	13
Customer relocation	1	8
Route ejection	1	5
Route divide	2	9

Table 9. Inter-route operation effects on solution quality of bar-n100-5 [29].

Inter-Route Operator	(Vehicle Count, Distance) in Front 0	Hypervolume
All	(6, 780), (7, 772)	248
Double-pair relocation	(6, 785), (7, 782)	233
Single-pair relocation	(6, 784), (7, 761)	255
Best-customer relocation	(6, 805), (7, 776)	219
Customer relocation	(6, 806), (7, 777)	217
Route ejection	(7, 746)	154
Route divide	(6, 788), (7, 770)	242

Reference Point 8, 900.

Table 10. Inter-route operation effects on solution quality of bar-n100-5 [29] Part 2.

Inter-Route Operator	Mean Z Score	Mean # of Fronts
All	1	1
Double-pair relocation	3	19
Single-pair relocation	2	6
Best-customer relocation	3	6
Customer relocation	1	3
Route ejection	3	13
Route divide	3	10

Reference Point 8, 900.

6. Discussion and Conclusions

In this work we formulated a genetic algorithm based on NSGA-II for solving the multiobjective capacitated pickup and delivery problem with time windows. We built two generic metaheuristics which allowed solution construction and insertion and explored six different inter-route operations and sixteen different intraroute operations. We found that adding intraroute operations in addition to inter-route operations greatly improved solution quality, with 2-opt being the best operator we trialed.

Of the inter-route operations, all of the tested operators benefited the end result. The variety of each operator enabled a good diversity within the population. The intraroute operators had more interesting results. Standard genetic algorithm operators such as mutation tended to perform poorly. Operators that took structure into account such as k-opt performed much better. Blocking the results did not have as much success despite taking more of the problem into consideration.

Our work does contain some limitations. Without proper benchmarks within the literature, it is difficult to compare results directly. Additionally, future work would involve exploring different ways to maintain diversity, as our algorithm was occasionally stuck in local optima. There are many ways that this could be implemented, either by speciation, island models, or geographical encodings.

Author Contributions: Conceptualization, C.L., K.S., T.H. and S.C.; methodology, C.L.; software, C.L.; validation, C.L.; investigation, C.L.; data curation, C.L.; writing—original draft preparation, C.L.; writing—review and editing, K.S., T.H. and S.C.; visualization, C.L.; supervision, K.S., T.H. and S.C.; project administration, C.L. and S.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research is funded by Vector Scholarship in Artificial Intelligence and NSERC CGS-M 2021 scholarships of Connor Little, as well as NSERC discovery grant received by Dr. Salimur Choudhury (RGPIN-2018-1507).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Li Lim dataset [30]: https://www.sintef.no/projectweb/top/pdptw/li-lim-benchmark/ (accessed on 25 September 2022).

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

PDP	Pickup and delivery problem
MOCPDPTW	Multiobjective capacitated pickup and delivery problem with time windows

Appendix A

Below are Tables A1 and A2 and displaying the found and best solutions for each instance in the Li Lim [30] and Sulzbach Sartori and Buriol [29] datasets, respectively.

Table A1. Results on Li-Lim's benchmark.

Instance	Best Solution	Found Solution
lc101	(10, 828.94)	(10, 828.94)
lc102	(10, 828.94)	(10, 828.94)
lc103	(9, 1035.35)	(10, 829.56) *
lc104	(9, 860.01)	(10, 818.60) *
lc105	(10, 828.94)	(10, 828.94)
lc106	(10, 828.94)	(10, 828.94)
lc107	(10, 828.94)	(10, 828.94)
lc108	(10, 826.44)	(10, 828.94)
lc109	(9, 1000.60)	(10, 828.94) *
lc201	(3, 591.56)	(3, 591.56)
lc202	(3, 591.56)	(3, 591.56)
lc203	(3, 591.17)	(3, 648.05), (4, 637.87)
lc204	(3, 590.60)	(3, 674.84)
lc205	(3, 588.88)	(3, 588.88)
lc206	(3, 588.49)	(3, 597.31)
lc207	(3, 588.29)	(3, 588.29)
lc208	(3, 588.23)	(3, 589.44)

Table A1. *Cont.*

Instance	Best Solution	Found Solution
lr101	(19, 1650.80)	(20, 1700)
lr102	(17, 1487.57)	(17, 1519.98)
lr103	(13, 1292.68)	(13, 1337.34), (14, 1333.39)
lr104	(9, 1013.39)	(10, 1067.19)
lr105	(14, 1377.11)	(15, 1486.85)
lr106	(12, 1252.62)	(12, 1287.67), (13, 1271.09)
lr107	(10, 1111.31)	(10, 1111.31)
lr108	(9, 968.97)	(9, 970.74)
lr109	(11, 1208.96)	(12, 1251.53)
lr110	(10, 1159.35)	(12, 1226.60)
lr111	(10, 1108.90)	(12, 1203.15)
lr112	(9, 1003.77)	(11, 1075.88)
lr201	(4, 1253.23)	(4, 1311.34), (5, 1291.02), (6, 1289.23)
lr202	(3, 1197.67)	(4, 1287.83)
lr203	(3, 949.40)	(3, 1042.40)
lr204	(2, 849.05)	(3, 1053.58)
lr205	(3, 1054.02)	(3, 1342.12), (4, 1211.80), (5, 1191.29)
lr206	(3, 931.63)	(3, 970.70)
lr207	(2, 903.06)	(3, 1120.39), (4, 1116.43)
lr208	(2, 734.85)	(3, 891.17)
lr209	(3, 930.59)	(4, 1030.60)
lr210	(3, 964.22)	(3, 1188.75), (4, 1179.80)
lr211	(2, 911.52)	(3, 1097.17), (4, 1007.62)
lrc101	(14, 1708.80)	(14, 1708.80)
lrc102	(12, 1558.07)	(13, 1608.49)
lrc103	(11, 1258.74)	(11, 1275.32)
lrc104	(10, 1128.40)	(10, 1128.40)
lrc105	(13, 1637.62)	(14, 1734.76)
lrc106	(11, 1424.73)	(13, 1531.88)
lrc107	(11, 1230.14)	(12, 1399.75)
lrc108	(10, 1147.43)	(11, 1188.12)
lrc201	(4, 1406.94)	(5, 1494.95)
lrc202	(3, 1374.27)	(4, 1440.29)
lrc203	(3, 1089.07)	(4, 1153.20)
lrc204	(3, 818.66)	(3, 914.81), (4, 887.48)
lrc205	(4, 1302.20)	(4, 1310.28)
lrc206	(3, 1159.03)	(3, 1159.03)
lrc207	(3, 1062.05)	(3, 1066.68)
lrc208	(3, 852.76)	(4, 953.59)

Results in which distance was improved are indicated wtih an *.

Table A2. Results on Sulzbach Sartori and Buriol's benchmark.

Instance	Best Solution	Found Solution
bar-n100-1	(6, 733)	(6, 776), (7, 766)
bar-n100-2	(5, 554)	(5, 608), (6, 579)
bar-n100-3	(6, 746)	(6, 803), (7, 777)
bar-n100-4	(12, 1154)	(13, 1217), (14, 1193)
bar-n100-5	(6, 838)	(6, 909), (7, 877)
bar-n100-6	(3, 788)	(4, 933), (5, 894), (6, 890), (7, 881)

Table A2. *Cont.*

Instance	Best Solution	Found Solution
ber-n100-1	(12, 1857)	(14, 1848) *
ber-n100-2	(6, 1491)	(7, 1569)
ber-n100-3	(3, 713)	(4, 745), (5, 702), (6, 695) *
ber-n100-4	(3, 494)	(3, 555)
ber-n100-5	(5, 944)	(5, 966)
ber-n100-6	(14, 2147)	(16, 2108) *
ber-n100-7	(7, 1935)	(8, 2043), (9, 2039)
nyc-n100-1	(6, 634)	(6, 731), (7, 665)
nyc-n100-2	(4, 567)	(4, 603), (5, 587), (6, 583)
nyc-n100-3	(3, 492)	(4, 477)
nyc-n100-4	(2, 535)	(3, 589)
nyc-n100-5	(2, 671)	(3, 702)
poa-n100-1	(12, 1589)	(14, 1637)
poa-n100-2	(15, 1539)	(16, 1645), (17, 1640)
poa-n100-3	(10, 1301)	(11, 1329), (12, 1311)
poa-n100-4	(7, 1668)	(9, 1586) *
poa-n100-5	(6, 624)	(6, 630), (7, 626)
poa-n100-6	(3, 562)	(3, 601), (4, 574), (5, 572)
poa-n100-7	(5, 779)	(6, 743), (7, 731)

Results in which distance was improved are indicated with an *.

References

1. Koç, Ç.; Laporte, G.; Tükenmez, İ. A review of vehicle routing with simultaneous pickup and delivery. *Comput. Oper. Res.* **2020**, *122*, 104987. [CrossRef]
2. Xu, Z.; Elomri, A.; Kerbache, L.; El Omri, A. Impacts of COVID-19 on Global Supply Chains: Facts and Perspectives. *IEEE Eng. Manag. Rev.* **2020**, *48*, 153–166. [CrossRef]
3. Parragh, S.N.; Doerner, K.F.; Hartl, R.F. A survey on pickup and delivery problems: Part II: Transportation between pickup and delivery locations. *J. Betriebswirtschaft* **2008**, *58*, 81–117. [CrossRef]
4. Deb, K.; Agrawal, S.; Pratap, A.; Meyarivan, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **2002**, *6*, 182–197. [CrossRef]
5. Tomasz, M. Visual Attractiveness of the Routes in the Vehicle Routing Problem. Mater's Thesis, University Wien, Wien, Austria, 2018.
6. Wang, J.; Zhou, Y.; Wang, Y.; Zhang, J.; Chen, C.L.; Zheng, Z. Multiobjective Vehicle Routing Problems with Simultaneous Delivery and Pickup and Time Windows: Formulation, Instances, and Algorithms. *IEEE Trans. Cybern.* **2016**, *46*, 582–594. [CrossRef]
7. Abdoun, O.; Abouchabaka, J.; Tajani, C. Analyzing the Performance of Mutation Operators to Solve the Travelling Salesman Problem. *arXiv* **2012**, arXiv:1203.3099.
8. Berbeglia, G.; Cordeau, J.F.; Gribkovskaia, I.; Laporte, G. Static pickup and delivery problems: A classification scheme and survey. *Top* **2007**, *15*, 1–31. [CrossRef]
9. Bravo, M.; Rojas, L.P.; Parada, V. An evolutionary algorithm for the multi-objective pick-up and delivery pollution-routing problem. *Int. Trans. Oper. Res.* **2019**, *26*, 302–317. [CrossRef]
10. Chami, Z.A.; Manier, H.; Manier, M.A.; Fitouri, C. *A Hybrid Genetic Algorithm to Solve a Multi-Objective Pickup and Delivery Problem*; Elsevier B.V.: Amsterdam, The Netherlands, 2017; Volume 50, pp. 14656–14661. [CrossRef]
11. Fatemi-Anaraki, S.; Mokhtarzadeh, M.; Rabbani, M.; Abdolhamidi, D. A hybrid of K-means and genetic algorithm to solve a bi-objective green delivery and pick-up problem. *J. Ind. Prod. Eng.* **2021**, *39*, 146–157. [CrossRef]
12. Garcia-Najera, A.; Gutierrez-Andrade, M.A. An evolutionary approach to the multi-objective pickup and delivery problem with time windows. In Proceedings of the 2013 IEEE Congress on Evolutionary Computation, Cancun, Mexico, 20–23 June 2013; pp. 997–1004. [CrossRef]
13. Gong, G.; Deng, Q.; Gong, X.; Zhang, L.; Wang, H.; Xie, H. A Bee Evolutionary Algorithm for Multiobjective Vehicle Routing Problem with Simultaneous Pickup and Delivery. *Math. Probl. Eng.* **2018**, *2018*, 2571380. [CrossRef]
14. Naik, N.; Jenkins, P.; Savage, N.; Yang, L.; Naik, K.; Song, J.; Boongoen, T.; Iam-On, N. Fuzzy hashing aided enhanced YARA rules for malware triaging. In Proceedings of the 2020 IEEE Symposium Series on Computational Intelligence (SSCI), Canberra, Australia, 1–4 December 2020.
15. Velasco, N.; Dejax, P.; Guéret, C.; Prins, C. A non-dominated sorting genetic algorithm for a bi-objective pick-up and delivery problem. *Eng. Optim.* **2012**, *44*, 305–325. [CrossRef]
16. Wang, H.F.; Chen, Y.Y. A genetic algorithm for the simultaneous delivery and pickup problems with time window. *Comput. Ind. Eng.* **2012**, *62*, 84–95. [CrossRef]

17. Zhu, Z.; Xiao, J.; He, S.; Ji, Z.; Sun, Y. A multi-objective memetic algorithm based on locality-sensitive hashing for one-to-many-to-one dynamic pickup-and-delivery problem. *Inf. Sci.* **2016**, *329*, 73–89. [CrossRef]
18. Grandinetti, L.; Guerriero, F.; Pezzella, F.; Pisacane, O. The Multi-objective Multi-vehicle Pickup and Delivery Problem with Time Windows. *Procedia—Soc. Behav. Sci.* **2014**, *111*, 203–212. [CrossRef]
19. Ren, X.; Huang, H.; Feng, S.; Liang, G. An improved variable neighborhood search for bi-objective mixed-energy fleet vehicle routing problem. *J. Clean. Prod.* **2020**, *275*, 124155. [CrossRef]
20. Zou, S.; Li, J.; Li, X. A hybrid particle swarm optimization algorithm for multi-objective pickup and delivery problem with time windows. *J. Comput.* **2013**, *8*, 2583–2589. [CrossRef]
21. Karakatič, S.; Podgorelec, V. A survey of genetic algorithms for solving multi depot vehicle routing problem. *Appl. Soft Comput. J.* **2015**, *27*, 519–532. [CrossRef]
22. Carrabs, F.; Cordeau, J.F.; Laporte, G. Variable Neighbourhood Search for the Pickup and Delivery Travelling Salesman Problem with LIFO Loading. *INFORMS J. Comput.* **2007**, *19*, 618–632. [CrossRef]
23. Cordeau, J.F.O. A Branch-and-Cut Algorithm for the Dial-a-Ride Problem. *Oper. Res.* **2003**, *54*, 573–586. [CrossRef]
24. Furtado, M.G.S.; Munari, P.; Morabito, R. Pickup and delivery problem with time windows: A new compact two-index formulation. *Oper. Res. Lett.* **2017**, *45*, 334–341. [CrossRef]
25. Katoch, S.; Chauhan, S.S.; Kumar, V. A review on genetic algorithm: Past, present, and future. *Multimed. Tools Appl.* **2021**, *80*, 8091–8126. [CrossRef] [PubMed]
26. Mitchell, M. *An Introduction to Genetic Algorithms*; MIT Press: Cambridge, MA, USA, 1998.
27. Alvarenga, G.B.; Mateus, G.R. A Two-Phase Genetic and Set Partitioning Approach for the Vehicle Routing Problem with Time Windows. In Proceedings of the Fourth International Conference on Hybrid Intelligent Systems (HIS'04), Kitakyushu, Japan, 5–8 December 2004.
28. Yanik, S.; Bozkaya, B.; Dekervenoael, R. A new VRPPD model and a hybrid heuristic solution approach for e-tailing. *Eur. J. Oper. Res.* **2014**, *236*, 879–890. [CrossRef]
29. Sartori, C.S.; Buriol, L. Instances for the Pickup and Delivery Problem with Time Windows based on open data. In *Mendeley Data*; 2020. [CrossRef]
30. Li, H.; Lim, A. A Metaheuristic for the Pickup and Delivery Problem with Time Windows. *Int. J. Artif. Intell. Tools* **2001**, *12*, 173–186. [CrossRef]
31. Zitzler, E.; Thiele, L. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Trans. Evol. Comput.* **1999**, *3*, 257–271. [CrossRef]