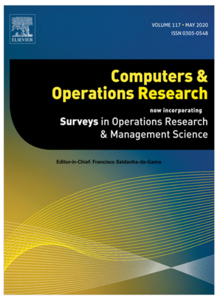

Adjusting the order crossover operator for capacitated vehicle routing problems

Lars Magnus Hvattum

PII: S0305-0548(22)00224-6
DOI: https://doi.org/10.1016/j.cor.2022.105986
Reference: CAOR 105986

To appear in: *Computers and Operations Research*

Received date : 6 June 2022
Accepted date : 2 August 2022

Adjusting the order crossover operator for capacitated vehicle routing problems

Lars Magnus Hvattum

[1]Faculty of Logistics, Molde University College, Norway

hvattum@himolde.no

June 6, 2022

Abstract

The capacitated vehicle routing problem is a much studied combinatorial optimization problem, reflecting its practical importance within areas such as logistics. The problem is computationally intractable, and heuristics are commonly applied for solving large instances. Among the best heuristics available is a hybrid genetic search that consists of mechanisms from evolutionary algorithms and a range of local search operators. This heuristic applies an order crossover operator that takes as input two existing solutions and produces as output a new solution for the search to explore. An open-source implementation of the heuristic is available, in which the order crossover operator represents 1.4% of the code. This work discusses potential short-comings of the traditional order crossover operator and proposes an adjusted operator. The new operator is evaluated on standard benchmark test instances, and is shown to reduce the gaps to best-known solutions by 4.2%.

Keywords: vehicle routing problem; genetic algorithm; open source; genetic operator.

1 Introduction

The *capacitated vehicle routing problem* (CVRP) is a well-known optimization problem that is easy to describe and hard to solve. We are given a depot where a fleet of identical vehicles is located, each vehicle with a given capacity. A set of customers is also given, where each customer has a given location and a given demand. The task is to design routes such that each customer is visited by exactly one vehicle and such that the sum of demands of customers visited by a vehicle does not exceed the capacity of the vehicle. Using the locations of the customers and the depot, travel distances can be calculated, and the goal is to design routes that minimizes the total distance travelled.

The CVRP is computationally intractable (Lenstra and Kan, 1981), and is frequently solved using heuristic algorithms. Some heuristics for the CVRP are based on genetic algorithms, a metaheuristic that has proven effective for dealing with a wide range of optimization problems (Reeves, 2010). However, for a long time no effective genetic algorithm was known for solving the CVRP, and Gendreau et al. (2002) concluded that, based on scarce results, genetic algorithms were not competitive for solving CVRPs. Prins (2004) changed this perspective by creating a successful method. In his implementation, solutions are represented as permutations of the customers visited, without including information about trip delimiters in the encoding. Decoding a permutation involves inserting optimal trip delimiters, thereby reaching a feasible solution of the CVRP. By representing solutions as a giant tour without trip delimiters, it is possible to apply crossovers known from the literature for solving the *traveling salesman problem* (TSP). Oliver et al. (1987) had compared three permutation crossover operators for the TSP. One of these was found to be superior to the others when solving the TSP with a genetic algorithm. The superior crossover, a generalization of an earlier "modified crossover" proposed by David (1985), is known as the order crossover, or OX crossover. Prins (2004) chose to apply the order crossover for solving CVRPs, finding it superior to an alternative linear order crossover.

Among the most successful heuristics for various routing problems, we currently find a hybrid genetic search that incorporates many of the ideas put forth by Prins (2004). The first version, presented by Vidal et al. (2012), combined population-based evolutionary search, neighborhood-based search, and advanced population-diversity management schemes to solve vehicle routing problems with multiple depots and multiple periods. Next, Vidal et al. (2013) focused on the challenges of duration and time-window constraints, after which Vidal et al. (2014) tackled the presence of an even wider range of possible problem attributes. Later, Vidal (2022) published an open-source implementation of the hybrid genetic search that specifically targeted the CVRP.

To handle multiple periods and depots, Vidal et al. (2012) developed a specialized crossover operator called the periodic crossover with insertions. Vidal et al. (2013) also used this when solving periodic problems, but for non-periodic problems the OX crossover was applied. Similarly, Vidal et al. (2014) developed another new crossover, the assignment and insertion crossover, but still used the OX crossover, except when the problem solved included a specific type of attribute. When addressing the CVRP, Vidal (2022) applied the OX crossover exclusively.

Since the work of Prins (2004), the OX crossover has been used in many successful applications of genetic algorithms to solve vehicle routing problems. However, it seems that no critical assessment has been published regarding the operator's suitability for solving the CVRP. The aim of this paper is to point out some potentially undesirable behaviors of the OX crossover.

After this we propose an adjustment of the OX operator, which we hypothesize may avoid some of the undesirable behavior and thereby lead to an improved performance of the overall heuristic. Finally, we empirically test this hypothesis on standard benchmark test instances of the CVRP.

This research would not be feasible to execute without access to an open-source implementation of the hybrid genetic search. While this implementation is simple, it is still extremely powerful, and Vidal (2022) argued that performance gains through additional operators and method hybridizations are inexorably connected to the reduction of conceptual simplicity. We support this claim, and therefore try to improve the existing method by making minimal changes to the code base: it is an interesting challenge to improve the performance by making as few changes as possible, while simultaneously trying to create a deeper understanding of what makes the method so powerful. Vidal (2022) stated this eloquently in his concluding remarks: "the goal of heuristic design should be to identify methodological concepts that are as simple and effective as possible, and to properly understand the role of each component."

The remainder of this paper is structured as follows. In Section 2 we summarize the state-of-the-art of heuristics for the CVRP. Section 3 provides a short summary of the hybrid genetic search and its open-source implementation. Next, Section 4 discusses the OX crossover and illustrates some of its potential shortcomings. Following this, two new variants of the crossover are presented: one that aims to improve the performance of the operator, and one that functions as an additional benchmark. Then, Section 5 presents a computational study to empirically evaluate the three different variants of the OX operator. Finally, conclusions are drawn in Section 6.

2 Heuristics for the capacitated vehicle routing problem

Since the CVRP was introduced by Dantzig and Ramser (1959), much research has gone into solving the problem efficiently. For solving the problem to proven optimality, the current best exact algorithms are based on combining multiple mechanisms such as cut generation and column generation, as discussed by Pecin et al. (2017) and Pessoa et al. (2020). However, given the these methods require prohibitively long computation times for instances with many customers, a significant amount of research has been focusing on the development of heuristic algorithms.

In the following, the focus is on the currently best performing heuristics. For an overview of older contributions to the literature on heuristics for the CVRP, we recommend the book chapter by Gendreau et al. (2002) and the paper by Laporte (2009). When presenting the hybrid genetic search for CVRP, Vidal (2022) evaluated the results by comparing to six other heuristics and the original hybrid genetic search by Vidal et al. (2012). The four best performing heuristics out of these are discussed below, in addition to a fifth, more recent heuristic.

Subramanian et al. (2013) proposed a hybrid iterated local search for several variants of routing problems, including the CVRP. The method is matheuristic that combines iterated local search and the use of a mathematical programming solver to find a combination of routes based on a set partitioning model. Arnold and Sörensen (2019) presented a method called the knowledge-guided local search. The method combines three local search techniques and uses problem-specific knowledge to guide the search towards promising solutions. The authors also show that the heuristic, in addition to performing well on the CVRP, can be applied to problem variants with multiple depots or multiple trips. Following this, Arnold et al. (2019) used the knowledge-guided local search framework to solve very large-scale instances of the CVRP.

Christiaens and Vanden Berghe (2020) developed a large neighborhood search with specialized operators for removing customers from a solution and then inserting them back. The method is called slack induction by string removals, and also considered a hierarchical objective including the minimization of vehicles used as a primary target. This contrasts most research on the CVRP, which typically only minimizes the total distance travelled. Accorsi and Vigo (2021) created a fast iterative localized optimization algorithm to solve large-scale instances of the CVRP. The method is based on iterated local search, but includes novel strategies to localize and control the search. Together with the paper, the authors made their source codes openly available.

Following the publication of the aforementioned heuristics, Vidal (2022) presented the hybrid genetic search as adapted to the CVRP. By measuring the performance using the gaps to best-known solutions after a given time limit, the hybrid genetic search was found to perform better than all the other methods tested. Only very early in the search, and on certain subsets of instances, the hybrid genetic search had a slightly worse performance than the fast iterative localized optimization algorithm of Accorsi and Vigo (2021).

Most recently, Simensen et al. (2022) reimplemented the hybrid genetic search while adding the operators from slack induction by string removals (Christiaens and Vanden Berghe, 2020) as a separate improvement method. The performance was measured in two ways: looking at the gaps to best-known solutions after a given time limit and also looking at the average gaps to the best-known solutions when sampled during several points in time during the run. Two different parameter settings were proposed by the authors, both of which led to gaps after a given time limit that were better than the corresponding gaps for the hybrid genetic search. Evaluated

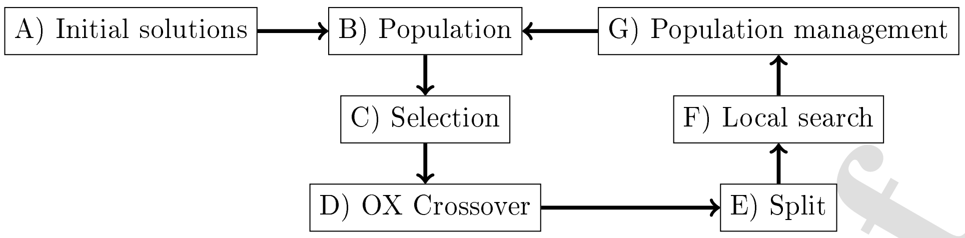

Figure 1: An overview of the hybrid genetic search for solving the CVRP.

using the average gaps during several points in time, one of the parameter settings was better than the hybrid genetic search and one was worse.

3 Hybrid genetic search

A detailed exposition of the hybrid genetic search specialized to the CVRP was provided by Vidal (2022). This section gives a brief overview, so as to understand the role that the OX crossover operator has within the method. Figure 1 labels seven algorithmic components and shows their interactions. This involves entering a loop that continues until a time limit has been reached.

The first search component (A) creates a set of initial solutions. These are simply random solutions that are improved by local search. The set of solutions is stored, and is referred to as a population (B). The population has two parts, consisting of feasible solutions and infeasible solutions, respectively. When storing solutions, they are expressed in an encoded form. This encoding is illustrated in Figure 2 for two solutions that differ only in the direction of travel in some of the vehicle routes. The solutions are represented by considering the sequence of visits, but without including any delimiters between routes. That is, solutions are stored as a permutation of the customers only.

An iteration of the hybrid genetic search beings by selecting two solutions from the population. This selection (3) is made using two binary tournaments. In each binary tournament, two solutions are selected at random, and the best solution is taken as the winner and is thus selected. Using this technique, it can happen that the same solution is selected as the winner in both binary tournaments.

Once two solutions have been selected, they are combined using the OX crossover (D). This operator is discussed in more detail in Section 4. Its application produces a new permutation of customers. However, a given permutation of customers can represent several different solutions,

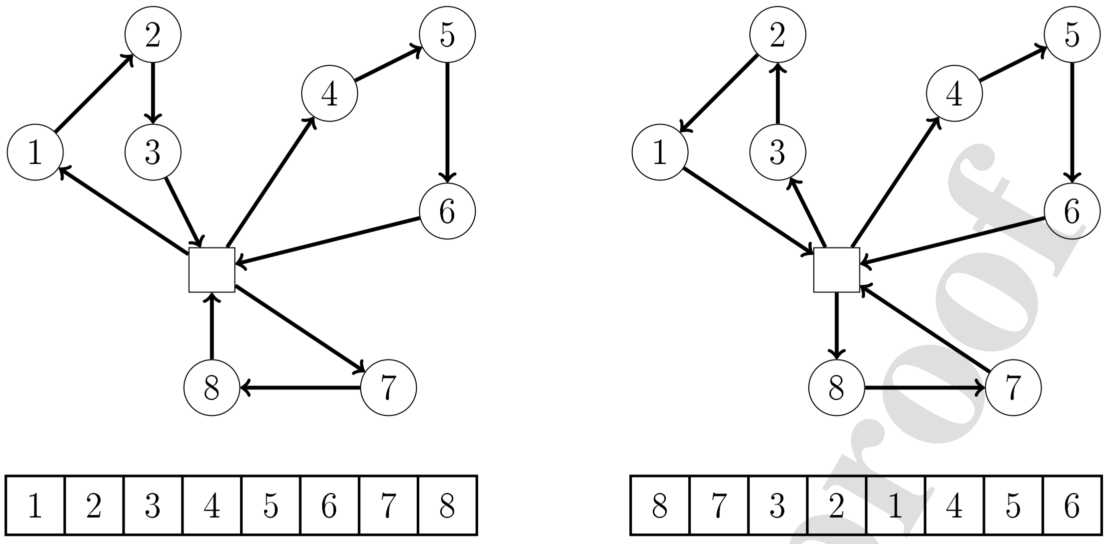

Figure 2: Two solutions to a CVRP instance with eight customers, each with a possible encoded representation.

depending on where the routes start and end. The split algorithm (E) is able to determine the optimal way to form routes from a given permutation in linear time (Vidal, 2016), and is thus used to decode the permutation into an actual solution.

After the method has obtained a new solution, it applies local search operators (F) to improve the solution. Several different neighborhoods are used. As the search allows the exploration of infeasible solutions, there is also a probability of applying a repair operation aiming to recover a feasible solution. The population management (G) handles the feasible and infeasible solutions separately. Once a subpopulation reaches a certain size, it is trimmed down by removing any repeated solutions, or by removing solutions with a worse evaluation. The evaluation is based on both the solution quality and on a diversity measure.

4 The order crossover

The OX crossover operator constitutes only one of seven components in the hybrid genetic search, as depicted in Figure 1. However, it is potentially an important operator, as it dictates how solutions are combined in order to generate new solutions. In the hybrid genetic search, the OX crossover is used to create one new solution from a given pair of existing solutions. Figure 3 illustrates the procedure when applied on the encoded solutions from Figure 2.

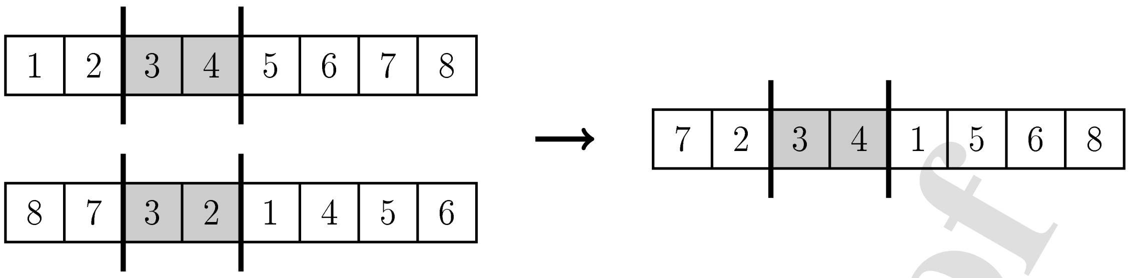

Figure 3: Example of the OX crossover applied to two permutation vectors (on the left), providing a new permutation vector (on the right).

The crossover starts by randomly selecting two cutting points that split the encoded solution into two segments, as indicated by two different background colors in Figure 3. It is possible that the cutting points are selected so that the segment between them will loop around the permutation vector, but this is not the case in the example illustrated. The crossover then continues by copying the segment between the cutting points from the first of the two combined solutions into a new permutation vector. In the example, this corresponds to the segment consisting of customers 3 and 4. This leaves six open spaces in the permutation vector that is going to be used for the new solution generated.

To complete the new solution, the second solution combined is processed. Starting from the element corresponding to the position directly after the copied segment (from the first solution), the second solution is read and the missing elements from in the new solution are added one by one in the order provided by the second solution. Hence, customers 5 and 6 are copied to the new solution first, then the processing of elements from the second solution wraps around and continues with 8 and 7. Then, we loop around also in the new solution. At this point, the next customer read in the second solution is 3, but since this was already present in the copied segment, it is skipped. Thus, next follows customer 2, and then finally customer 1.

It has been claimed that the OX crossover leads to new solutions where the relative order of elements are similar to the orders in the combined solutions (Gendreau et al., 2002). This is in contrast to other crossovers that tend to preserve the position of elements or the edges of the implied routes. In the illustrating example, the solutions that are combined are essentially identical, as shown in Figure 2. When we inspect the giant tour implied by the new permutation vector obtained in our example, as shown to the left in Figure 4, it becomes apparent that several long edges are introduced in the process.

On one hand, this suggests that the OX crossover may be integral in diversifying the search

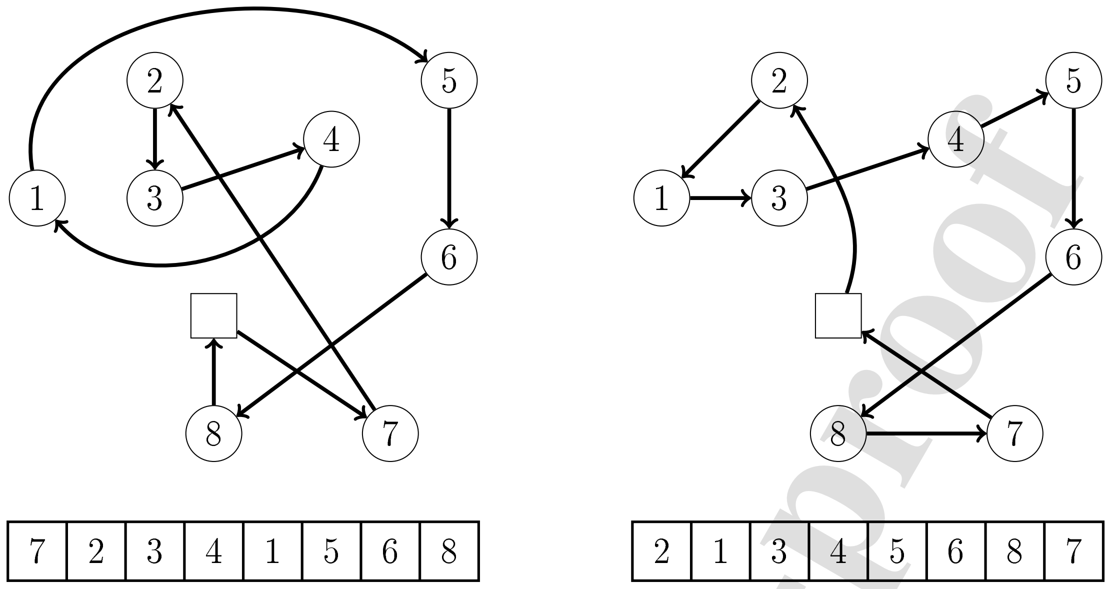

Figure 4: Giant tours resulting from applying the original OX crossover (to the left) and an adjusted OX crossover (to the right) when combining the solutions shown in Figure 2.

process: it can introduce new edges, even though these edges do not immediately appear to be advantageous. On the other hand, scrambling solutions in this manner may waste resources as the resulting solution must be improved using expensive local search operators afterwards.

The example illustrates one type of behavior that we hereby question: when filling in elements from the second solution, the process starts immediately after the cutting point used to identify the segment copied from the first solution. However, given that the order of routes in a permutation is arbitrary, it does not follow that this location in the permutation vector of solution two is related to the same location in the permutation vector of solution one. In the example, even though the beginning of the copied segment contains part of the same route (in fact the same customer), the end of the segment contains customers from different routes.

Does it really make sense to continue filling in the new solution from the location of the cutting point? We will argue here that it makes more sense to continue filling in the new solution from the location of the last customer that was copied, not from the location that was last copied. In other words, we should continue filling in starting from customer 4 in the second solution, not from the location in the second solution where customer 4 was located in the first solution. A problematic behavior can occur according to the original OX crossover because the customer that

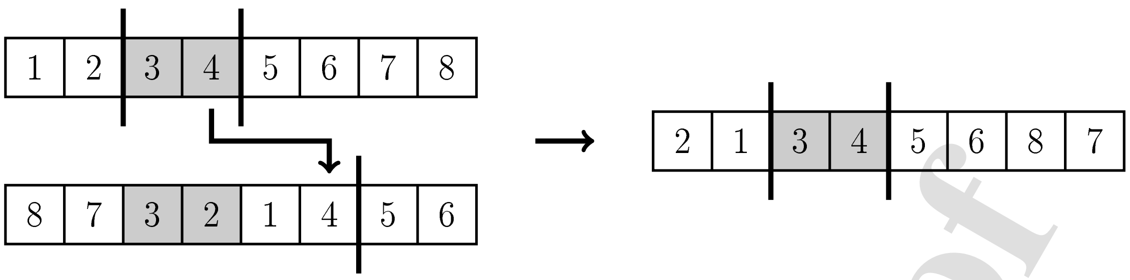

Figure 5: Example of the adjusted OX crossover applied to two permutation vectors (on the left), providing a new permutation vector (on the right).

is located just after the original cutting point in the second permutation vector is likely unrelated to the customer that is located just before the original cutting point in the first permutation vector.

This immediately brings us to an adjusted OX crossover, which is illustrated for the same example in Figure 5. Filling in the remainder of the new solution from the described location in the second solution provides a very different new solution. Its giant tour, shown to the right in Figure 4 has not introduced many long edges. It is, however, quite similar to the solutions combined. This is not unreasonable, however, given that the combined solutions were originally identical except for the direction of travel. In addition, the new solution is not entirely identical, and allows the exploration of a different route for visiting customers 1, 2, and 3.

There are other situations where both the original OX crossover and the proposed adjustment are counter-productive. When two relatively similar solutions are combined, and when the segment to copy is relatively small, the segment that is copied from the first solution may be identical to a segment in the second solution (albeit at a different location in the permutation vector). When the copied segment is relatively large, the non-copied segment in the first solution may be identical to a corresponding segment in the second solution. In these cases, the adjusted OX crossover would not create new permutations, but rather end up recreating one of the permutations of the combined solutions.

To deal with these situations, an additional adjustment is made. In the adjusted OX crossover we check if the segment copied from the first solution is found in an identical form in the second solution. If that is the case, for example when the combined solutions are identical, we instead choose a crossover point in the second solution at random, while avoiding crossover points that are inside the identical segment.

We refer to the new variant as the adjusted order crossover (AOX). As a third variant, to be considered as a benchmark, we define a random order crossover by always taking the cutting point in the second solution as a completely random point. We label this as a random order crossover (ROX).

Based on the discussion above, we would expect the ROX crossover to perform worse, as it does not exploit any structural information from the combined solutions when deciding on the relevant cutting point for the second solution. Furthermore, as the AOX crossover avoids what is presumably an unwanted behavior in the OX crossover, we believe that the AOX crossover could be able to improve the performance of the heuristic due to improving the intensification of the search. Although the illustrating example could suggest that the AOX leads to less diversification in the search, the occasional inclusion of a randomized cutting points will help the search to diversify whenever the combined solutions become relatively similar.

In the appendix to this paper, Table 2 shows the code, written in C++, of the original implementation of the OX crossover. The entire open-source code of Vidal (2022) encompasses 2073 non-empty lines of code (including comments) across 15 files. The OX crossover only takes 29 lines of code (also including comments), which is about 1.4% of the entire code base. The new implementation of the AOX crossover takes 43 lines of code, many of which are identical to the original implementation. The new code is provided in Table 3 of the appendix.

5 Computational study

To evaluate whether the AOX crossover leads to a better performance of the hybrid genetic search than the OX crossover, we design a computational experiment. In the experiment we test the performance across 100 benchmark test instances created by Uchoa et al. (2017). The instances, known as the X-set, have between 100 to 1,000 customers, and are commonly used to evaluate the performance of both heuristic and exact algorithms for the CVRP. Best-known solutions to the instances were retrieved from CVRPLIB, using the link `http://vrp.atd-lab.inf.puc-rio.br/index.php/en/` accessed on April 26, 2022.

The code is compiled using Microsoft Visual C++ 2019 for a 64-bit architecture, and the experiments are conducted on a standard desktop computer with an Intel i9-9900 CPU at 3.1GHz and with 32 GB of RAM. For each instance, a time limit is set to 2.4 seconds times the number of customers. Since there are several random elements in the heuristic, each instance is solved 10 times using different random seeds. Three versions of the code are executed, corresponding to using the OX crossover, the AOX crossover, or the ROX crossover.

We next compare the performances of the alternative crossovers from three different perspectives. We first consider the method's ability to find solutions such that the average gaps to the best-known solutions are smaller. Then, we consider the ability to actually find the best-known solutions for each instance. Finally, we consider direct comparisons between the crossovers, and whether each crossover leads to a method that is more or less likely to find a better solution than another crossover operator.

5.1 Primal gaps

To evaluate the results, we first consider the gaps to best-known solutions as calculated at the end of the runs. For the CVRP and the given instances, this corresponds to the primal gaps as defined by Berthold (2013). In Figure 6 we plot these primal gaps as a function of the running time, after normalizing the running times to the interval $[0, 1]$.

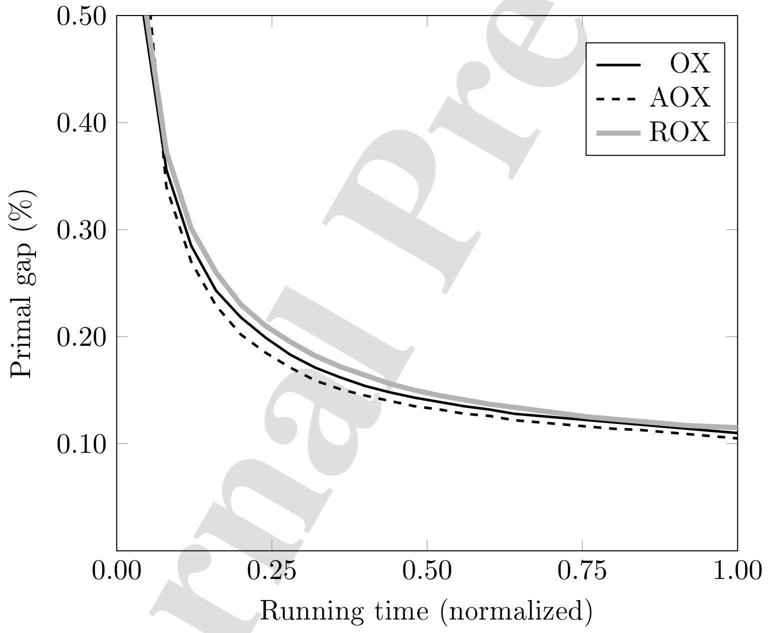

Figure 6: Primal gaps as a function of time when considering all 100 instances.

The figure gives a visual indication of how the new AOX performs better than both OX and ROX for most of the running time. At the end of the runs, the primal gaps are 0.110% for OX, 0.105% for AOX, and 0.115% for ROX. This means that the primal gap when using AOX is 4.2%

lower than the primal gap when using OX. The relative improvement for AOX is slightly larger when lower running times are considered. For example, after a normalized running time of 0.2, the improvement is 7%, lowering the average gaps from 0.218% to 0.203%.

In Figure 7 we show the primal gaps for the runs of the 50 smallest instances in the test set. Again, AOX is the best performing crossover, but on these smaller instances the randomized crossover gives a slightly smaller average gap than the original crossover. The final primal gaps for OX, AOX, and ROX are, respectively, 0.026%, 0.021%, and 0.025% when considering the smallest instances.

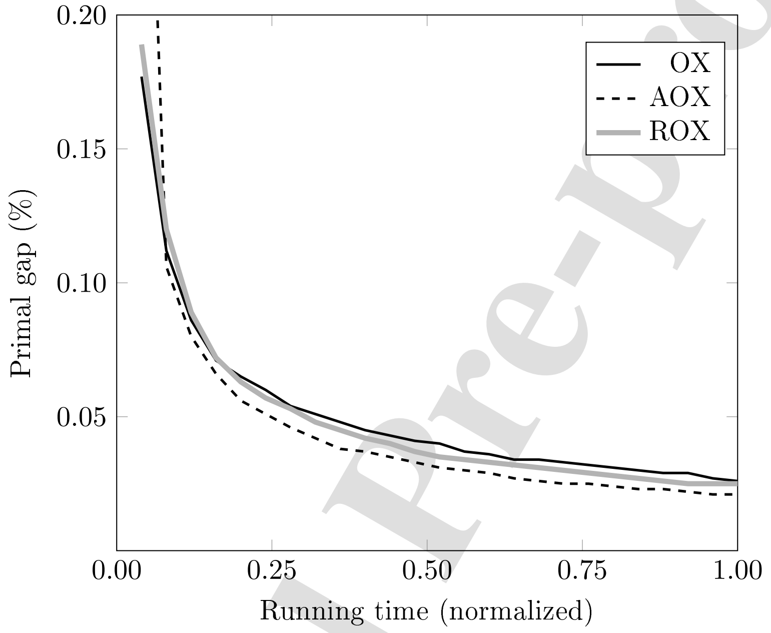

Figure 7: Primal gaps as a function of time when considering the 50 smallest instances.

Then, in Figure 8 the primal gaps are shown for the 50 largest instances. This time the OX crossover performs better than the randomized crossover, but nevertheless, the new adjusted AOX crossover provides the best average gaps also for the largest instances. The final primal gaps for OX, AOX, and ROX are here 0.194%, 0.190%, and 0.205%.

5.2 Best-known solutions

Next, we consider the ability of the hybrid genetic search to find the best-known solutions using three variants of the crossover operator. This measures a different aspect of the performance

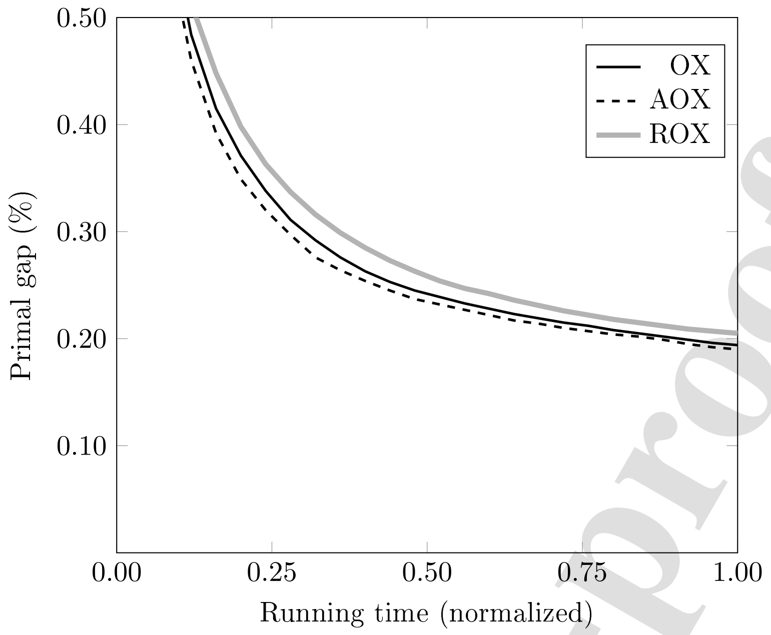

Figure 8: Primal gaps as a function of time when considering the 50 largest instances.

than simply measuring the average primal gaps: it is possible to have very small gaps without ever actually finding the best possible solution, and it is possible to have a large gap on average, despite finding many optimal solutions. Figure 9 shows the number of best-known solutions found, out of 1,000 runs for each search variant considered.

The ROX crossover, where the entry point for the second solution combined is totally randomized, appears to be better than the original OX crossover in terms of finding the best-known solutions. The randomness of the ROX likely leads to more varied performances, and it is thus more likely to find optimal solutions, at the expense of having a worse average performance. However, the AOX is the method that finds the most best-known solutions, with 371 runs successfully identifying the best solution, compared to 362 successful runs for ROX and 347 successful runs for OX. For all methods, the number of best-known solutions found is steadily increasing, even as we get closer to the full running time allotted, which is an indication that even better results should be expected if the running time is increased.

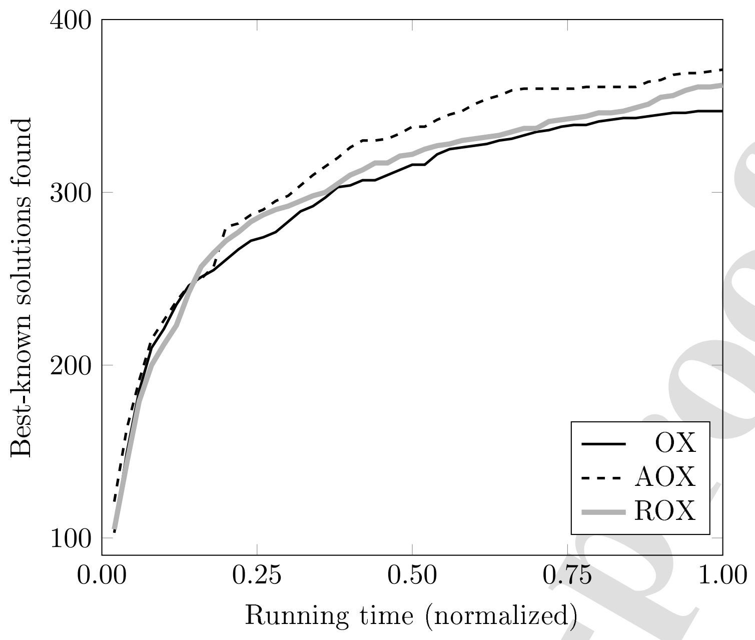

Figure 9: Number of best-known solutions identified as a function of time, out of 1,000 total runs.

5.3 Pair-wise comparisons

Finally, we evaluate the performance of the methods using a non-parametric sign test (Derrac et al., 2011). Here, for a given pair of methods, we count the number of runs where a given method performs better than another method. A method is then considered to perform better than another if either it obtains a better solution at the end of the run, or if it obtains the same quality solution but finds it after spending less computational time. The output of the sign tests are P-values that indicate how likely we are to observe the actual result, or a more skewed result, if the two methods compared are in reality equally likely to produce the best outcome. A low P-value then indicates that the better performing method is statistically better than the worse performing method. This type of comparison is facilitated by using the same random seeds for each method, so that runs are directly comparable: we start from the same initial set of solutions, and the runs only diverge once a crossover operator is applied.

Table 1 summarizes the pair-wise comparisons, where each pair of methods is evaluated on the full set of instances. The P-values are sufficiently small in all three tests to conclude that one of the methods is more likely to produce a better result than the other. That is, if we solve an instance using both AOX and OX, it is more likely that the AOX-run performs better than the

OX-run. Similarly, both AOX and OX are more likely to produce a better run than ROX.

Table 1: Pairwise comparisons of runs and P-values from a two-sided sign-test (Derrac et al., 2011).

	Wins	Losses	P-value
AOX vs. OX	554	446	0.0007
AOX vs. ROX	559	441	0.0002
OX vs. ROX	535	465	0.0291

We also performed similar tests when considering only the largest instances and only the smallest instances. Focusing on a subset of instances like this does change the results somewhat. For the smallest instances, AOX is significantly better than both OX and ROX, but OX is no longer significantly better than ROX from a statistical point of view. The latter seems consistent with the primal gaps shown in Figure 7. When it comes to the largest instances, AOX is significantly better than ROX, whereas the difference between AOX and OX and the difference between OX and ROX are not significant.

6 Concluding remarks

The hybrid genetic search proposed by Vidal (2022) is one of the best available heuristics for solving the capacitated vehicle routing problem. The method uses many different operators to manipulate solutions during a local search phase, but relies solely on an order crossover (OX) operator when combining pairs of solutions. After arguing that the OX crossover has certain drawbacks when used in the setting of a vehicle routing problem, we proposed an adjusted order crossover (AOX) operator and, as a benchmark, a random order crossover (ROX) operator.

In a computational study, using standard benchmark test instances, it is shown that the AOX is superior in three different aspects: 1) using the AOX operator provides a smaller average gap to the best-known solutions, reducing the gap obtained by using the OX operator by an additional 4.2%; 2) using the AOX operator is more likely to result in finding the best-known solution for the test instances examined; and 3) for a given run, the AOX operator is more likely to either find a better solution or to find the same solution in less computational time, when comparing to the OX operator.

References

L. Accorsi and D. Vigo. A fast and scalable heuristic for the solution of large-scale capacitated vehicle routing problems. *Transportation Science*, 55:832–856, 2021.

F. Arnold and K. Sörensen. Knowledge-guided local search for the vehicle routing problem. *Computers and Operations Research*, 105:32–46, 2019.

F. Arnold, M. Gendreau, and K. Sörensen. Efficiently solving very large-scale routing problems. *Computers and Operations Research*, 107:32–42, 2019.

T. Berthold. Measuring the impact of primal heuristics. *Operations Research Letters*, 41:611–614, 2013.

J. Christiaens and G. Vanden Berghe. Slack induction by string removals for vehicle routing problems. *Transportation Science*, 54:417–433, 2020.

G.B. Dantzig and J.H. Ramser. The truck dispatching problem. *Management Science*, 6:80–91, 1959.

L. David. Applying adaptive algorithms to epistatic domains. In *IJCAI'85: Proceedings of the 9th international joint conference on Artificial intelligence*, volume 1, pages 162–164. 1985.

J. Derrac, S. García, D. Molina, and F. Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1:3–18, 2011.

M. Gendreau, G. Laporte, and J.-Y. Potvin. Metaheuristics for the capacitated VRP. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, Discrete Mathematics and Applications, pages 129–154. SIAM, Philadelphia, USA, 2002.

G. Laporte. Fifty years of vehicle routing. *Transporation Science*, 43(4):408–416, 2009.

J.K. Lenstra and A.R. Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11:221–227, 1981.

I.M. Oliver, D.J. Smith, and J.R.C. Holland. A study of permutation crossover operators on the traveling salesman problem. In J. Grefenstette, editor, *Genetic Algorithms and their Applications: Proceedings of the Second International Conference*, pages 224–230. Lawrence Erlbaum, Hillsdale, NJ, USA, 1987.

D. Pecin, A. Pessoa, M. Poggi, and E. Uchoa. Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, 9:61–100, 2017.

A. Pessoa, R. Sadykov, E. Uchoa, and F. Vanderbeck. A generic exact solver for vehicle routing and related problems. *Mathematical Programming*, 183:483–523, 2020.

C. Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers and Operations Research*, 31(12):1985–2002, 2004.

C.R. Reeves. Genetic algorithms. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 109–140. Springer, New York, NY, USA, second edition, 2010.

M. Simensen, G. Hasle, and M. Stålhane. Combining hybrid genetic search with ruin-and-recreate for solving the capacitated vehicle routing problem. *Journal of Heuristics*, 2022.

A. Subramanian, E. Uchoa, and L.S. Ochi. A hybrid algorithm for a class of vehicle routing problems. *Computers and Operations Research*, 40:2519–2531, 2013.

E. Uchoa, D. Pecin, A. Poessoa, M. Poggi, T. Vidal, and A. Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257:845–858, 2017.

T. Vidal. Tehcnical note: Split algorithm in O(n) for the capacitated vehicle routing problem. *Computers and Operations Research*, 69:40–47, 2016.

T. Vidal. Hybrid genetic search for the CVRP: Open-source implementation and SWAP* neighborhood. *Computers and Operations Research*, 140:105643, 2022.

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.

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 and Operations Research*, 40:475–489, 2013.

T. Vidal, T.G. Crainic, M. Gendreau, and C. Prins. A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research*, 234:658–673, 2014.

A Code for crossover operators

To facilitate reproduction, this appendix provides both the original code for the order crossover in the implementation by Vidal (2022) and the new code suggested for the adjusted order crossover. The original code in Table 2 has 29 lines, whereas the new code in Table 3 has 43 lines. The lines indicated with black line numbers are new, whereas the lines with red line numbers have been modified, due to renaming some variables. The randomized crossover used as a benchmark is identical to the original code except that two new lines are added; these are shown in Table 4.

Table 2: Original C++ code for the order crossover (OX).

```cpp
void Genetic::crossoverOX(Individual * result, const Individual * parent1,
                                    const Individual * parent2)
{
   // Frequency table to track customers already inserted
   std::vector <bool> freqClient = std::vector <bool> (params->nbClients + 1, false);
   // Picking the beginning and end of the crossover zone
   int start = std::rand() % params->nbClients;
   int end = std::rand() % params->nbClients;
   while (end == start) end = std::rand() % params->nbClients;
   // Copy in place the elements from start to end
   int j = start;
   while (j % params->nbClients != (end + 1) % params->nbClients)
   {
     result->chromT[j % params->nbClients] = parent1->chromT[j % params->nbClients];
     freqClient[result->chromT[j % params->nbClients]] = true;
     j++;
   }
   // Fill the remaining elements in the order given by the second parent
   for (int i = 1; i <= params->nbClients; i++)
   {
     int temp = parent2->chromT[(end + i) % params->nbClients];
     if (freqClient[temp] == false)
     {
       result->chromT[j % params->nbClients] = temp;
       j++;
     }
   }
   // Completing the individual with the Split algorithm
   split->generalSplit(result, parent1->myCostSol.nbRoutes);
}
```

Table 3: New C++ code for the adjusted order crossover (AOX).

```cpp
void Genetic::crossoverOX(Individual * result, const Individual * p1,
                              const Individual * p2)
{
   // Frequency table to track customers already inserted
   std::vector <bool> freqClient = std::vector <bool> (params->nbClients + 1, false);
   // Picking the beginning and end of the crossover zone
   int start1 = std::rand() % params->nbClients;
   int end1 = std::rand() % params->nbClients;
   while (end1 == start1) end = std::rand() % params->nbClients;
   // Shift zone in p2 to match final customer of zone in p1
   int start2 = start1, end2 = end1;
   while (p2->chromT[end2 % params->nbClients] !=
             p1->chromT[(end1) % params->nbClients]) start2++; end2++;
   // Test if zone in p1 is different to zone in p2
   bool same = true;
   int size = (start1 < end1 ?  end1 - start1 :  params->nbClients - start1 + end1);
   for (int j = 0; j < size && same; j++)
   {
     if (p1->chromT[(start1 + j) % params->nbClients] !=
            p2->chromT[(start2 + j) % params->nbClients])
       same = false;
   }
   // If same, randomize point in p2
   if (same)
     end2 = end2 + rand() % (params->nbClients - size);
   // Copy in place the elements from start to end
   int j = start1;
   while (j % params->nbClients != (end1 + 1) % params->nbClients)
   {
     result->chromT[j % params->nbClients] = p1->chromT[j % params->nbClients];
     freqClient[result->chromT[j % params->nbClients]] = true;
     j++;
   }
   // Fill the remaining elements in the order given by p2
   for (int i = 1; i <= params->nbClients; i++)
   {
     int temp = p2->chromT[(end2 + i) % params->nbClients];
     if (freqClient[temp] == false)
     {
       result->chromT[j % params->nbClients] = temp;
       j++;
     }
```

```
    }
    // Completing the individual with the Split algorithm
    split->generalSplit(result, p1->myCostSol.nbRoutes);
}
```

Table 4: Changes to original C++ code for the randomized crossover (ROX).

```
16  (...)
N       // randomize end, giving the start point for second parent
N       end = std::rand() % params->nbClients;
17  (...)
```