Experimental Evaluation of Traveling Salesman Problem Hybrid Genetic Algorithm Implementation with 2-opt Heuristic Local Search

Gökay Gülsoy
Computer Engineering Department
Izmir Institute of Technology
Gülbahçe, İzmir YTE, 35433 Barbaros/Urla/İzmir
E-Mail: gokaygulsoy@iyte.edu.tr

***Abstract*—In this paper, I introduce a comprehensive comparison of different crossover and mutation strategies along with application of heuristic search algorithm, which is known as 2-opt local search to further improve fitness values in each generation [1] in an genetic algorithm implementation of Traveling Salesman problem [2]. I have provided a genetic algorithm implementation of the Traveling Salesman problem with shortest path construction animation simulated in Java with JavaFX and tabulated the experimental results for different crossover and mutation strategy combinations and indicated which combinations yielded best results on two well known benchmark datasets namely, berlin52.tsp and att48.tsp from TSPLIB which is a benchmark library for various traveling salesman problems [3].**

***Index Terms*—genetic algorithm, traveling salesman problem, crossover, mutation, 2-opt local search, Java, JavaFX**

I. Introduction

First basic principles of genetic algorithm concept (GA) were proposed by Holland [4]. Genetic algorithm is inspired from the biological process known as natural selection in which robust individuals are likely to survive in a competing environment. Potential solution to a problem is represented as an individual called *chromosome* and each chromosome has a positive value named *fitness* which is used to indicate level of "goodness" for chromosome in solving problem. In the course of genetic evolution, chromosomes that have higher fitness tend to produce better-quality offsprings. In practice, population can be set randomly at the beginning of the genetic algorithm. Even though population size differs from problem to problem there are some principles that can be followed in order to determine population size [5]. In each iteration of genetic algorithm next generation chromosomes are created from chromosomes currently in population. Those chromosomes selected to yield next generation offsprings are called "parents". Parent genes recombined and mixed in order to produce offsprings for next generation. The idea behind the creation of new generation is that "'better" chromosomes generate more, higher-fitness offsprings, which reflects the "survival-of-fittest" mechanism. One of the most widely used scheme is the roulette wheel selection [6] described as follows:

1) Sum the fitness of all chromosomes in population denoted as total fitness(N)
2) Generate random number in range 0 to total fitness(N)
3) Get the first chromosome in population whose fitness is greater than or equal to n

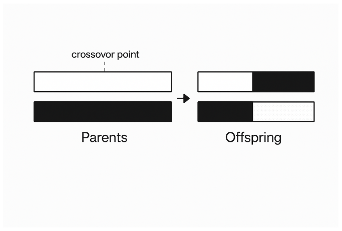

Fig. 1: One-point crossover

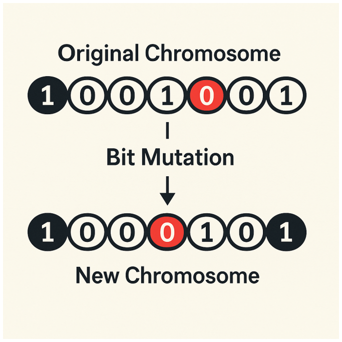

Fig. 2: Bit mutation on fourth and fifth bits

Genetic algorithm's evolution cycle repeats until a termination condition is reached where termination condition can be certain number of evolution cycles or specified value of fitness. Two operators, namely crossover and mutation, are used to realize the evolution cycle of the genetic algorithm. Simple example of crossover is one-point crossover mechanism shown in Fig 1. in which crossover point is set randomly and parts of chromosomes beyond that cut-off point to the right are exchanged to produce offsprings [7].

Crossover operation probability (p_c) generally takes value in range 0.6 to 1.0. Different from crossover, mutation operation is applied to each offspring after crossover operation is applied. It changes each bit with a small probability (p_m) with value less than 0.1 [7] as given in the Fig 2.

The implementation of the generic genetic algorithm pseudocode is given in Algorithm 1. which is the procedure followed in every genetic algorithm implementation as a general schema.

While there is no strictly determined rule to perform parent selection, there are some best practices that can be utilized [8, 9] as follows:

- for large population sizes (such as 100 or greater)
  crossover rate:, $p_c = 0.6$
  mutation rate, $p_m = 0.001$
- for small population sizes (such as 30 or smaller)
  crossover rate:, $p_c = 0.9$
  mutation rate, $p_m = 0.01$

Algorithm 1 Generic Genetic Algorithm Pseudocode

```
1: // initialize random population of individuals
2: initialize population $P(t)$
3: // evaluate fitness of all individuals in population
4: evaluate $P(t)$
5: // initialize termination condition
6: terminated := false
7: // check for termination condition (iteration count, fitness, etc.)
8: while not terminated do
9:     // select a parents for offspring production
10:    $P'$ := select_parents($P(t)$)
11:    // perform crossover with selected parents
12:    recombine $P'(t)$
13:    // mutate created offsprings stochastically
14:    mutate $P'(t)$
15:    // evaluate fitness for new offsprings
16:    evaluate $P'(t)$
17:    // select the survivors according to fitness values
18:    $P(t)$ := survive($P(t), P'(t)$)
19:    if termination condition is reached then
20:        terminated := true
21:    end if
22: end while
```

In my work I have implemented a Traveling Salesman problem with an genetic algorithm which is a well-known combinatorial optimization problem in which goal is to find shortest tour that visits each city only once and returns back to city that tour started from [10]. Implementation is done in Java with shortest path construction simulation utilizing JavaFX library and experiment different crossover types which are: Order One crossover, Partially Mapped crossover, and Cycle crossover along with various mutation strategies which are: Swap mutation, Insert mutation, Scramble mutation, and Inversion mutation in two well-known TSP datasets, namely, att48.tsp and berlin52.tsp from TSPLIB which is a library of TSP problems [3]. I have determined the best crossover and mutation strategy combination with population sizes 30, 80, and 150 for both the att48.tsp and berlin52.tsp datasets.

II. Related Works

The work in [10] provides an Java implementation of TSP problem which allows user to provide population size and mutation rate from GUI and allows distance data to be loaded into program and reports shortest path, shortest path distance, number of generations passed until reaching termination condition, and runtime of genetic algorithm in GUI. Simple and fast solution to TSP problem in spherical 3D coordinates employing genetic algorithm and 2-opt local search is provided by [11] which differs from the 2D TSP in that all cities are on a sphere and it is only allowed to move on the surface of the sphere to complete a tour. Another hybrid algorithm named genetic ant colony optimization (GACO) applied to TSP problem makes use of ant colony optimization (ACO) with genetic algorithm which has the advantage of ACO, the ability to discover feasible solutions and avoid early convergence, and with GA, the ability to refrain from being trapped in local optima has shown to find optimum solutions effectively [12]. The study that utilizes distributed computing with ant colony system (ACS) to find optimal or suboptimal solutions to the symmetric and asymmetric TSP problem is done by [13] in which the solution is built incrementally via the transition of the ant state followed by the local update rule of the pheromone combined with the global update rule of the pheromone which overcomes stochastic multigreedy behavior. Using genetic algorithms alone does not often lead to best solution in terms of optimality, genetic algorithms are mostly combined with heuristic search strategies to reach better results [14]. One such study is done in [15] proposed new crossover operator named distance preserving crossover (DPX) which performs jumps within the search space of local minima and new mutation operator named non-sequential 4-change which performs random jumps near neighborhood of local optimum along with parent replacement scheme which tries to avoid emergence of individuals that are very similar. The idea behind the design of DPX is that global optimum solution can be found within other near-optimum solutions where each near-optimum solution is approximately equally distanced [16]. Non-sequential 4-change mutation operator transforms tour T into T' such that distance between T and T' is 4, and it makes the assumption that tour length does not change considerably, hence it will not diverge current solution from good near-optimum solution solution [17]. Also designed parent replacement scheme is aimed at preserving population diversity which selects the individual from population that is most similar to offspring to be added to population and replaces the individual with offspring if distance between them is falling below pre-defined threshold. This offspring will only be replaced if and only if it has higher fitness; otherwise individual with worst fitness from population will be replaced with offspring. Variation of ant-colony optimization algorithm for TSP problem proposed by [18] partitions ants into two sets named as scout ants and common ants, where common ants work according to search strategy of basic ant-colony algorithm and scout ants shorten the search time and retain the characteristics of optimal solution while accelerating convergence speed of the algorithm. Basic ant-colony optimization for TSP can be enhanced by allowing only the best ant to update trails in every cycle, but that approach suffers from problem of early stagnation of the search space which makes tour improvements unfeasible because ants will all the time construct same tour again and again. Max-Min ant-colony system was proposed to solve that problem via introducing maximum and minimum trail strengths on paths [19]. Because of the pre-defined min trail limit, probability that a specific path is chosen can be very small, but will never be zero. This minimum limit reduce the risk of early stagnation and thus leading to further exploration of search space. Trail strengths are initialized to maximum trail limit and after each iteration, evaporation will reduce trail strengths in such a way that trails on bad paths decrease slowly, on the other hand, trails on good paths can maintain high level of trail and therefore be selected more often by the ants. One study that compares sequential and parallel version of ant-colony optimization problem for solving TSP problem is done by [20] showed that as the number of cities to be visited or number of ants in colony increases, gap between the time that takes algorithm to terminate for sequential and parallel version increases regularly. Results suggested that in order for very large scale optimization problems to be solved in reasonable amount of time data parallelism is crucial. Distributed and decentralized implementation of ant-colony optimization with a fuzzy parameter adaptation for TSP [21] divides processes into two groups where one process is used as a controller and remaining processes are workers. Controller process is used for maintaining system via coordinating the distribution of statistics of processes to each other and does not involve in computation of solutions. Worker processes are responsible for constructing solutions. Results have shown that adaptive approach in which controller process has the responsibility of monitoring measurements for exploration versus exploitation and convergence to adapt ACO parameters outperforms non-adaptive version in terms of both cost of best tour and number of iterations required for convergence.

III. Methodology

I have provided a hybrid implementation for TSP that combines the genetic algorithm and the 2-opt local search heuristic in Java programming language with the JavaFX library to create shortest path construction simulations for two benchmark datasets, namely att48.tsp and berlin52.tsp from TSPLIB [3]. The implemented genetic algorithm provides swap, insert, scramble, and inversion mutations along with order one (O1), partially mapped (PMX), and cycle crossover strategies [22, 23]. Each mutation and crossover strategy, along with 2-opt local heuristic search [24] is described in detail in the following subsections. Numerical results obtained from experiments are provided in detail in section IV.

A. *Mutation Strategies Used*

Implementation of 4 different mutation strategies which are swap, insert, scramble and inversion mutation are used in experiments.

1) Swap Mutation: Swap mutation swaps the allele on the chromosome currently being considered with another allele located at randomly chosen location. Fig. 3 shows the swap mutation.

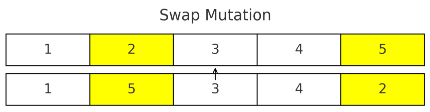

Fig. 3: Swap mutation

2) Insert Mutation: Insert mutation picks two alleles at random and moves the second to follow the first, shifting the rest to accommodate. Insert mutation preserves most of the order and adjacency information and it is depicted in Fig. 4.

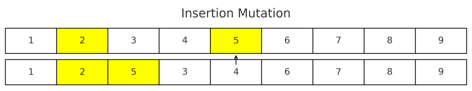

Fig. 4: Insert mutation

3) Scramble Mutation: Scramble mutation picks a subset of alleles at random, then randomly rearranges alleles in that subset. Fig. 5 depicts the scramble mutation.

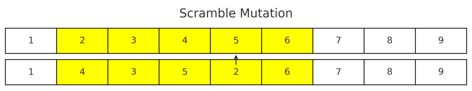

Fig. 5: Scramble mutation

4) Inversion Mutation: Inversion mutation picks two alleles at random and then inverts the alleles in between them. Inversion mutation preserves most of the adjacency but it is disruptive in terms of information. Fig. 6 depicts inversion mutation.

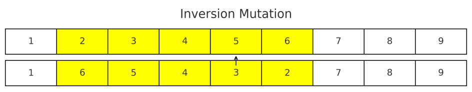

Fig. 6: Inversion mutation

B. *Crossover Strategies Used*

Implementation of 3 different crossover strategies which are order 1 (O1), partially mapped (PMX), and cycle crossover [25] are used in experiments.

1) Order-1 Crossover: Idea behind the order-1 crossover is to preserve the relative order in which alleles occur and Fig. 7 depicts the order-1 crossover. Procedure for order-1 crossover is defined as follows:

1) Choose arbitrary part from the first parent
2) Copy this part to first child
3) Copy alleles that are not in the first part to the first child
    a) right from the cut point of the the copied part
    b) using the order of the second parent
    c) wrapping around at the end
4) Analogously for the second child, with parent roles exchanged.

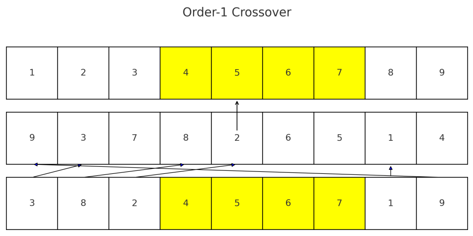

Fig. 7: Order-1 Crossover

2) Partially Mapped Crossover: Procedure for partially mapped crossover is depicted in Fig. 8 and defined as follows:

1) Choose random segment and copy it from first parent to child
2) Starting from the first crossover point look for elements in that segment of second parent that have not been copied to child
3) For each allele i look in the child to see what element j has been copied in its place from first parent
4) Place i into the position occupied by j in second parent, since element j is already in child
5) If the place occupied by j in second parent has already been filled in child by element k, put i in the position occupied by k in second parent
6) Having dealt with the elements from crossover segment, the rest of the child can be filled from second parent
7) Second child is created analogously, with parent roles exchanged.

3) Cycle Crossover: Procedure for cycle crossover is depicted in Fig. 9 and defined as follows:

1) Make a cycle of alleles from the first parent as follows:
    a) Start with the first allele of first parent
    b) Look at the allele at the same position in second parent
    c) Go to the position with the same allele in first parent
    d) Add this allele to the cycle
    e) Repeat step b through d until first allele reached
2) Put the alleles of the cycle in the first child on the positions they have in the first parent
3) Take next cycle from second parent

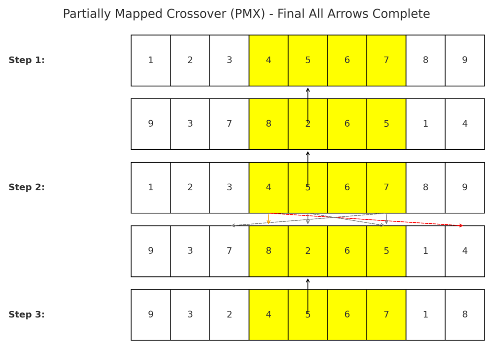

Fig. 8: Partially Mapped Crossover

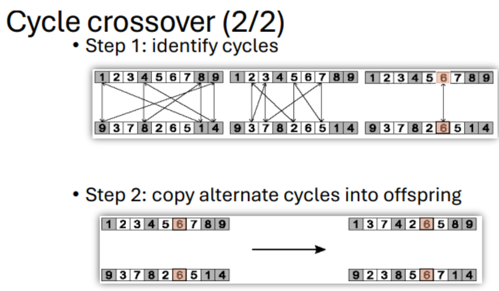

Fig. 9: Cycle Crossover

C. 2-Opt Heuristic Local Search

2-opt is an simple heuristic local search algorithm that can be used in the solution of TSP [26]. The idea behind the 2-opt is to delete two edges from a tour and reconnect them to see whether cost of tour decrease. 2-opt decreases the cost of tour when the sum of length of two deleted edges is greater than sum of the two reconnected edges which is equivalent to inversion of sub-path between two edges. In my implementation in order for 2-opt local search to scale large scale problems I have have constrained the fitness improvement to 5 times, so after 5 fitness improvements no further improvement is done. Algorithm 2 gives the pseudocode for 2-opt local search procedure as follows:

D. Technologies Used and System Structure

Implementation of TSP with genetic algorithm and 2-opt heuristic local search is done with Java programming language, utilizing JavaFX library for animating the shortest path construction procedure through generations and for final near-optimal path. Simulator can be accessed from given GitHub repository[1] to experiment different combinations of crossover and mutation strategies along with different population sizes via either enabling or disabling 2-opt heuristic local search algorithm with a boolean flag. System decomposed into 6 different packages which are *tsp.genetic.algorithm*, *tsp.genetic.animator*, *tsp.genetic.context*, *tsp.genetic.entities*,

[1]Link to TSP hybrid genetic algorithm with 2-opt local search simualtor

Algorithm 2 2-opt Local Search Algorithm Pseudocode

```
// initialize fitness improvement flag to true
$improvedFitness = true$
// initialize variable to keep track of improvement count
$improvementCount = 0$
$permutation \leftarrow getRandomInitialTour()$
$fitnessLoop :$
while $improvedFitness$ do
    $improvedFitness = false$
    for $i \leftarrow 1, permutationSize - 2$ do
        for $j \leftarrow i + 1, permutationSize$ do
            // skip adjacent locations
            if $j - i = 1$ then
                continue
            end if
            // create a copy of permutation
            $newPermutation \leftarrow copy(permutation)$
            // reverse the sub-path between $(i, j)$
            $reversed \leftarrow reverse(newPermutation, i, j)$
            // compute fitness after inversion of sub-path
            $computeFitness(reversed)$
            // get the original fitness value
            $orgFitness \leftarrow getFitness(permutation)$
            // get the reversed version's fitness value
            $newFitnnes \leftarrow getFitness(reversed)$

            // compare fitness of reversed with original
            if $newFitness > orgFitness$ then
                // assign permutation to reversed version
                $permutation \leftarrow reversedPermutation$
                // set the new fitness value
                $computeFitness(permutation)$
                $improvedFitness = true$
                $improvementCount + +$
            end if
            // stop after 5 fitness improvement
            if $imporvementCount == 5$ then
                $improvedFitness \leftarrow false$
                break fitnessLoop
            end if
        end for
    end for
end while
```

tsp.genetic.fileio, *tsp.genetic.simulator* respectively. *tsp.genetic.algorithm* package contains class named TSPGA which implements the genetic algorithm flow as depicted in Fig. 1. *tsp.genetic.animator* package contains class named TSPAnimator which implements shortest path construction animation. *tsp.genetic.context* package contains MutationManager, CrossoverManager, and GAContextManager classes in which MutationManager class provides an implementation of mutation strategies discussed under the section III-A, CrossoverManager class provides an implementation of crossover strategies discussed under the section III-B, and GAContextManager class provides an implementation for subordinate tasks which are population generation, parent selection, 2-opt local search, and creation of distance matrix. *tsp.genetic.entities* package contains Chromosome and Point classes where Chromosome class represents an individual solution to TSP and Point class represents the coordinate of city. *tsp.genetic.fileio* package contains FileIO class to parse csv files to create coordinates to be used in the genetic algorithm. *tsp.genetic.simulator* package contains TSPSimulator class which contains main method and takes parameters which are number of iterations, population size, and boolean flag indicating whether to apply 2-opt local search. Genetic algorithm can be configured by directly changing these parameters.

I have evaluated 150 models for berlin52.tsp and 150 models for att48.tsp datasets from TSPLIB [3]. In total 300 experiments conducted and best performing models are indicated under the section IV.

In the items 79^{th} through 150^{th}, Combination-n refers to a subset of crossover and mutation strategies applied randomly. All the used combinations are defined as follows where c represents crossover strategy and m represents mutation strategy:

1) Combination-1
   $c \in \{Order - 1\ Crossover, Cycle\ Crossover\}$
   and
   $m \in \{Scramble\ Mutation, Inversion\ Mutation\}$
2) Combination-2
   $c \in \{Order - 1\ Crossover, Cycle\ Crossover\}$
   and
   $m \in \{Swap\ Mutation, Inversion\ Mutation\}$
3) Combination-3
   $c \in \{Order - 1\ Crossover, Cycle\ Crossover\}$
   and
   $m \in \{Swap\ Mutation, Scramble\ Mutation\}$
4) Combination-4
   $c \in \{Order - 1\ Crossover, Cycle\ Crossover\}$
   and
   $m \in \{Insert\ Mutation, Inversion\ Mutation\}$
5) Combination-5
   $c \in \{Order - 1\ Crossover, Cycle\ Crossover\}$
   and
   $m \in \{Swap\ Mutation, Insert\ Mutation\}$
6) Combination-6
   $c \in \{Order - 1\ Crossover, Cycle\ Crossover\}$
   and
   $m \in \{Insert\ Mutation, Scramble\ Mutation\}$
7) Combination-7
   $c \in \{Order{-}1\ Crossover, Partially\ Mapped\ Crossover\}$
   and
   $m \in \{Scramble\ Mutation, Inversion\ Mutation\}$
8) Combination-8
   $c \in \{Order{-}1\ Crossover, Partially\ Mapped\ Crossover\}$
   and
   $m \in \{Swap\ Mutation, Inversion\ Mutation\}$

9) Combination-9
$c \in \{Order{-}1\ Crossover, Partially\ Mapped\ Crossover\}$
and
$m \in \{Swap\ Mutation, Scramble\ Mutation\}$
10) Combination-10
$c \in \{Order{-}1\ Crossover, Partially\ Mapped\ Crossover\}$
and
$m \in \{Insert\ Mutation, Inversion\ Mutation\}$
11) Combination-11
$c \in \{Order{-}1\ Crossover, Partially\ Mapped\ Crossover\}$
and
$m \in \{Swap\ Mutation, Insert\ Mutation\}$
12) Combination-12
$c \in \{Order{-}1\ Crossover, Partially\ Mapped\ Crossover\}$
and
$m \in \{Insert\ Mutation, Scramble\ Mutation\}$

All the models used in experiments defined are as follows:

1) Order-1 Crossover and Swap Mutation without 2-opt Local Search with population size = 30
2) Order-1 Crossover and Swap Mutation with 2-opt Local Search with population size = 30
3) Order-1 Crossover and Swap Mutation without 2-opt Local Search with population size = 80
4) Order-1 Crossover and Swap Mutation with 2-opt Local Search with population size = 80
5) Order-1 Crossover and Swap Mutation without 2-opt Local Search with population size = 150
6) Order-1 Crossover and Swap Mutation with 2-opt Local Search with population size = 150
7) Order-1 Crossover and Insert Mutation without 2-opt Local Search with population size = 30
8) Order-1 Crossover and Insert Mutation with 2-opt Local Search with population size = 30
9) Order-1 Crossover and Insert Mutation without 2-opt Local Search with population size = 80
10) Order-1 Crossover and Insert Mutation with 2-opt Local Search with population size = 80
11) Order-1 Crossover and Insert Mutation without 2-opt Local Search with population size = 150
12) Order-1 Crossover and Insert Mutation with 2-opt Local Search with population size = 150
13) Order-1 Crossover and Scramble Mutation without 2-opt Local Search with population size = 30
14) Order-1 Crossover and Scramble Mutation with 2-opt Local Search with population size = 30
15) Order-1 Crossover and Scramble Mutation without 2-opt Local Search with population size = 80
16) Order-1 Crossover and Scramble Mutation with 2-opt Local Search with population size = 80
17) Order-1 Crossover and Scramble Mutation without 2-opt Local Search with population size = 150
18) Order-1 Crossover and Scramble Mutation with 2-opt Local Search with population size = 150
19) Order-1 Crossover and Inversion Mutation without 2-opt Local Search with population size = 30
20) Order-1 Crossover and Inversion Mutation with 2-opt Local Search with population size = 30
21) Order-1 Crossover and Inversion Mutation without 2-opt Local Search with population size = 80
22) Order-1 Crossover and Inversion Mutation with 2-opt Local Search with population size = 80
23) Order-1 Crossover and Inversion Mutation without 2-opt Local Search with population size = 150
24) Order-1 Crossover and Inversion Mutation with 2-opt Local Search with population size = 150
25) Partially Mapped Crossover and Swap Mutation without 2-opt Local Search with population size = 30
26) Partially Mapped Crossover and Swap Mutation with 2-opt Local Search with population size = 30
27) Partially Mapped Crossover and Swap Mutation without 2-opt Local Search with population size = 80
28) Partially Mapped Crossover and Swap Mutation with 2-opt Local Search with population size = 80
29) Partially Mapped Crossover and Swap Mutation without 2-opt Local Search with population size = 150
30) Partially Mapped Crossover and Swap Mutation with 2-opt Local Search with population size = 150
31) Partially Mapped Crossover and Insert Mutation without 2-opt Local Search with population size = 30
32) Partially Mapped Crossover and Insert Mutation with 2-opt Local Search with population size = 30
33) Partially Mapped Crossover and Insert Mutation without 2-opt Local Search with population size = 80
34) Partially Mapped Crossover and Insert Mutation with 2-opt Local Search with population size = 80
35) Partially Mapped Crossover and Insert Mutation without 2-opt Local Search with population size = 150
36) Partially Mapped Crossover and Insert Mutation with 2-opt Local Search with population size = 150
37) Partially Mapped Crossover and Scramble Mutation without 2-opt Local Search with population size = 30
38) Partially Mapped Crossover and Scramble Mutation with 2-opt Local Search with population size = 30
39) Partially Mapped Crossover and Scramble Mutation without 2-opt Local Search with population size = 80
40) Partially Mapped Crossover and Scramble Mutation with 2-opt Local Search with population size = 80
41) Partially Mapped Crossover and Scramble Mutation without 2-opt Local Search with population size = 150
42) Partially Mapped Crossover and Scramble Mutation with 2-opt Local Search with population size = 150
43) Partially Mapped Crossover and Inversion Mutation without 2-opt Local Search with population size = 30
44) Partially Mapped Crossover and Inversion Mutation with 2-opt Local Search with population size = 30
45) Partially Mapped Crossover and Inversion Mutation without 2-opt Local Search with population size = 80
46) Partially Mapped Crossover and Inversion Mutation with 2-opt Local Search with population size = 80
47) Partially Mapped Crossover and Inversion Mutation without 2-opt Local Search with population size = 150

48) Partially Mapped Crossover and Inversion Mutation with 2-opt Local Search with population size = 150
49) Cycle Crossover and Swap Mutation without 2-opt Local Search with population size = 30
50) Cycle Crossover and Swap Mutation with 2-opt Local Search with population size = 30
51) Cycle Crossover and Swap Mutation without 2-opt Local Search with population size = 80
52) Cycle Crossover and Swap Mutation with 2-opt Local Search with population size = 80
53) Cycle Crossover and Swap Mutation without 2-opt Local Search with population size = 150
54) Cycle Crossover and Swap Mutation with 2-opt Local Search with population size = 150
55) Cycle Crossover and Insert Mutation without 2-opt Local Search with population size = 30
56) Cycle Crossover and Insert Mutation with 2-opt Local Search with population size = 30
57) Cycle Crossover and Insert Mutation without 2-opt Local Search with population size = 80
58) Cycle Crossover and Insert Mutation with 2-opt Local Search with population size = 80
59) Cycle Crossover and Insert Mutation without 2-opt Local Search with population size = 150
60) Cycle Crossover and Insert Mutation with 2-opt Local Search with population size = 150
61) Cycle Crossover and Scramble Mutation without 2-opt Local Search with population size = 30
62) Cycle Crossover and Scramble Mutation with 2-opt Local Search with population size = 30
63) Cycle Crossover and Scramble Mutation without 2-opt Local Search with population size = 80
64) Cycle Crossover and Scramble Mutation with 2-opt Local Search with population size = 80
65) Cycle Crossover and Scramble Mutation without 2-opt Local Search with population size = 150
66) Cycle Crossover and Scramble Mutation with 2-opt Local Search with population size = 150
67) Cycle Crossover and Inversion Mutation without 2-opt Local Search with population size = 30
68) Cycle Crossover and Inversion Mutation with 2-opt Local Search with population size = 30
69) Cycle Crossover and Inversion Mutation without 2-opt Local Search with population size = 80
70) Cycle Crossover and Inversion Mutation with 2-opt Local Search with population size = 80
71) Cycle Crossover and Inversion Mutation without 2-opt Local Search with population size = 150
72) Cycle Crossover and Inversion Mutation with 2-opt Local Search with population size = 150
73) Random Crossover and Random Mutation without 2-opt Local Search with population size = 30
74) Random Crossover and Random Mutation with 2-opt Local Search with population size = 30
75) Random Crossover and Random Mutation without 2-opt Local Search with population size = 80
76) Random Crossover and Random Mutation with 2-opt Local Search with population size = 80
77) Random Crossover and Random Mutation without 2-opt Local Search with population size = 150
78) Random Crossover and Random Mutation with 2-opt Local Search with population size = 150
79) Combination-1 without 2-opt Local Search with population size = 30
80) Combination-1 with 2-opt Local Search with population size = 30
81) Combination-1 without 2-opt Local Search with population size = 80
82) Combination-1 with 2-opt Local Search with population size = 80
83) Combination-1 without 2-opt Local Search with population size = 150
84) Combination-1 with 2-opt Local Search with population size = 150
85) Combination-2 without 2-opt Local Search with population size = 30
86) Combination-2 with 2-opt Local Search with population size = 30
87) Combination-2 without 2-opt Local Search with population size = 80
88) Combination-2 with 2-opt Local Search with population size = 80
89) Combination-2 without 2-opt Local Search with population size = 150
90) Combination-2 with 2-opt Local Search with population size = 150
91) Combination-3 without 2-opt Local Search with population size = 30
92) Combination-3 with 2-opt Local Search with population size = 30
93) Combination-3 without 2-opt Local Search with population size = 80
94) Combination-3 with 2-opt Local Search with population size = 80
95) Combination-3 without 2-opt Local Search with population size = 150
96) Combination-3 with 2-opt Local Search with population size = 150
97) Combination-4 without 2-opt Local Search with population size = 30
98) Combination-4 with 2-opt Local Search with population size = 30
99) Combination-4 without 2-opt Local Search with population size = 80
100) Combination-4 with 2-opt Local Search with population size = 80
101) Combination-4 without 2-opt Local Search with population size = 150
102) Combination-4 with 2-opt Local Search with population size = 150
103) Combination-5 without 2-opt Local Search with population size = 30

104) Combination-5 with 2-opt Local Search with population size = 30
105) Combination-5 without 2-opt Local Search with population size = 80
106) Combination-5 with 2-opt Local Search with population size = 80
107) Combination-5 without 2-opt Local Search with population size = 150
108) Combination-5 with 2-opt Local Search with population size = 150
109) Combination-6 without 2-opt Local Search with population size = 30
110) Combination-6 with 2-opt Local Search with population size = 30
111) Combination-6 without 2-opt Local Search with population size = 80
112) Combination-6 with 2-opt Local Search with population size = 80
113) Combination-6 without 2-opt Local Search with population size = 150
114) Combination-6 with 2-opt Local Search with population size = 150
115) Combination-7 without 2-opt Local Search with population size = 30
116) Combination-7 with 2-opt Local Search with population size = 30
117) Combination-7 without 2-opt Local Search with population size = 80
118) Combination-7 with 2-opt Local Search with population size = 80
119) Combination-7 without 2-opt Local Search with population size = 150
120) Combination-7 with 2-opt Local Search with population size = 150
121) Combination-8 without 2-opt Local Search with population size = 30
122) Combination-8 with 2-opt Local Search with population size = 30
123) Combination-8 without 2-opt Local Search with population size = 80
124) Combination-8 with 2-opt Local Search with population size = 80
125) Combination-8 without 2-opt Local Search with population size = 150
126) Combination-8 with 2-opt Local Search with population size = 150
127) Combination-9 without 2-opt Local Search with population size = 30
128) Combination-9 with 2-opt Local Search with population size = 30
129) Combination-9 without 2-opt Local Search with population size = 80
130) Combination-9 with 2-opt Local Search with population size = 80
131) Combination-9 without 2-opt Local Search with population size = 150
132) Combination-9 with 2-opt Local Search with population size = 150
133) Combination-10 without 2-opt Local Search with population size = 30
134) Combination-10 with 2-opt Local Search with population size = 30
135) Combination-10 without 2-opt Local Search with population size = 80
136) Combination-10 with 2-opt Local Search with population size = 80
137) Combination-10 without 2-opt Local Search with population size = 150
138) Combination-10 with 2-opt Local Search with population size = 150
139) Combination-11 without 2-opt Local Search with population size = 30
140) Combination-11 with 2-opt Local Search with population size = 30
141) Combination-11 without 2-opt Local Search with population size = 80
142) Combination-11 with 2-opt Local Search with population size = 80
143) Combination-11 without 2-opt Local Search with population size = 150
144) Combination-11 with 2-opt Local Search with population size = 150
145) Combination-12 without 2-opt Local Search with population size = 30
146) Combination-12 with 2-opt Local Search with population size = 30
147) Combination-12 without 2-opt Local Search with population size = 80
148) Combination-12 with 2-opt Local Search with population size = 80
149) Combination-12 without 2-opt Local Search with population size = 150
150) Combination-12 with 2-opt Local Search with population size = 150

IV. Experimental Results

In this section, 150 experiment results for each data set are provided which are berlin52.tsp and att48.tsp and the best performing models for each configuration are highlighted in bold. The best known solutions for **berlin52.tsp** and **att48.tsp** are **7542** and **10628**. The results for the att48.tsp data set without the 2-opt local search are given in table I and with the 2-opt local search in table II, respectively. The results for the berlin52.tsp data set without 2-opt local search are given in Table III and with 2-opt local search in Table IV, respectively. Comparison of experimental results for the att48.tsp data set with population sizes 30, 80, and 150 are given in Fig. 10 and Fig. 11 and for the berlin52.tsp data set with population sizes 30, 80, and 150 are given in Fig. 12 and Fig. 13.

Experimental results carried out with 500 iterations when the 2-opt heuristic local search algorithm is not applied and 50 iterations when the 2-opt heuristic local search algorithm

TABLE I: att48.tsp Experimental Results without 2-opt Heuristic Local Search

Model	**Pop. Size 30**	**Pop. Size 80**	**Pop. Size 150**
Model-{1-3-5}	13615.51	12250.56	11671.00
Model-{7-9-11}	13602.94	12239.94	11844.44
Model-{13-15-17}	15730.11	12807.74	11907.26
Model-{**19**-21-23}	**11582.37**	11121.03	10986.05
Model-{25-27-29}	16047.70	14528.86	14173.30
Model-{31-33-35}	15683.23	14077.52	13149.35
Model-{37-39-41}	20174.83	17110.32	15559.87
Model-{43-45-47}	11679.08	11186.76	11131.26
Model-{49-51-53}	16885.93	15639.97	14999.31
Model-{55-57-59}	16314.05	15023.28	14667.83
Model-{61-63-65}	22518.67	20805.10	19635.78
Model-{67-69-71}	11987.61	11275.67	11164.55
Model-{73-75-77}	12291.19	11310.09	11066.80
Model-{79-81-83}	12057.41	11260.85	11103.36
Model-{85-87-89}	11872.83	11191.04	11059.39
Model-{91-93-95}	15978.64	13387.30	12544.06
Model-{97-99-**101**}	11846.71	11097.71	**10953.84**
Model-{103-105-107}	14833.06	13049.97	12256.02
Model-{109-111-113}	15291.72	13289.76	12370.55
Model-{115-117-119}	11915.65	11275.26	11064.56
Model-{121-123-125}	11952.38	11205.81	11072.44
Model-{127-129-131}	15089.76	12723.44	11845.32
Model-{133-**135**-137}	11839.17	**11078.27**	10967.31
Model-{139-141-143}	14057.34	12411.70	11685.31
Model-{145-147-149}	14523.31	12557.09	11736.92

TABLE II: att48.tsp Experimental Results with 2-opt Heuristic Local Search

Model	**Pop. Size 30**	**Pop. Size 80**	**Pop. Size 150**
Model-{2-4-6}	10799.27	10713.51	10681.92
Model-{8-10-12}	10814.42	10713.47	10675.07
Model-{14-16-18}	10862.77	10742.99	10707.15
Model-{20-22-**24**}	10800.17	10771.00	**10669.92**
Model-{26-28-30}	11080.07	10887.69	10882.14
Model-{32-34-36}	11081.16	10901.10	10808.74
Model-{38-40-42}	11222.99	11002.20	10942.75
Model-{44-46-48}	10959.35	10847.69	10789.70
Model-{50-52-54}	10954.30	10829.97	10750.97
Model-{56-58-60}	11268.44	11047.46	10916.44
Model-{62-64-66}	11357.17	11095.88	10966.34
Model-{68-70-72}	10925.51	10814.64	10751.94
Model-{74-76-78}	10818.17	10725.22	10682.94
Model-{80-82-84}	10832.73	10726.46	10687.60
Model-{86-88-90}	10816.01	10714.20	10679.29
Model-{92-94-96}	10813.34	10723.78	10682.14
Model-{**98-100**-102}	**10789.55**	**10703.30**	10678.73
Model-{104-106-108}	10794.27	10713.10	10685.32
Model-{110-112-114}	10816.59	10712.63	10684.62
Model-{116-118-120}	10945.78	10807.80	10748.07
Model-{122-124-126}	10862.05	10801.31	10726.90
Model-{128-130-132}	10957.84	10831.51	10795.30
Model-{134-136-138}	10871.92	10776.03	10721.81
Model-{140-142-144}	10916.81	10805.53	10757.77
Model-{146-148-150}	10860.69	10782.21	10766.81

TABLE III: berlin52.tsp Experimental Results without 2-opt Heuristic Local Search

Model	**Pop. Size 30**	**Pop. Size 80**	**Pop. Size 150**
Model-{1-3-5}	10010.15	8944.99	8509.35
Model-{7-9-11}	10019.37	8951.27	8566.92
Model-{13-15-17}	11570.49	9309.72	8675.53
Model-{**19-21**-23}	**8608.90**	**8193.95**	8084.41
Model-{25-27-29}	11133.45	10467.37	9950.16
Model-{31-33-35}	11024.53	9950.24	9542.71
Model-{37-39-41}	14321.13	12406.80	10989.53
Model-{43-45-47}	8787.87	8310.0	8214.17
Model-{49-51-53}	11642.39	10851.48	10632.33
Model-{55-57-59}	11657.20	10937.30	10509.75
Model-{61-63-65}	15902.94	14940.95	14253.14
Model-{67-69-71}	9039.42	8427.64	8226.46
Model-{73-75-77}	9087.82	8437.10	8232.75
Model-{79-81-83}	9078.88	8364.73	8147.96
Model-{85-87-89}	8951.10	8336.01	8225.35
Model-{91-93-95}	11200.34	10515.35	9251.53
Model-{97-99-**101**}	8900.39	8271.43	**8071.93**
Model-{103-105-107}	10638.06	9360.96	9003.68
Model-{109-111-113}	11347.74	9719.47	9153.94
Model-{115-117-119}	9045.83	8301.52	8158.29
Model-{121-123-125}	8936.82	8336.21	8156.06
Model-{127-129-131}	10688.02	9180.19	8803.27
Model-{133-135-137}	10193.44	8252.87	8101.60
Model-{139-141-143}	10118.63	9069.21	8659.35
Model-{145-147-149}	10397.52	9074.25	8727.39

TABLE IV: berlin52.tsp Experimental Results with 2-opt Heuristic Local Search

Model	**Pop. Size 30**	**Pop. Size 80**	**Pop. Size 150**
Model-{2-4-6}	7921.73	7746.16	7656.49
Model-{8-10-12}	7845.55	7674.15	7613.19
Model-{14-16-18}	7955.94	7802.24	7667.32
Model-{20-22-24}	7822.25	7659.44	7591.41
Model-{26-28-30}	8171.07	8008.30	7952.11
Model-{32-34-36}	8130.68	7957.10	7852.37
Model-{38-40-42}	8197.94	8145.92	8090.67
Model-{44-46-48}	8112.92	7898.33	7810.75
Model-{50-52-54}	8010.54	7837.83	7740.62
Model-{56-58-60}	8143.76	8037.82	7950.89
Model-{62-64-66}	8191.70	8102.89	7998.84
Model-{68-70-72}	7941.27	7802.95	7719.04
Model-{74-76-78}	7838.47	7663.50	7609.85
Model-{80-82-84}	7807.37	7646.49	7616.75
Model-{86-88-90}	7864.30	7629.88	7577.95
Model-{92-94-96}	7847.99	7742.52	7594.44
Model-{**98-100**-102}	**7771.57**	**7625.70**	7581.09
Model-{104-106-108}	7792.41	7662.67	7580.17
Model-{110-112-**114**}	7822.27	7639.69	**7576.70**
Model-{116-118-120}	7982.68	7857.23	7808.09
Model-{122-124-126}	7991.38	7836.06	7735.67
Model-{128-130-132}	8055.44	7977.73	7860.26
Model-{134-136-138}	7921.98	7802.55	7711.26
Model-{140-142-144}	8021.55	7854.05	7787.65
Model-{146-148-150}	8050.83	7908.04	7761.99

is applied for both data sets. According to results, the best performing models for the att48.tsp data set without 2-opt heuristic local search for populations sizes 30, 80, 150 are model-19, model-135, and model-101. Best performing models for att48.tsp data set with 2-opt heuristic local search for populations sizes 30, 80, 150 are model-98, model-100, and model-24. Best performing models for berlin52.tsp data set without 2-opt heuristic local search for populations sizes 30, 80, 150 are model-19, model-21, and model-101. Best performing models for berlin52.tsp data set with 2-opt heuristic local search for populations sizes 30, 80, 150 are model-98, model-100, and model-114.

Among the 12 best performing models, all 12 experiments use order-1 crossover, 8 experiments use cycle crossover, 11

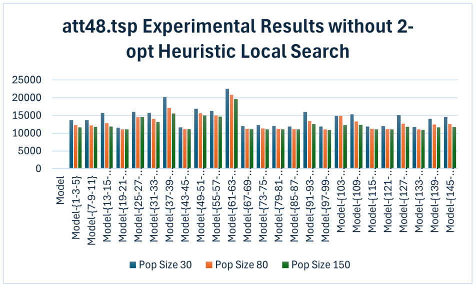

Fig. 10: Comparison of Experimental Results for att48.tsp dataset without 2-opt heuristic local search with population sizes 30, 80, and 150

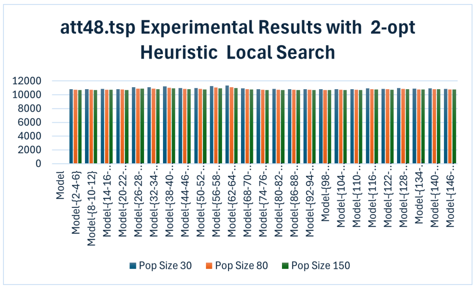

Fig. 11: Comparison of Experimental Results for att48.tsp dataset with 2-opt heuristic local search with population sizes 30, 80, and 150

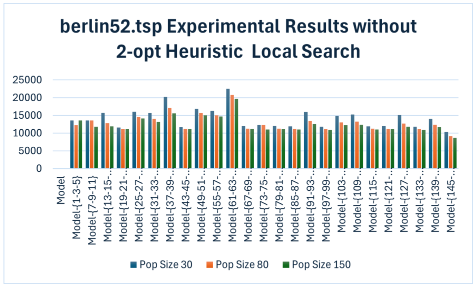

Fig. 12: Comparison of Experimental Results for berlin52.tsp dataset without 2-opt heuristic local search with population sizes 30, 80, and 150

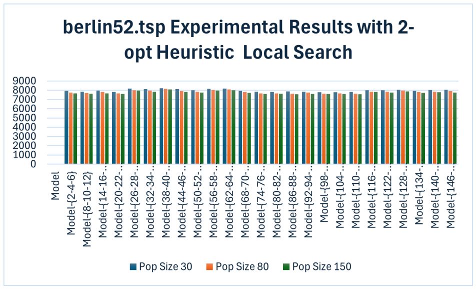

Fig. 13: Comparison of Experimental Results for berlin52.tsp dataset with 2-opt heuristic local search with population sizes 30, 80, and 150

experiments use inversion mutation, 7 experiments use insert mutation, and only 1 experiment uses scramble mutation. As results indicate, **order-1 crossover** and **inversion mutation** in combination **with 2-opt heuristic local search** give most promising results in most experiments.

V. Conclusion

I have carried out 150 experiments for two well-known datasets att48.tsp and berlin52.tsp each from TSPLIB [3] on the genetic algorithm implementation of TSP problem with 2-opt heuristic local search. Experimental results have shown that among 12 best performing models order-1 crossover and inversion mutation gave the best results in majority of the cases. Also Fig. 10, Fig. 11, Fig. 12, and Fig. 13 given under the section IV has shown that increase in population size where population size $\in \{30, 80, 150\}$ resulted in better results. Also it is clearly observed by comparing table I and table II for att48.tsp data set; table III and table IV for berlin52.tsp data set that genetic algorithm combined with 2-opt heuristic local search resulted in values that are much closer to best known solutions compared to versions that does not perform 2-opt heuristic local search. Another realistic evaluation approach that this study brings is that application of genetic algorithm with 2-opt heuristic local search if not restricted with certain number of fitness improvements easily reaches best known solutions in the implementation provided under section III footnote. Whereas, in real-life scenarios with very large search spaces usually it is not possible to perform full search with 2-opt algorithm, so I have restricted 2-opt heuristic local search algorithm to execute until reaching 5 fitness improvements and not doing full search which greatly reduces execution time. This study indicated that we can improve solutions provided by genetic algorithms combined with heuristic techniques further by selecting specific subset of combinations of crossover and mutation strategies along with populations sizes. This fine-tuning procedure requires great amount of experimentation and comparison among different models, so as a future work, simulator software provided in section III footnote can be extended to cover different crossover and mutation strategies with heuristic search algorithms to discover better ensembles via experimentation.

References

[1] O. Mersmann, B. Bischl, J. Bossek, H. Trautmann, M. Wagner, and F. Neumann, "Local search and the traveling salesman problem: A feature-based characterization of problem hardness," in *Learning and Intelligent Optimization*, Y. Hamadi and M. Schoenauer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 115–129.

[2] S. S. Juneja, P. Saraswat, K. Singh, J. Sharma, R. Majumdar, and S. Chowdhary, "Travelling salesman problem optimization using genetic algorithm," in *2019 Amity International Conference on Artificial Intelligence (AICAI)*, 2019, pp. 264–268.

[3] G. Reinelt, "Tsplib—a traveling salesman problem library," *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376–384, 1991. [Online]. Available: https://doi.org/10.1287/ijoc.3.4.376

[4] J. R. Sampson, "Adaptation in natural and artificial systems (john h. holland)," *SIAM Review*, vol. 18, no. 3, pp. 529–530, 1976. [Online]. Available: https://doi.org/10.1137/1018105

[5] S. W. Mahfoud, "Population sizing for sharing methods," *Foundations of Genetic Algorithms*, vol. 3, 1994.

[6] L. D. Chambers, *The practical handbook of genetic algorithms: applications*. Chapman and Hall/CRC, 2000.

[7] K. Man, K. Tang, and S. Kwong, "Genetic algorithms: concepts and applications [in engineering design]," *IEEE Transactions on Industrial Electronics*, vol. 43, no. 5, pp. 519–534, 1996.

[8] K. A. De Jong and W. M. Spears, "An analysis of the interacting roles of population size and crossover in genetic algorithms," in *International Conference on Parallel Problem Solving from Nature*. Springer, 1990, pp. 38–47.

[9] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Transactions on systems, man, and cybernetics*, vol. 16, no. 1, pp. 122–128, 2007.

[10] P. Adewole, A. Akinwale, and K. Otubamowo, "A genetic algorithm for solving travelling salesman problem," *International Journal of Advanced Computer Sciences and Applications*, vol. 2, 01 2011.

[11] A. Uğur, S. Korukoğlu, A. Çalıskan, M. Cinsdikici, and A. Alp, "Genetic algorithm based solution for tsp on a sphere," *Mathematical and Computational Applications*, vol. 14, no. 3, pp. 219–228, 2009. [Online]. Available: https://www.mdpi.com/2297-8747/14/3/219

[12] Z.-J. Lee, "A hybrid algorithm applied to travelling salesman problem," in *IEEE International Conference on Networking, Sensing and Control, 2004*, vol. 1, 2004, pp. 237–242 Vol.1.

[13] M. Dorigo and L. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997.

[14] B. Freisleben and P. Merz, "New genetic local search operators for the traveling salesman problem," in *Parallel Problem Solving from Nature — PPSN IV*, H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 890–899.

[15] ——, "A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems," in *Proceedings of IEEE International Conference on Evolutionary Computation*, 1996, pp. 616–621.

[16] K. D. Boese, *Cost versus distance in the traveling salesman problem*. Citeseer, 1995.

[17] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations research*, vol. 21, no. 2, pp. 498–516, 1973.

[18] R. Gan, Q. Guo, H. Chang, and Y. Yi, "Improved ant colony optimization algorithm for the traveling salesman problems," *Journal of Systems Engineering and Electronics*, vol. 21, no. 2, pp. 329–333, 2010.

[19] T. Stutzle and H. Hoos, "Max-min ant system and local search for the traveling salesman problem," in *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97)*, 1997, pp. 309–314.

[20] E. Fejzagić and A. Oputić, "Performance comparison of sequential and parallel execution of the ant colony optimization algorithm for solving the traveling salesman problem," in *2013 36th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2013, pp. 1301–1305.

[21] J. Collings and E. Kim, "A distributed and decentralized approach for ant colony optimization with fuzzy parameter adaptation in traveling salesman problem," in *2014 IEEE Symposium on Swarm Intelligence*, 2014, pp. 1–9.

[22] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future," *Multimedia tools and applications*, vol. 80, pp. 8091–8126, 2021.

[23] K. Deep and H. Mebrathu, "Combined mutation operators of genetic algorithm for the travelling salesman problem," *IJCOPI*, vol. 2, pp. 2–24, 01 2011.

[24] M. Dordevic, M. Tuba, and B. Djordjevic, "Impact of grafting a 2-opt algorithm based local searcher into the genetic algorithm," 08 2009, pp. 485–490.

[25] A. Saxena and J. Pandey, "Review of crossover techniques for genetic algorithms," pp. 2394–9333, 08 2019.

[26] D. S. Johnson and L. A. McGeoch, "The traveling salesman problem: A case study in local optimization," 2008. [Online]. Available: https://api.semanticscholar.org/CorpusID:208903251