LNCS, vol. 13352, pp. 369–382, DOI: 10.1007/978-3-031-08757-8_31, ICCS, June 2022



# Local Search in Selected Crossover Operators

Mirosław Kordos[1], Rafał Kulka[1], Tomasz Steblik[1], and Rafał Scherer[2]

[1] Department of Computer Science and Automatics, University of Bielsko-Biała, Willowa 2, 43-309 Bielsko-Biała, Poland mkordos@ath.bielsko.pl
[2] Department of Intelligent Computer Systems, Częstochowa University of Technology,
al. Armii Krajowej 36, 42-200 Częstochowa rafal.scherer@pcz.pl

**Abstract.** The purpose of the paper is to analyze an incorporation of local search mechanisms into five crossover operators (KPoint, AEX, HGreX, HProX and HRndX) used in genetic algorithms, compare the results depending on various parameters and draw the conclusions. The local search is used randomly with some probability instead of the standard crossover procedure in order to generate a new individual. We analyze injecting the local search in two situations: to resolve the conflicts and also without a conflict with a certain probability. The discussed mechanisms improve the obtained results and significantly accelerate the calculations. Moreover, we show that there exists an optimal degree of the local search component, and it depends on the particular crossover operator.

**Keywords:** genetic algorithms · crossover · local search.

## 1 Introduction

Genetic algorithms (GA) is a highly researched topic for a long time and especially in recent years. GA have two important advantages: a fast intelligent search mechanism, which allows finding a good solution after analyzing only a small fraction of possible candidates, and a high level of universality, which allows for a broad range of practical applications, and also optimization of training data and parameters of other artificial intelligence methods [2, 19, 12]. GA are very good at exploring the whole solution space, however, they are not so good at exploiting the local areas of the most promising solutions. For that purpose some hybrid methods have been proposed [18, 3, 20].

Hybrid genetic algorithms use an additional local search method, which cooperates with the genetic algorithm in order to achieve better results, by leveraging the power of both: the global search capabilities of GA and the local search. There are two basic families of approaching the joint genetic and local search: the Lamarckian approach and the Baldwinian one [14]. In the Baldwinian approach the effects of the local search improve the fitness of the individual, but its chromosome remains unchanged. In the Lamarckian approach, the effects of the local search are reflected in the chromosome. The advantage of the former one is

better exploration of the search space, and of the second one is better exploitation. Also combinations of the both approaches are possible [8]. In the method presented in this paper, we use the Lamarckian approach, because it accelerates the search process more effectively [102]. On the other hand, by changing the chromosome of individuals, it can disrupt the building blocks created by genetic algorithms, what may lead to a fast conversion, but only to a local minimum. To prevent this, we apply the local search only with a limited frequency.

An Adaptive Hybrid Genetic Algorithm (AHGA) was proposed in [4], which contains two dynamic learning mechanisms to guide and combine the exploration and exploitation search processes adaptively. The first learning mechanism aims to assess the worthiness of conducting the local search. The second learning mechanism uses instantaneously learned probabilities to select from a set of predefined local search operators which compete against each other for selection which is the most appropriate at any particular stage of the search to take over from the evolutionary-based search process. The authors of [6] presented a hybrid genetic algorithm (HGA) that uses a sequential constructive crossover, a local search approach along with an immigration technique to find good solutions in the multiple traveling salesman problem. In [23] an inversion operation was discussed to solve this problem, which was similar to an RMS mutation, but performed only if it improves the fitness. Hybrid genetic search with dynamic population management and adaptive diversity control with a problem-tailored crossover and local search operators were analyzed in [24]. A hybrid genetic algorithm given by applying 2-opt selection strategy to two edges chosen by replacement probability, and add the new edges by 2-opt permutation algorithm was proposed in [15].

## 2 Crossover Operators

The purpose of the crossover operator to combine information from two or more different chromosomes (parents) into one chromosome (child) that can represent a better solution than its parents.

In this paper we consider this kind of problems, where each item can occupy only one location (one position of the chromosome) at a time and each location must be occupied by exactly one item. For example, the traveling salesman problem or product placement optimization in a warehouse. In these cases, the crossover operator must ensure that there will be no duplicate elements and that each element will be present in the newly created individual.

Several crossover operators have been developed for this purpose. This includes the well known crossover operators as order crossover (OX), partially mapped crossover (PMX), order-based crossover (OBX), and position-based crossover (PBX) and cycle crossover (CX) [1, 11]. Also several newer crossover operators were developed, and we present some of them below.

Tan proposed heuristic greedy crossover (HGreX) and its variants HRndX and HProX [21], which we will explain in detail later. Other popular crossover operator is edge recombination crossover (ERX) and also several its variants

were proposed [22, 10], alternating edges crossover (AEX) [16] and the uniform partially matched crossover [7]. AEX also performs well after the improvements that we introduced, and for this reason we also used it in this paper.

A crossover method similar to HGreX, but with the difference that four candidates for each next position in the child were considered, was presented in [14]. These four candidates were the position placed before and after the current position in both parents, and the nearest position was selected.

Hassanat and Alkafaween [9] proposed several crossover operators, such as cut on worst gene crossover (COWGC) and collision crossover, and selection approaches, as select the best crossover (SBC). COWGC exchanges genes between parents by cutting the chromosome at the point that maximally decreases the cost. The collision crossover uses two selection strategies for the crossover operators. The first one selects this crossover operator from several examined operators, which maximally improves the fitness, and the other one randomly selects any operator. This algorithm applies multiple crossover operators at the same time on the same parents, and finally selects the best two offspring to enter the population.

In this paper we analyze incorporation of the local search in the following five crossover operators: KPoint, AEX, HRndX, HProX and HGreX. We chose them, because they perform very well and at the same time they are widely used. All the five operators work in a similar way. They use two parents to generate a child by alternatively taking the elements from one or from the other parent. They differ in the way in which the element alternation is organized.

The AEX operator can start from any position in the chromosome. In the example, we will start from the value on the first position in the first parent, this is A. Thus, A becomes the first position in the child. Then AEX looks at the second parent to find what it contains after A. This is actually D. Then again a value from Parent1 that is after D is used and so on. If following this rule would cause a conflict leading to repeating some values in the child, then randomly one from the available elements is chosen.

A sample explanation is presented in Figure 1, where we start from two parents; P1 and P2, and perform the following steps:

1. In step 1, we start creating a child, by taking the first element from the first parent and removing that element from both parents.
2. In step 2, we must find what is after A in P2 - this is D. So the next element of the child will be D and we remove it from the parents.
3. In step 3, we look, what is after D in P1 - this is E and this will be the next element in the child.
4. In a similar way in step 4.
5. In step 5, we would normally add D to the child. However, D has already been used, so a conflict appear here, and we select randomly any of the remaining elements. Let us say G.
6. In steps 6-8, we follow the standard rule and since there are no more conflicts, we add to the child F and then H and then B.

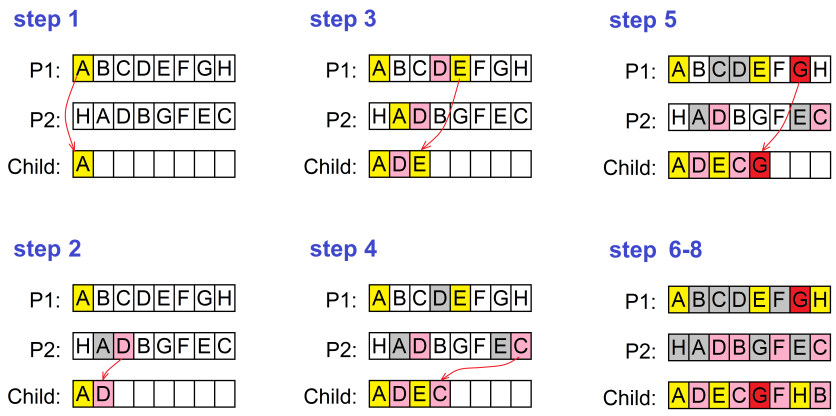


**Fig. 1.** Explanation of the AEX Crossover operator.

The HRndX crossover operator works similarly to AEX. The difference is, that it does not take alternatively the elements from both parents, but always randomly decides if the next element is taken from the first or from the second parent. HGreX always chooses the next element from this parent, to which the distance is closer. HProX chooses the closest element with a higher probability. The conflicts are resolved in the same way as in AEX.

The KPoint crossover operator does not take into account, which element occurs after which in the parents, but takes them sequentially alternating between the two parents. So for this example, it will generate the following child in the first three steps:

P1 = ABCDEFGH P2 = HADBGFEC Child = HBD_____

and a conflict occurs here. The conflicts are resolved in the same way as in AEX, so any unused value will be taken randomly, and then it continues with its standard scheme.

## 3 The Local-Search Enhancements of the Crossover Operators

### 3.1 Local search implementation in conflict and non-conflict resolvers

Let us analyze the case of a conflict first. A conflict appears, if following the standard crossover rule would lead to setting the next position in the child chromosome to a value that has already been used, as discussed in the previous section. The conflicts must be resolved in order to build a valid child chromosome in which each value appears exactly once.

**Random/Native Resolver** is the method of conflict resolving used originally by all the five crossover operators. In case of a conflict, this resolver takes

randomly any of the not used so far values and makes it the next element of the child.

**Algorithm 1** The Extended Crossover Operators

```
1: Input: Two parents
2: Output: The child

3: if Both parents are equal then
4:    mutate one parent with RSM mutation
5: end if
6: for k = 1 to chromocomeLength do
7:    if No Conflict then
8:       if Radom()>probability1 then
9:          set the k-th position in the child according to the crossover operator rule
10:      else
11:         if random resolver = Nearest Neighbor Resolver then
12:            set the k-th position in the child to the nearest from the unused positions
13:         else if random resolver = Tournament Resolver then
14:            set the k-th position in the child with the tournament selection of the
               unused positions
15:         end if
16:      end if
17:   else if Conflict then
18:      if conflict resolver = Random Resolver then
19:         set the k-th position in the child according to the crossover rule (by selecting
            a random unused position)
20:      else if conflict resolver = Nearest Neighbor Resolver then
21:         set the k-th position in the child to the nearest from the unused positions
22:      else if conflict resolver = Tournament Resolver then
23:         set the k-th position in the child with the tournament selection of the
            unused positions
24:      end if
25:   end if
26: end for
27: Return child
```

**Nearest Neighbor Resolver** - in this case, if a conflict occurs, the cost between the last position added to the child and all unused positions is calculated and the point with the lowest cost is chosen.

**Tournament Resolver** performs a tournament selection of all the unused values. A predefined number $n$ or percentage $p$ of the unused elements are randomly selected, and then the one with the lower cost to the recently added position in the child is chosen.

As a matter of fact, the tournament conflict resolver with $n = 1$ is equivalent to the random selection (random conflict resolver) and with $n$ being equal to the number of remaining points is equivalent to the nearest neighbor resolver.

The Nearest Neighbor Resolver and the Tournament Resolver can also be applied with a certain probability for dealing with the situations, where there are no conflicts. In these situations, normally the next child position would be determined by the base crossover operator rule. However, using the resolver here introduces the local search component.

## 4 Experimental Evaluation

We performed the experiments using two sample problems: the Warehouse Optimization Problem and the Traveling Salesman Problem (TSP). The source code of our software and the experimental data can be found at *kordos.com/iccs2022*.

In this section, first we describe these two problems along with the encoding schemes and local search methods that we used, and then we present the obtained results.

The HGreX and HProX operators are only suitable for the TSP example. The KPoint operator, although being suitable for both examples, works very well for the warehouse problem, but quite poor for the TSP. For that reason we evaluate AEX, HRndX for both problems, HGreX and HProX only for TSP and KPoint only for the warehouse problem.

### 4.1 The Example of Warehouse Optimization Problem

For the evaluation we used 6 different warehouse structures with the size between 60 and 240 locations, which can be found at *kordos.com/iccs2022*

**Problem description** Order picking is the most time-consuming task in warehouse operations, and thus also the task generating most costs. It was discovered that about 60% of warehouse operation costs are those of picking up products when completing orders [2]. Thus, reducing the order completion time is a crucial challenge, as it gives an opportunity to significantly reduce costs and to increase sales using the same resources. The problem was studied in many literature positions [4, 6, 5, 17, 13]. A review of the scientific literature investigating order picking and planning problems can be found in [23]. With $n$ items in the warehouse, the number of all possible their placements is $n!$ Already for 100 products it gives $100! = 9.3e157$ possible product placements. Thus, it can be easily seen that this cannot be optimized by brute force.

Thus, we consider this as a good example that can be used to analyze the performance of the crossover operators with incorporated local search.

In a paper co-authored by one of us [13] presented an optimization of product layout in the warehouse with a genetic algorithm, which used only the standard global search with the standard AEX crossover operator. The reader is referenced to that paper for more details about the problem, because due to space limitations we cannot explain them here.

In the current experiment we analyze the different ways of adding local search components to the AEX, KPoint, and HRndX crossovers to solve the warehouse optimization problem.

**Objective function** The objective function is expressed by the sum of all orders picking times within a certain time frame, e.g. one month (the smaller, the better). This objective can be measured by the length of the route that the workers must cover to complete all the orders. In practice, the length should be measured rather in time units than in distance units, as for example covering the same distance in a straight line is faster than going around a corner and especially covering the same distance in horizontal direction is faster than in vertical direction (to reach the product located on upper shelves). However, it does not matter for the functioning of the genetic algorithm.

As a byproduct of the optimization, we obtain shorter completion routes for all orders. (which is another NP-hard problem). To keep the example simple, we use the nearest neighbor algorithm followed by 2-opt to find the shortest order completion routes. This method is used only as an example to calculate the value of the objective function used by the genetic algorithm. Thus, we do not try to improve this mechanism of the order completion route generation, because it is not the purpose of the paper.

**Problem encoding** The particular locations in the warehouse (e.g. shelves) are represented by the positions in the chromosome. Each position corresponds to one location. The products are represented by particular numbers on the chromosome positions. For example, if a product number 1 is placed on the location number 3, product number 2 is placed on location number 2 and product number 3 is placed on location number 1, then the encoding of this product distribution is represented as: [3 2 1]. To optimize the objective function, the genetic algorithm needs to decide upon appropriate locations of particular products, that is, upon the appropriate order of the numbers in the chromosome.

**Local search component** To construct the local search for the warehouse example, we use the two commonly known facts. First, the products which are most frequently purchased should be placed close to the warehouse entry. We call this Single Product Frequency Search (SPFS). Second, the products which are frequently purchased together within one order should be placed close to each other. Pairwise Product Frequency Search (PPFS). However, optimizing the two criteria together is an NP-hard problem, as the orders differ one from another and particular products appear in many orders in different combinations with other products. For that reason, we cannot use the local search only, without the genetic algorithm, which provides the global search.

FPS can be used with some probability to determine the next position in the child instead of determining this position according to the base crossover operator rule.

Before applying SPFS, two arrays are created: an array of warehouse locations $A_{WL}$ sorted by the distance from the warehouse entry and an array of single product frequencies $A_{PS}$ (how many times each product occurred in all the orders together). The idea of SPFS is to try placing a product in a location which is on similar position in the $A_{WL}$ as the product position in $A_{PS}$. In this

way, more frequently purchased product get placed closer to the entry and the less frequently get placed further. However, SPFS cannot always place the most optimal product on each location, because it can use only the products, which are not positioned yet by the crossover operator, so it searches for free products and selects this one, which is most close to the optimal one.

Before applying PPFS, a pairwise product frequency matrix $M_{PF}$ is calculated. $M_{PF}$ contains the frequencies with which each two products occur together in the same orders. The idea is to select such a product for the current location, which coexists frequently in the same orders with the product already placed in the closest neighbor locations in the warehouse.

SPFS and PPFS can be used as well to resolve the conflicts in the crossover operator (replacing the default method of conflict resolving, which takes randomly any available product) as to select the next product in the chromosome (instead of the given crossover rule). In the second case, it is used with a certain probability.

Also, PPFS cannot always place the most optimal product on each location, but chooses the most optimal of the still available products.

### 4.2 The Example of Traveling Salesman Problem

For the evaluation we used 10 travels comprising between 50 and 1600 locations, which can be found at *kordos.com/iccs2022*

**Problem description and objective function** In the traveling salesman problem (TSP) the task is to visit all the cities from a given list, starting from the first one and returning to the first one in such a way that the total length of the travel will be minimal. The objective function is expressed by the total length of the travel $Tdist$ (the smaller, the better). To obtain this, the cities must be visited in a proper order.

**Problem encoding** The order of visiting the cities is encoded in the chromosome. Let us say that there are six cities on the list: A, B, C, D, E, F, and A is the starting city (and the ending one). For example, the following chromosome:

[A D B C F E]

represents the following order of visiting the cities: A->D->B->C->F->E->A and this order also determines the total distance $Tdist$ that must be covered to visit the cities in this order as the $Tdist = dist(A, D)+dist(D, B)+dist(B, C)+dist(C, F) + dist(F, E) + dist(E, A)$.

**Local search component** The common knowledge in the TSP is that cities that are close to each another should be rather visited one by one. This knowledge is implemented by the nearest neighbor algorithm, which always connects a given city with the closest city. Because of lack of the global search, it produces worse results than GA in this case. Nerveless, when used as the local search component inside GA, it allows improving the results.

### 4.3 Experimental setup and parameters

The following parameters of the genetic algorithm were used for all the experiments:

- base crossover operators: AEX, HGreX, HProX, HRndX for TSP, and KPoint, AEX, HRndX for the Warehouse Optimization Problem
- population size: 100
- number of children: 80 (80 children and the best 20 parents were promoted to the next generation)
- selection: tournament selection with 8 candidates per each parent.
- The GA was run for 1000 epochs, because no further improvement of results was observed while running the optimization for more than 1000 epochs.
- mutation: RSM mutation with the probability of 10%.
- resolver type 1: tournament selection with the number of candidates set to 25% of the available locations
- resolver type 2: nearest neighbor
- resolver type 3: random selection
- random resolver probability: we performed the experiments with 11 different probabilities: 0, 0.1, 0.2, ... 1.0.

We repeated each experiment 100 times. The average results for selected configurations are presented in Tables 1 and 2. We analyzed the fitness obtained after each number of epochs, so that the speed of achieving the results can be observed, and this is presented in Table 1, and in Figure 1 for the warehouse optimization problem with optimal probabilities of the tournament selection resolver.

For the TSP the results were similar, so due to the limited space here we decided not to present them in this form, but instead to show a detailed snapshot of the process, to show in detail how the types and probabilities of the resolvers influence the results. We show the snapshot at the 100-th epoch, because it shows also well the speed of the convergence of the optimization with various configurations. Obviously, the differences at the end of the optimization (1000-th epoch) are smaller, but still significant - proportionally to what we can see in Table 1.

To compare the results over all the dataset in Table 2, we introduced the relative route length for each dataset as the proportion of the route length obtained with a given set of parameters to the shortest obtained route lengths. These values were calculated as the average length obtained over 100 runs of the genetic algorithms for each dataset, and then averaged over all the datasets. Thus, the lowest possible value is 1.0. In fact, 1.0 does not appear in the table, because it would mean that some set of parameters were the best for each dataset and in practice it was not the case.

### 4.4 Analysis of the Results obtained for the Warehouse Problem

The first conclusion visible from Table 1, Table 2, and Figure 1 is that the addition of local search not only improves the final results, but also drastically accelerates the genetic algorithms process.

**Table 1.** Fitness for the warehouse optimization problem (sum of the order picking route lengths for the warehouse-orders set) obtained with different crossover operators with and without the local search (the lower, the better). The name of the warehouse-orders set indicates the number of position in the warehouse (p), number of orders (o) and number of items in all orders (i). For example 233p25o131i is the set of 233 positions, 25 orders and 131 items.

| warehouse-orders | epoch | Kpoint | Kpoint with local search | AEX | AEX with local search | HRndX | HrndX with local search |
|---|---|---|---|---|---|---|---|
| 233p25o131i | 10 | 7087 | 3745 | 8987 | 4558 | 8848 | 5153 |
| | 100 | 4607 | 3512 | 6541 | 3599 | 5663 | 3619 |
| | 1000 | 3905 | 3450 | 5251 | 3513 | 3691 | 2931 |
| 118p25o666i | 10 | 7495 | 6197 | 12088 | 9218 | 12049 | 9444 |
| | 100 | 6435 | 5997 | 9871 | 8139 | 9469 | 8184 |
| | 1000 | 6187 | 5946 | 8476 | 7768 | 8335 | 7846 |
| 81p25o804i | 10 | 33213 | 24146 | 35004 | 26983 | 34710 | 28235 |
| | 100 | 22230 | 20415 | 26938 | 23624 | 24158 | 24079 |
| | 1000 | 19600 | 19427 | 22828 | 22012 | 20892 | 21927 |
| 71p34o392i | 10 | 10053 | 7174 | 11066 | 8037 | 11069 | 7730 |
| | 100 | 7330 | 6554 | 7656 | 6794 | 7469 | 6701 |
| | 1000 | 6796 | 6497 | 6836 | 6577 | 6879 | 6516 |
| 106p25o668i | 10 | 30456 | 17578 | 37250 | 19568 | 34094 | 23196 |
| | 100 | 20611 | 15032 | 25888 | 17176 | 23285 | 17504 |
| | 1000 | 16351 | 14662 | 21544 | 16478 | 19722 | 16416 |
| 127p25o406i | 10 | 21191 | 18364 | 28898 | 20897 | 30093 | 21807 |
| | 100 | 16089 | 13824 | 17709 | 20060 | 17864 | 17400 |
| | 1000 | 15588 | 12768 | 16939 | 15674 | 16803 | 14957 |

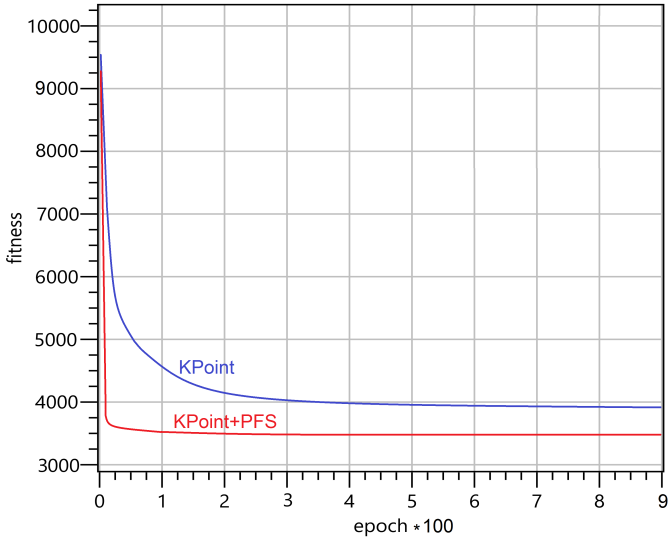


**Fig. 2.** Comparison of optimization progress with standard KPoint crossover (blue) and KPoint crossover with product frequency search (red) for the *100p34o434* warehouse.

**Table 2.** Relative fitness value after 100 iterations averaged over 100 runs on 10 datasets for TSP (the lower the better).

| | | | non-conflict resolver probability | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| crossover | conflict | non-c. | p=0 | p=0.1 | p=0.2 | p=0.3 | p=0.4 | p=0.5 | p=0.6 | p=0.7 | p=0.8 | p=0.9 | p=1 |
| AEX | Rnd | Rnd | 8.76 | 9.91 | 10.15 | 10.30 | 10.42 | 10.51 | 10.59 | 10.66 | 10.72 | 10.77 | 10.80 |
| AEX | NN | Rnd | 1.21 | 2.48 | 3.84 | 5.05 | 6.12 | 7.08 | 7.96 | 8.73 | 9.47 | 10.16 | 10.80 |
| AEX | Trn | Rnd | 1.56 | 3.64 | 4.92 | 6.01 | 6.93 | 7.70 | 8.44 | 9.11 | 9.70 | 10.27 | 10.80 |
| AEX | Rnd | NN | 8.76 | 5.45 | 4.45 | 3.73 | 3.12 | 2.60 | 2.11 | 1.69 | 1.38 | 1.15 | 1.18 |
| AEX | NN | NN | 1.21 | 1.17 | 1.14 | 1.11 | 1.10 | 1.09 | 1.09 | 1.08 | 1.09 | 1.09 | 1.18 |
| AEX | Trn | NN | 1.56 | 1.55 | 1.40 | 1.29 | 1.22 | 1.17 | 1.12 | 1.11 | 1.10 | 1.10 | 1.18 |
| AEX | Rnd | Trn | 8.76 | 6.20 | 5.42 | 4.70 | 4.13 | 3.59 | 3.06 | 2.52 | 1.97 | 1.43 | 1.17 |
| AEX | NN | Trn | 1.21 | 1.36 | 1.35 | 1.29 | 1.23 | 1.17 | 1.12 | 1.08 | 1.07 | 1.09 | 1.17 |
| AEX | Trn | Trn | 1.56 | 1.60 | 1.46 | 1.35 | 1.26 | 1.18 | 1.13 | 1.11 | 1.11 | 1.12 | 1.17 |
| HProX | Rnd | Rnd | 4.17 | 6.93 | 8.17 | 8.95 | 9.49 | 9.87 | 10.15 | 10.38 | 10.54 | 10.71 | 10.80 |
| HProX | NN | Rnd | 1.19 | 2.30 | 3.68 | 5.05 | 6.28 | 7.34 | 8.24 | 9.00 | 9.67 | 10.26 | 10.80 |
| HProX | Trn | Rnd | 1.47 | 3.14 | 4.65 | 5.95 | 7.05 | 7.94 | 8.68 | 9.31 | 9.85 | 10.37 | 10.80 |
| HProX | Rnd | NN | 4.17 | 2.49 | 2.12 | 1.79 | 1.58 | 1.41 | 1.26 | 1.19 | 1.10 | 1.10 | 1.18 |
| HProX | NN | NN | 1.19 | 1.11 | 1.09 | 1.09 | 1.08 | 1.08 | 1.08 | 1.08 | 1.08 | 1.09 | 1.18 |
| HProX | Trn | NN | 1.47 | 1.25 | 1.19 | 1.14 | 1.12 | 1.10 | 1.10 | 1.09 | 1.09 | 1.09 | 1.18 |
| HProX | Rnd | Trn | 4.17 | 3.79 | 3.52 | 3.17 | 2.77 | 2.39 | 2.00 | 1.62 | 1.36 | 1.20 | 1.16 |
| HProX | NN | Trn | 1.19 | 1.28 | 1.27 | 1.23 | 1.18 | 1.12 | 1.09 | 1.07 | 1.09 | 1.10 | 1.16 |
| HProX | Trn | Trn | 1.47 | 1.39 | 1.33 | 1.25 | 1.19 | 1.13 | 1.11 | 1.10 | 1.11 | 1.12 | 1.16 |
| HGreX | Rnd | Rnd | 2.97 | 5.57 | 7.07 | 8.09 | 8.86 | 9.42 | 9.85 | 10.17 | 10.44 | 10.64 | 10.80 |
| HGreX | NN | Rnd | 1.11 | 2.05 | 3.36 | 4.68 | 5.91 | 7.03 | 8.00 | 8.85 | 9.59 | 10.22 | 10.80 |
| HGreX | Trn | Rnd | 1.23 | 2.64 | 4.08 | 5.40 | 6.57 | 7.56 | 8.42 | 9.15 | 9.77 | 10.31 | 10.80 |
| HGreX | Rnd | NN | 2.97 | 1.80 | 1.55 | 1.41 | 1.30 | 1.18 | 1.12 | 1.11 | 1.11 | 1.11 | 1.18 |
| HGreX | NN | NN | 1.11 | 1.06 | 1.07 | 1.07 | 1.08 | 1.08 | 1.08 | 1.09 | 1.09 | 1.10 | 1.18 |
| HGreX | Trn | NN | 1.23 | 1.07 | 1.08 | 1.09 | 1.09 | 1.09 | 1.09 | 1.09 | 1.09 | 1.10 | 1.18 |
| HGreX | Rnd | Trn | 2.97 | 2.85 | 2.62 | 2.39 | 2.10 | 1.88 | 1.64 | 1.45 | 1.26 | 1.21 | 1.17 |
| HGreX | NN | Trn | 1.11 | 1.07 | 1.08 | 1.08 | 1.06 | 1.05 | 1.06 | 1.07 | 1.09 | 1.10 | 1.17 |
| HGreX | Trn | Trn | 1.23 | 1.11 | 1.11 | 1.10 | 1.07 | 1.08 | 1.09 | 1.10 | 1.11 | 1.12 | 1.17 |
| HRndX | Rnd | Rnd | 7.11 | 9.33 | 9.77 | 10.06 | 10.26 | 10.39 | 10.52 | 10.60 | 10.69 | 10.76 | 10.79 |
| HRndX | NN | Rnd | 1.47 | 3.47 | 5.07 | 6.19 | 7.07 | 7.81 | 8.51 | 9.17 | 9.77 | 10.34 | 10.79 |
| HRndX | Trn | Rnd | 2.17 | 4.63 | 6.06 | 6.99 | 7.73 | 8.36 | 8.92 | 9.46 | 9.95 | 10.40 | 10.79 |
| HRndX | Rnd | NN | 7.11 | 4.86 | 3.74 | 2.88 | 2.23 | 1.72 | 1.44 | 1.24 | 1.13 | 1.10 | 1.18 |
| HRndX | NN | NN | 1.47 | 1.26 | 1.18 | 1.13 | 1.11 | 1.10 | 1.08 | 1.08 | 1.08 | 1.09 | 1.18 |
| HRndX | Trn | NN | 2.17 | 1.64 | 1.42 | 1.28 | 1.21 | 1.15 | 1.11 | 1.10 | 1.09 | 1.09 | 1.18 |
| HRndX | Rnd | Trn | 7.11 | 5.70 | 4.79 | 4.04 | 3.42 | 2.88 | 2.35 | 1.86 | 1.43 | 1.20 | 1.16 |
| HRndX | NN | Trn | 1.47 | 1.58 | 1.50 | 1.40 | 1.30 | 1.21 | 1.14 | 1.10 | 1.09 | 1.10 | 1.16 |
| HRndX | Trn | Trn | 2.17 | 1.85 | 1.62 | 1.46 | 1.33 | 1.22 | 1.16 | 1.12 | 1.11 | 1.11 | 1.16 |

For the warehouse optimization problems the optimal probabilities, in a non-conflict resolvers, based on our experiments, the optimal probability of using FTS was about 0.3. In each place, where SPFS can be applied, also PPFS can be applied and the decision about which operator to use is taken randomly each time (we use SPFS with probability 0.65 and PPFS with probability 0.35).

The best performing base crossover operator was KPoint, followed by AEX, followed by HRndX. As it can be seen from Table 1 and from Figure 1, the incorporation of the product frequency search (PFS) in the crossover operators significantly improved the results, while still preserving the order of the quality: the best one was the improved KPoint, followed by the improved AEX, followed by the improved HRndX.

The progress with the improved operators was faster, and the final results were better. Significant differences were obtained already after 10 epochs. This allows to efficiently use the crossover operators with local search for running multiple optimizations for only 100 or even only 10 epochs, and then to choose the best performing run to continue the optimization. That is because in most cases, the best run after 10 or 100 epochs is also be the best run after 1000 epochs [13]. The choice of the best run with the improved crossover operators can be done earlier than with the base crossover operators, because the improvement is much faster in the initial epochs. This is an additional benefit in terms of computational time of the improved crossover operators.

### 4.5 Analysis of the Results Obtained for TSP

The values that represent the basic crossover operators in Table 2 are those with random conflict resolver and with zero probability of random resolver, that is: 8.76 for AEX, 4.17 for HProX, 2.97 for HGreX, and 7.11 for HRndX.

The best results are obtained for all the four crossover operators with a combination of nearest neighbor (NN) resolver and tournament resolver (Trn) or with two nearest neighbor resolvers. For AEX, HProX and HRndX, the optimal probability of the random resolver in these cases is between 0.4 and 0.8 for two NN resolvers and between 0.6 and 0.8 for one NN and one Trn resolver. The highest optimal probability of the mixed resolver corresponds to a weaker preference for the shortest distance inside the resolvers than inside two NN resolvers. Thus, in both cases, the optimal local component is similar, but can be obtained either by the second NN resolver or by an increased random resolver probability in the mixed configuration.

HGreX, on the other hand, has already embedded some local search mechanism, as was already mentioned. This implies two results. First, in its basic form, is the best performing from the crossover operators (the relative route length 2.97). Second, it does need so strong additional local search as the other crossover operators. For that reason it obtained the best performance with random resolver probability of 0.1 for NN/NN and Trn/NN configuration and about 0.4-0.5 for NN/Trn, and also it achieved very good performance for Trn/Trn resolvers with the optimal probability of 0.4-0.5.

As it can be seen, the random resolvers used in the situation, where there is no conflict do not make any sense, as they only worsens the results, especially if they are used with high probabilities.

Although in some cases the standard crossover operators were able to find the optimal route and there was nothing to improve in terms of the results, incorporating the local search always allowed to reduce dramatically the number of epochs required to find the solution.

## 5 Conclusions

We analyzed the AEX, HGreX, HProX, HRndX and KPoint crossover operators with local search mechanisms and investigated the optimal balance between exploration and exploitation in the crossover operators for the genetic algorithm based optimization. In most cases, HGreX was the superior crossover operator TSP (it is worth noting that HGreX has already embedded some local search component in its basic form), while KPoint performed best for the warehouse problem. Nevertheless, by adding the resolvers to the crossover operators, the results could be improved in many cases and could be obtained faster. Especially, the great acceleration of the optimization process was observed.

Although, in the paper, we used two examples: the warehouse optimization and TSP, the approach is more universal and can be extended to to other problems, where the concept of cost between two locations can be defined. This is especially the case in many production optimization, planing and scheduling problems, which we are going to consider in our future research and practical implementations.

An important issue is to provide the optimal amount of the local search (see Table 2). It is also likely, that the results can still be improved if the local search frequency is adjusted dynamically during the optimization process, which we are going to further investigate. It is also worth including other crossover operators in the future experiments.

## References

1. Abdoun, O., Abouchabaka, J.: A comparative study of adaptive crossover operators for genetic algorithms to resolve the traveling salesman problem. International Journal of Computer Applications **31** (2011)
2. Affenzeller, M., Wagner, S., Winkler, S., Beham, A.: Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications. CRC Press (2018)
3. Al-Furhud, M.A., Ahmed, Z.H.: Experimental study of a hybrid genetic algorithm for the multiple travelling salesman problem. Mathematical Problems in Engineering **20**, 3431420 (2020)
4. Avdeikins, A., Savrasovs, M.: Making warehouse logistics smart by effective placement strategy based on genetic algorithms. Transport and Telecommunication **20**(4), 318–324 (2019)

5. Bartholdi, J.J., Hackman, S.T.: Warehouse and Distribution Science. Available online: https://www.warehouse-science.com/book/index.html (2019)
6. Bolaños Zuñiga, J., et. al.: Optimization of the storage location assignment and the picker-routing problem by using mathematical programming. Applied Sciences **10(2)**(534) (2020)
7. Cicirello, V.A.: Non-wrapping order crossover: An order preserving crossover operator that respects absolute position. In: GECCO 2006. p. 1125–1131 (2006)
8. Ha, Q.M., Deville, Y., Pham, Q.D., Hà, M.H.: A hybrid genetic algorithm for the traveling salesman problem with drone. https://arxiv.org/abs/1812.09351 (2018)
9. Hassanat, A.B.A., Alkafaween, E.: On enhancing genetic algorithms using new crossovers. Int. Journal of Computer Applications in Technology **55** (2017)
10. Hongx, Z., Guohui, Z., Chili, C.: On directed edge recombination crossover for atsp. In: ICNC 2006: Advances in Natural Computation. pp. 783–791 (2006)
11. Hwang, H.: An improvement model for vehicle routing problem with time constraint based on genetic algorithm. Computers and Industrial Engineering **42**, 361–369 (2002)
12. Kordos, M., Blachnik, M., Scherer, R.: Fuzzy clustering decomposition of genetic algorithm-based instance selection for regression problems. Information Sciences **587**, 23–40 (2021)
13. Kordos, M., Boryczko, J., Blachnik, M., Golak, S.: Optimization of warehouse operations with genetic algorithms. Applied Sciences **10(14)**, 4817 (2020)
14. Lin, B.L., Sun, X., Salous, S.: Solving travelling salesman problem with an improved hybrid genetic algorithm. Journal of computer and communications **4(15)**, 98–106 (2016)
15. Ma, M., Li, H.: A hybrid genetic algorithm for solving bi-objective traveling salesman problems. J. Phys.: Conf. Ser **887**, 012065 (2017)
16. Puljić, K., Manger, R.: Comparison of eight evolutionary crossover operators for the vehicle routing problem. Mathematical Communications **18**, 359–375 (2013)
17. Rakesh, V., Kadil, G.: Layout optimization of a three dimensional order picking warehouse. IFAC-PapersOnLine **48**, 1155–1160 (2017)
18. Santos, J., Ferreira, A., Flintsch, G.: An adaptive hybrid genetic algorithm for pavement management. International Journal of Pavement Engineering **20(3)** (2019)
19. Simon, D.: Evolutionary Optimization Algorithms. Wiley (2013)
20. Singh, K., Sundar, S.: A hybrid genetic algorithm for the degree-constrained minimum spanning tree problem. Soft Computing **24(3)**, 2169–2186 (2020)
21. Tan, H., Lee, L.H., Zhu, Q., Ou, K.: Heuristic methods for vehicle routing problem with time windows. Artificial Intelligence in Engineering **16**, 281–295 (2001)
22. Ting, C.K.: Improving edge recombination through alternate inheritance and greedy manner. In: EvoCOP 2004: Evolutionary Computation in Combinatorial Optimization. pp. 210–219 (2004)
23. Van Gils, T., Ramaekers, K., Caris, A., De Koster, R.: Designing efficient order picking systems by combining planning problems: State-of-the-art classification and review. European Journal of Operational Research **267**, 1–15 (2018)
24. Zhang, W., Zhu, J., Yuan, R.: Optimization of automated warehouse storage location assignment problem based on improved genetic algorithm. In: The 9th Int. Conference on Logistics, Informatics and Service Sciences. pp. 297–311 (2019)