An Investigative Study of Genetic Algorithms to Solve the DNA Assembly Optimization Problem

Hachemi Bennaceur [1], Meznah Almutairy[2], Nora Alqhtani[3]
Faculty of Computer and Information Sciences, Computer Science Department
Al Imam Mohammad Ibn Saud Islamic University (IMSIU)
Riyadh, Saudi Arabia

***Abstract*—This paper aims to highlight the motivations for investigating genetic algorithms to solve the DNA Fragments Assembly problem (DNA_FA). DNA_FA is an optimization problem that attempts to reconstruct the original DNA sequence by finding the shortest DNA sequence from a given set of fragments. We showed that the DNA_FA optimization problem is a special case of the two well-known optimization problems: The Traveling Salesman Problem (TSP) and the Quadratic Assignment Problem (QAP). TSP and QAP are important problems in the field of combinatorial optimization and for which there exists an abundant literature. Genetic Algorithms (GA) applied to these problems have led to very satisfactory results in practice. In the perspective of designing efficient genetic algorithms to solve DNA_FA we showed the existence of a polynomial-time reduction of DNA-FA into TSP and QAP enabling us to point out some technical similarities in terms of solutions and search space complexity. We then conceptually designed a genetic algorithm platform for solving the DNA-FA problem inspired from the existing efficient genetic algorithms in the literature solving TSP and QAP problems. This platform offers several ingredients enabling us to create several variants of GA solvers for the DNA assembly optimization problems.**

Keywords*—Genetic Algorithms; Traveling Salesman Problem; Quadratic Assignment Problem; DNA fragments assembly problem

I. INTRODUCTION

The DNA fragment assembly problem attempts to reconstruct the original DNA sequence by finding the shortest DNA sequence from a large number of fragments [1]. DNA_FA is a hard optimization problem due to its high complexity, the larger the sequence, the larger the fragments set input and consequently the higher and harder computation [2]. Metaheuristics have been shown to be the best alternative techniques to solve this problem. Especially, the DNA_FA problem was tackled with some metaheuristic algorithms such as Genetic Algorithms [3], Tabu Search [4], Simulated Annealing [5], Particle Swarm Optimization [6], and Ant Colony Optimization [7].

The famous Traveling Salesman Problem (TSP) and the Quadratic Assignment Problem (QAP) are two important problems in the field of combinatorial optimization and for which there exists an abundant literature. Genetic Algorithms applied to these problems have led to very satisfactory results in practice [8],[9],[10].

Our main contributions are: first, we presented a formal proof of the existence of a polynomial-time reduction of DNA_FA into TSP and QAP enabling us to point out some technical similarities in terms of solutions and search space complexity. Particularly, we have theoretically demonstrated that all these three optimization problems have a similar topological structure and they need to explore a search space of solutions with a same complexity to find an optimal solution. Notice that although many works mentioned the relationship between DNA_FA with the TSP problem, but to the best of our knowledge none provided a formal demonstration enabling to take advantages from the existent solvers of TSP and QAP problems to treat efficiently the DNA_FA problem. For this purpose, we revisited the relationship and the similarities between DNA_FA and TSP and established a new relationship and similarities between DNA_FA and QAP. Second, as the TSP and QAP problems were solved efficiently using GA algorithms, we believe it is worth to exploit these similarities in order to deeply investigate the use of GA algorithms for solving the DNA_FA problem. Based on these facts, we proposed a genetic algorithm platform including main GA concepts and tools (selection, crossover, mutation, …) enabling us to design several variants of GA solvers for the DNA_FA. The platform regroups the best GA concepts inspired from the existing efficient genetic algorithms in the literature solving TSP and QAP problems such that when they are used synergistically, we lead to efficient GA solvers. Few research works based on GA have been developed to solve the DNA_FA problem; the more recent GA algorithm used a basic schema with traditional simple GA concepts [11]. We think there is ample room for improvements by integrating the more recent advanced GA concepts used for solving TSP and QAP problems.

The remaining sections of the paper are structured as follows. The related work is discussed in Section II. Section III presents the DNA_FA, TSP, and QAP problems formally and provides formal proof of the equivalence between these three problems. In Section IV the existing GA solutions to solve the DNA_FA problem, TSP, and QAP are presented and discuss these solutions in Section V. Finally, we proposed, in section VI a GA platform for designing efficient GA solvers for the DNA_FA problem.

II. RELATED WORK

Through research in the previous literature, we found some studies that indicated a relationship between TSP and DNA_FA problem [12],[13],[14]. In 1995 Parsons et al. [12]

noticed the similarity between DNA_FA and TSP but they argued that the mapping is not easy for some issues; some of these issues are (i) the DNA_FA problem is a maximization problem where the TSP is a minimization problem, and (ii) the TSP seeks to find a Hamiltonian circuit where the DNA_FA seeks to find a Hamiltonian path. In 2013 Mallén and Fernández [15] tried to overcome these issues by adding a dummy city with zero distance to all the other cities to form an open Hamiltonian path instead of a Hamiltonian circuit. Also, they multiplied the objective function by (-1) to convert the maximization into minimization.

There are many studies mentioned the relationship between TSP and QAP [16],[17],[18],[19]. However, no study has formally shown the relationship between the DNA_FA problem with the TSP and QAP problems. Moreover, no study has explored the numerous similarities between the DNA_FA problem with the TSP and QAP problems to take advantage of the many efficient techniques developed for these two problems in order to solve the DNA_FA problem.

Few genetic algorithms have been developed to solve the DNA_FA problem (see section IV for more details), compared to what have been developed for the TSP and QAP problems [9], [10], [20],[21], [22]. Moreover, these GA algorithms did not exploit the recent progress performed in the context of TSP and QAP problems. For instance, if we consider the crossover operator which is the main concept of GA algorithms, recently many modern crossover types (e.g. the Sequential Constructive Crossover) have been designed and successively tested for the TSP and QAP problems that have yielded to better performances. Due to the numerous similarities with the TSP and QAP problems shown in this paper, we believe and expect that well-designed GA solvers inspired from existing GA for these two problems would produce similar performance for DNA_FA problem. Unfortunately, no work investigated deeply these similarities in order to exploit the more advanced genetic algorithms designed for TSP and QAP for solving the DNA_FA problem. For this purpose, it is worth to investigate how these advanced operators originally designed for the TSP and QAP behave when they are applied to the DNA_FA problem.

III. DNA_FA versus TSP and QAP

In this section we describe the optimization problems DNA_FA, TSP and QAP, then we provide formal proofs showing the existence of polynomial-time reductions of the DNA_FA problem into the TSP and QAP problems respectively.

A. DNA_FA Problem

Given a set of DNA fragments drown from a finite alphabet {A, T, C, G}, Adenine (A), Thymine (T), Guanine (G), and Cytosine (C). The goal of the DNA_FA problem is to find the shortest superstring sequence covering all fragments that is a superset of all input fragments. Intuitively, the goal is to find an optimal permutation of the fragments maximizing the number of overlaps between the fragments.

Formally, given a set of n fragments $F = \{f_1, f_2, \dots f_n\}$, drown from a finite alphabet Σ=$\{A,C,G,T\}$, the problem consists in finding an optimal permutation of the fragments $F' =< f_1', f_2', \dots, f_n' >$ that maximizes the number of overlaps between every pair of two consecutive fragments and thus minimizes the length of F'.

$$MAX_{<f_1',f_2',\dots,f_n'>}\left(\sum overlapping\ (f_i', f_{i+1}')\right)$$
$$where\ 1 \le i \le n-1) \quad (1)$$

B. TSP Problem

Given a set of *n* cities along with the distance information between every pair of those cities, the goal of the Travelling Salesman Problem is to find the shortest tour that visits all cities once and returns to the starting city.

The TSP problem can be modeled as follows. Let $< x_1, x_2, \dots, x_n >$ refers to a tour, the goal is to find the lowest cost tour such that.

$$MIN_{<x_1,x_2,\dots,x_n>} \sum_{i=1}^{n-1} d_{(i)(i+1)} + d_{(n)(1)} \quad (2)$$

where $d_{(i)(i+1)}$ is the distance between city x_i and city x_{i+1}, and $d_{(n)(1)}$ is the distance between city x_n and the starting city x_1.

C. QAP Problem

Given two sets of equal size "facilities" and "locations", for each pair of locations, a distance is specified and for each pair of facilities a weight or a flow is specified. The problem is to assign all facilities to different locations with the goal is to minimize the total distances weighted by the corresponding flows.

Formally, given two sets P ("facilities") and L ("locations"), with a weight function w: P × P → R and a distance function d: L × L → R. The goal is to find the assignment f : P → L such that the cost function (the distance multiplied by the weight) is minimized:

$$MIN_f \sum_{a,b\in P} w(a,b) \times d\big(f(a), f(b)\big) \quad (3)$$

{\displaystyle\sum_{a,b\in P}w(a,b)\cdot d(f(a),f(b))}

D. Polynomial Reduction of DNA_FA into TSP and QAP

Notice that although many works mentioned the relationship between DNA_FA with the TSP problem [12], [13], [14], but to the best of our knowledge none provided a formal demonstration enabling to take advantages from the existent solvers of TSP and QAP problems to treat efficiently the DNA_FA problem. For this purpose, we revisited the relationship and the similarities between DNA_FA and TSP presented in [15] and established a new relationship and similarities between DNA_FA and QAP. Namely, we showed that DNA_FA is a special case of TSP and QAP problems.

The DNA_FA problem can be represented as a directed complete weighted graph $G = (V, E)$ where V represents the set of fragments and E is the set of edges. The weight of an arc of G expresses the overlap cost between the two corresponding fragments and it is set to zero if there is no overlap. An optimal solution of DNA_FA corresponds to a Hamiltonian path in G. DNA_FA can be easily transformed into a TSP problem. Let's consider $G' = (V \cup \{s\}, E \cup E')$ the complete graph G of DNA_FA augmented by the virtual

vertex s connected to every vertex of V. For each vertex v of V, two arcs $(s,v) \in E'$ and $(v,s) \in E'$ with zero weights are added to G. We can easily see that the problem of finding a Hamiltonian path in G amounts to find an optimal solution to the TSP represented by the graph G'. Hence, DNA_FA is a special case of the TSP problem.

Regarding the QAP, we established a new relationship between the DNA_FA problem and the QAP, we showed that DNA_FA is a special case of QAP problem as follows. Let' us consider the set of fragments of the problem as a set of *n* facilities and the possible positions of each fragment in the solution sequence as a set of *n* locations. Taking into account the distance between a pair of locations as the overlap cost between their corresponding fragments, and set the flow equal to 1 for every pair of facilities, we can see that finding the optimal permutation assigning to each location exactly one facility so as to minimize the total cost (the distances multiplied by the weight) amounts to find an optimal solution to the associate DNA_FA problem. Thus, clearly DNA_FA is a special case of the QAP problem.

As the famous TSP and QAP problems are NP-Hard [23], it follows automatically from this demonstration DNA_FA is also NP-hard.

In the perspective of designing an effective and robust genetic algorithm platform for solving DNA_FA, Table I. points out some technical similarities in order to get out the most profit from the efficient genetic algorithms designed to solve TSP and QAP problems.

From Table I, we can see that the three optimization problems need to explore a same size search space to find an optimal solution.

TABLE I. TECHNICAL SIMILARITIES BETWEEN DNA-FA, TSP, AND QAP

Characteristics	**DNA_FA**	**TSP**	**QAP**
Variables	Fragments	Cities	Facilities, locations
Constraints	Every fragment must be present in the final sequence. No duplicate fragment is allowed.	Every city should be present in the tour. No duplicate city is allowed.	Every facility should be assigned to one location. No duplicate facility or location is allowed.
Objective Function	Find an optimal permutation of the fragments in which the length of the sequence is minimized.	Find an optimal permutation of cities (an optimal tour) in which the total cost of the tour is minimized.	Find an optimal permutation of facilities (assignment of facilities to locations) in which the cost of the assignment is minimized.
Feasible solution	A sequence that contains all the fragments.	A tour that visits all cities once.	An assignment of each facility to one location
Complexity	NP-hard	NP-hard	NP-hard

IV. GENETIC ALGORITHM (GA) SOLUTIONS

Basic genetic algorithm schema contains various concepts such as population encoding, population initialization, fitness function, selection, crossover, mutation and replacement operators, and stopping conditions.

Each concept has its importance in the algorithm; for instance, encoding the population represents a feasible solution of the problem. Basically, the type of the problem determines the appropriate encoding type (e.g., in the TSP a tour (individual) is usually encoded as a sequence of integers, where each integer represents a city). The initial population may influence the overall behavior of finding solutions. Getting a good initial population can strongly influence the performance of the search. The fitness function is used to evaluate the quality of each individual within the population. The fitness function has to be designed to accurately assess the individual's quality in order to select the best and the fittest individuals for crossover and mutation operations. Selection is the operation where the parents (individuals) are selected from the population according to the fitness function for crossover and mutation operations. The purpose of selection is to ensure that the fitter individuals in the population will be maintained so that the offspring produced has a higher fitness. The crossover operator plays the most crucial role in GA as it aims to explore the huge search space of the problem. Traditionally, it is a binary operation taking two individuals as parents to create new offspring. Recently, the crossover operator has been generalized to take more than two parents to generate new offspring. This type of crossover is called Multi-Parents Crossover (MPX) [24]. The mutation operator aims to ensure diversity in the population by allowing a certain change in the individual, which helps to escape local optima. The replacement allows individuals from the current generation to be replaced by better newly generated offspring. The stopping condition is a vital operator; it is necessary to identify a tradeoff between the algorithm stopping criteria and the algorithm performance.

The fundamental concept behind GAs is inspired by natural evolution, where the GA operators evolve generations of potential solutions of a given problem. More in detail, the GA initiates a population of possible solutions (i.e., individuals) after defining the appropriate solution encoding. Then, calculates the fitness for each individual within the population using the fitness function. The selection operator then selects individuals (parents) based on their fitness to produce a new offspring by carrying out the crossover and mutation operations. The new offspring will inherit the parent's characteristics and will be added to the next generation via a replacement strategy. This process keeps on iterating until the stopping condition is reached, and hence the solution with the best fitness value is returned. The GA procedure is illustrated in Fig. 1.

```
Genetic Algorithm ()
{Initialize the population
Evaluate the population    // fitness function
Generation = 0

While termination condition is not meet
   {
   Generation = Generation +1
    selection operation
    Crossover operation with probability (Pc)
    Mutation operation with probability (Pm)
    Evaluate the population
   }
Return the best individual found
}
```

Fig. 1. GA Procedure.

A. GA solutions for the DNA_FA problem

Here we review the previous works using genetic algorithms for the DNA_FA. As aforementioned, the DNA_FA problem reconstructs the original DNA from a large number of fragments. To illustrate the problem, consider the following example:

An input of five fragments could look like:

TCGG, GCAG, ATCG, CAGC, GATC.

Two possible final sequences involving all input fragments are:

CAGCAGATCGG (length = 11)

GATCGGCAGC (length = 10)

The latter sequence is better since its length is 10, compared to 11 for the former sequence.

In order to deal with this problem, let us introduce the following terms:

Prefix: A substring comprising the first *n* characters of fragment *f*.

Suffix: A substring comprising the last *n* characters of fragment *f*.

Overlap (w): Common sequence between the suffix of one fragment and the prefix of another fragment.

Contig: overlapped fragments without gaps.

By applying the overlapping measure to the above example, we found that **w** (GCAG, CAGC) = 3 whereas **w** (CAGC, GCAG) = 2. This means that these two fragments can be represented in a sequence of 5 letters (4+4-3): GCAGC, which is better the other sequence of length 6: CAGCAG.

The overlaps between fragments can be represented as a directed weighted complete graph. The set V of nodes in this graph corresponds to the set of fragments. A directed edge from fragment *a* to a different fragment *b* with weight $t \geq 0$ exists if the suffix of *a* with *t* characters is a prefix of *b*.

Table II shows the overlap (w) matrix for the aforementioned example, the symbol (-) means there is no edge.

TABLE II. THE OVERLAP MATRIX

w	**TCGG**	**GCAG**	**ATCG**	**CAGC**	**GATC**
TCGG	-	1	0	0	1
GCAG	0	-	0	3	1
ATCG	3	1	-	0	1
CAGC	0	2	0	-	0
GATC	2	0	3	1	-

Once the overlap weighted directed complete graph is built as a pre-processing step, the DNA fragments assembly problem can be transformed into the problem of finding a Hamiltonian path that goes through every vertex (i.e. a complete order of the fragments). The quality of each path (i.e. sequence of fragments) is measured by the sum of the weights of its edges, which represents the total overlaps between fragments.

There were relatively few historical studies in the area of DNA fragments assembly using genetic algorithms. Some preliminary works were carried out in the early 1990s [3], [12]. In later works GA algorithms were enhanced by combining them with other searching metaheuristics in order to achieve better results. For instance [25] proposed a method named SAX, it combines GA algorithm enhanced with the simulated annealing metaheuristic to solve the DNA-FA. SAX implemented a greedy approach to generate the initial population, the order crossover (OX) operator, and the inversion mutation operator. SAX enhanced the GA approach with the simulated annealing metaheuristic to escape local optima. The proposed method was able to achieve relatively better overlapping scores on different datasets that reach the size of 400k bp (characters). However, the drawback of this GA solver is its high computational time. The authors in [26] examined different types of the population initialization including random initialization, 2-opt heuristics, and greedy methods. The crossover operators implemented in this study were the Cycle crossover (CX), Edge Recombination (ERX), Order crossover OX, and Partial Mapped crossover (PMX). According to their results, when using the 2-opt to initialize the population, the quality of the solution is improved without a significant increase in the overall execution time, regardless of the type of the applied crossover.

Another work using GA approach is presented in [27], it combines different GA variations in different ways: (1) Recentering-Restarting GA (RRGA) in order to avoid getting stuck on local optima, (2) Island Model GA (RRGA+IM) which divides the population into multiple islands. And finally, (3) GA which uses Ring Species (RRGA+RS), where the population is treated as a ring. The first and last individuals of the population are considered adjacent. They used two methods to initialize the population; the identity permutation and the 2-opt heuristic method proposed in [26]. The PMX crossover and swap mutation operators are implemented for all GA variations. These methods proved to be better since it records more overlapping score for all the tested datasets than the results obtained in [25]. However, among these three methods (RRGA, RRGA+IM, RRGA+RS) there is not a dominatrix; they all have the same convergence

performance. For that reason, some restarting strategies were implemented in the next work [28] in order to distinguish the performance of the above three GA variations. These strategies include dynamic restart where the search restarts after convergence, forcing initial restart where the search is forced to restart again in order to escape local optima. The experimental results showed that the RRGA has better convergence performance than RRGA+IM and RRGA+RS. In [29] the authors used RRGA along with the Power Aware Local Search (PALS) as a genetic operator and the 2-opt heuristic for initializing the population. Three types of experiments were performed with the objective function is minimizing the number of contigs and maximizing the overlap score. In the first experiment, PALS was used as a genetic operator, in the second experiment PALS is used after executing the GA. Whereas, in the last experiment, PALS was used as a genetic operator and utilized after GA execution as well. The results show that the first experiment using PALS as a genetic operator performs better than the other two experiments.

Despite the importance of the DNA fragments assembly problem, most of the previously mentioned studies ignore reporting the accuracy of their works (the degree to which the assembler covers the reference genome). However, there are some studies that assessed the accuracy as in [11]. In contracts with previous works that focused on maximizing the overlapping score as a fitness function, in [11] the fitness function is defined to minimizing the total length of a scaffold (the sum of the length of the contigs), and the number of contigs on the scaffold. The basic one-point order crossover has been implemented in this study. They assessed the accuracy and measured how the assembler actually covers the reference genome. In the next work [30] some improvements have been applied; the first one is to merge any two contigs that have overlapped between them longer than the length of the fragment. The other improvement is a post-processing step to merge each left chaff contigs into the appropriate location on long contigs (chaff means a contig of length shorter than 3–4 times the fragment length). The experimental results showed that the algorithm found a single correct contig identical to the reference in over 95% of 200 runs for most instances and decreases to 87.5% for the largest instances. However, as they mentioned, the main drawback of their method is the higher computational time.

Table III shows genetic algorithm performances in terms of the overlapping score for the DNA_FA problem. The first column represents the datasets used mainly in the literature with its mean fragment's length and sequence length (within brackets), obtained from the National Center for Biotechnology Information NCBI[1]. The first 10 datasets were fragmented (i.e., cutting the original sequence into fragments) using a tool called GenFrag and denoted by the dataset name-the coverage. These datasets are obtained from four sequences, ranging in length from 3 to 77 thousand bp. The "Acin" datasets fragmented using a different tool called the DNAgen; the "Acin" sequences are longer (except "Acin1", which is the smallest one in this table) and more difficult since they contain longer and more fragments). The "Acin" datasets are obtained from six sequences, ranging in length from 2 to 426 thousand bp. The rest columns are the references with the used method name (reference, method name), and for each method the sum of the generated overlapping score for each dataset is recorded. The symbol (-) means that this reference has not applied this method to the corresponding dataset. We considered in this table only references that report the overlapping score for their work.

From Table III, SAX presents competitive results for the small datasets. RRGA_RS was more performant for the long datasets ("Acin") comparing with the other methods, namely, it obtained more overlapping scores for three out of six "Acin" datasets. RRGA_IM gave satisfying results for eight out of sixteen reported datasets. GA2o is outperformed by the two other methods on the tested datasets.

In view of all that has been mentioned so far, such studies suffer from a lack of efficiently dealing with the accuracy, the time and space complexities. While DNA fragments assembly is a growing field, although GA gave very satisfactory results for similar hard optimization problems TSP and QAP, research works based on genetic algorithms for DNA assembly remains relatively poor. From reviewing and studying the previous related works of the GA, we can see that there are different ways to initialize and to represent the population. Also, it is obvious that the type of crossover affects the produced results in terms of solution quality and computational time. However, combining the GA with other good metaheuristics algorithms could improve the solution quality.

B. GA Solutions for TSP and QAP

This section reviews GA algorithms designed to solve the TSP and QAP problems, and their associated experimental results. This review is not exhaustive, we considered only more recent GA algorithms designed with advanced concepts.

GA design mainly begins with encoding the population. Different types of encoding were used for the optimization problems TSP and QAP. Most works of the wide literatures used the identity permutation such as for TSP in [10] [31] and QAP in [9]. Another advanced types of population encoding were used for TSP such as value encoding [20], and real number encoding [21].

The common strategies of generating of initial populations are the random generation as investigated for TSP in [21] and the greedy procedure as in [21]. Recently more advanced strategies have been designed; the Multi-Agent Reinforcement Learning (MARL) was proposed in [31] for solving TSP problems. The sequential sampling method has been implemented in [9] for solving QAP problem in order to improve the GA algorithm and to speed up the convergence.

[1] The National Center for Biotechnology Information (NCBI) is part of the United States National Library of Medicine(NLM), a branch of the National Institutes of Health (NIH). The NCBI houses a series of databases relevant to biotechnology and biomedicine and is an important resource for bioinformatics tools and services. Major databases include GenBank for DNA sequences. https://www.ncbi.nlm.nih.gov/guide/.

TABLE III. THE OVERLAPPING SCORE GENERATED BY THE GA FOR THE DNA FRAGMENTS ASSEMBLY PROBLEM. (* INDICATES THAT THE REFERENCE IMPLEMENTED MORE THAN FOUR METHODS, THE TABLE DISPLAYS THE RESULT OF THE BETTER ONE)

Dataset	**(reference, method)**				
Dataset (fragment length, sequence length)	***([25],SAX)***	***([26], GA2o*)***	***[27],RRGA)***	***([27],RRGA_RS)***	***[27],RRGA_IM)***
x60189-4 (395, 3835)	**11478**	-	**11478**	**11478**	**11478**
x60189-5 (286, 3835)	14027	13988.20	**14161**	**14161**	**14161**
x60189-6 (343, 3835)	**18301**	18293.03	**18301**	**18301**	18184
x60189-7 (387, 3835)	**21268**	21221	21228	21257	21218
m15421-5 (398, 10089)	**38726**	37967.13	38675	38668	38667
m15421-6 (350, 10089)	48048	-	48034	48048	**48052**
m15421-7 (383, 10089)	55072	53041.89	**55094**	55020	54986
j02459-7 (405, 20000)	115301	109513.62	116198	116110	**116336**
bx842596-4 (708, 77292)	223029	-	227151	227090	**227171**
bx842596-7 (703, 77292)	417680	-	441893	441867	**442100**
Acin1 (182, 2170)	46865	-	47436	**47450**	47437
Acin2 (1002, 147200)	144567	-	**151285**	151253	151243
Acin3 (1001, 200741)	155789	-	167035	166882	**167214**
Acin5 (1003, 329958)	145880	-	163061	**163066**	163027
Acin7 (1003, 426840)	157032	-	179835	**179932**	179886
Acin9 (1003, 156305)	314354	-	342936	342949	**342965**

The selection operator can have an impact on the overall performance of the GA algorithm [21]. The roulette wheel is the common selection operator used for optimization problems [20],[31],[32] [21], the tournament selection was implemented for TSP [22], and the stochastic remainder selection was used for QAP [33]. More recently in [21], a greedy method was designed as a selection operator for TSP.

The crossover operator is the main operator of GA as it plays a crucial role to explore efficiently the search space of the optimization problem. Hence, several advanced crossover operators have been designed for solving TSP as well as QAP using GA algorithms. The parents' characteristics are mainly inherited by crossover operators. The Sequential Constructive Crossover SCX is an intelligent crossover designed by Ahmed [10] to solve the TSP. A comparative study between SCX, ERX and generalized N-point crossover (GNX) for some benchmark TSPLIB[2] instances found SCX outperforms ERX and GNX in term of the solution quality. Most recently, a modified version of sequential constructive crossover, named greedy SCX (GSCX) was proposed for solving TSP [34]. The reverse greedy sequential constructive crossover (RGSCX) and the comprehensive sequential constructive crossover (CSCX) are two new crossover operators enhancing SCX for solving TSP [35].

The encouraging results obtained using SCX proved its effectiveness to solve the TSP problem. Wherefore, Ahmed [33] investigated its effectiveness for the QAP problem. He compared SCX with one-point crossover (OPX) and swap path crossover (SPX), and concluded that SCX was better in terms of solution quality. An Improved Genetic Algorithm (IGA) adapting and implementing SCX with the combined mutation for finding effective solution to the QAP was proposed in [9]. The performance of IGA using the adaptive and exchange mutation was evaluated on some QAPLIB instances and compared to a simple GA algorithm. The results showed that the IGA was better in terms of solution quality. The gaps with the best-known solutions were improved by 0.83% to 5.82% over the simple GA. However, the IGA takes longer time than the simple GA. SCX was also applied successfully for solving QAP with a combination of sequential sampling and random algorithms to generate the initial population [36]. Notice as well, the multi-point crossover implemented in [31] showed satisfying performance for solving TSP problems relatively to the classical crossover operators.

Another types of advanced crossover operators were designed in [32] to solve the QAP relying on the idea of a frequency model. Three crossover operators were introduced for enhancing GA, namely, the Highest Frequency crossover (HFX), the Greedy HFX (GHFX), and the Highest Frequency Minimum Cost crossover (HFMCX). The authors presented a detailed comparative study between One Point crossover, Swap Path crossover, SCX, and the new three crossover operators. The experimental results showed that the frequency models were better in term of computational time, precisely, HFX and GHFX were faster by 2 and 1.5 times than SCX, respectively. However, in term of solution quality all crossover operators found good near-optimal solutions on the tested benchmarks.

In order to compare the effectiveness of different types of crossover operators for solving the QAP using GA algorithm,

[2] The TSPLIB is a library of samples for the TSP and other problem such as Hamiltonian cycle problem, Sequential ordering problem, and Capacitated vehicle routing problem. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.

Misevicius and Kilda [37] implemented twelve different crossover operators; including the uniform like crossover (ULX) and its modifications (the randomized ULX crossover (RULX), the ULX crossover combined with repair procedure (RX), the block crossover (BX), the uniform partially-mapped crossover (UPMX), the distance preserving crossover (DPX), the cycle crossover, the swap path crossover, the one point crossover, the order-based crossover, the cohesive crossover (COHX), and, finally, the multi-parent crossover (MPX). The comparison showed that the multi-parents' crossover led to better performances.

Various types of mutation have been investigated for the TSP and QAP problems, including the Exchange Mutation [31],[21],[22], the Reciprocal Exchange Mutation[10]. The principle of these two mutation operators is similar to the Swap Mutation, which randomly selects two positions in the individual and swap the corresponding values. More advanced mutation operators have been designed for the TSP and QAP problems such as the interchange mutation in [20], and the inversion mutation in [22] which it selects two positions within an individual and then inverts the substring between these two positions. The adaptive and combined mutation operators were proposed for solving QAP in [36]. The adaptive mutation assigns highest probability values for the fittest individuals; therefore, all individuals will not have the same likelihood of mutation. Whereas, the combined mutation combines more than one type of mutation operators.

Table IV shows the results obtained for TSP in terms of the tour cost for some TSPLIB instances. The rows correspond to the results obtained for a problem instance by different types of crossover operators. The first column represents the dataset and its best-known solution (within parentheses), (e.g., bayg29 means the instance named "bayg" with 29 cities and the best-known solution 1610). The remaining columns display for each reference the name of the GA crossover operator as well as the obtained solution (tour cost). The symbol (-) means this crossover type has not been applied in this reference to the corresponding dataset. The table clearly shows that the Sequential Constructive Crossover (SCX) and the Smart Multi-Point Crossover (SMX) achieved better solutions for the tested datasets.

Table V shows the results obtained for solving the QAP problem in terms of the solution quality, which measured by the percentage of deviation (excess%) of average solution value over the best-known solution value reported in QAPLIB. The lower the percentage the better the solution quality. The datasets used for the QAP were obtained from the QAPLIB [38], which is a library with instances size varies from 12 to 256 facilities (or locations). The table displays in the first column the names and the sizes of datasets (e.g., "tai20a" means the dataset named tai contains 20 facilities or locations; the character "a" means random instances, and the character "b" as in "tai20b" means real-life like instances) [38]. The remaining columns represent for each reference the name of the GA crossover operator types as well as the obtained solutions. The symbol (-) means this crossover type has not been applied in this reference to the corresponding dataset. From this table we can notice that the multi-parents' crossover (MPX) followed by the Sequential Constructive Crossover (SCX) dominate the other crossover types.

TABLE IV. THE IMPACT OF GA CROSSOVER OPERATORS ON THE RESULTING TOURS OF TSP USING DIFFERENT DATASETS (TSPLIB INSTANCES)

TSPLIB instances	**Ref [10]**	**Ref [31]**	**Ref [21]**	**Ref [34]**	**Ref [35]**	**Ref [35]**
	SCX	***SMX***	***PMX***	***GSCX***	***RGSCX***	***CSCX***
bayg29 (1610)	**1610**	-	-	1634	-	-
Ftv33 (1286)	-	-	-	1380	1396	**1341**
Ftv35 (1473)	-	-	-	1531	1583	**1499**
Ftv38 (1530)	-	-	-	1613	1672	**1550**
P43 (5620)	-	-	-	5631	**5625**	5627
Ftv44 (1613)	-	-	-	1706	1627	**1613**
Ftv47 (1776)	-	-	-	1846	1919	**1833**
Ry48p (14422)	-	-	-	15469	15293	**14983**
att48 (33522)	-	**33522**	33523.06	-	-	-
eil51 (426)	**426**	**426**	-	436	-	-
berlin 52 (7542)	**7542**	**7542**	-	7926	-	-
eil76 (538)	**538**	**538**	-	-	-	-
pr76 (108159)	**108159**	**108159**	-	116844	-	-
kroa100 (21282)	**21282**	**21282**	-	-	-	-
eil101 (629)	**629**	**629**	-	-	-	-
lin105 (14379)	**14379**	**14379**	-	15921	-	-
bier127 (118282)	-	**118678**	-	-	-	-

TABLE V. THE IMPACT OF GA CROSSOVER OPERATORS ON THE RESULTING SOLUTION QUALITY OF QAP USING DIFFERENT DATASETS (QALIP INSTANCES). (** INDICATES THAT THE REFERENCE IMPLEMENTED 12 DIFFERENT TYPES OF CROSSOVER OPERATORS, THE TABLE DISPLAYED THE BEST OBTAINED RESULT).

QALIP instances	Ref [32]			Ref [33]			Ref [37]**	Ref [36]
	HFX	***GHFX***	***HFMCX***	***OPX***	***SPX***	***SCX***	***MPX***	***SCX***
Tai20a	6.81	5.59	4.82	6.57	5.59	4.50	**0.246**	1.20
Tai20b	8.45	8.31	7.67	8.73	9.12	6.56	**0.000**	0.44
Tai25a	6.18	5.82	5.33	5.01	5.61	4.58	**0.150**	1.71
Tai25b	10.05	10.73	7.45	12.92	15.42	5.24	**0.000**	0.14
Tai30a	6.11	6.12	5.66	5.16	5.49	4.29	**0.035**	2.30
Tai30b	11.32	10.04	9.05	13.4	10.55	8.78	**0.001**	0.18
Tai35a	5.78	5.21	5.71	4.92	5.35	4.95	**0.194**	2.42
Tai35b	8.45	7.11	6.02	7.49	6.76	5.00	**0.019**	0.33
Tai40a	7.44	6.23	4.23	4.97	5.76	4.53	**0.374**	2.48
Tai40b	9.22	10.52	8.11	9.44	11.57	7.30	**0.000**	0.03
Tai50a	7.13	5.47	5.46	5.00	4.91	4.51	**0.583**	3.07
Tai50b	8.15	8.43	5.28	7.46	6.68	5.60	**0.014**	0.56
Tai60a	6.73	6.00	5.74	4.89	4.60	4.54	**0.572**	3.28
Tai60b	7.83	7.11	5.11	6.95	8.14	5.12	**0.010**	0.34
Tai80a	5.41	4.55	4.02	4.32	3.83	4.35	**0.218**	3.41
Tai80b	7.05	6.41	5.39	6.15	6.07	6.63	**0.016**	2.17
Tai100a	5.49	5.04	3.65	4.04	3.22	4.02	**0.108**	2.92
Tai100b	8.34	7.09	5.23	9.33	5.39	5.08	**0.064**	0.95
Tai150b	-	-	-	-	-	-	**0.241**	1.81

From the literature review we noticed that the TSP and QAP problems have been treated with various GA designs, including different advanced GA concepts. The SCX crossover emerges from the lot as it is shown experimentally to be the most performant crossover for the TSP and QAP problems. We also noticed that the exchange mutation operator is mostly used for TSP in the literature. The datasets used for the TSP were obtained from the TSPLIB, which is a library of instances of the TSP and related problems from various sources and of various types with number of cities varies from 14 to more than 33k cities.

The QAP literature review included other types of performant crossover operators; as the frequency models and the multi-parents' crossover.

V. DISCUSSION

In the light of what has been studied and reviewed so far, we noticed that GA algorithms applied to the TSP and QAP problems have led to very satisfactory performances in practice. We likewise noticed that some advanced types of genetic algorithm operators have been widely used for the two problems, and their effectiveness has been investigated to solve these two problems. Unfortunately, these advanced GA operators have not yet been exploited for solving the DNA_FA problem. The similarities between DNA_FA and these two problems pointed out in this paper are strong indicators of the benefit that the use of the advanced GA algorithms can bring to solve the DNA_FA problem. That is why it is worth to design GA platform including advanced GA operators and to investigate its effectiveness for solving the DNA_FA problem.

More in detail, we have noticed that the most efficient crossover operators SCX, frequency model and the multi-parent crossovers have not been adapted and exploited for the DNA_FA problem. Likewise, the 2-opt heuristics for initializing the population was used few times for the DNA_FA problem. Regarding the mutation operators, the Adaptive and Combined mutation designed for the TSP and QAP problems can be exploited for the DNA_FA problem.

In order to design GA platform including advanced GA operators, Table VI summarizes the main GA operators and their values and Table VII shows the used GA operators so far for each problem (DNA_FA, TSP, QAP). As well as the different GA parameters tunings: such as the population size, the crossover and mutation probabilities, the number of generations, and the number of runs. Defining a proper setting for GA parameters can drastically improve the algorithm performance. However, it is not an easy task, generally the parameters are set experimentally, and Table VII supplies some GA parameter settings for the three optimization problems. From Table VII, we can clearly see the advanced operators that have not been adapted and implemented so far for the DNA_FA problem.

VI. GA Platform to Solve the DNA_FA Problem

Our GA platform design to solve the DNA_FA problem is inspired from the efficient GA approaches developed for the TSP and QAP problems. The GA platform gathers several advanced GA operators and tools which have shown their effectiveness in the TSP and QAP contexts. The GA platform contains the best advanced GA tools for our problem DNA_FA and one could build many variants of GA algorithms for solving it by integrating in different and judicious ways the ingredients of this platform.

A. The Flowchart of the GA Platform

Fig. 2 presents the flowchart of our GA platform to solve the DNA_FA problem.

B. The GA Concepts of the Designed Platform

1) Initial population: Our GA platform design includes the random, the greedy and the 2-opt heuristics strategies which yielded to good performances as shown in [27][28][29]. The previous works showed that the computational time when using the 2-opt and greedy initialization strategies was better than when using the random way.

2) Fitness function: As the fitness function is repeatedly applied to each individual of each generation it should be relatively easy to compute and should also give an accurate evaluation of the quality of each individual. We reported two related fitness functions from the literature. The first one is a simple fitness function that sums the overlap for each of the adjacent fragment's pairs, as expressed by the formula (4) in [3].

$$F1 = \sum_{i=1}^{n-1} w[i, i+1] \quad (4)$$

where $w[i, i+1]$ is the overlap between fragment i and fragment $i + 1$. This function attempts to maximize the value F1.

TABLE VI. GA Operators and their Values

GA operator	values
Population encoding	Integer encoding, value encoding.
Population initialization	Random initialization, greedy initialization, 2-opt heuristics initialization, and sequential sampling initialization.
Crossover operator	Multi-parents crossover, order crossover, cycle crossover, edge recombination crossover, partial mapped crossover, sequential constructive crossover, greedy sequential constructive crossover, reverse greedy sequential constructive crossover, comprehensive sequential constructive crossover, one-point crossover, highest frequency crossover, greedy highest frequency crossover, highest frequency minimum cost crossover, uniform like crossover, randomized uniform like crossover, block crossover, uniform partially mapped crossover, distance preserving crossover, cohesive crossover, and smart multi-point crossover.
Mutation operator	Inversion mutation, swap mutation, reciprocal mutation, exchange mutation, interchange mutation, combined mutation, and adaptive mutation.
Selection operator	Roulette wheel selection, tournament selection, greedy selection, and stochastic reminder selection.
Stopping condition	Number of generations, CPU time, no improvement for number of iterations.

TABLE VII. The Existing GA Design and Experimental Settings for DNA_FA, TSP and QAP

GA design and experimental settings	DNA_FA	TSP	QAP
Population encoding	Integer number (sequence of integer numbers, each of which represents a city to be visited)	integer numbers, value encoding (sequence of some values such as real numbers, characters, each of which represents a city to be visited)	Integer numbers
Population initialization	Random, greedy, 2-opt heuristics	Random, greedy, MARL	Sequential sampling, random
Population size	Varies from 11 to 2500 Individuals.	Varies from 20 to 200 individuals.	Varies from 30 to 200 individuals.
Selection	Tournament.	Roulette wheel, tournament, greedy.	Roulette wheel, stochastic reminder selection.
Crossover	OX, ER, PMX, one-point order. CX	SCX, ERX, GNX, PMX, smart multi point crossover, order insert crossover.	SCX, OPX, SPX, HFX, GHFX, HFMCX, MPX.
Mutation	Inversion mutation, swap mutation	Reciprocal mutation, exchange mutation, interchange mutation, inversion mutation	Reciprocal exchange mutation, combined mutation, adaptive mutation, swap mutation
Crossover probability	Varies from (60% to 100%)	Varies from (90% to 100%)	100%
Mutation probability	2%	Varies from (1% to 20%)	Varies from (5% to 15%)
Stopping condition	No improvement for number of iterations.	Optimal rout, number of generations.	Number of generations, CPU time.
Number of runs	From 5 runs to 30 runs	From 10 runs to 30 runs.	20 runs.
Number of generations	Varying from (1 K to 512 K) generations	Varying from (20 to 10k) generations.	Varying from (5000 to 10k) generations.

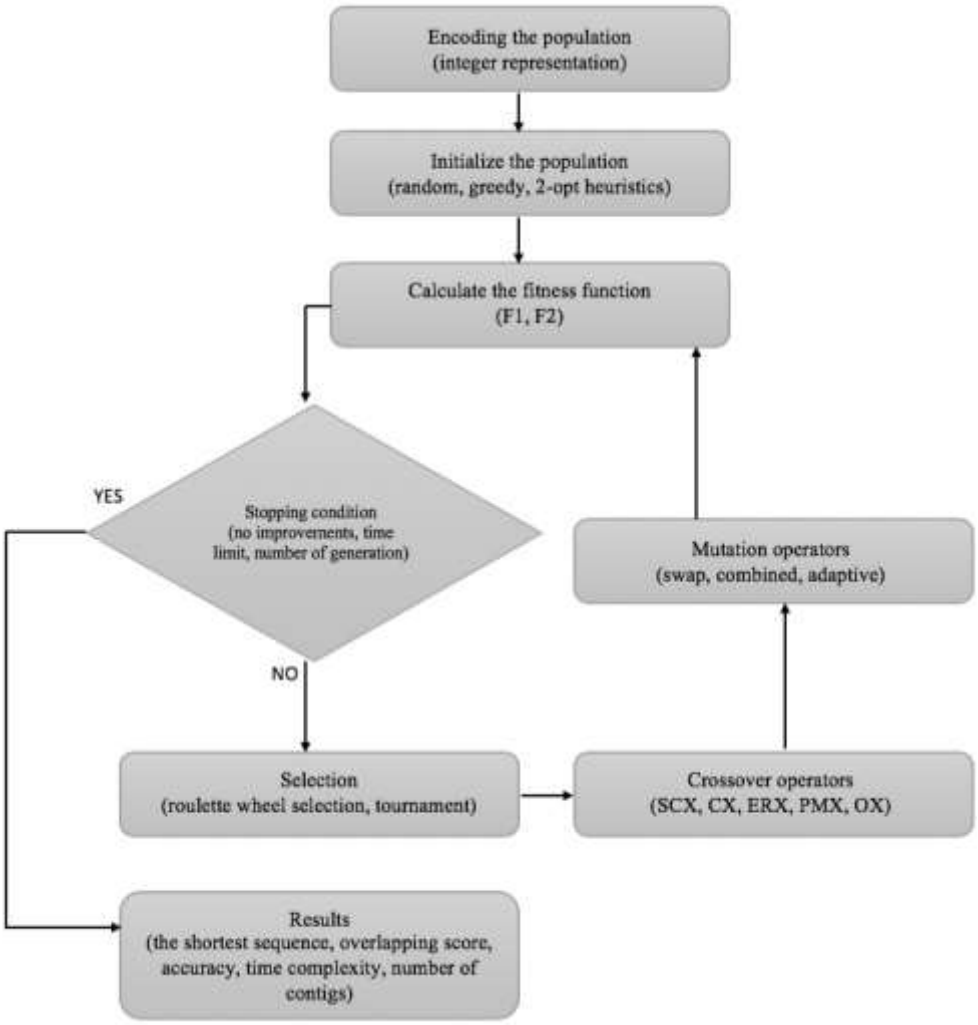

Fig. 2. GA Platform Design for the DNA Fragments Assembly Problem.

The second fitness function F2 is expressed by the formula (5) in [3].

$$F2 = \sum_{i=1}^{n} \sum_{j=1}^{n} |i - j| * w[i, j] \quad (5)$$

The fitness function F2 considers the overlap between adjacent fragments and the overlap between non-adjacent fragments as well. However, the complexities of F1 and F2 are different, F1 takes $O(n)$ where F2 is of $O(n^2)$ because all pairs of fragments must be considered [3]. These two fitness functions are included in the platform.

3) Selection operator: Several selection operators were used for these three problems as shown in Table VI (e.g., roulette wheel selection, rank selection, elitist selection, and tournament selection). As roulette wheel selection widely used and consumes least amount of time, and tournament selection can maintain diversity by giving an equal chance to all the individuals to compete [39]; The roulette wheel selection and the tournament selection are selected to be added to the platform.

4) Crossover operator: Several crossover operators SCX, CX, PMX, ERX, and Order Crossover are candidate to be included in the platform. A special attention should be given to the SCX crossover as it was one of the best operators for the TSP and QAP problems and we predict same performance in the DNA_FA context.

5) Mutation operator: The swap mutation operator with its variants were widely used for DNA_FA, TSP, and QAP. Combined and adaptive mutation was designed for the QAP problem. The last one seems more suitable as it was performant for the QAP problem.

VII. Conclusion

This paper aims to show why it is worth to investigate genetic algorithms for solving the DNA fragment assembly problem. We have provided a simple formal proof showing the relationship between this problem and the famous TSP and QAP problems enabling us to extract some similarities between these three optimization problems. TSP and QAP have been solved efficiently using GA algorithms designed with advanced GA operators and tools. For this reason, we exploited the extracted similarities between the DNA_FA problem and the TSP and QAP problems to design an efficient GA platform integrating several advanced operators used for

TSP and QAP. Our future work is to implement the designed GA platform and to conduct comprehensive experiments in order to get the best combination of integrating the different operators of GA to build a robust solver for the DNA_FA problem.

REFERENCES

[1] G. Luque and E. Alba, "Metaheuristics for the DNA Fragment Assembly Problem," 2005, doi: 10.5019/j.ijcir.2005.28.

[2] Pavel A. Pevzner, Computational Molecular Biology An Algorithmic Approach. The MIT Press, 2000.

[3] R. J. Parsons, S. Forrest, and C. Burks, "Genetic Algorithms for DNA Sequence Assembly," Proc. Int. Conf. Intell. Syst. Mol. Biol., vol. 1, pp. 310–318, 1993.

[4] J. Błażewicz, P. Formanowicz, M. Kasprzak, W. T. Markiewicz, and J. Węglarz, "Tabu search for DNA sequencing with false negatives and false positives," Eur. J. Oper. Res., vol. 125, no. 2, pp. 257–265, Sep. 2000, doi: 10.1016/S0377-2217(99)00456-7.

[5] G. Minetti and E. Alba, "Metaheuristic assemblers of DNA strands: Noiseless and noisy cases," in IEEE Congress on Evolutionary Computation, Jul. 2010, pp. 1–8, doi: 10.1109/CEC.2010.5586524.

[6] K. Huang, J. Chen, and C. Yang, "A Hybrid PSO-Based Algorithm for Solving DNA Fragment Assembly Problem," in 2012 Third International Conference on Innovations in Bio-Inspired Computing and Applications, Sep. 2012, pp. 223–228, doi: 10.1109/IBICA.2012.8.

[7] C. Blum, M. Y. Vallès, and M. J. Blesa, "An ant colony optimization algorithm for DNA sequencing by hybridization," Comput. Oper. Res., vol. 35, no. 11, pp. 3620–3635, Nov. 2008, doi: 10.1016/j.cor.2007.03.007.

[8] A. Hussain, Y. S. Muhammad, M. Nauman Sajid, I. Hussain, A. Mohamd Shoukry, and S. Gani, "Genetic Algorithm for Traveling Salesman Problem with Modified Cycle Crossover Operator," Comput. Intell. Neurosci., vol. 2017, pp. 1–7, 2017, doi: 10.1155/2017/7430125.

[9] Z. H. Ahmed, "An improved genetic algorithm using adaptive mutation operator for the quadratic assignment problem," in 2015 38th International Conference on Telecommunications and Signal Processing (TSP), Jul. 2015, pp. 1–5, doi: 10.1109/TSP.2015.7296481.

[10] Z. H. Ahmed, "Genetic Algorithm for The Traveling Salesman Problem Using Sequential Constructive Crossover," 2010.

[11] D. Bucur, "De Novo DNA Assembly with a Genetic Algorithm Finds Accurate Genomes Even with Suboptimal Fitness," in Applications of Evolutionary Computation, vol. 10199, G. Squillero and K. Sim, Eds. Cham: Springer International Publishing, 2017, pp. 67–82.

[12] R. J. Parsons, S. Forrest, and C. Burks, "Genetic algorithms, operators, and DNA fragment assembly," Mach. Learn., vol. 21, no. 1–2, pp. 11–33, 1995, doi: 10.1007/BF00993377.

[13] A. B. Ezzeddine, S. Kasala, and P. Navrat, "Applying the Firefly Approach To The Dna Fragments Assembly Problem," p. 13.

[14] W. Wetcharaporn, N. Chaiyaratana, and S. Tongsima, "DNA Fragment Assembly by Ant Colony and Nearest Neighbour Heuristics," in Artificial Intelligence and Soft Computing – ICAISC 2006, vol. 4029, L. Rutkowski, R. Tadeusiewicz, L. A. Zadeh, and J. M. Żurada, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1008–1017.

[15] G. M. Mallén-Fullerton and G. Fernández-Anaya, "DNA fragment assembly using optimization," in 2013 IEEE Congress on Evolutionary Computation, Jun. 2013, pp. 1570–1577, doi: 10.1109/CEC.2013.6557749.

[16] E. Çela, The Quadratic Assignment Problem, vol. 1. Boston, MA: Springer US, 1998.

[17] E. Çela, "Problem Statement and Complexity Aspects," in The Quadratic Assignment Problem, vol. 1, Boston, MA: Springer US, 1998, pp. 1–25.

[18] E. Çela, V. G. Deineko, and G. J. Woeginger, "The multi-stripe travelling salesman problem," Ann. Oper. Res., vol. 259, no. 1, pp. 21–34, 2017, doi: 10.1007/s10479-017-2513-4.

[19] R. E. Burkard, E. Çela, P. M. Pardalos, and L. S. Pitsoulis, The Quadratic Assignment Problem. .

[20] 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), Feb. 2019, pp. 264–268, doi: 10.1109/AICAI.2019.8701246.

[21] W. Xueyuan, "Research on Solution of TSP Based on Improved Genetic Algorithm," in 2018 International Conference on Engineering Simulation and Intelligent Control (ESAIC), Aug. 2018, pp. 78–82, doi: 10.1109/ESAIC.2018.00025.

[22] R. Liu and Y. Wang, "Research on TSP Solution Based on Genetic Algorithm," in 2019 IEEE/ACIS 18th International Conference on Computer and Information Science (ICIS), Jun. 2019, pp. 230–235, doi: 10.1109/ICIS46139.2019.8940186.

[23] S. Sahni and T. Gonzalez, "P-Complete Approximation Problems," J. ACM JACM, vol. 23, no. 3, pp. 555–565, Jul. 1976, doi: 10.1145/321958.321975.

[24] Alfonsas Misevičius, Dalius Rubliauskas, "Performance of Hybrid Genetic Algorithm for The Grey Pattern Problem♣," Information technology and controL, 2005.

[25] G. Minetti, G. Leguizamon, and E. Alba, "SAX: a new and efficient assembler for solving DNA Fragment Assembly Problem," p. 12, 2012.

[26] G. Minetti, E. Alba, and G. Luque, "Seeding strategies and recombination operators for solving the DNA fragment assembly problem," Inf. Process. Lett., vol. 108, no. 3, pp. 94–100, Oct. 2008, doi: 10.1016/j.ipl.2008.04.005.

[27] J. Hughes, S. Houghten, G. M. Mallén-Fullerton, and D. Ashlock, "Recentering and Restarting Genetic Algorithm variations for DNA Fragment Assembly," in 2014 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology, May 2014, pp. 1–8, doi: 10.1109/CIBCB.2014.6845500.

[28] J. A. Hughes, S. Houghten, and D. Ashlock, "Restarting and recentering genetic algorithm variations for DNA fragment assembly: The necessity of a multi-strategy approach," Biosystems, vol. 150, pp. 35–45, Dec. 2016, doi: 10.1016/j.biosystems.2016.08.001.

[29] Uzma and Z. Halim, "Optimizing the DNA fragment assembly using metaheuristic-based overlap layout consensus approach," Appl. Soft Comput., vol. 92, p. 106256, Jul. 2020, doi: 10.1016/j.asoc.2020.106256.

[30] D. Bucur, "A stochastic de novo assembly algorithm for viral-sized genomes obtains correct genomes and builds consensus," Inf. Sci., vol. 420, pp. 184–199, Dec. 2017, doi: 10.1016/j.ins.2017.07.039.

[31] M. M. Alipour, S. N. Razavi, M. R. Feizi Derakhshi, and M. A. Balafar, "A hybrid algorithm using a genetic algorithm and multiagent reinforcement learning heuristic to solve the traveling salesman problem," Neural Comput. Appl., vol. 30, no. 9, pp. 2935–2951, Nov. 2018, doi: 10.1007/s00521-017-2880-4.

[32] H. Bennaceur and Z. Ahmed, "Frequency model based crossover operators for genetic algorithms applied to the quadratic assignment problem," Int Arab J Inf Technol, vol. 14, pp. 138–145, 2017.

[33] Z. H. Ahmed, "A Simple Genetic Algorithm using Sequential Constructive Crossover for the Quadratic Assignment Problem," vol. 73, p. 4, 2014.

[34] Zakir Hussain Ahmed, "Solving the Traveling Salesman Problem using Greedy Sequential Constructive Crossover in a Genetic Algorithm," Fabruary 2020.

[35] Zakir Hussain Ahmed, "Genetic Algorithm with Comprehensive Sequential Constructive Crossover for the Travelling Salesman Problem," IJACSA Int. J. Adv. Comput. Sci. Appl. Vol 11 No 5, 2020.

[36] Z. H. Ahmed, H. Bennaceur, M. H. Vulla, and F. Altukhaim, "A Hybrid Genetic Algorithm for the Quadratic Assignment Problem," p. 7.

[37] A. Misevicius and B. Kilda, "Comparison of Crossover Operators for The Quadratic Assignment Problem," 2015, doi: 10.5755/j01.itc.34.2.11999.

[38] R. E. Burkard, S. Karisch, and F. Rendl, "QAPLIB-A quadratic assignment problem library," Eur. J. Oper. Res., vol. 55, no. 1, pp. 115–119, Nov. 1991, doi: 10.1016/0377-2217(91)90197-4.

[39] N. Saini, "Review of Selection Methods in Genetic Algorithms," Int. J. Eng. Comput. Sci., vol. 6, no. 12, Art. no. 12, Dec. 2017.

APPENDIX

TABLE VIII. SYMBOLS AND NOTATIONS

Symbol	**Refer to**
DNA_FA	DNA Fragments Assembly problem.
TSP	Traveling Salesman Problem.
QAP	Quadratic Assignment Problem.
GA	Genetic Algorithms.
A	Adenine (A).
T	Thymine (T).
G	Guanine (G).
C	Cytosine (C).
MPX	Multi-Parents Crossover.
OX	order crossover.
CX	Cycle Crossover.
ERX	Edge Recombination Crossover.
PMX	Partial Mapped crossover.
SCX	Sequential Constructive Crossover.
GSCX	Greedy Sequential Constructive Crossover.
RGSCX	Reverse Greedy Sequential Constructive Crossover.
CSCX	Comprehensive Sequential Constructive Crossover.
OPX	One-Point Crossover.
HFX	Highest Frequency Crossover.
GHFX	Greedy Highest Frequency Crossover.
HFMCX	Highest Frequency Minimum Cost Crossover.
ULX	Uniform Like Crossover.
RULX	Randomized Uniform Like Crossover.
BX	Block Crossover.
UPMX	Uniform Partially Mapped Crossover.
DPX	Distance Preserving Crossover.
COHX	Cohesive Crossover.
SMX	Smart Multi-Point Crossover.
MARL	Multi-Agent Reinforcement Learning.
RRGA	Recentering-Restarting Genetic Algorithm.
RRGA+IM	Island Model Genetic Algorithm.
RRGA+RS	Ring Species Genetic Algorithm.
PALS	Power Aware Local Search.
NCBI	The National Center for Biotechnology Information.
TSPLIB	TSP library.
QAPLIB	QAP library.