Experimental Evaluation 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[1, 2, 3]

***Abstract*—This paper aims to highlight the motivations for investigating genetic algorithms (GAs) to solve the DNA Fragment Assembly (DNAFA) problem. DNAFA problem is an optimization problem that attempts to reconstruct an original DNA sequence by finding the shortest DNA sequence from a given set of fragments. This paper is a continuation of our previous research paper in which the existence of a polynomial-time reduction of DNAFA into the Traveling Salesman Problem (TSP) and the Quadratic Assignment Problem (QAP) was discussed. Taking advantage of this reduction, this work conceptually designed a genetic algorithm (GA) platform to solve the DNAFA problem. This platform offers several ingredients enabling us to create several variants of GA solvers for the DNAFA optimization problems. The main contribution of this paper is the designing of an efficient GA variant by carefully integrating different GAs operators of the platform. For that, this work individually studied the effects of different GAs operators on the performance of solving the DNAFA problem. This study has the advantage of benefiting from prior knowledge of the performance of these operators in the contexts of the TSP and QAP problems. The best designed GA variant shows a significant improvement in accuracy (overlap score) reaching more than 172% of what is reported in the literature.**

Keywords*—Genetic algorithms; traveling salesman problem; quadratic assignment problem; DNA fragments assembly problem

I. Introduction

The DNA fragment assembly problem is the process of reconstructing an original DNA sequence from a given set of DNA fragments. This is achieved by ordering and aligning these DNA fragments such that the resulting DNA sequence is as short as possible. It is a complex combinatorial optimization problem belonging to the class of NP-hard problems, where there is a need to find the right order of the DNA fragments to assemble them. Several metaheuristic techniques have been developed to solve this problem [1], [2], [3]. This paper exploit the genetic algorithm platform (GAP) developed in the former preliminary paper [4], in which the existence of a polynomial-time reduction of DNAFA into the Traveling Salesman Problem and the Quadratic Assignment Problem was discussed. Then, conceptually designed a GA platform for solving the DNAFA problem, inspired by the existing efficient GAs in the literature for solving the TSP and QAP problems. This platform gathers and offers several GA operators designed to solve hard optimization problems such as TSP, QAP, and DNAFA. The GAP enables the researchers to easily design an adequate variant GA algorithm for hard optimization problems in particular. This work implementing and experimenting on some GA variants judiciously built from the platform (GAP) aims to identify the best variant that efficiently deals with the DNAFA problem. Using this platform, this work is able to individually study the effects of genetic algorithm components on selected metrics, which were presented in terms of time and overlap score. This work focused on examining and discussing the effects of population size, population generation methods, selection types, and crossover types and figure out which component has the most impact on GA performance. Some of these GA components have never been tested in the context of the DNAFA problem, such as SCX crossover, which is worth to be investigated experimentally. Other components have been tested before, but when retest was done on them, a different result was found, such as greedy as a population initialization method. Because of SCX effectiveness in TSP and QAP, we believe the SCX crossover is a smart crossover that will outperform other crossovers. As a result of these comprehensive experiments, this work identifies the best-designed GA variant that outperforms the existing GA algorithms in solving the DNAFA problem. This GA variant features the use of 200 individuals for the population size, along with the greedy method for initializing the population, tournament selection, and SCX crossover. This GA variant showed a significant increase in overlap score compared to what is reported in the literature. The results showed that the SCX crossover was the best crossover among the studied crossovers and gave good results. Furthermore, the results showed that the greedy method is a very powerful method that improved the algorithm's performance by 37%, demonstrating that the population generation method has the greatest impact on improving the results than the other GA components. The experimental results demonstrated the efficiency of the designed approach, as it got a better result for the overlap score ranging from 56.16% to 172.74% than the previous recorded results for most data sets. This work demonstrate experimentally that the best designed GA variant outperforms existing GA algorithms in solving the DNAFA problem for some data sets.

A. The DNAFA Problem

The DNAFA problem is defined as follows: Given a set of fragments, f1, f2,…, fn, drawn from a finite alphabet $\Sigma = \{A,C,G,T\}$, the goal is to find the shortest superstring that contain all the input 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'.

$$MAX_{<f'_1,f'_2,\ldots,f'_n>} \sum overlapping\ (f'_i f'_{i+1}) \quad (1)$$

$where\ 1 \leq i \leq n-1.$

B. Assembly Process

To understand the assembly process, we've defined some key terms.

- Fragment: A short sequence of DNA bases. It is also called read.
- Coverage: The number of fragments at a specific position in the DNA.
- Prefix: A substring from the first characters of a fragment.
- Suffix: A substring from the last characters of a fragment.
- Overlap: Common sequence between the suffix of one fragment and the prefix of another.
- Layout: An arrangement of the collection of fragments based on their overlapping order.
- Contig: Contiguous overlapped fragments without gaps.
- Scaffold: The overlapped contigs, which may contain gaps.
- Consensus: Reconstruction of the complete sequence

In the assembly process, the input for the DNA fragment assembly is a set of fragments. The traditional assembly approach works in the following order: overlap, layout, and consensus [5].

- *Overlap stage:* Finding the overlapping fragments and computing their similarity score (overlap score). This means finding the longest match between the suffix of one fragment and the prefix of another.
- *Layout stage:* Finding the order of fragments based on the computed overlap score.
- *Consensus stage*: Reconstructing the complete sequence from the layout.

This paper is organized as follows: the second section discusses the related works, then the proposed design is discussed in detail in the third section. In the fourth section, the experiments and the method of conducting the investigations are detailed, and then the results are listed in the fifth section. The sixth section discusses these results by comparing them with previous works, and finally, the paper is concluded in the seventh section.

II. Related Works

This section presents the previous related works organized into two subsections: the first subsection summarizes the works solving the DNAFA with GA, and the second subsection introduces the works solving the TSP and QAP with GA.

A. Genetic Algorithm for DNAFA Problem

The basic genetic algorithm schema contains various concepts such as population encoding, population initialization, fitness function, selection, crossover, and mutation. Each concept has its own importance in the algorithm. By studying previous works, it can be noted that each concept can be done in a different way. In more detail, the population can be encoded in different ways in the GA, one such way is through segmented permutation [6], identity permutation [7]. Random generation, as in [8], the greedy approach, and the 2-opt heuristics, as proposed by Minetti et al. [8] and [9], are common strategies for generating the initial populations.

For the fitness function, the most commonly used fitness function is to maximize the overlap score, where the smith-waterman algorithm is used to calculate the overlap between the fragments [7]. The smith water man algorithm takes a lot of time but, even though it is the most precise algorithm for identifying similarity regions between fragments. Overlap score is considered the best measure for measuring the quality of the solution. It was used in most of the previous works [7], [8], and [9].

The crossover operator is the main operator of GA, as it plays a crucial role in efficiently exploring the search space of the optimization problem. The parents' characteristics are mainly inherited by crossover operators. Among the crossovers that were used in solving DNAFA, there was the order-based crossover (OX) as in [5], [10], [6], the edge-recombination (ER) [5], and the partially mapped crossover (PMX) [7], [9]. For the mutation, inversion mutation operators [10], and swap mutation [7], [9]were used for the DNAFA.

GA can be combined with other metaheuristics to achieve good results. For this purpose, Minetti et al. [10] designed a hybrid method named SAX that combined the GA with a simulated annealing metaheuristic. Another work, by Hughes et al. [7] combines different variations of GA in different ways. Also, the authors in [5] applied multiple algorithms, such as simulated annealing and scatter search with the GA. Another recent work is provided by Uzma and Halim [9] they combine GA and Power Aware Local Search (PALS).

The studies of Bucur [6], [11] focus on minimizing the total length of the scaffold (summing the length of the overlapped contigs). Unlike previous works, Bucur used simulated data sets where the fragments were of uniform length, they were able to measure the accuracy since they had the reference genome. However, as they mentioned, the main disadvantage of their method is its increased time complexity.

B. Genetic Algorithm for TSP and QAP

This section reviews GA algorithms designed to solve the TSP and QAP problems. 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 [12], [13] and QAP in [14]. Another advanced types of population encoding were used for TSP such as value encoding [15], and real number encoding [16]. The common strategies of generating the initial populations are the random generation as investigated for TSP in [16] and the greedy method as in [17]. Recently, more advanced strategies have been developed, the

author in [13] proposed Multi-Agent Reinforcement Learning (MARL) for solving TSP problems, and [14] implemented the sequential sampling method for solving QAP problems.

For the selection, the roulette wheel is the common selection operator used for optimization problems [15], [13], [18], [16], the tournament selection was implemented for TSP [17], and the stochastic remainder selection was used for QAP [19]. More recently, in [16], a greedy method was designed as a selection operator for TSP.

Several advanced crossover operators have been designed for solving TSP as well as QAP using GA algorithms. The Sequential Constructive Crossover (SCX) is an intelligent crossover designed by Ahmed [12] to solve the TSP. Recently, a modified version of sequential constructive crossover, named greedy SCX (GSCX), was proposed for solving TSP [20]. The reverse greedy sequential constructive crossover (RGSCX) and the comprehensive sequential constructive crossover (CSCX) are two new crossover operators that enhance SCX for solving TSP [21]. Other types of advanced crossover operators were designed in [18] 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).

Various types of mutations have been investigated for the TSP and QAP problems, including the exchange mutation [13], [16], [17], and the reciprocal exchange mutation [12]. More advanced mutation operators have been designed for the TSP and QAP problems, such as the interchange mutation in [15], and the inversion mutation in [17]. In [22], the adaptive and combined mutation operators were proposed for solving QAP.

C. Other Metaheuristics Algorithms for DNA Fragments Assembly

Particle swarm optimization (PSO) was reported in the literature for the DNA fragment assembly problem. Verma and Kumar [2] used the PSO with the smallest position value (SPV) rule. The PSO can be enhanced when combined with other algorithms, such as in Huang et al. [23], who proposed a hybrid particle swarm optimization algorithm (HPSO). The algorithm was divided into two parts: (1) Tabu search combined with PSO to improve solution quality and (2) simulated annealing combined with variable neighborhood local search (VNS). Additionally, the parallel approach can reduce the computation time, so, Mallén-Fullerton and Fernández-Anaya [2] presented a parallel heuristic based on the PSO and the differential evolution (DE), which is similar to GA, but DE relies on mutation operation, while GA relies on crossover operation to assemble better solutions. Mallén-Fullerton and Fernández-Anaya used a variation of the TSP (the Lin-Kernighan algorithm [24]) with some modifications to be applied for DNA fragment assembly. Another study is that of Huang et al [25], who presented a memetic PSO algorithm with a variable neighborhood search (VNS) approach as well as TS and SA, each of these algorithms is used in different ways and in different combinations. Indumathy and Maheswari [26] used a variant of the standard PSO called the constriction factor PSO (CPSO). Another proposed metaheuristic algorithm for solving the DNA fragment assembly problem is the problem aware local search (PALS) [27]. The main drawback of PALS is its quick convergence to local optima but combining it with other algorithms can overcome this drawback. Minetti et al. [28] used PALS by combining it with SA, this suggested method shows improved performance on the largest data sets when compared with SA and PALS separately. Another algorithm founded for the DNA fragment assembly is the bee colony, Firoz et al. [29] presented the artificial bee colony (ABC) algorithm and the queen bee evolution based on the genetic algorithm (QEGA). Majid al-Rifaie [30] investigated a new algorithm, stochastic diffusion search (SDS), which follows a different strategy for calculating the overlaps, picking a model from given fragments and trying to find the same model in the rest of the fragments. Among the fragments containing the model, the one with the highest similarity is picked, assembled, and then removed from the search space.

The previous paper [4] showed that the DNAFA optimization problem is a special case of two well-known optimization problems: the traveling salesman problem and the quadratic assignment problem. Particularly, that paper theoretically demonstrated that all three optimization problems have a similar topological structure and that they need to explore a search space of solutions with the same complexity to find an optimal solution. For this reason, the GA platform designed to solve the DNAFA problem is inspired by the efficient GA approaches developed for the famous combinatorial optimization problems, TSP and QAP. The GA platform gathers several advanced GA operators and tools that have demonstrated their effectiveness in the context of TSP and QAP.

Table I illustrates the GA parameters' settings from the literature for DNA, TSP, and QAP.

III. The GA Platform

The GA platform consists of the best and most advanced GA tools for the DNAFA problem (shown in Fig. 1.). One could build several variants of the GA to solve it by judiciously integrating the ingredients of this platform in different ways.

A. The GA Operations of the Designed Platform

This section describe the different GA operations involved in the platform that suggested earlier [4]. This paper will study all these operations, test them experimentally, and try different versions by combining different tools from the platform to create the best version that will be compared with other algorithms from the literature.

1) Encoding: For the encoding, the work will use the integer encoding, where the fragments encode as numbers, such that fragment one encodes as "1", fragment two encodes as "2" and so on.

TABLE I. GA PARAMETERS SETTING FROM THE LITERATURE FOR DNA, TSP AND QAP

GA Design and Experimental Settings	DNA_FA	TSP	QAP
Individual encoding	Integer numbers (an ordered sequence of integer numbers, each of which represents a fragment number).	Integer numbers (an ordered sequence of integer numbers, each of which represents a city to be visited)	Integer numbers (an ordered sequence of integer numbers, each of which represents an assignment of a task to a resource).
Population initialization	Random, greedy, 2-opt heuristics	Random, greedy, MARL (Multiagent Reinforcement Learning) [13].	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 [31]	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 [12], [14].
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 (1000 to 512 K) generations	Varying from (20 to 10,000) generations.	Varying from (5000 to 10k) generations.

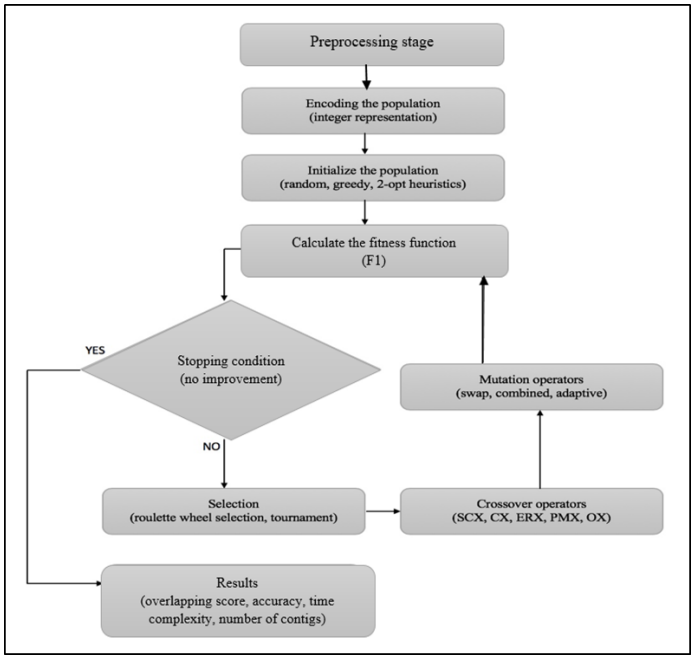

Fig. 1. GA platform design for the DNA fragments assembly problem.

2) Initial Population: Initial population includes the population size and the population generation method. For the population size, the work selects two sizes (200 and 500 individuals) and discusses how much time is saved if the population is small and how accurate it is.

For the population generation method, the GA platform design includes the random, greedy, and 2-opt heuristic strategies that have previously yielded good performances as shown in [7], [31], and [9]. Since the primary results showed that the greedy initialization method gave the best solutions, this paper displays its results. However, because the greedy method searches and generates populations intelligently, further experiments will investigate whether the greedy can find the solution from the beginning without relying on the rest of the GA operator.

3) Fitness Functions: As the fitness function is repeatedly applied to each individual of each generation, it should be relatively easy to compute and should also accurately evaluate the quality of each individual [12]. A simple fitness function aims to maximize the overlap score by summing the overlap for each of the adjacent fragment pairs, as expressed by the expression (2) in [9].

$$F = \sum_{i=1}^{n-1} w[i, i+1] \qquad (2)$$

where $w\ [i,\ i\ +\ 1]$ is the overlap score between fragment i and fragment i + 1. F is simple in complexity since it takes O(n).

To measure the solution quality, the work will use the following formula, which is often used in TSP and QAP problems to measure the solution quality.

Gap = ((ASV-BSV)/BSV) *100

Where ASV refers to the average solution value (average overlap) and BSV refers to the best-known solution value reported in the literature.

4) Selection operators: As roulette wheel selection is widely used and consumes the least amount of time and tournament selection can maintain diversity by giving an equal chance to all the individuals to compete [32], the roulette wheel and the tournament selections are selected to be added to the platform.

5) Crossover operators: Several crossover operators, including SCX, OX, CX, PMX, and ERX, have been chosen for inclusion in the platform. Special attention should be paid to SCX, as it is a smart crossover and was one of the best operators for the TSP and QAP problems and is expected to have the same performance in the DNA_FA context. Moreover, SCX has never been used to solve the DNA assembly problem before; therefore, this paper will present the results related to this crossover.

6) Mutation operator: The swap mutation operator and its variants were widely used for DNA_FA, TSP, and QAP [9] and [17]. Combined and adaptive mutations were designed for the QAP problem [22]. These three mutation types are included in the platform.

7) Stopping condition: The GA platform will stop if the solution is not improved at all during a certain number of iterations or a time limit is reached.

IV. Experiments

This section describes the data sets, the experimental setting, and the variable's values in this study.

A. Data Sets

The GA platform will be assessed on data sets produced by next-generation sequencing, the data sets are obtained from the National Center for Biotechnology Information (NCBI)[1]. These data sets are the same benchmarks used in the previous works mentioned in the related works section. This work used 17 data sets with a varying number of fragments, from 25 fragments to 352 fragments. The mean length of the fragment varies between 286 and 512 pb, the description of these data sets is given in Table II.

B. Experimental Setting

The designed algorithm has been implemented in C++ on a Windows 10 computer with a 2 GHz CPU and 16 GB of RAM. The work maintained the parameter values used in TSP and QAP that led to the best results. Based on Table I, this work chose the values for the GA operators, which are described in detail in Table III. The work applied to each data set 60 experiments using different GA parameters and operators. In detail, it applied two types of population size, three types of initialization, two types of selection, and five types of crossover (2*3*2*5 = 60). Since the GA is a stochastic process, each experiment was run 30 times to ensure the satiability of the given results (30*60=1800 experiments). Since there were 17 datasets, the total number of experiments reached more than 30 thousand (17 * 1800=30,600 experiments)

TABLE II. Benchmarks' Datasets, where Total Data Size is Number of Fragments * Mean Fragment Length

Benchmark	Mean fragment length	Number of fragments	Total data size
x60189 4	395	39	15405
x60189 5	286	48	13728
x60189 6	343	66	22638
x60189 7	387	68	26316
m15421 5	398	127	50546
m15421 6	350	173	60550
m15421 7	383	177	67791
j02459 7	405	352	142560
f25_305	307	25	7675
f25_400	400	25	10000
f25_500	500	27	13500
f50_315	315	50	15750
f50_412	412	50	20600
f50_498	498	50	24900
f100_307	307	100	30700
f100_415	415	100	41500
f100_512	512	100	51200

TABLE III. Parameters' and Operators' Values Used in the Experiments

Parameter	Value
Population size	200, 500 individuals.
Initialize type	Randomly, 2-opt heuristics, and greedy
Number of runs	30 runs.
Stopping condition	No improvement for 300 consecutive generations or running time < 5 second.
Selection type	Tournament with size 5, roulette wheel
crossover	SCX, OX, CX, PMX, ERX.
mutation	Randomly pick one of (Swap, adaptive, and combined mutation)
Mutation probability	0.001
Crossover probability	1.0
Total experiments	30,600

[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/

C. Evaluation Metrics

The performance of the GA algorithm will be measured in terms of the following:

- Overlapping scores: should be high. The overlap score measured by calculating the length of the overlap between each fragment and all the existing fragments. The overlap scores were computed using the Smith-Waterman algorithm. Two forms of overlap scores were reported in the results: the best overlap scores out of 30 runs, and the average overlap score for 30 runs.
- Computational complexity (time complexity): should be minimized. The time for the complete assembly process was divided into two stages: the time for calculating the overlap score in the preprocessing stage, and the time for the GA to find the best solution. This paper only showed the time of the GA because this work studying the change in the performance of the GA and also because the time for SW is constant for each dataset regardless of which GA variant is studying.

D. Aspects of Investigations

This work study the effect of some algorithm components on two metrics: the GA running time and the overlap score. It will discuss the effect of population generation, including the population size and the population generation method, the effect of the crossover types, as well as the effect of the selection types. According to the comprehensive experiments, below are the major interesting investigations aspects.

- Crossover types on overlap score when varying the population generation method.
- Population size on overlap score and GA running time.
- Selection types on overlap score and GA running time when varying the population generation method.

V. Results

This section presents and discusses the results obtained from the experiments conducted in this study. The results in this section are organized in subsections. Each subsection reports the results of a specific investigation, as mentioned earlier. In each subsection, the results will be illustrated with tables or pictures and discussed, in addition to summarizing the findings at the end of each subsection.

A. The Effect of Crossover Type on Overlap Score when Varying the Population Generation Method

This section studies the effect of the crossover types on the overlap score when varying the population generation methods while the other GA operators remain constant. For simplicity, the type of selection operation is the tournament, and the population size is 200.

Each of the figures below represents the effect of a specific population generation method on the best overlap score.

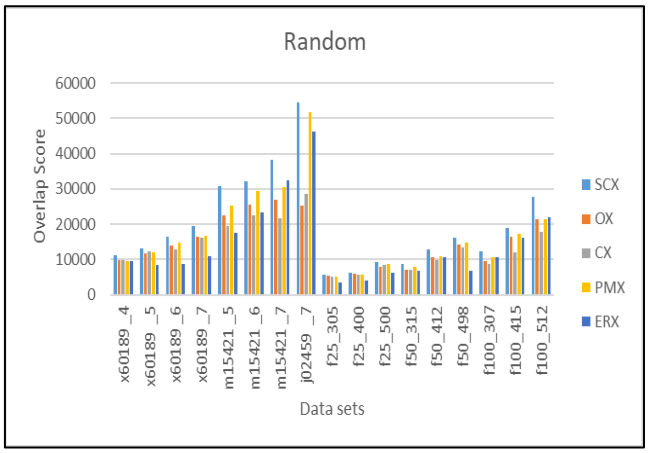

Fig. 2. The effect of crossover types on overlap score when generation population method is random.

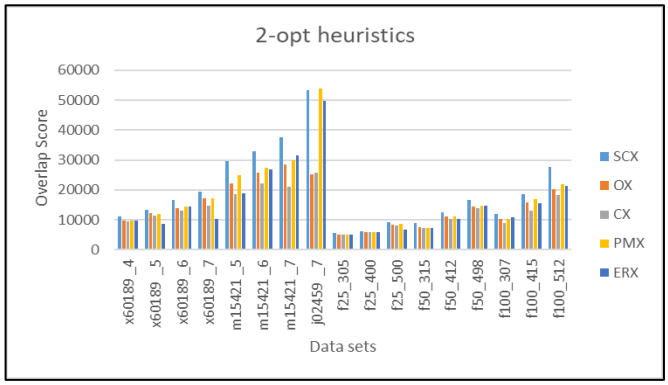

Fig. 3. The effect of crossover types on overlap score when generation population method is 2-opt heuristics.

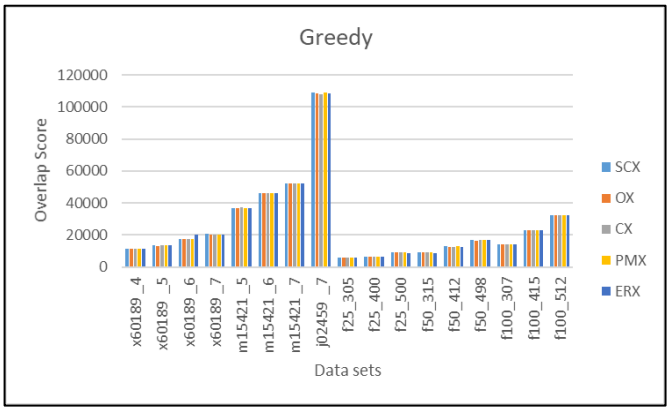

Fig. 4. The effect of crossover types on overlap score when generation population method is greedy.

From Fig. 2, Fig. 3, and Fig. 4, it can be seen that the greedy initialization type is the one that gives the highest overlap score for all the data sets. Moreover, the 2-opt heuristics and random method gave different results but clearly showed that SCX is the best crossover in the majority of cases. Clearly, the SCX has the best accuracy regardless of the population generation methods. SCX is less sensitive to the type of initialization, whatever the type of initialization, it gives good results in every case. Also, the greedy approach

improves the performance of all crossover operators. It seems only when population generation method is greedy, it is not clear if SCX is still the best. This reveals the impact of the population generation method and how creating the population in a smart way from the beginning has been a strong factor in improving the results and reducing the differences between the types of crossovers. In more detail, the greedy improved the SCX overlap results by 11.7% compared to random and 2_opt heuristics, also improved the OX and CX results by 25.9% and 31.01%, respectively. And improve the PMX and ERX by 22.01% and 37.7%, respectively. This demonstrates that the SCX was the least sensitive crossover to the population generation method among the crossover types and maintained the highest overlap score. Thus, further investigation is done by checking the time cost for the greedy method using small and large population sizes.

The results of this study are summarized as follows:

- SCX is not sensitive to the type of initialization, whatever the type of initialization, it gives good results in every case.
- OX, CX, PMX, and ERX are sensitive to the type of generating population, as they perform better with the greedy initialization than with the random and the 2-opt heuristics. Because the greedy is good at generating a good initial population.
- The population generation method has a strong impact on improving the results.

B. *The Effect of the Population Size on Overlap Score and GA Running Time*

This subsection investigates the effect of population size on the overlap score, including the best overlap score and average overlap score. In addition, the effect of population size on the GA's running time.

1) The effect of population size on overlap score: Recall that this paper only report for the SCX; Table IV shows the effect when the initialization method is greedy, the crossover type is SCX, and the selection type is tournament. The "gap" column represents the gap on overlap which calculated by the formula in Section III and the "absolute difference" represents the pure difference between 500 and 200 individuals.

In this investigation, when the crossover is SCX, and the generation method is greedy, and the selection type is tournament, the results show that when the initial population is 60% less, the GA still gives high overlap score in all datasets with an average difference of 0.14%. This is important since decreasing the initial population size decreases the computation significantly.

Moreover, for datasets (f*) this work show a significant increase in accuracy compared to what is reported in the literature. The increase reaches 172.57%, with the 200-population size, and 171.9% with the 500-population size. Moreover, when computing the difference between the best overlap score and average overlap score, there was not a significant difference in performance, where the best overlap score is only 0.44% and 0.42% better than the average overlap score on 200 and 500, respectively. Thus, the paper only reports the best overlap score in the rest of the paper. With regard to the gap, the table shows that our results are better in 9 data sets out of 17. When the absolute difference is negative, that means the 200 size is better than the 500. This was clear for eight data sets, and their performance was equal in four data sets, this makes the smaller size more suitable.

The results of this study are summarized as follows:

- Increase the size of the population increase the computational time, however it may give chance to have good results for the big data sets.
- The population sizes of 200 and 500 individuals do not have a noticeable difference in the quality of the solution; therefore, it is preferable to take the smaller size.

TABLE IV. THE EFFECT OF POPULATION SIZE ON OVERLAP SCORE FOR SCX WHEN THE GENERATION METHOD IS GREEDY AND SELECTION TYPE IS TOURNAMENT

Data set	BSV	200 individuals-greedy			500 individuals-greedy			Gap on the overlap (%)		Absolute difference (500, 200)
		Best overlap	Average overlap	Standard deviation	Best overlap	Average overlap	Standard deviation	200-individuals	500-individuals	
x60189 4	11478	**11298**	**11203.8**	46.99	11272	11195.9	**41.95**	-2.39	-2.46	-26
x60189 5	14161	**13594**	**13498.4**	67.11	13540	13478.4	**12.38**	-4.68	-4.82	-54
x60189 6	18301	**17401**	**17318.9**	41.02	17304	17273.8	**6.96**	-5.37	-5.61	-97
x60189 7	21271	**20546**	**20433.8**	30.40	20527	20431.6	**30.03**	-3.94	-3.95	-19
m15421 5	38746	36972	**36916.6**	20.08	**36975**	36908	**12.55**	-4.72	-4.74	3
m15421 6	48052	**46304**	46226.2	32.61	46240	**46233.7**	**18.38**	-3.80	-3.78	-64
m15421 7	55171	52069	51977.7	74.56	**52077**	**52005.6**	**37.98**	-5.79	-5.74	8
j02459 7	116700	109043	108855	**58.85**	**109056**	**108876**	76.40	-6.72	-6.70	13
f25_305	2271	**5594**	**5594**	**0**	**5594**	**5594**	**0**	146.32	146.32	0
f25_400	3139	**6307**	**6307**	**0**	**6307**	**6307**	**0**	100.92	100.92	0
f25_500	5777	**9170**	**9021.5**	71.45	**9170**	8974.13	**45.71**	56.16	55.34	0
f50_315	4013	**9076**	9072.87	16.87	**9076**	**9076**	**0**	126.09	126.16	0
f50_412	5835	12990	12896.4	**29.14**	**13095**	**12915.3**	48.60	121.02	121.34	105
f50_498	9050	**17070**	**16935.8**	54.22	17012	16910	**27.48**	87.14	86.85	-58
f100_307	7035	**14319**	**14265.3**	16.78	14282	14260.7	**3.96**	102.78	102.71	-37
f100_415	9202	**23008**	**22896.5**	45.55	22993	22894.2	**27.49**	148.82	148.80	-15
f100_512	11881	**32384**	32285.2	41.36	**32340**	**32305**	**26.11**	172.57	171.90	1

2) The effect of population size on GA running time: This section studies the effect of population size on the GA running time. The following tables reveal this effect when the type of crossover is SCX, the type of initialization is greedy, and the selection type is tournament. In addition, Table VI compare the use of complex SCX crossover operators with small population size to the use of simple PMX, ERX, OX and CX crossover operators with large population sizes. The 200 and 500 refer to small and large population sizes, respectively.

- The "GA time" column represents the time for the whole GA to find the result,
- The "greedy time" column represents the time for initializing the population with the greedy method.
- The "overlap after greedy" column represents the overlap score after creating the population.
- The "overlap after GA" column represents the overlap score when the algorithm is done.
- The "Increase in overlap "column represents the percentage increase in the overlap score between the overlap score after greedy and the overlap score at the end of the GA.
- The "Absolute difference in time" column shows the difference in time between 500 and 200 individuals for the GA time.

As indicated earlier, the question posed for discussion is whether the greedy might override the algorithm's performance, is the solution comes from the greedy or the GA finds the solution, is the time spent in creating the population or in finding the solution. For this matter, the following table illustrate the overlap score after creating the population with the greedy method, as well as the overlap score when the algorithm is done. In addition to the time taken to create the population and the time taken by the algorithm to find the solution, the least time and the best solution for every data set are marked in bold.

TABLE V. THE EFFECT OF POPULATION SIZE ON GA RUNNING TIME (IN SEC.). FOR SCX WHEN THE GENERATION METHOD IS GREEDY AND SELECTION TYPE IS TOURNAMENT

Data sets	200 individual-greedy					500 individual-greedy					Absolute difference in time (500-200)
	GA time	Greedy initialization time	Overlap after greedy	Overlap after GA	Increase in overlap (%)	GA time	Greedy initialization time	Overlap after greedy	Overlap after GA	Increase in overlap (%)	
x60189_4	1.93	0.36	10765	**11298**	4.72%	**1.07**	0.81	10865	11252	3.44%	-0.86
x60189_5	1.6	0.59	12250	**13594**	9.89%	**1.83**	1.4	12250	13493	9.21%	0.23
x60189_6	**2.5**	0.99	16272	**17401**	6.49%	**2.5**	2.2	16275	17295	5.90%	0
x60189_7	**1.9**	1.1	19419	20546	5.49%	3.3	2.6	19529	**20641**	5.39%	1.4
m154215	**5.1**	4.1	36606	**36972**	0.99%	6.9	6.4	36616	36951	0.91%	1.8
m154216	**6.4**	4.7	46154	**46304**	0.32%	13.1	12.3	46154	46262	0.23%	6.7
m154217	**5.4**	3.6	51708	**52069**	0.69%	13.7	12.5	51738	52012	0.53%	8.3
j024597	**27.4**	23.1	107398	**109094**	1.55%	60.2	56.7	107504	109051	1.42%	32.8
f25_305	**0.2**	0.1	5594	**5594**	0.00%	0.43	0.36	**5594**	**5594**	0.00%	0.23
f25_400	**0.23**	0.17	6107	**6307**	3.17%	0.5	0.4	**6307**	**6307**	0.00%	0.27
f25_500	1.6	0.2	8550	**9170**	6.76%	**1.2**	0.5	8640	**9170**	5.78%	-0.4
f50_315	**0.7**	0.6	8576	**9076**	5.51%	1.6	1.4	8576	8982	4.52%	0.9
f50_412	**1.03**	0.7	12486	12990	3.88%	2.3	1.8	12492	**13011**	3.99%	1.27
f50_498	**1.5**	0.8	16802	**17051**	1.46%	2.2	1.9	16802	17014	1.25%	0.5
f100_307	**2.6**	2	14100	**14319**	1.53%	5.5	5.3	14103	**14260**	1.10%	2.9
f100_415	**3.6**	1.9	22154	**23008**	3.71%	6.2	4.7	22154	**22993**	3.65%	2.6
f100_512	**4.8**	2.9	31187	**32384**	3.56%	8.6	7.6	31189	32339	3.56%	3.8

The table above shows that a small population size takes less time in the majority of the data sets and gives more opportunity for the algorithm to improve the solution. However, the larger population size takes more time to generate but less time to find the solution. In the case of small data sets, most of the time is taken to create population, while the time taken to find the solution is very small. These results showed that when using a larger population size with the greedy method, the improvement in the solution is small and may be nonexistent in the case of small data sets such as (f25_305, f25_400). Small population sizes are better suited to the greedy method because they allow the algorithm to improve the solution while also taking less time. When the datasets (x60189_4 and f25_500) are excluded, the results in Table V for total time show that there is a 49.21% reduction in time when using a 40% smaller population. Moreover, this table showed us that the greedy method contributed 95% to improving the solution, and the GA improved the solution by 3.51% for the 200-population size and 2.99% for the 500 size. This supports the previous investigation, that SCX with a smaller population size is better.

But it may come to mind that if we choose the larger population size with a simple crossover, could it give an overlap higher than the SCX with the small population size in a reasonable time? So, recalling what previously raised for discussion the following table compare SCX with the smaller population size, with other types of crossovers with a larger population size, for time and overlap score.

The results in Table VI show in 14 out of 17 data sets, using SCX with 40% less population size leads to better results than other crossovers with larger size. In addition to having a 40% smaller population size but comparable accuracy, SCX is also significantly faster than the other crossovers. The results show that SCX is 26.92% faster than PMX in all data sets, except for "x60189 _6" and "f25_500" datasets, and in some datasets, it is 59.46% faster (as in f50_498). Also, SCX is 38.38% faster than ERX in all data sets, except for "x60189 _4" and "f25_500" datasets, and in some datasets, it is 68.75% faster (as in f50_498). Also, SCX is 34.64% faster than OX in all data sets except for x60189_4, and SCX is 32.89% faster than CX in all data sets except for "x60189_4", "x60189_6", and "f25_500". There is a similarity in the performance of all crossovers in the small data sets. But the SCX is still the dominant one. This confirms the results obtained previously that with smallest population size SCX still gives the best solution.

TABLE VI. COMPARING SCX WITH THE SMALLER POPULATION SIZE, WITH OTHER CROSSOVERS WITH A LARGER POPULATION SIZE

Data sets	SCX- 200 individual		PMX-500 individual		ERX-500 individual		OX-500 individual		CX-500 individual	
	Best overlap	Time	Best overlap	Time	Best overlap	Time	Best overlap	time	Best overlap	time
x60189 4	11298	1.93	***11318***	2.2	10817	**1.4**	**11316**	**1.4**	**11252**	**1.6**
x60189 5	***13594***	**1.6**	13304	1.8	12942	1.7	13474	2.1	13324	1.7
x60189 6	***17401***	2.5	16997	**1.8**	16997	3.5	17344	2.6	16272	1.4
x60189 7	***20546***	**1.9**	20536	2.9	19898	3.3	20467	2.8	20477	2.8
m15421 5	***36972***	**5.1**	36964	6.2	36662	6.4	36922	7.3	36899	7
m15421 6	***46304***	**6.4**	46240	7.3	46162	7.4	46287	9.3	46240	7.3
m15421 7	***52069***	**5.4**	52011	11.1	51728	11.9	52011	8.1	52011	11.4
j02459 7	***109094***	**27.4**	108832	34.8	107557	52.1	108832	50	109012	36.3
f25_305	***5594***	**0.2**	***5594***	0.3	**5594**	0.3	5594	0.4	5594	0.3
f25_400	***6307***	**0.2**	***6307***	0.3	6114	0.4	6307	0.4	6307	0.4
f25_500	***9170***	1.6	9013	0.8	8650	**0.4**	9130	2	9122	0.7
f50_315	***9076***	**0.6**	***9076***	1.4	8585	1.7	9076	1.5	9076	1.4
f50_412	***12990***	**1.03**	12960	1.3	12564	1.5	12192	1.7	12886	1.3
f50_498	17051	**1.5**	***17163***	3.7	16868	4.8	17089	4.3	17163	3.2
f100_307	***14319***	**2.6**	14260	3.9	14105	4.8	14314	4.6	14314	6.5
f100_415	***23008***	**3.6**	22854	**3.6**	22163	6.3	22854	3.7	22878	3.9
f100_512	32384	**4.8**	***32352***	5.1	31823	6.5	32266	6.3	32254	6.3

The results of this study are summarized as follows:

- The greedy method had a clear impact on the GA performance and contributed to improving the solution by 95%.
- Most of the time is spent on creating the population especially with the large population size.
- Smart crossover like SCX with small population size is better than simple crossovers with large population size.

C. The Effect of Selection Types Varying the Population Generation on the Overlap Score

This section studies the effect of the initialization and selection types on the overlap score. The following figures show this effect when the type of crossover is SCX and the population size is 200, since previous investigations show that a small population size is more suitable.

Fig. 5, Fig. 6, and Fig. 7 show the effect of initialization types and selection types on the best overlap score. Clearly, the greedy initialization type gives a better overlap score than the random and 2-opt heuristics in 15 data sets out of 17.

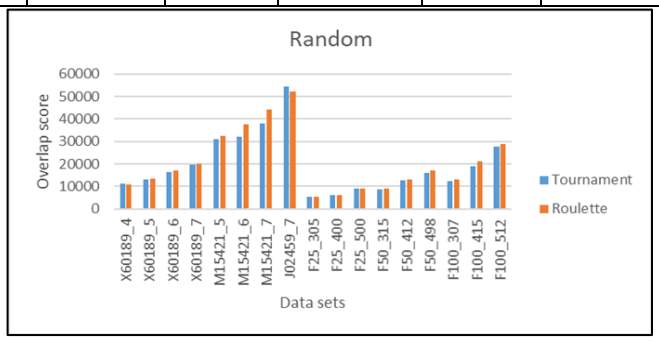

Fig. 5. Effect of selection types with random initialization on best overlap score.

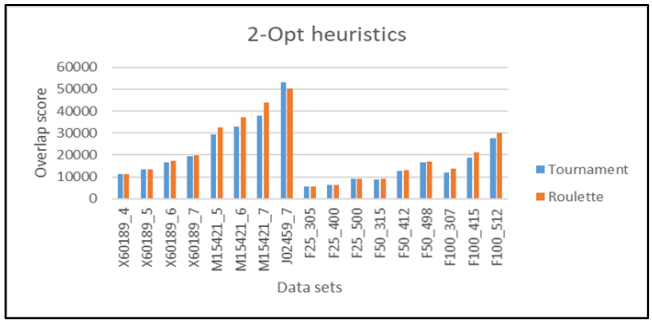

Fig. 6. Effect of selection types with 2-opt heuristics initialization on best overlap score.

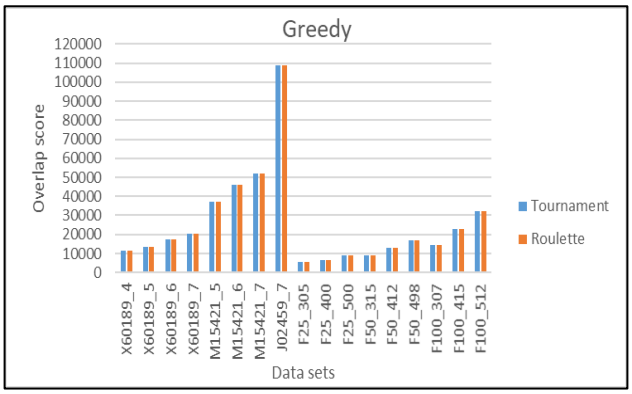

Fig. 7. Effect of selection types with greedy initialization on best overlap score.

Fig. 8, Fig. 9, and Fig. 10 show the effect of initialization and selection types on the average overlap score. The greedy initialization still dominates the random and 2-opt heuristics for the average overlap score as well, giving a higher average in 16 out of 17 data sets. Additionally, the roulette wheel selection is still better than the tournament with the random and 2-opt heuristics. However, with the greedy initialization, the tournament is better.

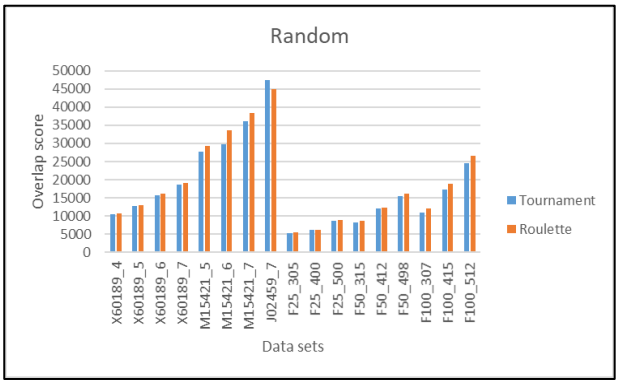

Fig. 8. Effect of selection types with random initialization on average overlap score.

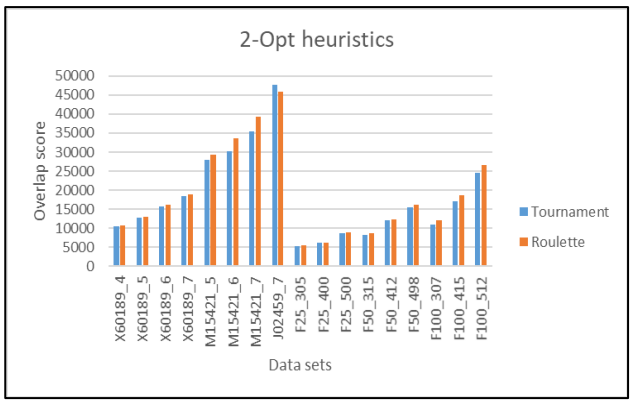

Fig. 9. Effect of selection types with 2-opt heuristics initialization on average overlap score.

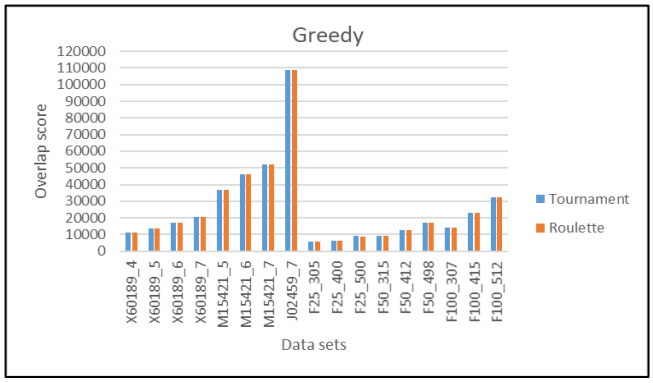

Fig. 10. Effect of selection types with greedy initialization on average overlap score.

The results of this study are summarized as follows:

- The greedy initialization type is the best among the majority of the data sets for the best overlap scores and the best among all the data sets for the average overlap scores.
- The random initialization type is better than the 2-opt for the best overlap score, but their performance is almost similar for the average overlap score.
- The tournament selection type is better than the roulette wheel selection for the best and average overlap scores with the greedy.

D. *The Effect of Selection Types Varying the Population Generation Method on GA Running Time*

Table VII illustrates the results obtained when studying the effect of initialization and selection types on the GA running time. It shows this effect when the type of crossover is SCX, and the population size is 200. The GA time column represents the time it took for the GA to find the result. The least time for every data set is marked in bold green when the selection type is the tournament and marked in bold blue if the selection type is the roulette wheel. The gap column shows the difference in time between the generating methods (i.e., greedy, random, 2-opt heuristic).

Table VII shows that the greedy initialization type is the best from the viewpoint of time complexity. The random and 2-opt heuristics types take more time than the greedy, but the random-type records less time for 12 data sets out of 17, while the 2-opt heuristics record less time for three data sets and equal time for two. As for the selection type, the roulette wheel selection dominates the tournament selection by recording the least time for 14 data sets out of 17. The gap confirms as in the previous section that the greedy generating method is better than the random and 2-opt, and the random is better than 2-opt. In more detail, the results show that the greedy initialization results are fast compared to both random and the 2-opt heuristic in most of the datasets. Except for datasets M15421_7 and J02459_7, the greedy approach is 47.39% and 48.17% faster than random, and 2-opt heuristics when selection type is tournament. Also, greedy approach is 66.90% and 67.19% faster than random, and 2-opt heuristics when selection type is roulette. For the other crossovers OX, CX, PMX, and ERX, the

tournament was the best in terms of solution quality, and the roulette was the fastest, because the tournament chose multiple parents every time and compared them to pick the better one.

The results of this study are summarized as follows:

- The greedy initialization type takes less time than the random and the 2-opt heuristics in almost all cases except for the large data set (J02459_7), because the greedy method takes time to create the population, the larger the data, the more comparisons that greedy makes, and therefore it takes longer time.
- The random initialization type is faster than the 2-opt heuristics in 12 data sets out of 17, this is an expected result.
- The roulette wheel selection type is faster than the tournament selection, but the tournament is better for solution quality.

TABLE VII. THE EFFECT OF INITIALIZATION TYPES AND SELECTION TYPES ON THE GA RUNNING TIME (IN SEC.) FOR SCX CROSSOVER ON REAL DATASETS THE BOLD GREEN COLOR REFERS TO THE LEAST TIME FOR THE CORRESPONDING DATA SET WHEN THE SELECTION TYPE IS TOURNAMENT, AND THE BOLD BLUE COLOR REFERS TO THE LEAST TIME WHEN THE SELECTION IS ROULETTE

Data set	Select	random	2-opt	greedy	gap	gap	gap
		GA time	GA time	GA time	Greedy-random	Greedy-2-opt	Random-2-opt
X60189_4	Tournament	4.27	4.3	2.17	-2.1	-2.13	-0.03
	Roulette	4.27	4.43	**0.93**	-3.34	-3.5	-0.16
X60189_5	Tournament	4.4	4.57	3.63	-0.77	-0.94	-0.17
	Roulette	4.7	4.77	**1.37**	-3.33	-3.4	-0.07
X60189_6	Tournament	4.6	4.73	3.63	-0.97	-1.1	-0.13
	Roulette	4.73	4.77	**1.27**	-3.46	-3.5	-0.04
X60189_7	Tournament	4.7	4.7	2.57	-2.13	-2.13	0
	Roulette	4.8	4.8	**1.3**	-3.5	-3.5	0
M15421_5	Tournament	4.87	4.97	4.1	-0.77	-0.87	-0.1
	Roulette	5	5	**3.7**	-1.3	-1.3	0
M15421_6	Tournament	5.03	5	5	-0.03	0	0.03
	Roulette	4.87	5	**4.13**	-0.74	-0.87	-0.13
M15421_7	Tournament	4.93	4.93	6.13	1.2	1.2	0
	Roulette	**4.77**	5	**4.77**	0	-0.23	-0.23
J02459_7	Tournament	**5.1**	**5.1**	25.97	20.87	20.87	0
	Roulette	5.17	5.17	38.27	33.1	33.1	0
F25_305	Tournament	4	4.07	0.23	-3.77	-3.84	-0.07
	Roulette	4.57	4.4	**0.2**	-4.37	-4.2	0.17
F25_400	Tournament	4.33	4.4	**0.2**	-4.13	-4.2	-0.07
	Roulette	4.47	4.53	**0.2**	-4.27	-4.33	-0.06
F25_500	Tournament	4.3	4.47	1.6	-2.7	-2.87	-0.17
	Roulette	4.6	4.5	**1.13**	-3.47	-3.37	0.1
F50_315	Tournament	4.4	4.53	0.87	-3.53	-3.66	-0.13
	Roulette	4.47	4.63	**0.73**	-3.74	-3.9	-0.16
F50_412	Tournament	4.3	4.63	**1.17**	-3.13	-3.46	-0.33
	Roulette	4.6	4.67	1.23	-3.37	-3.44	-0.07
F50_498	Tournament	4.47	4.63	2.37	-2.1	-2.26	-0.16
	Roulette	4.73	4.63	**0.8**	-3.93	-3.83	0.1
F100_307	Tournament	4.77	4.67	**2.53**	-2.24	-2.14	0.1
	Roulette	4.93	4.97	2.97	-1.96	-2	-0.04

F100_415	Tournament	4.8	4.77	4.1	-0.7	-0.67	0.03
	Roulette	4.93	4.93	**2.27**	-2.66	-2.66	0
F100_512	Tournament	4.77	4.8	4.53	-0.24	-0.27	-0.03
	Roulette	4.93	4.97	**3.8**	-1.13	-1.17	-0.04

VI. DISCUSSION

This section discusses the findings that emerged from the results presented in the Results section. And conclude that the small population size (i.e., 200 individuals) is more suitable for most cases. And the greedy type of initialization is the best when look for good overlap score results and time. Furthermore, the results show that the roulette wheel selection type is more suitable than the tournament selection in the context of time, but the tournament is better in the quality of the solution. Also, this work shows that the SCX crossover is the best in the context of best overlap score and average overlap score.

This study has multiple GA versions, but in comparison to the previous works, we selected the best version we got. Moreover, the comparisons are divided as follows:

- Previous works that used the GA, the comparison is presented in Table VIII.
- Previous works that used other metaheuristics algorithms, the comparison is presented in Table IX.

Table VIII compares the designed GA results and the other previous GA work's results in the context of the overlap score. The best results are marked in bold. The "difference in percentage" column shows the difference between our best results and those of the previous works. Clearly, our results for the F-series data sets (from F25_305 to F100_512) dominate all the previous work's results. This work got less than the best results of previous works in eight data sets out of 17, however, our results are still better than [23], [25], and [9] for these data sets.

Moreover, this work obtained better results than the results of all the previous works in nine data sets out of 17.

With regard to the time, the results were given in a reasonable time and there is no significant change or difference in time, because the dominant time is actually not the GA time but the assembly time (i.e., in our case, the Smith-Waterman algorithm.). GA is useful when the data set is large, and this is expected because GA avoids large search space. The results show that the designed GA gives the results in less time for large data sets such as M15421_6, M15421_7, and J02459_7, which have several fragments that vary from 173 to 352 characters.

TABLE VIII. COMPARISON OF BEST SOLUTIONS BETWEEN OUR GA RESULTS AND OTHER GA ALGORITHMS RESULTS FROM THE LITERATURE IN THE CONTEXT OF OVERLAP SCORE

Data sets	Our best GA	REF. [9]	REF. [10]	REF. [31]	Difference in percentage %
X60189_4	11272	6488	**11478**	**11478**	-1.79%
X60189_5	13475	8655	14027	**14161**	-4.84%
X60189_6	17357	9943	**18301**	**18301**	-5.16%
X60189_7	20559	11546	**21268**	21212	-3.33%
M15421_5	36972	22598	**38726**	38694	-4.53%
M15421_6	46240	29469	48048	**48052**	-3.77%
M15421_7	52077	32744	**55072**	55071	-5.44%
J02459_7	109043	68736	115301	**116487**	-6.39%
F25_305	**5594**	2271	-	596	146.32%
F25_400	**6307**	3139	-	777	100.92%
F25_500	**9170**	5777	-	921	58.73%
F50_315	**9076**	4013	-	1578	126.16%
F50_412	**12967**	5835	-	1572	122.23%
F50_498	**16902**	9050	-	1570	86.76%
F100_307	**14318**	7035	-	2780	103.53%
F100_415	**22911**	9202	-	2846	148.98%
F100_512	**32384**	11881	-	2717	172.57%

TABLE IX. COMPARISON OF BEST SOLUTIONS BETWEEN OUR GA RESULTS AND OTHER METAHEURISTICS NON-GA ALGORITHMS RESULTS FROM THE LITERATURE IN THE CONTEXT OF OVERLAP SCORE

Data set	Our best GA	REF. [23]	REF. [3]	REF. [25]	REF. [28]	REF. [27]	REF. [33]	Difference in Percentage %
X60189_4	11272	-	**11478**	3046	**11478**	11451	**11478**	-1.79%
X60189_5	13475	-	13642	-	**14161**	13932	**14161**	-4.84%
X60189_6	17357	-	**18301**	-	**18301**	18204	**18301**	-5.16%
X60189_7	20559	-	20921	3022	**21271**	20968	**21271**	-3.35%
M15421_5	36972	5821	38686	6443	**38746**	38454	**38746**	-4.58%
M15421_6	46240	6713	47669	7041	**48052**	-	**48052**	-3.77%
M15421_7	52077	6291	54891	6537	**55171**	54666	**55171**	-5.61%
J02459_7	109043	-	114381	-	**116700**	115405	**116700**	-6.56%
F25_305	**5594**	-	-	-	596	-	596	838.59%
F25_400	**6307**	-	-	-	777	-	777	711.71%
F25_500	**9170**	-	-	-	921	-	921	895.66%
F50_315	**9076**	-	-	-	1581	-	1581	474.07%
F50_412	**12967**	-	-	-	1573	-	1573	724.35%
F50_498	**16902**	-	-	-	1570	-	1570	976.56%
F100_307	**14318**	-	-	-	2793	-	2793	412.64%
F100_415	**22911**	-	-	-	2860	-	2860	701.08%
F100_512	**32384**	-	-	-	2732	-	2732	1085.36%

VII. CONCLUSION

This paper is a continuation of our previous work [4] to solve the DNA fragment assembly problem. As was pointed out in the introduction to this paper, the DNAFA is an optimization problem that attempts to reconstruct an original DNA sequence by finding the shortest DNA sequence from a given set of fragments. We have designed a platform for the genetic algorithm, from which more than one version of the genetic algorithm can be deduced to solve this problem. The design was inspired by the good designs that solved TSP and QAP problems. This study is the first to our knowledge that examines the genetic algorithm for the DNAFA problem from this perspective. In more detail, this study has gone a long way towards investigating the effect of genetic algorithm operators on the quality of the solution to the DNAFA problem. The study focused on investigating the effect of the initial population, size of the population, selection types, and crossover types.

This paper recorded the important results and came out with some findings, the most obvious finding to emerge from this study is that the SCX crossover is a smart crossover and has never been used before with DNA_FA, SCX crossover gave better results compared to the rest of the studied crossover types. Furthermore, the results show that the population generation method has the greatest influence on GA performance in terms of time and solution quality. Also, we configured the best-designed GA variant that outperforms the existing GA algorithms solving the DNAFA problem. This GA variant features the use of 200 individuals for the population size along with the greedy method for initializing the population, tournament selection, and SCX crossover. This study has found that generally, the size of the population does not significantly affect the quality of the solution, especially if the type of initialization is good. The results were good and competitive compared to the results of previous works. Our design showed that the results were better than all previous results from the literature for some data sets.

There is still ample scope to study and solve this problem, an interesting point will be how to find a way to create the population intelligently and without consuming a lot of time, given that the greedy is time consuming. Moreover, further research might explore or investigate the effect of other GA operators (e.g., mutation types and stooping conditions). Also, investigate the effect of combining different types of GA operators (initialization types, crossover operators, and mutation operators) on the results. Another possible area of future research would be to combine the data sets (next generation with the third generation).

ACKNOWLEDGMENT

The authors extend their appreciation to the Deanship of Scientific Research at Imam Mohammad Ibn Saud Islamic University for funding and supporting this work through Graduate Students Research Support Program.

REFERENCES

[1] R. S. Verma and S. kumar, “DSAPSO: DNA sequence assembly using continuous Particle Swarm Optimization with Smallest Position Value rule,” in 2012 1st International Conference on Recent Advances in

Information Technology (RAIT), Mar. 2012, pp. 410–415. doi: 10.1109/RAIT.2012.6194455.

[2] 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.

[3] E. Alba and G. Luque, "A Hybrid Genetic Algorithm for the DNA Fragment Assembly Problem," in Recent Advances in Evolutionary Computation for Combinatorial Optimization, vol. 153, C. Cotta and J. van Hemert, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 101–112. doi: 10.1007/978-3-540-70807-0_7.

[4] H. Bennaceur, M. Almutairy, and Alqhtani, Nora, "An Investigative Study of Genetic Algorithms to Solve the DNA Assembly Optimization Problem," Int. J. Adv. Comput. Sci. Appl., vol. 11, no. 10, p. 12, 2020.

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

[6] 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. doi: 10.1007/978-3-319-55849-3_5.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] Z. H. Ahmed, "GENETIC ALGORITHM FOR THE TRAVELING SALESMAN PROBLEM USING SEQUENTIAL CONSTRUCTIVE CROSSOVER," 2010.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

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

[21] 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.

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

[23] 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.

[24] S. Lin and B. W. Kernighan, "An Effective Heuristic Algorithm for the Traveling-Salesman Problem," Oper. Res., vol. 21, no. 2, pp. 498–516, 1973.

[25] K.-W. Huang, J.-L. Chen, C.-S. Yang, and C.-W. Tsai, "A memetic particle swarm optimization algorithm for solving the DNA fragment assembly problem," Neural Comput. Appl., vol. 26, no. 3, pp. 495–506, Apr. 2015, doi: 10.1007/s00521-014-1659-0.

[26] R. Indumathy, S. Uma Maheswari, and G. Subashini, "Nature-inspired novel Cuckoo Search Algorithm for genome sequence assembly," Sadhana, vol. 40, no. 1, pp. 1–14, Feb. 2015, doi: 10.1007/s12046-014-0300-3.

[27] E. Alba and G. Luque, "A New Local Search Algorithm for the DNA Fragment Assembly Problem," in Evolutionary Computation in Combinatorial Optimization, vol. 4446, C. Cotta and J. van Hemert, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 1–12. doi: 10.1007/978-3-540-71615-0_1.

[28] G. Minetti, G. Leguizamón, and E. Alba, "An improved trajectory-based hybrid metaheuristic applied to the noisy DNA Fragment Assembly Problem," Inf. Sci., vol. 277, pp. 273–283, Sep. 2014, doi: 10.1016/j.ins.2014.02.020.

[29] J. S. Firoz, M. S. Rahman, and T. K. Saha, "Bee algorithms for solving DNA fragment assembly problem with noisy and noiseless data," in Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference - GECCO '12, Philadelphia, Pennsylvania, USA, 2012, p. 201. doi: 10.1145/2330163.2330192.

[30] F. Majid al-Rifaie and M. Majid al-Rifaie, "Maximising Overlap Score in DNA Sequence Assembly Problem by Stochastic Diffusion Search," in Intelligent Systems and Applications, vol. 650, Y. Bi, S. Kapoor, and R. Bhatia, Eds. Cham: Springer International Publishing, 2016, pp. 301–321. doi: 10.1007/978-3-319-33386-1_15.

[31] 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.

[32] 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.

[33] A. Ben Ali, G. Luque, and E. Alba, "An efficient discrete PSO coupled with a fast local search heuristic for the DNA fragment assembly problem," Inf. Sci., vol. 512, pp. 880–908, Feb. 2020, doi: 10.1016/j.ins.2019.10.026.