Soft Computing (2020) 24:12855–12885
https://doi.org/10.1007/s00500-020-04712-2

METHODOLOGIES AND APPLICATION

A novel ODV crossover operator-based genetic algorithms for traveling salesman problem

P. Victer Paul[1] · C. Ganeshkumar[2] · P. Dhavachelvan[3] · R. Baskaran[4]

Published online: 7 February 2020

Abstract

The genetic algorithm is a popular meta-heuristic optimization technique whose performance depends on the quality of the initial population and the crossover operator used to manipulate the individuals to obtain the final optimal solution. It is evident that when similar principle is followed for population seeding and crossover operators, it can enhance the speed of convergence and the quality of final individuals. The recent and popular population seeding technique for combinatorial genetic algorithm is ordered distance vector-based population seeding which works best with respect to convergence rate and diversity. However, the technique could not achieve the zero error rate convergence for the large-sized test instances. Thus, in this paper, an ordered distance vector-based crossover operator is proposed that exclusively exploits the advantages of individuals' generated using the same initialization methods to attain the complete convergence, particularly for most of the large-sized test instances considered. One of the famous combinatorial problems of traveling salesman problem obtained from standard library is chosen as the testbed. From the experimental results, the proposed genetic algorithm model outshines the other existing and popular working genetic algorithm models in the literature.

Communicated by V. Loia.

✉ P. Victer Paul
victerpaul@gmail.com

C. Ganeshkumar
gcganeshkumar@gmail.com

P. Dhavachelvan
dhavachelvan@gmail.com

R. Baskaran
baaski@annauniv.edu

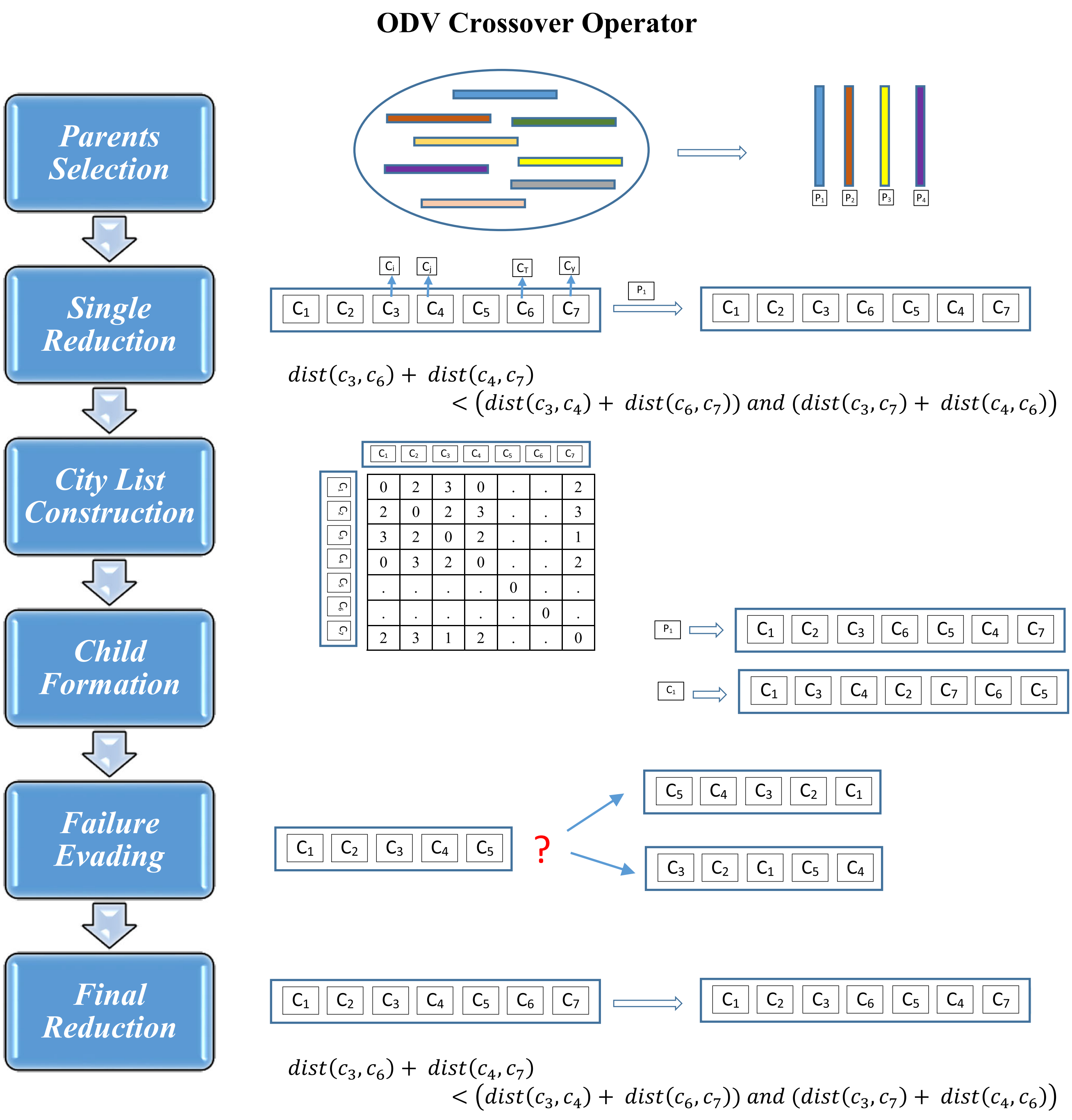
[1] Department of Computer Science and Engineering, Indian Institute of Information Technology Kottayam, Kottayam, Kerala, India

[2] Indian Institute of Plantation Management Bangalore (IIPMB), Bangalore, Karnataka, India

[3] Department of Computer Science, Pondicherry University, Puducherry, India

[4] Department of Computer Science and Engineering, Anna University, Chennai, India

Graphic abstract



	C1	C2	C3	C4	C5	C6	C7
C1	0	2	3	0	.	.	2
C2	2	0	2	3	.	.	3
C3	3	2	0	2	.	.	1
C4	0	3	2	0	.	.	2
C5	.	.	.	.	0	.	.
C6	.	.	.	.	.	0	.
C7	2	3	1	2	.	.	0

Keywords Genetic algorithm · Population seeding · Crossover · Ordered distance vector · Permutation coded

1 Introduction

Genetic algorithms (GAs), the bio-inspired and stochastic global optimization techniques are most popular in dealing complex problems with very large search space efficiently in the majority of the cases. GAs belong to the family of evolutionary computation models inspired by biological evolution, natural selection and survival of the fittest in living organisms. The life cycle of classical GA consists of the several phases like initial population (population seeding), selection, crossover, mutation and termination constraint (Paul et al. 2014). The first phase occurs only once, and the rest of the phases are repeated until the termination condition is satisfied. Generally, the classical GA takes more computation time to evolve at an optimal solution, which may be rectified using heuristics in a problem-specific manner. In other words, the heuristics may definitely reduce the computation time to improve the overall ability and to evolve the optimal solution, thus resulting in hybrid GA (Marinakis et al. Marinakis and Marinaki 2010; Chen and Chien 2011; Yingzi et al. 2007; Wang et al. 2014). Many researchers recommend modification or hybridization in each phase of the genetic algorithm using problem-specific knowledge to enhance the solution quality and/or efficiency of the algorithm. Hybridization of GA makes the algorithm more problem dependent (less robust), but the performance in terms of convergence and computation time can be improved significantly (Sivanandam and Deepa 2008). The performance of the GAs may be improved through various phases of GA. Normally in all cases, the issue of performance tuning may be carried out in a phase-specific manner and of course, the phase-specific improvement will automatically result in overall improvement.

In GA, crossover comprehends the construction of the offspring using the individuals selected from the population (Pandey 2016; Pan et al. 2016). In Nagata (2004), Nagata discussed the significances of the crossover operator in GA to evolve the final optimal solution and presented a set of critical factors to be considered in designing a crossover operator for GA.

- The ratio of new characteristics introduced into the offsprings should be adjusted according to the trade-off between the quality and variety of the offsprings.
- The quality of the new characteristics should be constructive (the probability that new characteristics are included in an optimal solution should be high).

Problem-specific crossover methods can be utilized to recuperate the feasibility of solutions produced by the traditional crossover operators (El-Mihoub et al. 2006). The crossover phase is highly associated with the population initialization phase; the crossover operator has to exploit the potential of the individuals generated initially and to improve the individual's quality in the successive generations. The overall performance of the GA highly depends on the quality of the initial population supplied and also the type of crossover operator used to manipulate the individuals to obtain the final optimal solution (Andal and Sathiamoorthy 2001). Specifically, the association between the principles of the population seeding technique and crossover operator plays a vital role in reducing the computation time and increasing the probability of finding the global optimal solution. If the population seeding and the crossover techniques are not principally associated together, the crossover would destroy the potential information induced at the initialization stage and try to enhance the quality of the solution in its own method at every generation, which may cause an extensive delay in attaining the optimal value. On the other hand, when the population seeding technique and crossover operator follow the similar principle of generating or improving the quality of individuals, it can enhance the speed of convergence as well as the quality of best individual (Andal and Sathiamoorthy 2001). From the previous sections, it has been observed that the ODV-based methods work best with respect to convergence rate and diversity. However, it is necessary to point that it could not achieve the zero error-rate convergence for any of the large-sized test instances after a certain point of generations. This would be because of the difference in the principles of the population seeding and crossover techniques (Poon and Carter 1995). From this perspective, designing an ODV-based crossover operator that exclusively exploits the advantages of individuals' generated using ODV initialization methods may assist to attain the best optimal solution, particularly for large-sized test instances.

2 Background study

In this section, the recent and popular population seeding and crossover techniques for combinatorial GA are discussed in Sects. 2.1 and 2.2, respectively, for a better understanding of the work proposed and the experimental setup.

2.1 Study on the population initialization techniques

Random Initialization Random population initialization is the simple and common population generation technique preferred when lacking prior information on the problem to solve. A variety of random number generation methods have been proposed such as quasi-random, Sobol random and uniform random sequence (Deng et al. 2015; Katayama et al. 2000).

Nearest Neighbor (NN) Technique Nearest Neighbor (NN) population seeding technique is a well-known replacement for random population initialization, to construct the initial population of solutions, in greedy fashion, for permutation-coded GAs (Kaur and Murugappan 2008; Ting 2013; Shubhra et al. 2007).

Selective Initialization (SI) Technique In Rong (1997), a selective initialization scheme using *k-nearest-neighbor* sub-graph has been proposed. In this technique, a k-*nearest-neighbor* sub-graph is formulated, from the distance matrix, as a graph includes all routes of cities c_i and c_j, such that the city c_i is among the *k-nearest neighbors* of city c_j or city c_j is among the *k-nearest neighbors* of city c_i, i.e., a list of k-nearest neighbors for each city is generated in advance.

The hybrid population seeding techniques have the benefit of good quality individuals and fast convergence, but lack in terms of randomness, individual diversity and ability to converge to the global optimal solution. Therefore, an efficient ordered distance vector (ODV)-based population seeding technique is proposed and detailed theoretical discussion can be found in Paul et al. (2013a, b), Shanmugam et al. (2013) and Moganarangan et al. (2014). Paul et al. (2013a, b) and Arthi et al. (2015). The two significant characteristics that make the ODV population seeding technique distinct from the other techniques are p*otential sequence and individual diversity*. Based on these two characteristics, three different varieties of ODV-based population seeding techniques have been proposed, namely ODV-EV, ODV-VE and ODV-VV techniques. In the same literature, *'ba'* value has been defined as follows: The Best Adjacent (*ba*) number is an important factor and derived using the assumption that, in an optimal solution, any city c_i is connected to city c_j such that c_j is one of the c_i's nearest *'ba'* number of cities.

2.2 Study on the crossover techniques

The quality preserved crossover operators conserve the good information from the parent individuals and add new information heuristically such that the quality of the offspring could be improved. Whitely et al. (1989) claim that the preservation of gene order-based quality is more important than the positions of the individual gene in the design of crossover operators for combinatorial problems. This kind of crossover operators is also called as edge-based operators in case of TSP and vehicle routing problems. The various quality preserved crossover techniques are discussed as follows:

Edge Recombination Crossover (ERX) Edge Recombination Crossover was developed by (Whitley et al. 1989), assuming that only the quality of genes is important not their direction. It inherits common sequence of genes from parent solutions as much as possible and randomly adds new genes to generate feasible solutions. Mathias and Whitley (1992) introduced subsequent versions of ERX, Edge-2 and Edge-3. The Edge-2 method improves ERX by increasing the preference for the information present in both parents and Edge-3 attempts to minimize the inclusion of new information on the occurrence of crossover failure. Edge-4 and Edge-5 proposed by Nguyen et al. (2000) try to recover from the crossover failures using different methods, and performance improvement has been investigated using TSP. Though several variants of ERX are available, the crossover failure could not be eradicated completely.

Edge Assembly Crossover (EAX) In Nagata and Kobayashi (1997), Nagata proposed an effective Edge Assembly Crossover (EAX) in which search block identifies the best individual to exchange the information among the parents for large problem instance in a considerable CPU time (Tsai et al. 2004). EAX shows the best performance being compared with other crossovers from the viewpoints of inheritance of good information (Fei and Guangzhou 2009).

Partition Crossover (PX) The partition crossover operator (Whitley et al. 2009; Whitley et al. 2010; Hains 2012) attempts to recombine two parent individuals that are locally optimal to build the offsprings which are expected to be local optimal individuals. Thus, the PX acts as a tunnel between the local optimal solutions in the search space.

Sub-tour Recombination Crossover operator (SRX) Masafumi et al. (2010) proposed Sub-tour Recombination Crossover operator (SRX) that segregates the parent individuals into several subsequences whose lengths are restricted to a predefined value and then recombines the subsequences from both parents to build a new solution which improves the effective recombination capability.

A multi-parent crossover is a new and interesting approach in GA to generate the offspring using the potential information of more than two parent individuals. In Liu et al. (2012), the authors proposed a novel cluster oriented differential evolution model using two multi-parent crossover techniques. The proposed crossover is the hybrid approach of the one-step k-means clustering using the objective space distance measure and multi-parent crossover with DE. A novel diversity-based hybrid evolutionary model had been proposed for the graph coloring problem (Porumbel et al. 2010). In the work, they introduced a special cluster-based multi-parent crossover operator which depends on the various significant features to identify the meaningful information for the offspring construction. Ting et al. (2010) proposed a multi-parent partially mapped crossover for combinatorial optimization problems. The model works by choosing a collection of parents and applying the partially mapped crossover based to construct the offspring.

A set of multi-parent crossovers for real-coded GAs had been proposed in Wang et al. (2016), Tsutsui and Ghosh (1998), Tsutsui and Jain (1998) such as center of mass crossover, feature-oriented multi-parent crossover, and seed crossover. The authors also showed that proposed multi-parent crossovers can explore and exploit the search space for improved performance though the performance is problem-specific. These studies validate the dominance of multi-parent crossover over traditional two-parent crossover. The performance of the multi-parent crossover approach is often validated on numerical optimization problems (Eiben 2002; Ting 2005; Tsutsui et al. 1999; Tsutsui and Ghosh 1998; Tsutsui and Jain 1998) rather than on combinatorial problems (Ting et al. 2010; Eiben et al. 1994). Thus, the multi-parent crossover-based approach to solve combinatorial optimization problems is still lacking.

3 Proposed system

3.1 Problem statement

As described in Sect. 1, the coordination between the population seeding and the crossover techniques used in the GA plays a vital role in exploring the search space and exploiting the quality of individuals generated at each generation. At the same time, the operational difference in the individual initialization and manipulation techniques may take a long time to converge to an optimal solution and also reduce the probability of obtaining the global optimal solution (Sivanandam and Deepa 2008). On the other hand, the ODV-based population seeding technique was proved as an effective method for population initialization for permutation-coded GA (Paul et al. 2014, 2015); the study also reveals that the existing random and quality preserved crossovers could not exploit the potential of individuals generated at the initial stage. These factors motivated to propose an effective ODV-based Crossover Operator, based on the ODV matrix used at the population seeding stage, to bridge the operational difference in the initialization and crossover methods and also to exploit the potentials of individuals generated using ODV initialization technique.

3.2 ODV-based crossover operator (ODVX)

3.2.1 Principles of ODVX operator

The ODV-based crossover operator (ODVX) uses the ODV matrix (Paul et al. 2014, 2015) and multi-parent techniques (Chuan 2007) to perform the crossover operation. In addition to these, the proposed crossover operator modifies and extends the failure reduction method dealt in Chuan (2007) to effectively handle the situation of crossover failures. There are four significant characteristics that make the ODV-based crossover technique distinct from the others and they are as follows:

- *Multi-parent crossover* In general, crossover operation is performed by recombining the information of two parent individuals. But, the multi-parent recombination technique (Chuan 2007) allows more than two parents to participate in the recombination process. The main advantage of the multi-parent method over the two parent method is that it helps to produce the offsprings with more possible combinations and quality information from different parents improves the diversity of the offsprings. The multi-parent method also has the inbuilt characteristic to overcome the crossover failures by extracting ample information from the parents.
- *Two-way potential extraction* The best performing crossovers, like PX and EAX, attempt to extract the goodness of the parent individuals in a single direction and completely neglecting the possible extort from another way. This reduces the quantity of potential information inherited from the parent individuals and also the possibility of generating different combinations of offspring for better search space exploration. The ODVX operator, similar to ERX, follows two-way potential extraction method to inherit the goodness of the parent individuals to the extent that possible.
- *Better new information inclusion* Inclusion of new information into the offspring is crucial to enhance the effectiveness and exploration capability of the crossover operator. The existing crossover techniques introduce the new information either by random (Maaranen et al. 2004) or greedy (Yingzi et al. 2007) methods; however, the random method may collapse the potential of the offspring produced and the greedy method increases the chance of getting trapped in the local optima (Sivanandam and Deepa 2008). In contrast, the proposed ODVX technique introduces fresh information into the offsprings using ODV matrix, which is formulated with features such as randomness, potential sequence and diversity, and helps to overcome the problems that exist with random and greedy methods.
- *Crossover Failure Evading* Occurrence of crossover failure results in the introduction of strange information into the offspring, either by random or greedy fashion, to overcome the failure. As discussed, this may get struck at the local optima and also agitate the quality of the offspring produced. The proposed ODVX technique uses a set of rule-based offspring (partially built) alteration schemes to effectively overcome the crossover failure condition.

These characteristics facilitate the proposed ODVX technique to effectively inherit the potential information from the parent individuals into the offspring and also to enhance the exploitation and exploration capabilities of the crossover operator.

3.2.2 Algorithm for ODVX operator

The algorithm for ODVX operator consists of five different stages as in Algorithm 1, and each stage performs a set of operations to produce the offspring(s) with better quality. The five stages of the ODVX, shown in Fig. 1, can be briefed as follows:

- *Stage-1: Parents Selection* This stage selects a precise number of parent individuals from the current population based on the size of the problem instance and the frequency of crossover failure.
- *Stage-2: Initial Reduction* This stage offers a single-step quality improvement operation on each parent individuals selected.
- *Stage-3: City List Formulation* A City List is formulated using the information present in these of improved parent individuals.
- *Stage-4: Offspring Construction* This stage performs a set of rule-based offspring generation operations using the City List generated at the Stage-3 of the algorithm.
- *Stage-5: Failure Evading* This stage is an on-demand scenario; invoked only on the occurrence of crossover failure at the Stage-4. This stage uses a set of rule-based

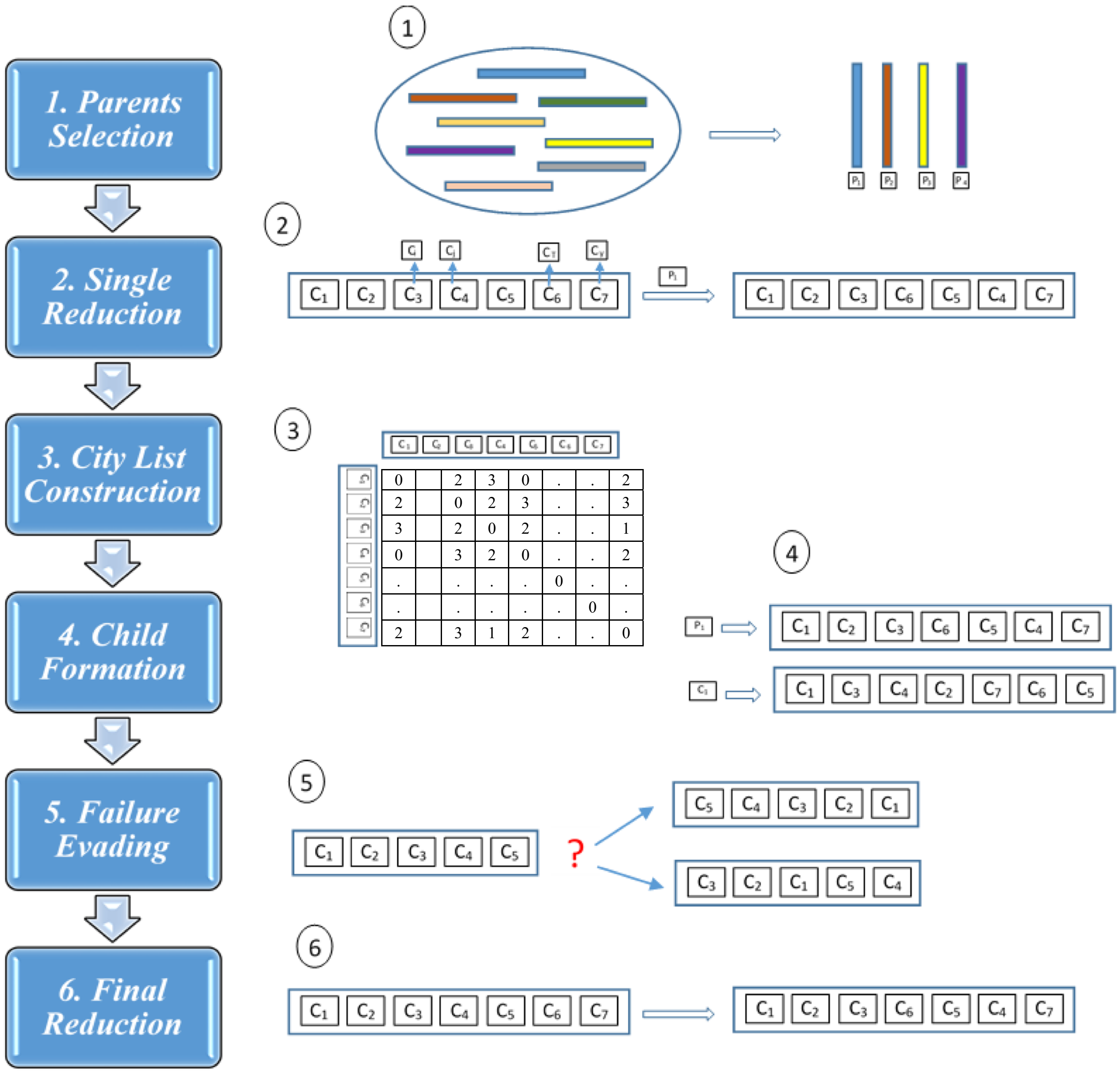


Fig. 1 Stages of ODV crossover operator

offspring (partially built) alteration schemes to effectively overcome the crossover failure condition.

- *Stage-6: Final Reduction* This stage offers a single-step quality improvement operation on each offspring individuals generated at the Stage-4 of the algorithm.

The functionalities of each stage of the algorithm are explained as follows:

Stage-1: Parents Selection

The Stage-1 decides the number parent individuals to be selected for recombination with respect to the factors such as the size of the problem, expected the quality of the offspring and the frequency of the crossover failure. Let *'k'* be the set of *'m'* number of parents selected from the current population,

$$k = \{k_0, k_1, k_2 \ldots k_{\mathrm{m}}\}$$

The mixture of *'m'* number of parents chosen can be given as,

$$m = m_{\mathrm{el}} + m_{\mathrm{pop}}$$

where

- 'm_{el}' refers to the number of parents selected from the elitist part of the current population.
- 'm_{pop}' refers to the number of parents selected from the current population (except elitist).

The value of *'m'* is decided based on the size of the problem and requirement to reduce the number of crossover failure. The value of *'m'* is inversely proportional to the number of crossover failures and directly proportional to the computation time. The value of 'm_{el}' is chosen based on the requirement of the transfer of quality information from the best (elitist) solutions of the current population. The 'm_{el}' is kept high on the need of fast convergence and low to increase the exploration of the search space. It is better to gradually increase the 'm_{el}' toward the termination of the algorithm. The 'm_{pop}' helps to evolve different combinations of the offspring and also decides the crossover failure evading and search space exploration abilities of the operator.

Stage-2: Single Reduction

This stage helps to improve the quality of the parent individuals in prior to the actual crossover process. Let 'k_i' be a parent individual from the set 'k'. Find c_i and c_j are the cities in k_i with the largest distance value and then discover the cities c_x and c_y in k_i which satisfy the following condition,

$$\mathrm{dist}(c_i, c_j) + \mathrm{dist}(c_x, c_y) > \min(\mathrm{dist}(c_i, c_x) + \mathrm{dist}(c_j, c_y), \mathrm{dist}(c_i, c_y) + \mathrm{dist}(c_j, c_x))$$

It should be also noted that the rank of cities to which c_i and c_j are joined (c_x and c_y, respectively) should be less than or equal to the *'ba'* value used at the population seeding stage of the GA (Paul et al. 2013a, b). If no cities satisfy such condition, then the condition has to be checked for the next largest distance cities in k_i. The total number of times the Single Reduction step $n(\mathrm{SR})$ is performed at the completion of the GA can be given as,

$$n(\mathrm{SR}) = n \times \mathrm{Gen}$$

where

- *'n'* is the size of the population.
- 'Gen' is the total number of generations of execution.

Thus, the Stage-2 of ODVX attempts to improve the quality of the individuals with a simple operation which helps to enhance the overall efficacy of the algorithm.

Stage-3: City List Formulation

In this stage, a City List *cityList* has been formulated, similar to Edge List in ERX (Whitley et al. 1989), which is a two-dimensional matrix of size $n \times 2m$ and filled up using the information present in the set of improved parent individuals. Each row in the *cityList* corresponds to the city c_i, where $i \in \{1, n\}$, can be represented as $cityList(c_i)$ and initialized as $cityList(c_i) = \emptyset$ at the start of the stage. The $cityList(c_i)$ can be updated with the cities adjacent to the city c_i in each of the parent individual in the set k. The same city can be adjacent to a particular city in more than one parent individual and such repeated adjacent city is marked with its corresponding repetition number. The repetition number of city c_i for city c_j, $\mathrm{rep}(c_i, c_j)$ can be given as,

$$\mathrm{rep}(c_i, c_j) = \sum_{i=1}^{m} k_i(\mathrm{adj}(c_i, c_j))$$

```
Stage - 1: Parents Selection
    Step 1.1: Initialize the set k with 'm' number of parents from the current population.
        $k = \{k_0, \quad k_1, \dots \quad k_m\}$

Stage – 2: Single Reduction
    Step 2.1: For each individual $k_i$ in k. Repeat through Step 2.4
    Step 2.2: Find the largest distance cities $c_i$ and $c_j$
    Step 2.3: Identify the cities $c_x and c_y$ in $k_i$ and check,
        $dist(c_i, c_j) + dist(c_x, c_y) > min(dist(c_i, c_x) + dist(c_j, c_y), dist(c_i, c_y) + dist(c_j, c_x))$
    Step 2.4: Disconnect the cities $c_i$ & $c_j$ and join with $c_x$ & $c_y$ as per Step 2.3.

Stage – 3: City List Construction
    Step 3.1: Initialize $cityList = \emptyset$
    Step 3.2: For each individual $k_i$ in k. Repeat through the Step 3.3
    Step 3.3: Update $cityList(c_i)$ for each city in $k_i$ using the adjacent cities and also the repetition number of the
    corresponding city on recurrence with respect to the following conditions.
        $rep(c_i, c_j) = \sum_{i=1}^{m} k_i(adj(c_i, c_j))$
    where,
        $k_i(adj(c_i, c_j)) = \begin{cases} 1, & if\ 'c_i'\ is\ adjacent\ city\ to\ 'c_j'\ in\ the\ individual\ 'k_i' \\ 0, & otherwise. \end{cases}$

Stage – 4: Child Initialization and Neighbor City Selection
    Step 4.1: Initialize the set l with m number of children individuals to empty.
        $l = \{l_0, \quad l_1, \dots \quad l_m\}$
    Step 4.2: Assign the initial city for $i^{th}$ children correspondingly as $i^{th}$ parent and refer it as current city $c_x$.
        $\forall i \in [1, m], \quad inicity(k_i) \rightarrow inicity(l_i)$
    Step 4.3: For each individual $l_i$ in l. Repeat through Step 4.5
    Step 4.4: The next city $next(c_x)$ for the current city $c_x$ of the child $l_i$ is chosen from the $cityList(c_x)$ based on
    satisfaction of condition 1 to 3.
    Condition 1: The city with highest repetition number $c_y$ is selected as the next city for the current city $c_x$.
        $rep(c_x, c_y) > \forall c_o \in [1, cityList(c_x) - 1], rep(c_x, c_o)$
    Condition 2: If $rep(c_x, c_y)$ is equal for more than one city then the city which belongs to the same parent of
    previously selected city $c_{x-1}$ is chosen as the next city $c_y$.
    If $c_{x-1}\ in\ l_i = c_{y-2}\ in\ k_i$ then $next(c_x) = c_y$
    Condition 3: If Condition 1 and 2 fails, then the next city $c_y$ which has the least number of cities in its $cityList$ is
    chosen from the cities with same repetition number $c_y\ and\ c_z$.
        $len(cityList(c_y)) < len(cityList(c_z))$
    Step 4.5: If all Conditions fail to select the next city $c_y$ for the current city $c_x$ then the situation is $fail(c_x)$ and
    Stage 5 is called upon to overcome the failure.

Stage – 5: Failure Evading
    Step 5.1: The following rules are followed in sequence for the current city $c_x$ of the partially built child $l_i$ to
    escape from the failure.
        Rule 1: If the other end of the partially built child $c_1$ has any unvisited city in its $cityList$, then continue to
        find $next(c_x)$ the from the city at the corresponding end.
            $l_i = \{c_1, c_2, c_3 \dots c_x\}$
            $l_i \leftarrow l_i = \{c_x, c_{x-1}, c_{x-2} \dots c_2, c_1\}$
        Rule 2: Check whether the first city $c_1$ is in the $cityList$ of the current city $c_x$. If so, connect the two cities $c_1$
        and $c_x$ to continue with fresh ends of $l_i$.
            $l_i = \{c_1, c_2, .. c_u, c_v, \dots c_{x-1}, c_x\}$
            $l_i \leftarrow l_i = \{c_u, c_{u-1}, .. c_2 c_1, c_x, c_{x-1} \dots c_{v+1}, c_v\}$
        Rule 3: Identify and save the cities of $cityList(c_x)$ present in $l_i$ as $list(c_x)$ and find the adjacent city with
        minimum $cityList$ length for each of city in the $list(c_x)$. The join the city the present in $cityList(c_x)$ with
        the city $c_x$ and continue to find the $next()$ for new end. Similarly attempt the same steps by formulating the
        $list()$ for the initial city $c_1$.
        Rule 4: Follow the step of Rule 3 whereas the $ODV(c_x)$ is used instead of $cityList(c_x)$ to construct $list(c_x)$
        and to find the adjacent city with minimum $cityList$ length from the $list(c_x)$. Similarly attempt the same
        steps by formulating the $list()$ for the initial city $c_1$.
        Rule 5: If all the above rules fail, then the $next(c_x)$ would be chosen randomly from the available unvisited
        cities.

Stage – 6: Final Reduction
    Step 6.1: For each individual $k_i$ in k. Repeat through Step 6.3
    Step 6.2: Find the distance of first and the last cities $dist(c_i, c_j)$ and identify the cities $c_x and c_y$ in $k_i$ and check,
        $dist(c_i, c_j) + dist(c_x, c_y) < min(dist(c_i, c_x) + dist(c_j, c_y), dist(c_i, c_y) + dist(c_j, c_x))$
    Step 6.4: Disconnect the cities $c_i$ & $c_j$ and join with $c_x$ & $c_y$ as per Step 6.2
```

Algorithm 1. Algorithm for ODVX Operator

where $k_i\left(\text{adj}\left(c_i, c_j\right)\right) =$

$$\begin{cases} 1, & \text{if } 'c_i' \text{ is adjacent city to } 'c_j' \text{ in the individual } 'k_i' \\ 0, & \text{otherwise} \end{cases}$$

In the existing works, the range of length of the *cityList* for any city can be given as

$2 \leq \text{len}(cityList) \leq 4$ [ERX]. But in the proposed ODVX technique, the range of the possible length of the *cityList* for a city can be $2 \leq \text{len}(cityList) \leq 2m$. This increased size of the *cityList* helps to achieve the aforesaid characteristics such as better crossover failure evading and search space exploration.

Stage-4: Child Initialization and Neighbor City Selection

This stage produces a set '*l*' with '*m*' number of offsprings $l = \{l_0, l_1, \ldots l_m\}$ using the *cityList* formulated at the previous stage of the algorithm. '*l*' will create a set of children '*l*' with '*m*' individuals. For each offspring l_i in '*l*', the initial city would be assigned from the parent individual k_i in the set '*k*'.

$\forall i \in [1, m],\ inicity(l_i) \leftarrow inicity(k_i)$

where

- $inicity(k_i)$ refers to the initial city of the *i*th parent individual in the set *k*.
- $inicity(l_i)$ refers to the initial city of the *i*th offspring in the set *l*.

For each offspring, the successive cities are identified, until $\text{len}(c_y) < l_i$, as follows: Let c_x be the current city of the offspring l_i, next city $next(c_x)$ has to be chosen from the $cityList(c_x)$ using the following conditions in sequence,

Condition 1 If each of the city in the $cityList(c_x)$ already present in the partially built offspring l_i, then the operation halts without the continuing city and the situation is referred as crossover failure at city c_x, the $\text{fail}(c_x)$.. The Stage-5 is called upon to overcome the failure using a set of offspring modification rules.

Condition 2 If any city c_y such that $c_y \in cityList(c_x)$ and has the highest repetition number corresponding to the c_x,, it would be chosen as the next city provided is not already present in the partially built offspring l_i.

$$\begin{aligned} &\text{rep}\left(c_x, c_y\right) > \forall o \in [1, cityList(c_x) - 1], \text{rep}(c_x, c_o) \\ &\text{and rep}\left(c_x, c_y\right) > 2 \\ &\therefore \text{next}(c_x) = c_y \end{aligned}$$

where

- $cityList(c_x) - 1$ refers to the *cityList* of corresponding city c_x excluding the city c_y.

This condition identifies the city which is adjacent to city c_x in most of the parents and consequently helps to inherit the combined better information from the set of parent individuals.

Condition 3 If two or more cities have the same repetition number corresponding to the c_x, the city which belongs to the same parent of the previous city of c_x in the partially built offspring l_i would be chosen as the next city. This condition can be represented as,

$$\begin{aligned} &c_y, c_z \in cityList(c_x) \text{ and rep}\left(c_x, c_y\right) = \text{rep}(c_x, c_z) \geq 2 \\ &\text{If } c_{x-1} \text{ in } l_i = c_{y-2} \text{ in } k_i \text{ then next}(c_x) = c_y \\ &\text{Similarly, If } c_{x-1} \text{ in } l_i = c_{z-2} \text{ in } k_i \text{ then next}(c_x) = c_z \end{aligned}$$

where

- c_{x-1} is the city at the previous position to the current city c_x in l_i.

This condition attempts to inherit the long better information from the same parent individual.

Condition 4 If the previous city c_{x-1} does not belong to position c_{y-2} or c_{z-2} in any of the parents corresponding to the cities c_y and c_z, the next city which has the least $\text{len}(cityList)$ would be chosen from the cities with the same repetition number.

$$\begin{aligned} &\text{If } c_{x-1} \text{ in } l_i \neq c_{y-2} \text{ in } k_i \text{ and } c_{x-1} \text{ in } l_i \neq c_{z-2} \text{ in } k_i \text{ then,} \\ &\text{Check len}\left(cityList(c_y)\right) < \text{len}(cityList(c_z)) \\ &\therefore \text{next}(c_x) = c_y \end{aligned}$$

If $\text{len}\left(cityList(c_y)\right) = \text{len}(cityList(c_z))$, then $\text{next}(c_x)$ is chosen among the cities c_y and c_z randomly. This condition tries to choose the city with minimum continuing cities and thus helps to improve the crossover evading feature of the operator.

If the conditions 2, 3 and 4 fail to select the $\text{next}(c_x)$ for the current city c_x, then this situation is also referred as the crossover failure, $\text{fail}(c_x)$ and Stage-5 is called upon to overcome the failure.

Stage-5: Failure Evading

The Stage-5 is invoked on the occurrence of crossover failure at the Stage-4. This stage applies a set of rule-based alteration schemes on the partially built offspring l_i to effectively overcome the crossover failure condition. If crossover failure $\text{fail}(c_x)$ occurred at the current city c_x, then following rules are employed, in sequence, to overcome the crossover failure,

Rule 1 Check whether the other end of the partially built offspring l_i has any unvisited city in its *cityList*. If so, continue from the city at the corresponding end. Let, the partially built individual is

$l_i = \{c_1, c_2, c_3 \ldots c_x\}$, where c_x is the current city with crossover failure.

If $len(cityList(c_1)) \geq 1$, then $l_i \leftarrow l_i = \{c_x, c_{x-1}, c_{x-2} \ldots c_2, c_1\}$

Now, the current city is c_1 and continue with Stage-4 of operations to find the $\text{next}(c_1)$.

Rule 2 If the Rule 1 fails, check whether the current city c_x has the first city c_1 in its *cityList*. If so, identify any city in the partially built child l_i has $\text{len}(cityList()) > 0$. If exist, join the ends of the partially built child and continue with two new ends. This can be exemplified as,

Let l_i be the partially built child,

$l_i = \{c_1, c_2, \ldots c_u, c_v, \ldots c_{x-1}, c_x\}$
$c_x \in cityList(c_1)$

$\text{fail}(c_1)$ and $\text{fail}(c_x)$ is TRUE,If $\text{len}(cityList(c_v)) > 0$Then modify l_i as, $l_i \leftarrow l_i = \{c_u, c_{u-1}, \ldots c_{2,} c_1, c_x, c_{x-1} \ldots c_{v+1}, c_v\}$

Now, the current city is c_v and continue with Stage-4 of operations to find the $\text{next}(c_v)$.

Rule 3 If both Rules 1 and 2 fail, find a list of cities $\text{list}(c_x)$ using the cities present in the partially built child l_i and also in the *cityList* of the current city c_x. Then, find a city in l_i which is right adjacent to the city in $\text{list}(c_x)$ and also has minimum unvisited cities in its *cityList*. If so, rotate the partially built child in such a way to connect the corresponding city in the $\text{list}(c_x)$ with the city c_x and continue with the corresponding adjacent city.

Let l_i be the partially built child,

$l_i = \{c_1, c_2, \ldots c_{u-1}, c_u, c_v, c_{v+1} \ldots c_{x-1}, c_x\}$
$\text{list}(c_x) \leftarrow \forall c_o, c_o \text{ in } l_i \in cityList(c_x)$

$\text{fail}(c_1)$, $\text{fail}(c_x)$ and $c_x \notin cityList(c_1)$ is TRUEIf, $c_u \leftarrow \exists c_o \in \text{list}(c_x), \text{adj}(c_o, c_{o+1})$ in l_i and $\min(\text{len}(cityList(c_{o+1})))$
Modify l_i as,

$l_i \leftarrow l_i = \{c_1, c_2, \ldots c_{u-1,} c_u, c_x, c_{x-1} \ldots c_{v+1}, c_v\}$

Now, the current city is c_v and continue with Stage-4 of operations to find the $\text{next}(c_v)$. Similarly, attempt the same steps for another end city c_1 by formulating the $\text{list}(c_1)$; however, it has to find a city in l_i which is left adjacent to the city in $\text{list}(c_1)$.

Rule 4 If the rules 1, 2 and 3 fail, formulate the $\text{list}(c_x)$ as in Step 3, but using the $\text{ODV}(c_x)$ instead of $cityList(c_x)$. And rotate the partially built individual l_i as in (iii) and continue with the new end of l_i.

Let l_i be the partially built child,

$l_i = \{c_1, c_2, \ldots c_{u-1}, c_u, c_v, c_{v+1} \ldots c_{x-1}, c_x\}$
$\text{list}(c_x) \leftarrow \exists c_o, c_o \in \text{ODV}(c_x)$

$\text{fail}(c_1)$, $\text{fail}(c_x)$ and $c_x \notin cityList(c_1)$ is TRUEIf $c_u \leftarrow \exists c_o \in \text{list}(c_x), \text{adj}(c_o, c_{o+1})$ in l_i and $\min(\text{len}(cityList(c_{o+1})))$
Modify l_i as,

$l_i \leftarrow l_i = \{c_1, c_2, \ldots c_{u-1,} c_u, c_x, c_{x-1} \ldots c_{v+1}, c_v\}$

Now, the current city is c_v and continue with Stage-4 of operations to find the $\text{next}(c_v)$. Similarly, attempt the same steps for another end city c_1 by formulating the $\text{list}(c_1)$; however, it has to find a city in l_i which is left adjacent to the city in $\text{list}(c_1)$.

Rule 5 If all the above Rules fail, possibly the last few cities to be included, then the $\text{next}(c_x)$ would be chosen randomly from the available unvisited cities.

After the successful crossover failure evasion, the offspring construction operation is continued with the Stage-4 of the algorithm.

Stage-6: Final Reduction

This stage works similar to the Stage-2, but it improves the quality of the offsprings generated after the successful recombination process. The main objective of this stage is to optimize the distance between the first and last cities of the offspring which are usually anonymous to each other. Let c_i and c_j are the first and the last cities in the completely built offspring l_i, find the two consecutive cities c_x and c_y in l_i such that,

$$\text{dist}(c_i, c_j) + \text{dist}(c_x, c_y) > \min\big(\text{dist}(c_i, c_x) + \text{dist}(c_j, c_y), \text{dist}(c_i, c_y) + \text{dist}(c_j, c_x)\big)$$

If the condition holds, then modify the offspring to connect c_i and c_j with the corresponding c_x and c_y cities. It should be noted that the rank of cities to which c_i and c_j are joined should be less than or equal to the 'ba' value used at the population seeding stage.

If no cities satisfy such condition, then the condition has to be re-checked for the city c_j and c_{j-1} in l_i and so on. Thus, the Stage-6 of ODVX attempts to improve the quality of the individuals with simple operations which helps to enhance the overall efficacy of the algorithm.

At the end of the algorithm, 'm' number of offsprings have been generated using the parent individuals chosen from the current population and the same procedure is repeated until the generation of 'n' offsprings which constitute a generation.

4 Experimentation and result analyses

4.1 Experimental setup

4.1.1 Test bed design

Experiments are carried out in a different combination of crossover operators and population seeding techniques under the similar environmental condition to validate the significances of the proposed crossover operator. The experimental setup framework is shown in Fig. 2 in which the initial populations generated with different population initialization techniques are made to recombine with recent and best working specific crossover operators until the termination condition is satisfied. The resultant populations of individuals, which are generated under similar test settings, are evaluated with the defined performance factors. The various GA parameters and their corresponding values used in the evaluation are given in Table 1. For experiments, the authors used Windows PC with Intel i5 processor with 8 GB RAM, MATLAB R2017a for implementation and SPSS for mean value-based analyses.

One of the famous combinatorial hard problems traveling salesman problem (TSP) obtained from standard TSPLIB is being chosen as the testbed [TSPLIB]. Classification of TSP instances based on the size is given in Table 2. For experiments, the performance of the ODVX crossover has been compared with the state-of-art crossover operators in combination with different population seeding techniques of the permutation-coded GA. The best performing population seeding techniques such as Random, Nearest Neighbor (NN), Selective Initialization (SI) and ODV-based techniques are chosen at the population initialization stage. The crossover techniques such as the Edge Assembly Crossover (EAX), Partition Crossover (PX), Modified Order Crossover (MOX), Sub-tour Recombination Crossover (SRX) and ODV-based crossover (ODVX) operators are selected to perform recombination operation at each generation of the GA. Therefore, with six different population seeding techniques and five different crossover operators, the final populations of thirty different GA models can be obtained which are analyzed with various assessment criteria defined in Sect. 4.1.2. This combinational style of analyses helps to validate the performance of the proposed ODVX crossover operator with different existing population seeding techniques and also the ODV-based population seeding technique proposed in (Paul et al. 2014).

4.1.2 Assessment criteria

There are three critical performance factors used to observe the importance of the proposed ODVX operator, and they are summarized as follows:

Computation Time The computational time is the total time taken to complete the 250 generations of GA with the corresponding crossover operator. This factor is used to measure the computational complexity of the complete GA

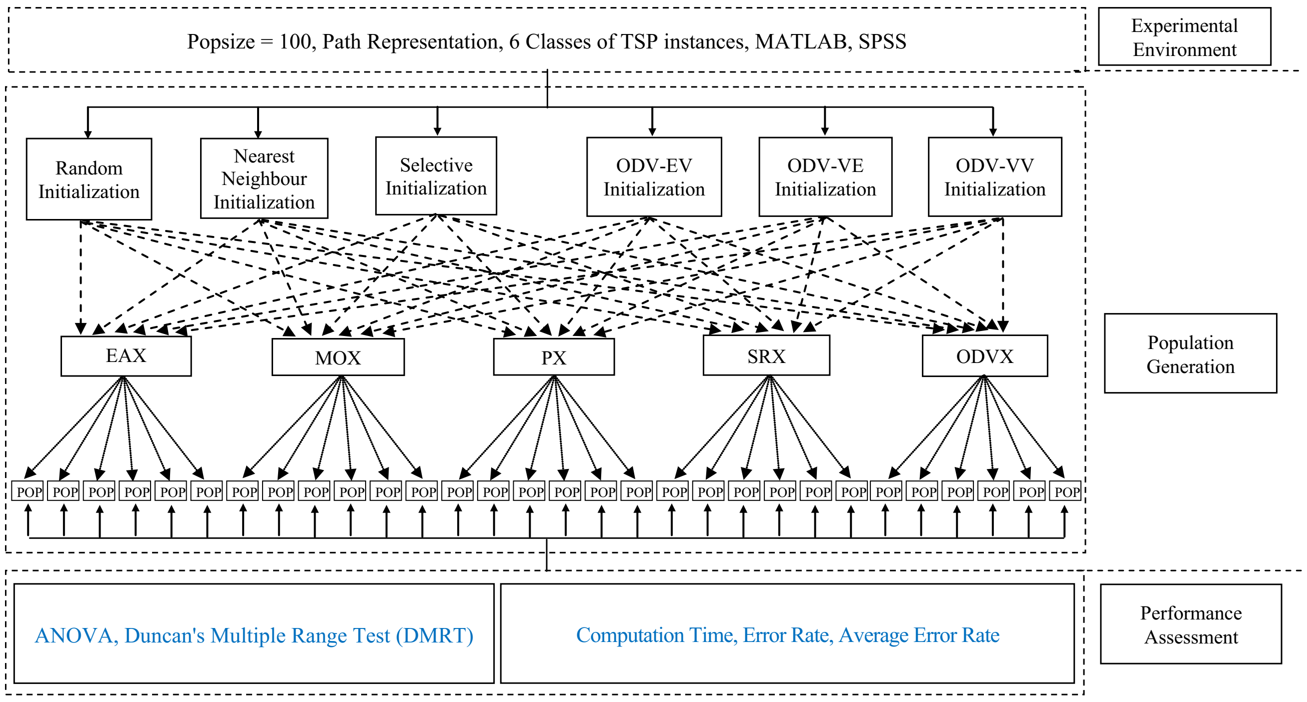

Popsize = 100, Path Representation, 6 Classes of TSP instances, MATLAB, SPSS
Experimental Environment
Random Initialization
Nearest Neighbour Initialization
Selective Initialization
ODV-EV Initialization
ODV-VE Initialization
ODV-VV Initialization
EAX
MOX
PX
SRX
ODVX
POP
Population Generation
ANOVA, Duncan's Multiple Range Test (DMRT)
Computation Time, Error Rate, Average Error Rate
Performance Assessment

Fig. 2 Experimental framework

Table 1 Genetic algorithm parameters

S.No	Parameter	Value/Technique
1	Pop Size	100 Individuals
2	Limit	250 Generations
3	Initialization models	Random, Selective Initialization, Nearest Neighbor and ODV
4	Crossover Technique	EAX, PX, SRX and ODVX
5	Crossover Probability	0.6 [3,35]
6	Mutation Technique	Swap [35]
7	Mutation Probability	0.02 [3,32]
8	Elite	5 individuals
9	Termination Condition	Generation Limit

Table 2 Classification of TSP instances based on the size

Sl. No	Instance class	Size (No. of Cites)	Name of the instances
1	Class I	Size $\leq$ 100	eil51, rat99, kroA100
2	Class II	100 < Size $\leq$ 500	tsp225, lin318, d493
3	Class III	500 < Size $\leq$ 1000	d657, u724, rat783
4	Class IV	1000 < Size $\leq$ 5000	fl1577, d2103, fnl4461
5	Class V	5000 < Size $\leq$ 15,000	rl5915, rl11849, brd14051
6	Class VI	Size $\geq$ 15,000	d15112, d18512

and is directly proportional to the size of the problem instance. The measuring unit of computation time is seconds(s).

Error Rate (%) Error Rate of a solution can be defined as the percentage of difference in the fitness value of the solution with the known optimal solution for the problem.

$$\text{Error Rate}(\%) = \frac{\text{Fitness}-\text{Optimal Fitness}}{\text{Optimal Fitness}} \times 100$$

Average Error Rate (%) It is the average of the error rate of the solutions in the population after the completion of the GA with a predefined number of generations and it is defined as follows:

$$\text{Average Error Rate}(\%) = \frac{\text{Average Fitness}-\text{Optimal Fitness}}{\text{Optimal Fitness}} \times 100$$

where

- Average Fitness is the average fitness value of solutions in the population
- Optimal Fitness is the known optimal value of the corresponding instance

This factor is used to measure the quality of the final population generated by finding the average fitness of individuals in the population.

4.2 Result analyses

In this section, the effectiveness of the various models of GA with Random, NN, SI, EV, VE and VV population seeding techniques in combination with EAX, PX, MOX, SRX and ODVX crossover operators for the defined number of generations is discussed based on the performance criteria discussed in Sect. 4.1.2. "Appendix G–Appendix L" show the performance of the different GA models for the Classes I, II, III, IV, V and VI of TSP test instances, respectively, under similar experimental setup. For each technique, the executions are carried out for 25 times and the average of each case has been considered for experimental analyses. "Appendix A–Appendix F" show the mean value-based assessment of different factors, and the best mean value obtained for each class of test instances is shown in '*bold*'.

4.2.1 Error rate

The error rate-based analyses on different models of GA show that the GA models with VV & ODVX, EV & ODVX, NN & ODVX, VV & EAX and NN & PX outperform other GA models comfortably. The mean values of the error rate of different GA models for Classes I, II & III and Classes IV, V & VI of instances are depicted in "Appendix A and Appendix B", respectively.

The performance of GA models with MOX and SRX crossover operators performs worst for all the class of test instances, regardless of the kind of population seeding techniques used. For most of the classes of test instances, the MOX and SRX operators yield three times the mean error rate value obtained by other crossover operators. This justifies that these recombination operators are not challengeable enough for the other crossover operators considered for evaluation. The EAX and PX operators offer better mean error rate value with the population seeding techniques such as NN, EV and VV than the Random, SI and VE techniques. It is also observed that the mean error rate value of EAX and PX operators degrades with an increase in the size of the problem instances. The PX operator outperforms the other recombination operators, for Classes IV & V of test instances, with NN as the population seeding technique.

The GA model with the proposed ODVX operator, regardless of the population seeding technique used, yields better mean error rate value than the other recombination operators. From Tables 3 and 4, it can be observed that the best performing GA models have the ODVX as their recombination operator and it can also be noted that the performance degradation in the mean error rate, due to the increase in the size of the problem instance, is relatively minimal in the ODVX operator. The ODVX operator offers to mean error rate lesser than 0.5%, for every class of test instance, with the NN, EV and VV population seeding techniques, whereas with SI and VE techniques the same has been increased to the maximum of 1.5%. The Error Rate (%) obtained for largest test instances of each class with respect to the different GA models is shown in Fig. 3. From the graph, it can be noted that the GA models VV & ODVX, NN & PX, EV & ODVX and NN & ODVX give better error rate (%) than the other GA models regardless of the class of the instance.

4.2.2 Average error rate

The average error rate (%)-based analyses reveal the ability of the different GA models to move toward the optimal point in the search space as a whole population rather than the single solution. "Appendix C and Appendix D" show the mean values of the average error rate of the different GA models for different classes of test instances. It can be observed that the different GA models with MOX and SRX as recombination operators give maximum mean values of average error rate and consequently these crossover operators could not exploit the potential of the initial populations generated by the VV, EV and NN population seeding techniques.

The GA models with NN & EAX obtained the best mean value of the average error rate % for the Class I of instances, whereas the GA model with ODV-based population seeding and the ODVX operator offers best mean value of the same for all the other Class of test instances. This justifies that the ODVX operator explores and exploits the potential sequence generated by the ODV-based population seeding techniques. It can also be observed that the ODVX operator yields a better average error rate in combination with NN, SI and Random seeding techniques than the other crossover operators. This shows the ability of the ODVX operator to extract better quality individuals from the initial populations that are generated with seeding techniques with a different principle of individual generation.

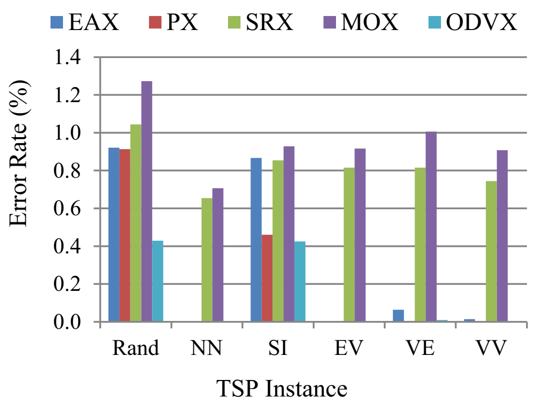
For the Class VI of test instances, the GA models VV & ODVX and EV & ODVX yield the mean value of the average error rate of 1.465% and 1.705%, respectively. This shows that the ODV-based GA models enable the whole population to converge toward the optimal point

Table 3 Performance Order of crossover operators for different performance criteria

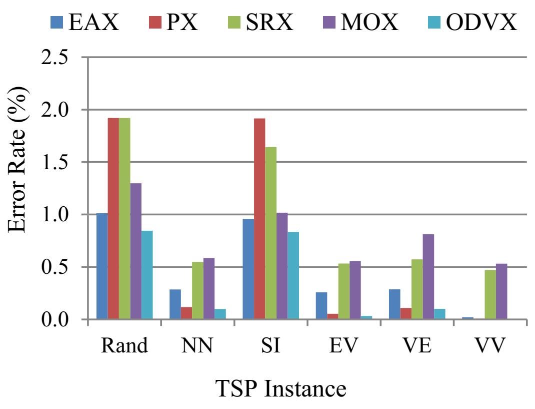
S.No	Performance criteria	Performance order (*best* → *worst*)
1	Error rate	ODVX → EAX → PX → SRX → MOX
2	Average error rate	ODVX → EAX → PX → SRX → MOX
3	Computation time	MOX → ODVX → SRX → PX → EAX
Overall	ODVX → EAX → PX → SRX → MOX	

Table 4 Best performing GA model for different class of test instances with respect to various performance criteria

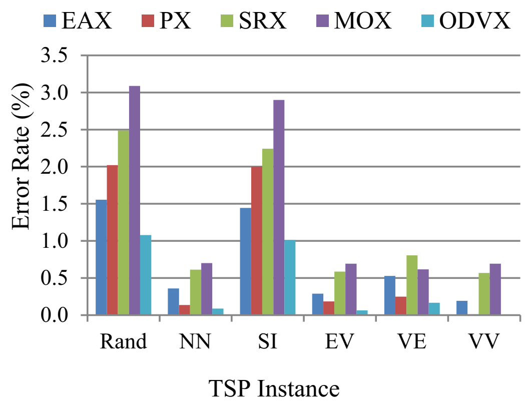
S. No	Performance criteria	Error rate	Average error rate	Computation time
1	Class I	NN & ODVX	NN & EAX	NN & SRX
2	Class II	VV & ODVX	VV & ODVX	SI & SRX
3	Class III	VV & ODVX	EV & ODVX	EV & MOX
4	Class IV	VV & ODVX	VV & ODVX	EV & MOX/ODVX
5	Class V	VV & ODVX	VV & ODVX	EV & ODVX
6	Class VI	VV & ODVX	VV & ODVX	EV & ODVX
Overall		VV & ODVX	VV & ODVX	EV & ODVX



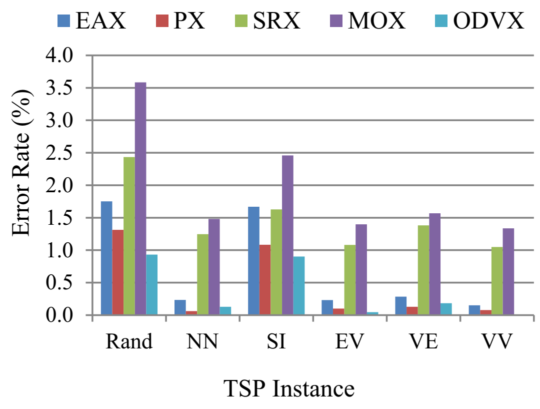
a Class I



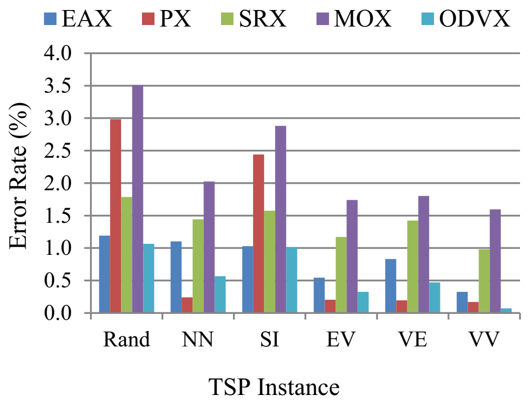
b Class II



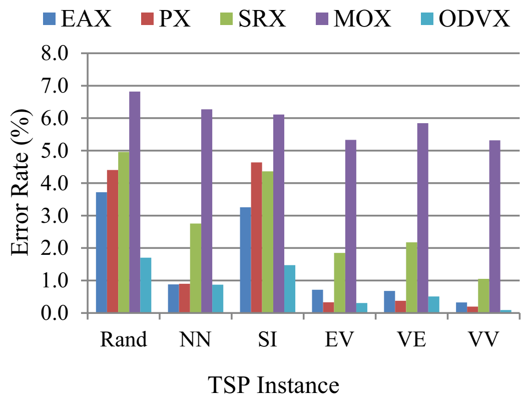
c Class III



d Class IV



e Class V



f Class VI

Fig. 3 Performance of different GA models w.r.t Error rate (%)

effectively for the problems with the large and complex search space. Figure 4 portrays the average error rate (%) obtained for largest test instances of each class with respect to the different GA models. From the graph, it can be noted that the GA models with ODVX as the recombination operator offer better average error rate (%) than the other GA models regardless of the class of the instance.

4.2.3 Computation time

The computation time-based analyses show that the crossover techniques such as EAX, NN and ODVX take more time to complete the defined GA termination condition. This is because of the complex decision system used to identify the potential information from the parent to

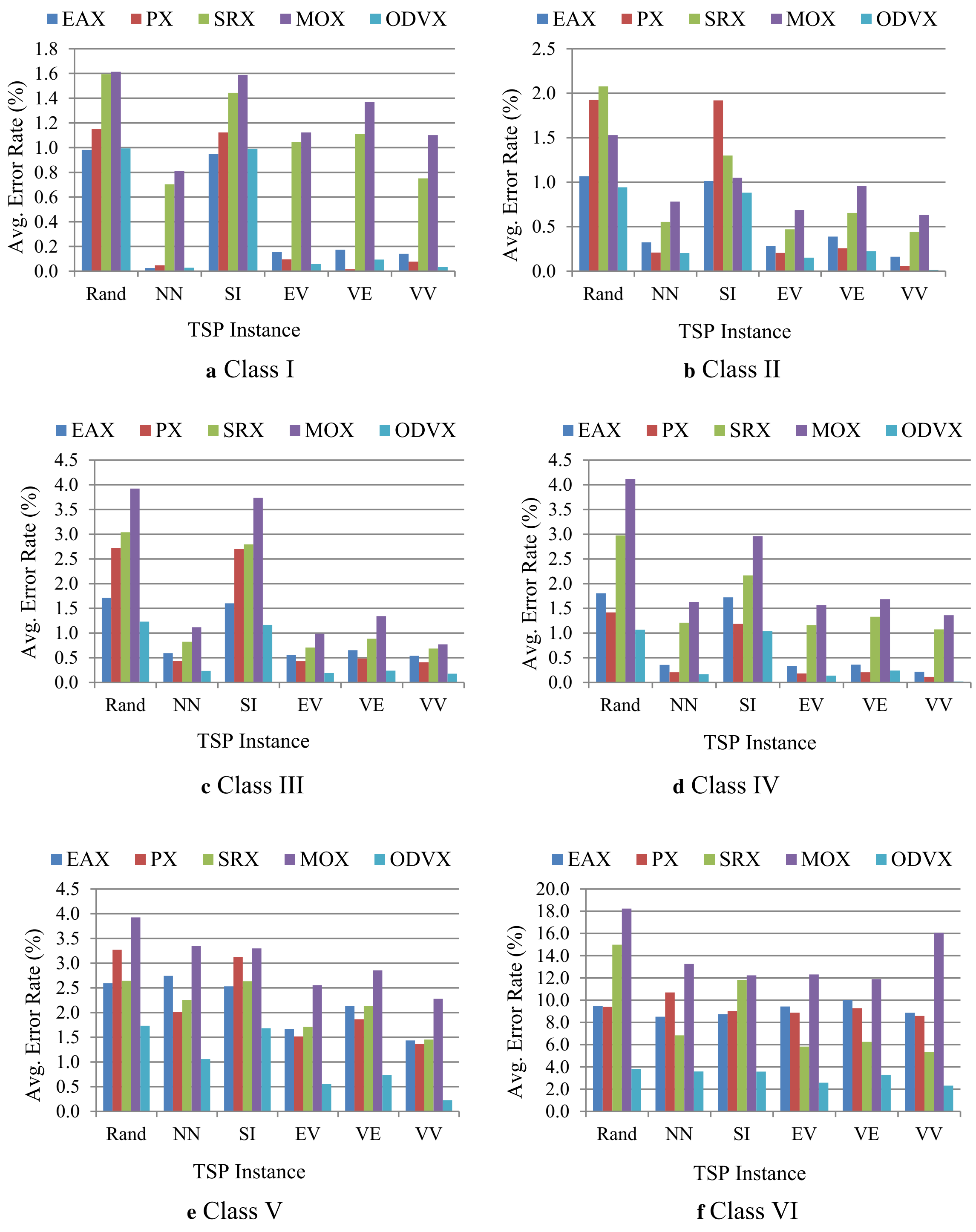


Fig. 4 Performance of different GA models w.r.t Average error rate (%)

generate the offspring individuals. On the other hand, the MOX and SRX techniques which have relatively lesser complexity in the crossover operation and consequently faster than the other crossover operators are considered for evaluation. "Appendix E and Appendix F" depict the mean values of computation time of different GA models for Classes I, II & III and Classes IV, V & VI of instances, respectively.

For Classes I and II of test instances, the SRX operator takes minimum time for the initial population generated with the NN and SI techniques, respectively. At the same time, the MOX operator offers better computation time Class III of TSP instances with EV population seeding

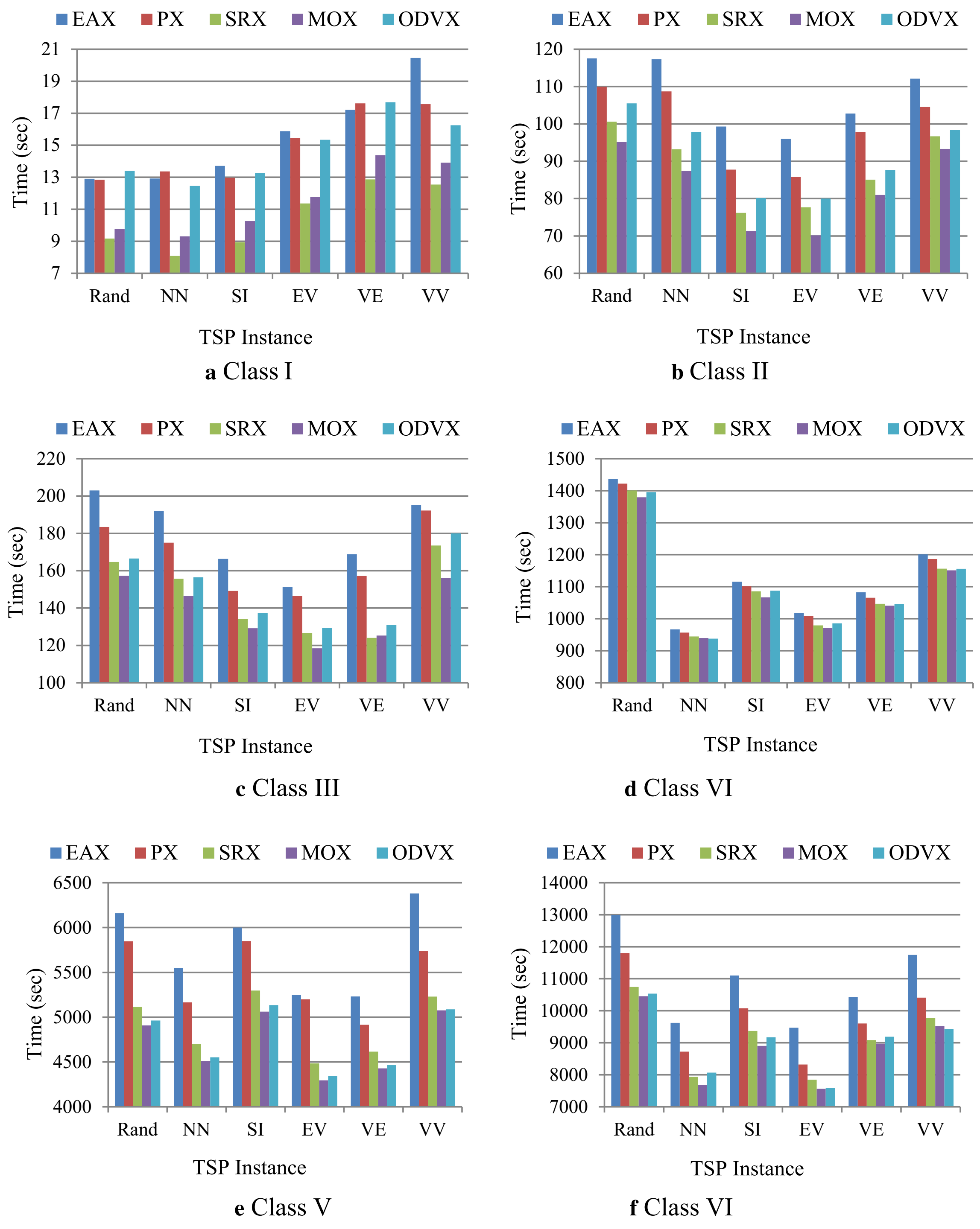


Fig. 5 Performance of different GA models w.r.t Computation time

technique. The interesting fact is that the MOX and ODVX techniques complete the GA at the same time using the EV seeding technique for the Class IV of test instances. Moreover, the ODVX & GA model outperforms the MOX & EV model of GA with respect to the computation time using the EV seeding technique for Classes V and VI of instances. These points support the following assessments:

- With respect to the computation time, the SRX operator performs better for small-sized TSP instances, MOX and ODVX recombination operators outperform other techniques for medium and large-sized test instances, respectively.
- The NN and SI seeding techniques offer minimum computation time for small-sized test problems,

whereas the EV technique shows outstanding performance with the computation time for the medium and large-sized TSP test instances.

The computation time measured for largest test instances of each class with respect to the different GA models is shown in Fig. 5. From the graph, it can be understood that the performance of the GA model with EV & ODVX improves with an increase in the size of the test instances.

4.3 Discussion

The performance of the proposed ordered distance vector-based crossover (ODVX) operator has been evaluated with the different recent and best working crossover operators based on various performance criteria defined. The performance order of the crossover operators of GA for different performance criteria based on the overall mean value using DMRT on different crossover techniques is shown in Table 3. The overall performance order of different crossover operators has been identified after neglecting the computation time factor which has no significant difference between the mean values of the different techniques.

From Table 3, it can be observed that the proposed ODVX operator outperforms the other well-known crossover operators studied for the Permutation-coded GA in solving TSP. Though PX yields a better quality solution for some of the test problems, EAX operator outperforms the PX in the mean values using DMRT analyses. Thus, the proposed ODVX operator has the following versatile capabilities: the multi-parent characteristic helps to effectively transfer the good information from the parents to the offsprings, the two-way potential extraction enables the individual to extract the goodness of the parents to the maximum possible extent, ability to overcome the crossover failure avoids the inclusion or addition of unnecessary random information into the offspring which may spoil the potential of the individual, and the selective new information inclusion facilitates the operator from getting stuck in the local optimal point that leads to premature convergence. This clearly evident that the proposed ODVX is better than the existing crossover operators by effectively utilizing the potential sequence generated at the initialization stage of the GA.

Table 4 shows the best performing GA model for different classes of test instances with respect to various performance criteria. From the table, it can be justified that the GA model with the proposed ODV-based population initialization techniques and the ODV-based crossover operators works outstanding for different classes of test instances. Especially, in case of medium and large-sized TSP test instances, the performance of ODV-based VV/EV technique with ODVX is exceptional. Thus, the significances of the proposed crossover operator have been validated with the suitable test environment and the performance factors.

5 Conclusion

To conclude, the research work is projected to design an effective GA model with ODV-based crossover technique and thereby to improve the overall performance of the GA with ODV initialization to solve the large-sized combinatorial problem. Precise experimental setup has been designed, and experiments are performed on different sized benchmark TSP instances obtained from standard TSPLIB in order to validate the proposed crossover technique. The assessment is intended to evaluate the performance of the proposed ODVX operator with respect to the different state-of-the-art crossover operators. A variety of GA models have been generated for examination with various population seeding techniques in combination with crossover operators, and the result values are noted after the 250 generations. From the experimental results, the GA model with ODV initialization and ODVX crossover operator outshines the other existing and best working GA model in the literature. This work can be extended to validate the performance of the proposed GA model with ODV-based seeding and crossover operator with different generation limits and with respect to the state-of-the-art metaheuristic approaches for TSP.

Compliance with ethical standards

Conflict of interest Author P. Victer Paul declares that he has no conflict of interest. Author P. Dhavachelvan declares that he has no conflict of interest. Author C. Ganeshkumar declares that he has no conflict of interest. Author R. Baskaran declares that he has no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

Informed consent Informed consent was obtained from all individual participants included in the study.

Appendix A: The mean values of the error rate of different GA models for Classes I, II and III of instances

Dependent variable: error rate			Mean
Class	Population seeding	Crossover	
Class I	Random	EAX	0.541
		PX	0.610
		SRX	0.687
		MOX	0.813
		ODVX	0.283
	NN	EAX	7.6E−16
		PX	1.3E−15
		SRX	0.280
		MOX	0.430
		ODVX	**7.7E−16**
	SI	EAX	0.480
		PX	0.243
		SRX	0.523
		MOX	0.600
		ODVX	0.283
	EV	EAX	4.96E−16
		PX	5.24E−16
		SRX	0.530
		MOX	0.473
		ODVX	1.93E−15
	VE	EAX	0.042
		PX	7.772E−16
		SRX	0.400
		MOX	0.680
		ODVX	0.003
	VV	EAX	0.005
		PX	1.277E−15
		SRX	0.460
		MOX	0.560
		ODVX	1.199E−14

Dependent variable: error rate			Mean
Class	Population seeding	Crossover	
Class II	Random	EAX	0.736
		PX	1.830
		SRX	1.740
		MOX	1.363
		ODVX	0.693
	NN	EAX	0.250
		PX	0.060
		SRX	0.477
		MOX	0.573
		ODVX	0.050
	SI	EAX	0.628
		PX	1.660
		SRX	1.270
		MOX	0.900
		ODVX	0.560
	EV	EAX	0.129
		PX	0.017
		SRX	0.447
		MOX	0.537
		ODVX	0.010
	VE	EAX	0.248
		PX	0.163
		SRX	0.490
		MOX	0.663
		ODVX	7.073
	VV	EAX	0.010
		PX	0.003
		SRX	0.320
		MOX	0.487
		ODVX	**2E−14**

Dependent variable: error rate			Mean
Class	Population seeding	Crossover	
Class III	Random	EAX	1.358
		PX	1.663
		SRX	2.107
		MOX	2.770
		ODVX	1.047
	NN	EAX	0.343
		PX	0.190
		SRX	0.530
		MOX	0.647
		ODVX	0.093
	SI	EAX	1.337
		PX	1.970
		SRX	2.110
		MOX	2.160
		ODVX	1.113
	EV	EAX	0.269
		PX	0.237
		SRX	0.523
		MOX	0.577
		ODVX	0.057

Dependent variable: error rate			Mean
Class	Population seeding	Crossover	
	VE	EAX	0.459
		PX	0.257
		SRX	0.757
		MOX	0.600
		ODVX	0.173
	VV	EAX	0.104
		PX	0.030
		SRX	0.420
		MOX	0.527
		ODVX	**1.94E−14**

Appendix B: The mean values of the error rate of different GA models for Classes IV, V and VI of instances

Dependent variable: error rate			Mean
Class	Population seeding	Crossover	
Class IV	Random	EAX	1.520
		PX	1.970
		SRX	1.820
		MOX	2.380
		ODVX	0.853
	NN	EAX	0.202
		PX	0.060
		SRX	1.073
		MOX	1.213
		ODVX	0.093
	SI	EAX	1.196
		PX	1.803
		SRX	1.387
		MOX	1.800
		ODVX	0.747
	EV	EAX	0.149
		PX	0.087
		SRX	0.980
		MOX	1.170
		ODVX	0.030
	VE	EAX	0.231
		PX	0.533
		SRX	1.067
		MOX	1.247
		ODVX	0.140
	VV	EAX	0.109
		PX	0.033
		SRX	0.887
		MOX	1.100
		ODVX	**0.007**

Dependent variable: error rate			Mean
Class	Population seeding	Crossover	
Class V	Random	EAX	1.435
		PX	2.030
		SRX	1.550
		MOX	2.863
		ODVX	0.927
	NN	EAX	0.585
		PX	0.147
		SRX	1.263
		MOX	1.967
		ODVX	0.307
	SI	EAX	1.174
		PX	1.843
		SRX	1.297
		MOX	2.480
		ODVX	0.827
	EV	EAX	0.328
		PX	0.133
		SRX	1.003
		MOX	1.807
		ODVX	0.160
	VE	EAX	0.454
		PX	0.160
		SRX	1.380
		MOX	1.893
		ODVX	0.293
	VV	EAX	0.207
		PX	0.110
		SRX	0.857
		MOX	1.727
		ODVX	**0.030**

Dependent variable: error rate			Mean
Class	Population seeding	Crossover	
Class VI	Random	EAX	3.357
		PX	4.515
		SRX	4.300
		MOX	6.615
		ODVX	1.755

Dependent variable: error rate			Mean
Class	Population seeding	Crossover	
	NN	EAX	1.400
		PX	0.995
		SRX	2.500
		MOX	5.435
		ODVX	0.930
	SI	EAX	2.724
		PX	4.460
		SRX	3.325
		MOX	6.115
		ODVX	1.720
	EV	EAX	1.023
		PX	0.650
		SRX	1.915
		MOX	4.320
		ODVX	0.375
	VE	EAX	1.132
		PX	0.770
		SRX	2.200
		MOX	4.610
		ODVX	0.610
	VV	EAX	0.692
		PX	0.465
		SRX	1.365
		MOX	4.230
		ODVX	**0.130**

Appendix C: The mean values of the average error rate of different GA models for Classes I, II and III of instances

Dependent variable: average error rate			Mean
Class	Population seeding	Crossover	
Class I	Random	EAX	0.642
		PX	0.777
		SRX	1.067
		MOX	1.030
		ODVX	0.643
	NN	**EAX**	**0.018**
		PX	0.037
		SRX	0.463
		MOX	0.473
		ODVX	0.030
	SI	EAX	0.587
		PX	0.710
		SRX	0.913
		MOX	1.023
		ODVX	0.597
	EV	EAX	0.056
		PX	0.043
		SRX	0.667
		MOX	0.747
		ODVX	0.030
	VE	EAX	0.101
		PX	0.023
		SRX	0.743
		MOX	0.853
		ODVX	0.053
	VV	EAX	0.054
		PX	0.040
		SRX	0.540
		MOX	0.690
		ODVX	0.040

Dependent variable: average error rate			Mean
Class	Population seeding	Crossover	
Class II	Random	EAX	0.882
		PX	1.967
		SRX	2.193
		MOX	1.850
		ODVX	0.827
	NN	EAX	0.296
		PX	0.180
		SRX	0.680
		MOX	0.750
		ODVX	0.160
	SI	EAX	0.774
		PX	1.800
		SRX	1.480
		MOX	1.190
		ODVX	0.673
	EV	EAX	0.268
		PX	0.160
		SRX	0.770
		MOX	0.667
		ODVX	0.137
	VE	EAX	0.339
		PX	0.203
		SRX	0.967

Dependent variable: average error rate			Mean
Class	Population seeding	Crossover	
		MOX	0.823
		ODVX	0.187
	VV	EAX	0.188
		PX	0.200
		SRX	0.697
		MOX	0.520
		ODVX	**0.083**

Dependent variable: average error rate			Mean
Class	Population seeding	Crossover	
Class III	Random	EAX	1.418
		PX	2.497
		SRX	2.703
		MOX	3.407
		ODVX	1.230
	NN	EAX	0.532
		PX	0.383
		SRX	0.727
		MOX	1.160
		ODVX	0.347
	SI	EAX	1.281
		PX	2.550
		SRX	2.520
		MOX	3.113
		ODVX	1.393
	EV	EAX	0.526
		PX	0.407
		SRX	0.670
		MOX	0.817
		ODVX	**0.157**
	VE	EAX	0.571
		PX	0.447
		SRX	0.823
		MOX	1.230
		ODVX	0.250
	VV	EAX	0.456
		PX	0.290
		SRX	0.593
		MOX	0.677
		ODVX	0.183

Appendix D: The mean values of the average error rate of different GA models for Classes IV, V and VI of instances

Dependent variable: average error rate			Mean
Class	Population seeding	Crossover	
Class IV	Random	EAX	1.644
		PX	2.040
		SRX	2.053
		MOX	2.680
		ODVX	0.983
	NN	EAX	0.296
		PX	0.160
		SRX	1.157
		MOX	1.433
		ODVX	0.203
	SI	EAX	1.319
		PX	1.873
		SRX	1.620
		MOX	2.090
		ODVX	0.843
	EV	EAX	0.266
		PX	0.143
		SRX	1.063
		MOX	1.357
		ODVX	0.107
	VE	EAX	0.334
		PX	0.427
		SRX	1.260
		MOX	1.500
		ODVX	0.247
	VV	EAX	0.226
		PX	0.103
		SRX	0.963
		MOX	1.273
		ODVX	**0.060**

Dependent variable: average error rate			Mean
Class	Population seeding	Crossover	
Class V	Random	EAX	2.007
		PX	2.480
		SRX	2.020
		MOX	3.067
		ODVX	1.227
	NN	EAX	1.416
		PX	1.003
		SRX	1.800

Dependent variable: average error rate			Mean
Class	Population seeding	Crossover	
		MOX	2.803
		ODVX	0.643
	SI	EAX	1.779
		PX	2.360
		SRX	1.833
		MOX	2.720
		ODVX	1.233
	EV	EAX	0.923
		PX	0.703
		SRX	1.443
		MOX	2.550
		ODVX	0.263
	VE	EAX	1.132
		PX	0.830
		SRX	1.730
		MOX	2.807
		ODVX	0.450
	VV	EAX	0.804
		PX	0.610
		SRX	1.287
		MOX	2.343
		ODVX	**0.097**

Dependent variable: average error rate			Mean
Class	Population seeding	Crossover	
Class VI	Random	EAX	8.200
		PX	8.655
		SRX	10.655
		MOX	13.545
		ODVX	3.520
	NN	EAX	6.374
		PX	8.240
		SRX	6.615
		MOX	10.400
		ODVX	2.630
	SI	EAX	7.447
		PX	8.640
		SRX	9.475
		MOX	10.145
		ODVX	3.690
	EV	EAX	6.956
		PX	6.035
		SRX	5.950
		MOX	9.580
		ODVX	1.705
	VE	EAX	7.411
		PX	6.465
		SRX	6.295
		MOX	9.485
		ODVX	2.250
	VV	EAX	6.404
		PX	5.720
		SRX	5.490
		MOX	11.300
		ODVX	**1.465**

Appendix E: The mean values of computation time of different GA models for Classes I, II and III of instances

Dependent variable: computation time			Mean
Class	Population seeding	Crossover	
Class I	Random	EAX	10.361
		PX	10.377
		SRX	7.910
		MOX	8.310
		ODVX	10.783
	NN	EAX	10.470
		PX	10.830
		SRX	**7.073**
		MOX	7.713
		ODVX	10.123
	SI	EAX	11.388
		PX	10.583
		SRX	7.817
		MOX	8.753
		ODVX	10.617
	EV	EAX	13.309
		PX	12.960
		SRX	9.883
		MOX	10.307
		ODVX	12.727
	VE	EAX	14.474
		PX	14.700
		SRX	11.210
		MOX	12.187
		ODVX	14.607
	VV	EAX	16.949
		PX	14.807

Dependent variable: computation time			Mean
Class	Population seeding	Crossover	
		SRX	11.637
		MOX	12.400
		ODVX	13.730

Dependent variable: computation time			Mean
Class	Population seeding	Crossover	
Class II	Random	EAX	78.962
		PX	74.610
		SRX	68.867
		MOX	65.807
		ODVX	72.443
	NN	EAX	73.161
		PX	67.477
		SRX	58.663
		MOX	54.017
		ODVX	61.240
	SI	EAX	66.219
		PX	58.803
		SRX	**47.963**
		MOX	48.537
		ODVX	55.123
	EV	EAX	66.090
		PX	60.963
		SRX	54.643
		MOX	51.787
		ODVX	57.863
	VE	EAX	71.648
		PX	77.437
		SRX	67.807
		MOX	66.697
		ODVX	71.567
	VV	EAX	74.592
		PX	71.730
		SRX	64.627
		MOX	64.833
		ODVX	66.577

Dependent variable: computation time			Mean
Class	Population seeding	Crossover	
Class III	Random	EAX	188.115
		PX	170.677
		SRX	153.583
		MOX	146.550
		ODVX	155.080
	NN	EAX	178.599
		PX	162.673
		SRX	145.187
		MOX	136.583
		ODVX	145.713
	SI	EAX	154.989
		PX	137.927
		SRX	123.103
		MOX	118.537
		ODVX	126.933
	EV	EAX	142.093
		PX	136.397
		SRX	117.913
		MOX	**110.457**
		ODVX	120.757
	VE	EAX	157.083
		PX	146.907
		SRX	114.403
		MOX	116.097
		ODVX	121.463
	VV	EAX	180.009
		PX	178.667
		SRX	160.643
		MOX	144.237
		ODVX	166.273

Appendix F: The mean values of Computation Time of different GA models for Classes IV, V and VI of instances

Dependent variable: computation time			Mean
Class	Population seeding	Crossover	
Class IV	Random	EAX	797.182
		PX	788.497
		SRX	765.300
		MOX	757.280
		ODVX	776.177
	NN	EAX	556.384
		PX	548.070
		SRX	536.020
		MOX	532.020
		ODVX	535.780
	SI	EAX	620.295
		PX	603.697

Dependent variable: computation time			Mean
Class	Population seeding	Crossover	
		SRX	589.613
		MOX	579.983
		ODVX	596.510
EV	EAX	537.149	
	PX	528.503	
	SRX	509.220	
	MOX	**502.133**	
	ODVX	**502.133**	
VE	EAX	600.023	
	PX	587.080	
	SRX	574.607	
	MOX	565.857	
	ODVX	574.817	
VV	EAX	631.661	
	PX	620.040	
	SRX	605.910	
	MOX	591.057	
	ODVX	604.690	

Dependent variable: computation time			Mean
Class	Population seeding	Crossover	
Class V	Random	EAX	4.14E3
		PX	3.98E3
		SRX	3.62E3
		MOX	3.59E3
		ODVX	3.61E3
	NN	EAX	3.49E3
		PX	3.35E3
		SRX	3.14E3
		MOX	3.00E3
		ODVX	3.05E3
	SI	EAX	4.08E3
		PX	3.95E3
		SRX	3.60E3
		MOX	3.55E3
		ODVX	3.59E3
	EV	EAX	3.42E3
		PX	3.39E3
		SRX	3.04E3
		MOX	2.98E3
		ODVX	**2.91E3**
	VE	EAX	3.49E3
		PX	3.36E3
		SRX	3.10E3
		MOX	3.09E3
		ODVX	3.09E3
	VV	EAX	3.73E3
		PX	3.78E3
		SRX	3.40E3
		MOX	3.26E3
		ODVX	3.37E3

Dependent variable: computation time			Mean
Class	Population seeding	Crossover	
Class VI	Random	EAX	1.115E4
		PX	1.015E4
		SRX	9.272E3
		MOX	9.004E3
		ODVX	9.062E3
	NN	EAX	8.372E3
		PX	7.832E3
		SRX	7.085E3
		MOX	6.862E3
		ODVX	7.287E3
	SI	EAX	9.754E3
		PX	8.852E3
		SRX	8.188E3
		MOX	7.845E3
		ODVX	8.187E3
	EV	EAX	7.727E3
		PX	6.902E3
		SRX	6.477E3
		MOX	6.257E3
		ODVX	**6.202E3**
	VE	EAX	8.416E3
		PX	7.828E3
		SRX	7.313E3
		MOX	7.295E3
		ODVX	7.282E3
	VV	EAX	1.015E4
		PX	9.039E3
		SRX	8.462E3
		MOX	8.637E3
		ODVX	8.221E3

Appendix G: Experimental results of Class I of test instances

Sl. no	Class of instance	Instance	Population seeding techniques	Crossover techniques														
				EAX			PX			SRX			MOX			ODVX		
				Err rate	Avg err rate	Time	Err rate	Avg err rate	Time	Err rate	Avg err rate	Time	Err rate	Avg err rate	Time	Err rate	Avg err rate	Time
1	Class I	Eil51	Random	0.00	0.00	6.00	0.02	0.04	5.97	0.00	0.03	5.71	0.01	0.02	5.43	0.00	0.00	5.61
			NN	0.00	0.00	5.92	0.00	0.00	5.84	0.00	0.00	5.68	0.02	0.04	5.48	0.00	0.00	5.54
			SI	0.00	0.00	6.85	0.00	0.00	6.27	0.00	0.00	6.00	0.01	0.02	5.78	0.00	0.00	6.23
			EV	0.00	0.00	8.21	0.00	0.00	8.11	0.00	0.00	7.91	0.00	0.00	7.81	0.00	0.00	7.81
			VE	0.00	0.00	9.16	0.00	0.00	9.06	0.01	0.01	8.86	0.02	0.02	8.66	0.00	0.00	8.61
			VV	0.00	0.00	10.07	0.00	0.00	9.97	0.00	0.00	9.87	0.00	0.00	9.62	0.00	0.00	9.67
2		rat99	Random	0.70	0.94	12.18	0.90	1.14	12.32	1.02	1.58	8.85	1.16	1.46	9.72	0.42	0.94	13.34
			NN	0.00	0.03	12.57	0.00	0.06	13.29	0.19	0.69	7.45	0.56	0.57	8.36	0.00	0.06	12.37
			SI	0.57	0.81	13.61	0.27	1.01	12.51	0.72	1.30	8.51	0.86	1.46	10.22	0.42	0.80	12.36
			EV	0.00	0.01	15.84	0.00	0.03	15.32	0.77	0.95	10.38	0.50	1.12	11.35	0.00	0.03	15.03
			VE	0.06	0.13	17.04	0.00	0.05	17.43	0.37	1.11	11.91	1.01	1.17	13.52	0.00	0.07	17.52
			VV	0.00	0.02	20.32	0.00	0.04	16.88	0.64	0.87	12.49	0.77	0.97	13.67	0.00	0.09	15.27
3		kroA100	Random	0.92	0.98	12.91	0.91	1.15	12.84	1.04	1.59	9.17	1.27	1.61	9.78	0.43	0.99	13.40
			NN	0.00	0.03	12.92	0.00	0.05	13.36	0.65	0.70	8.09	0.71	0.81	9.30	0.00	0.03	12.46
			SI	0.87	0.95	13.71	0.46	1.12	12.97	0.85	1.44	8.94	0.93	1.59	10.26	0.43	0.99	13.26
			EV	0.00	0.16	15.88	0.00	0.10	15.45	0.82	1.05	11.36	0.92	1.12	11.76	0.00	0.06	15.34
			VE	0.06	0.17	17.21	0.00	0.02	17.61	0.82	1.11	12.86	1.01	1.37	14.38	0.01	0.09	17.69
			VV	0.01	0.14	20.45	0.00	0.08	17.57	0.74	0.75	12.55	0.91	1.10	13.91	0.00	0.03	16.25

Appendix H: Experimental results of Class II of test instances

Sl. no	Class of instance	Instance	Population seeding techniques	Crossover techniques														
				EAX			PX			SRX			MOX			ODVX		
				Err rate	Avg err rate	Time	Err rate	Avg err rate	Time	Err rate	Avg err rate	Time	Err rate	Avg err rate	Time	Err rate	Avg err rate	Time
4	Class II	tsp225	Random	0.46	0.70	40.38	1.74	2.01	39.26	1.56	2.31	37.12	1.43	2.17	36.49	0.54	0.71	39.40
			NN	0.21	0.27	29.01	0.00	0.15	26.26	0.40	0.81	24.13	0.56	0.72	20.61	0.00	0.12	24.62
			SI	0.30	0.53	33.14	1.40	1.68	29.86	0.90	1.66	27.75	0.78	1.33	25.78	0.29	0.47	30.10
			EV	0.00	0.25	36.18	0.00	0.14	33.49	0.38	0.79	30.21	0.54	0.71	30.40	0.00	0.12	32.96
			VE	0.21	0.29	40.52	0.06	0.16	37.68	0.41	0.83	34.49	0.58	0.74	32.90	0.04	0.14	36.42
			VV	0.00	0.21	37.08	0.00	0.26	37.27	0.20	0.77	33.83	0.43	0.49	35.11	0.00	0.10	35.78
5		lin318	Random	0.74	0.88	78.96	1.83	1.97	74.61	1.74	2.19	68.87	1.36	1.85	65.81	0.69	0.83	72.44
			NN	0.25	0.30	73.16	0.06	0.18	67.48	0.48	0.68	58.66	0.57	0.75	54.02	0.05	0.16	61.24
			SI	0.63	0.77	66.22	1.66	1.80	58.80	1.27	1.48	51.96	0.90	1.19	48.54	0.56	0.67	55.12
			EV	0.00	0.32	70.76	0.00	0.13	63.64	0.43	0.75	56.05	0.51	0.60	54.81	0.00	0.14	60.61
			VE	0.24	0.32	102.20	0.32	0.19	96.81	0.49	0.70	83.87	0.60	0.77	86.23	21.08	0.19	90.58
			VV	0.00	0.31	74.09	0.00	0.28	73.39	0.29	0.68	63.36	0.50	0.44	66.10	0.00	0.14	65.53
6		d493	Random	1.01	1.07	117.55	1.92	1.92	109.96	1.92	2.08	100.61	1.30	1.53	95.12	0.85	0.94	105.49
			NN	0.29	0.32	117.31	0.12	0.21	108.69	0.55	0.55	93.20	0.59	0.78	87.42	0.10	0.20	97.86
			SI	0.96	1.01	99.30	1.92	1.92	87.75	1.64	1.30	76.18	1.02	1.05	71.29	0.83	0.88	80.15
			EV	0.26	0.28	96.00	0.05	0.21	85.76	0.53	0.77	77.67	0.56	0.69	70.15	0.03	0.15	80.02
			VE	0.29	0.39	102.77	0.11	0.26	97.82	0.57	1.37	85.06	0.81	0.96	80.96	0.10	0.23	87.70
			VV	0.02	0.16	112.11	0.01	0.06	104.53	0.47	0.64	96.69	0.53	0.63	93.29	0.00	0.01	98.42

Appendix I: Experimental results of Class III of test instances

Sl. no	Class of instance	Instance	Population seeding techniques	Crossover techniques														
				EAX			PX			SRX			MOX			ODVX		
				Err rate	Avg err Rate	Time	Err rate	Avg err rate	Time	Err rate	Avg err rate	Time	Err rate	Avg err rate	Time	Err rate	Avg err rate	Time
7	Class III	d657	Random	1.28	1.39	160.26	1.97	2.32	146.69	2.20	2.56	132.63	2.19	2.73	126.23	0.96	1.09	136.01
			NN	0.32	0.46	154.61	0.13	0.32	141.86	0.58	0.69	124.47	0.64	0.95	117.00	0.09	0.22	127.16
			SI	1.20	1.31	132.81	1.96	2.31	118.47	1.94	2.05	105.13	1.96	2.39	100.26	0.92	1.02	108.72
			EV	0.24	0.45	119.52	0.18	0.37	117.13	0.52	0.60	101.92	0.49	0.69	95.03	0.06	0.17	105.68
			VE	0.47	0.51	135.98	0.23	0.36	127.40	0.76	0.74	99.13	0.57	1.14	99.98	0.18	0.22	106.39
			VV	0.16	0.44	155.75	0.01	0.35	152.63	0.49	0.50	136.02	0.47	0.49	121.21	0.00	0.16	142.53
8		u724	Random	1.24	1.15	201.11	1	2.45	181.93	1.63	2.51	163.48	3.03	3.57	156.08	1.1	1.37	162.69
			NN	0.35	0.54	189.27	0.3	0.39	171.13	0.4	0.67	155.36	0.6	1.41	146.17	0.1	0.59	153.52
			SI	1.37	0.93	165.83	1.95	2.64	146.11	2.15	2.72	130.1	1.62	3.22	126.12	1.41	1.99	134.79
			EV	0.25	0.6	151.23	0.35	0.42	145.61	0.46	0.7	125.3	0.55	0.77	117.89	0.05	0.11	127.19
			VE	0.44	0.54	166.68	0.29	0.49	156.13	0.7	0.85	120.06	0.61	1.21	123.05	0.17	0.29	127.1
			VV	0.01	0.48	191.39	0.07	0.11	191.15	0.2	0.59	172.43	0.42	0.77	155.27	0	0.21	176.31
9		rat783	Random	1.56	1.71	202.98	2.02	2.72	183.41	2.49	3.04	164.64	3.09	3.92	157.34	1.08	1.23	166.54
			NN	0.36	0.59	191.91	0.14	0.44	175.03	0.61	0.82	155.73	0.70	1.12	146.58	0.09	0.23	156.46
			SI	1.44	1.60	166.33	2.00	2.70	149.20	2.24	2.79	134.08	2.90	3.73	129.23	1.01	1.17	137.29
			EV	0.29	0.56	151.37	0.18	0.43	146.45	0.59	0.71	126.52	0.69	0.99	118.45	0.06	0.19	129.40
			VE	0.53	0.65	168.79	0.25	0.49	157.19	0.81	0.88	124.02	0.62	1.34	125.26	0.17	0.24	130.90
			VV	0.19	0.54	195.06	0.01	0.41	192.22	0.57	0.69	173.48	0.69	0.77	156.23	0.00	0.18	179.98

Appendix J: Experimental results of Class IV of test instances

Sl. no	Class of instance	Instance	Population seeding techniques	Crossover techniques														
				EAX			PX			SRX			MOX			ODVX		
				Err rate	Avg err rate	Time	Err rate	Avg err rate	Time	Err rate	Avg err rate	Time	Err rate	Avg err rate	Time	Err rate	Avg err rate	Time
10	Class IV	fl1577	Random	1.88	2.15	382.96	2.23	2.30	375.75	1.41	1.55	353.23	1.57	1.86	357.69	0.88	1.11	365.98
			NN	0.23	0.35	311.86	0.05	0.16	305.06	1.32	1.54	298.36	1.34	1.82	285.00	0.10	0.36	298.95
			SI	1.28	1.55	313.24	2.17	2.24	296.10	1.12	1.26	276.39	1.53	1.82	275.43	0.63	0.76	289.61
			EV	0.10	0.32	243.93	0.00	0.14	242.91	1.24	1.42	227.10	1.30	1.73	218.90	0.01	0.13	228.41
			VE	0.24	0.42	299.25	1.31	0.88	299.04	1.23	1.67	287.35	1.37	1.85	271.70	0.14	0.39	290.56
			VV	0.09	0.33	264.36	0.00	0.16	261.35	1.10	1.29	252.29	1.23	1.72	232.32	0.00	0.11	248.13
11		d2103	Random	0.92	0.97	572.29	2.37	2.40	567.71	1.62	1.64	541.88	1.99	2.07	534.68	0.75	0.77	567.06
			NN	0.15	0.18	390.53	0.07	0.11	382.62	0.65	0.72	365.05	0.82	0.85	371.22	0.05	0.08	370.47
			SI	0.64	0.69	431.72	2.16	2.19	413.50	1.41	1.43	406.77	1.41	1.49	397.75	0.71	0.73	412.40
			EV	0.11	0.15	349.86	0.16	0.11	334.03	0.62	0.61	321.37	0.81	0.77	316.44	0.04	0.05	322.40
			VE	0.17	0.22	418.42	0.16	0.19	396.48	0.59	0.78	389.89	0.80	0.96	385.25	0.10	0.11	387.44
			VV	0.08	0.13	429.69	0.02	0.04	412.18	0.51	0.53	409.15	0.73	0.74	389.83	0.01	0.05	409.88
12		fnl4461	Random	1.75	1.81	1436.30	1.31	1.42	1422.03	2.43	2.97	1400.79	3.58	4.11	1379.47	0.93	1.07	1395.49
			NN	0.23	0.36	966.76	0.06	0.21	956.53	1.25	1.21	944.65	1.48	1.63	939.84	0.13	0.17	937.92
			SI	1.67	1.72	1115.93	1.08	1.19	1101.49	1.63	2.17	1085.68	2.46	2.96	1066.77	0.90	1.04	1087.52
			EV	0.23	0.33	1017.65	0.10	0.18	1008.57	1.08	1.16	979.19	1.40	1.57	971.06	0.04	0.14	985.44
			VE	0.28	0.36	1082.40	0.13	0.21	1065.72	1.38	1.33	1046.58	1.57	1.69	1040.62	0.18	0.24	1046.45
			VV	0.15	0.22	1200.94	0.08	0.11	1186.59	1.05	1.07	1156.29	1.34	1.36	1151.02	0.01	0.02	1156.06

Appendix K: Experimental results of Class V of test instances

Sl. no	Class of instance	Instance	Population seeding techniques	Crossover techniques														
				EAX			PX			SRX			MOX			ODVX		
				Err rate	Avg err rate	Time	Err rate	Avg err rate	Time	Err rate	Avg err rate	Time	Err rate	Avg err rate	Time	Err rate	Avg err rate	Time
13	Class V	rl5915	Random	1.69	1.72	2222.77	1.05	1.78	2189.56	1.24	1.27	2126.15	2.46	2.49	2107.82	0.73	0.74	2117.47
			NN	0.16	0.19	1533.87	0.11	0.10	1512.14	1.17	1.04	1463.47	1.72	1.98	1457.01	0.11	0.15	1459.84
			SI	1.62	1.65	1685.95	1.12	1.65	1655.63	1.10	1.13	1600.24	2.30	2.44	1596.92	0.90	1.12	1600.32
			EV	0.16	0.19	1247.65	0.09	0.19	1225.23	0.88	0.89	1183.01	1.65	1.75	1180.61	0.08	0.12	1185.20
			VE	0.19	0.24	1105.30	0.11	0.11	1224.32	1.59	1.07	1198.71	1.75	1.99	1182.76	0.20	0.17	1181.00
			VV	0.11	0.19	1560.42	0.07	0.27	1536.20	0.71	0.94	1498.37	1.57	1.68	1482.65	0.01	0.05	1474.97
14		rl11849	Random	1.43	1.71	4140.89	2.06	2.39	3917.85	1.62	2.15	3687.23	2.62	2.78	3570.45	0.98	1.21	3604.11
			NN	0.49	1.31	3386.47	0.09	0.90	3386.77	1.18	2.10	3175.64	2.15	3.08	3092.00	0.24	0.72	3212.17
			SI	0.87	1.15	4562.45	1.97	2.30	4329.72	1.21	1.74	4084.69	2.26	2.42	3974.64	0.57	0.90	4000.71
			EV	0.28	0.91	3862.37	0.11	0.40	3748.34	0.96	1.73	3524.59	2.03	3.35	3429.10	0.07	0.12	3435.98
			VE	0.34	1.02	4148.85	0.17	0.52	3928.68	1.13	1.99	3695.07	2.13	3.57	3596.23	0.21	0.45	3619.90
			VV	0.18	0.78	3407.29	0.09	0.20	3966.96	0.88	1.47	3530.99	2.01	3.07	3420.39	0.01	0.01	3596.86
15		brd14051	Random	1.19	2.59	6159.61	2.98	3.27	5846.84	1.79	2.64	5112.92	3.51	3.93	4908.39	1.07	1.73	4962.63
			NN	1.10	2.74	5546.74	0.24	2.01	5165.68	1.44	2.26	4702.79	2.03	3.35	4510.53	0.57	1.06	4551.70
			SI	1.03	2.53	6000.34	2.44	3.13	5849.43	1.58	2.63	5296.50	2.88	3.30	5062.26	1.01	1.68	5135.01
			EV	0.54	1.67	5246.40	0.20	1.52	5199.16	1.17	1.71	4484.25	1.74	2.55	4295.02	0.33	0.55	4342.63
			VE	0.83	2.14	5230.84	0.20	1.86	4915.52	1.42	2.13	4615.12	1.80	2.86	4428.73	0.47	0.73	4465.39
			VV	0.33	1.44	6380.08	0.17	1.36	5740.95	0.98	1.45	5229.68	1.60	2.28	5076.33	0.07	0.23	5088.02

Appendix L: Experimental results of Class VI of test instances

Sl. no	Class of instance	Instance	Population seeding techniques	Crossover techniques														
				EAX			PX			SRX			MOX			ODVX		
				Err rate	Avg err rate	Time	Err rate	Avg err rate	Time	Err rate	Avg err rate	Time	Err rate	Avg err rate	Time	Err rate	Avg err rate	Time
16	Class VI	d15112	Random	3.00	6.90	9312.08	4.63	7.91	8502.38	3.64	6.31	7799.02	6.41	8.85	7552.71	1.80	3.23	7589.26
			NN	1.92	4.23	7122.69	1.09	5.78	6942.29	2.25	6.39	6236.30	4.60	7.54	6036.68	0.99	1.66	6504.39
			SI	2.19	6.16	8404.12	4.28	8.25	7628.09	2.29	7.15	7006.34	6.12	8.05	6788.18	1.97	3.80	7198.14
			EV	1.33	4.48	5984.95	0.97	3.18	5484.71	1.98	6.07	5108.39	3.31	6.85	4954.07	0.44	0.83	5016.50
			VE	1.58	4.82	6407.34	1.16	3.65	6054.43	2.22	6.33	5541.80	3.37	7.07	5617.53	0.71	1.21	5374.76
			VV	1.06	3.94	8556.70	0.73	2.86	7668.57	1.68	5.65	7151.58	3.14	6.54	7751.38	0.17	0.62	7012.80
17		d18512	Random	3.72	9.50	12,995.87	4.40	9.40	11,803.98	4.96	15.00	10,745.12	6.82	18.24	10,454.73	1.71	3.81	10,535.70
			NN	0.88	8.52	9621.82	0.90	10.70	8722.62	2.75	6.84	7934.65	6.27	13.26	7687.89	0.87	3.60	8068.70
			SI	3.26	8.73	11,104.25	4.64	9.03	10,076.12	4.36	11.80	9369.79	6.11	12.24	8902.62	1.47	3.58	9174.92
			EV	0.71	9.44	9468.90	0.33	8.89	8320.10	1.85	5.83	7846.40	5.33	12.31	7560.24	0.31	2.58	7586.61
			VE	0.68	10.00	10,424.58	0.38	9.28	9601.47	2.18	6.26	9083.72	5.85	11.90	8972.73	0.51	3.29	9188.81
			VV	0.32	8.87	11,744.87	0.20	8.58	10,409.19	1.05	5.33	9772.12	5.32	16.06	9523.23	0.19	2.31	9428.34

References

Andal JG, Sathiamoorthy S (2001) A hybrid genetic algorithm: a new approach to solve traveling salesman problem. Int J Comput Eng Sci 2(2):339–355

Arthi J, Nanthini R, Sridevi S, Victer Paul P (2015) Enhanced ODV based population seeding technique for ATSP. In: International conference on innovations in information, embedded and communication systems (ICIIECS), India, pp 1–4

Chen SM, Chien CY (2011) Solving the travelling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques. Expert Syst Appl 38:14439–14450

Chuan KT(2007) Multi-parent extension of edge recombination. In: Proceedings of the 9th annual ACM conference on genetic and evolutionary computation (GECCO '07), pp 1535–1535

Deng Y, Liu Y, Zhou D (2015) An improved genetic algorithm with initial population strategy for symmetric TSP. Mathematical Problems in Engineering 2015

Eiben A (2002) Multiparent recombination in evolutionary computing. In: Advances in evolutionary computing, Springer, pp 175–192

Eiben A, Raué P-E, Ruttkay Z (1994) Genetic algorithms with multi-parent recombination. In: Parallel problem solving from nature—PPSN III. LNCS, vol 866, pp 78–87, Springer

El-Mihoub TA, Hopgood AA, Nolle L, Battersby A (2006) Hybrid genetic algorithms: a review. Eng Lett 13(2):124–137

Fei L, Guangzhou Z (2009) Study of genetic algorithm with reinforcement learning to solve the TSP. Expert Syst Appl 36:6995–7001

Hains DR (2012) Generalized partition crossover for the traveling salesman problem. Diss. Colorado State University, Libraries

Katayama K, Sakamoto H, Narihisa H (2000) The efficiency of hybrid mutation genetic algorithm for the travelling salesman problem. Math Comput Model 31:197–203

Kaur D, Murugappan MM (2008) Performance enhancement in solving travelling salesman problem using hybrid genetic algorithm. In: Annual meeting of the North American fuzzy information processing society, NAFIPS 2008, pp 1–6

Liu G, Yuanxiang L, Xin N, Hao Z (2012) A novel clustering-based differential evolution with 2 multi-parent crossovers for global optimization. Appl Soft Comput 12(2012):663–681

Maaranen H, Miettinen K, Makela MM (2004) Quasi-random initial population for genetic algorithms. Comput Math Appl 47:1885–1895

Marinakis Y, Marinaki M (2010) A hybrid genetic–particle swarm optimization algorithm for solving the vehicle routing problem. Expert Syst Appl 37(2):1446–1455

Masafumi K, Kunihito Y, Masaharu M, Moritoshi Y, Ikuo Y (2010) Development of a novel crossover of hybrid genetic algorithms for large-scale traveling salesman problems. Artif Life Robot 15:547–550

Mathias K, Whitley D (1992) Genetic operators, the fitness landscape and the traveling salesman problem. In: Manner R, Manderick B (eds) Parallel problem solving from nature. North Holland, Elsevier, pp 219–228

Moganarangan N, Raju R, Ramachandiran R, Paul PV, Dhavachelvan P, Venkatachalapathy VS (2014) Efficient crossover operator for genetic algorithm with ODV based population seeding technique. Int J Appl Eng Res 9:3885–3898

Nagata Y (2004). Criteria for designing crossovers for TSP. In: Proceeding of congress on evolutionary computation (CEC 2004), vol 2. https://doi.org/10.1109/cec.2004.1331069

Nagata Y, Kobayashi S (1997) Edge assembly crossover: a high-power genetic algorithm for the traveling salesman problem. In:

Proceeding of the seventh international conference on genetic algorithms (ICGA), pp 450–457

Nguyen HD, Yoshihara I, Yasunaga M (2000) Modified edge recombination operators of genetic algorithms for the traveling salesman problem. In: 26th annual IEEE confjerence of the industrial electronics society (IECON), Nagoya. pp 2815–2820. https://doi.org/10.1109/iecon.2000.972444

Pan G et al (2016) Hybrid immune algorithm based on greedy algorithm and delete-cross operator for solving TSP. Soft Comput 20(2):555–566

Pandey HM et al. (2016) Evaluation of genetic algorithm's selection methods. In: Information systems design and intelligent applications. Springer, New Delhi, pp 731–738

Paul PV, Ramalingam A, Baskaran R, Dhavachelvan P, Vivekanandan K, Subramanian R, Venkatachalapathy VS (2013a) Performance analyses on population seeding techniques for genetic algorithms. Int J Eng Technol 5(3):2993–3000

Paul PV, Dhavachelvan P, Baskaran R (2013b) A novel population initialization technique for genetic algorithm. In: IEEE international conference on circuit, power and computing technologies (ICCPCT), India, pp 1235–1238. ISBN: 978-1-4673-4921-5

Paul PV, Ramalingam A, Baskaran R, Dhavachelvan P, Vivekanandan K, Subramanian R (2014) A new population seeding technique for permutation-coded genetic algorithm: service transfer approach. J Comput Sci 5(2):277–297

Paul PV, Moganarangan N, Kumar SS, Raju R, Vengattaraman T, Dhavachelvan P (2015) Performance analyses over population seeding techniques of the permutation-coded genetic algorithm: An empirical study based on traveling salesman problems. Applied Soft Computing. 32:383–402

Poon PW, Carter JN (1995) Genetic algorithm crossover operators for ordering applications. Comput Oper Res. 22(1):135–147

Porumbel DC, Jin-Kao H, Pascale K (2010) An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. Comput Oper Res 37(2010):1822–1832

Rong Y (1997) Solving large travelling salesman problems with small populations. In: Genetic algorithms in engineering systems: innovations and applications, Conference Publication No. 446, pp 157–162

Shanmugam M, SaleemBasha MS, Victer Paul P, Dhavachelvan P, Baskaran R (2013) Performance assessment over heuristic population seeding techniques of genetic algorithm: benchmark analyses on traveling salesman problems. Int J Appl Eng Res 8(10):0973–4562

Shubhra SR, Sanghamitra B, Sankar KP (2007) Genetic operators for combinatorial optimization in TSP and microarray gene ordering. J Appl Intell 26(3):183–195

Sivanandam SN, Deepa SN (2008) Introduction to genetic algorithms. Springer, Berlin

Ting CK (2005) Design and analysis of multi-parent genetic algorithms, Ph.D. thesis, University of Paderborn, Germany

Ting JZ (2013) A genetic algorithm for finding a path subject to two constraints. Appl Soft Comput 13(2):891–898

Ting CK, Chien-Hao S, Chung-Nan L (2010) Multi-parent extension of partially mapped crossover for combinatorial optimization problems. Expert Syst Appl 37:1879–1886

Tsai HK, Yang JM, Tsai YF, Kao CY (2004) An evolutionary algorithm for large traveling salesman problems. IEEE Trans Syst Man Cybern Part B 34(4):1718–1729

TSPLIB http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/. Accessed 21 Sep 2012

Tsutsui S, Ghosh A (1998) A study on the effect of multi-parent recombination in real coded genetic algorithms. In: Proceedings of international conference on evolutionary computation, pp 828–833

Tsutsui S, Jain L (1998) On the effect on multi-parent recombination in real coded genetic algorithms. In: Proceedings of the 2nd international conference on knowledge-based intelligent electronic systems, pp 155–160

Tsutsui S, Yamamura M, Higuchi T (1999) Multi-parent recombination with simplex crossover in real coded genetic algorithms. Proc Genet Evolut Comput Conf 1:657–664

Wang Y (2014) The hybrid genetic algorithm with two local optimization strategies for traveling salesman problem. Comput Ind Eng 70:124–133

Wang J et al (2016) Multi-offspring genetic algorithm and its application to the traveling salesman problem. Appl Soft Comput 43:415–423

Whitely D, Starkweather T, D'Ann F (1989) Scheduling problems and traveling salesman: the genetic edge recombination operator. In: Proceedings 3rd international conference genetic algorithms, pp 133–140

Whitley D, Hains D, Howe A (2009) Tunneling between optima: partition crossover for the traveling salesman problem. In: Proceedings of the 11th annual conference on genetic and evolutionary computation, pp 915–922. ACM

Whitley D, Hains D, Howe A (2010) A hybrid genetic algorithm for the traveling salesman problem using generalized partition crossover. In: Schaefer R, Cotta C, Kolodziej J, Rudolph G (eds) PPSN XI LNCS, vol 6238. Springer, Heidelberg, pp 566–575

Yingzi W, Yulan H, Kanfeng G (2007) Parallel search strategies for TSPs using a greedy genetic algorithm. In: Third international conference on natural computation (ICNC 2007). vol. 3. IEEE, Haikou, China. https://doi.org/10.1109/ICNC.2007.537