Maximising Overlap Score in DNA Sequence Assembly Problem by Stochastic Diffusion Search

Fatimah Majid al-Rifaie and Mohammad Majid al-Rifaie

Abstract This paper introduces a novel study on the performance of Stochastic Diffusion Search (SDS)—a swarm intelligence algorithm—to address DNA sequence assembly problem. This is an NP-hard problem and one of the primary problems in computational molecular biology that requires optimisation methodologies to reconstruct the original DNA sequence. In this work, SDS algorithm is adapted for this purpose and several experiments are run in order to evaluate the performance of the presented technique over several frequently used benchmarks. Given the promising results of the newly proposed algorithm and its success in assembling the input fragments, its behaviour is further analysed, thus shedding light on the process through which the algorithm conducts the task. Additionally, the algorithm is applied to overlap score matrices which are generated from the raw input fragments; the algorithm optimises the overlap score matrices to find better results. In these experiments real-world data are used and the performance of SDS is compared with several other algorithms which are used by other researchers in the field, thus demonstrating its weaknesses and strengths in the experiments presented in the paper.

1 Introduction

Every cell in the body has a complete copy of about 3.2 billion[1] DNA base pairs or letters which build the human genome [22]. DNA has all the information necessary to build the whole living organism. Although the letters of the genetic alphabet

[1]In American English, 1 billion is equated to a thousand million (i.e. 1,000,000,000).

F. Majid al-Rifaie (✉)
Department of Computer Science, Kristianstad University,
S-291 88 Kristianstad, Sweden
e-mail: fatimah.al-rifaie0018@hkr.se

M. Majid al-Rifaie
Computing Department, Goldsmiths, University of London,
London SE14 6NW, UK
e-mail: m.majid@gold.ac.uk

Adenine (A), Thymine (T), Cytosine (C) and Guanine (G) are meaningless on their own, they are joined into useful instructions in genes. It is interesting to note that more than 99 % of human's structure is genetically identical [21].

Imagine having several copies of the same book written in a language you cannot understand. Every page of each copy has been randomly cut into horizontal strip and a piece from one copy may overlap a piece from another copy. Assuming that some of strips are missing and some are splashed with ink, and maybe some of the books have random typos and error throughout, in different places. Try to arrange all the strips and assemble a single copy of the original book without any typos or errors. This process is similar to the important task of DNA sequencing.

In this work, novel applications of a swarm intelligence technique is introduced as a proof of principle. The swarm intelligence algorithm used is Stochastic Diffusion Search (SDS) which has a good potential to work in large search spaces and noisy environments. This algorithm is explained in the paper and its application to the problem is detailed.

This paper starts by presenting the swarm intelligence algorithm along with a simple example demonstrating its use. Then a brief introduction is given to the DNA assembly problem and the solutions offered so far using swarm intelligence techniques. Subsequently, some experiments are designed and the performance of SDS is investigated using various benchmarks and then its performance is contrasted against several other techniques. Finally, the reason behind using SDS is further elaborated and the difference between utilising SDS and Smith–Waterman algorithm is discussed. Additionally in the second set of experiments, initially SDS is shown to be generating the overlap score matrices (a task that is historically accomplished by Smith–Waterman algorithm); and then the details of using the SDS generated matrices to optimise the overlap scores are described. This is followed by a conclusion and directions for future research.

2 Swarm Intelligence

The paper is based on swarm intelligence which is one of the categories of artificial intelligence. Swarm intelligence is based on the study of behaviour of simple individuals (e.g. ant colonies, bird flocking, and honey bees, animal herding) that mimics the behaviour of swarms of social insects or animals [5]. More and more researches are interested in this field as swarm intelligence offers new ways of designing intelligence systems.

Among the successful examples of optimisation techniques inspired by swarm intelligence are: ant colony optimisation (inspired by foraging behaviour of real ant colonies) and particle swarm optimisation (inspired by bird flocking) [5]. In this work, Stochastic Diffusion Search (SDS) [1] algorithm is used. This algorithm also belongs to the category of swarm intelligence and is based on mimicking the foraging behaviour of one type of ants *Leptothorax acervorum*. More details about SDS are provided in Algorithm 1 and a simple example is presented next.

```
Algorithm 1 SDS algorithm

Initialisation phase: Allocate agents to random hypotheses in the search space
Until (all agents congregate on the best hypothesis)
  • Test phase
      – Each agent evaluates its hypothesis
      – Each agent is classified into active or inactive
  • Diffusion phase
      – Each inactive agent randomly chooses another agent to communicate with. If the inactive
        agent selects another inactive agent, no information will be transferred between the agents.
        Therefore the selecting agent should choose another hypothesis randomly. If the selected agent
        is active, the active agent communicate its hypothesis to the selecting agent
End
```

2.1 *Search Example with SDS*

In the following example the aim is to find a 4-letter model (Table 1) in a 32-letter search space (Table 2).

There are four agents; and a hypothesis identifies four adjacent letters in the search space (e.g. hypothesis '6' refers to D-N-A-F; hypothesis '17' refers to A-S-S-E, etc.). In the first step, each agent initially picks a random hypothesis from the search space (see Table 3). Assume that:

- The first agent points to the 27th entry of the search space and randomly picks one of the letters (e.g. the fourth one, (B): $\boxed{\text{O}}\boxed{\text{B}}\boxed{\text{L}}\boxed{\textbf{E}}$
- The second agent points to the 14th entry and randomly picks the first letter (E): $\boxed{\textbf{E}}\boxed{\text{N}}\boxed{\text{T}}\boxed{\text{A}}$

Table 1 MODEL

Index:	0	1	2	3
Model:	D	N	A	F

Table 2 Search space

Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Search space	T	H	I	S	I	S	D	N	A	F	R	A	G	M	E	N
Index:	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Search space	T	A	S	S	E	M	B	L	Y	P	R	O	B	L	E	M

Table 3 Initialisation and iteration 1

Agent no	1	2	3	4	5
Hypothesis position	27	14	8	20	4
	OBL**E**	**E**NTA	A**F**RA	EM**B**L	I**S**DN
Letter picked	4th	1st	2nd	3rd	2nd
Status	×	×	×	×	×

- The third agent refers to the 8th entry in the search space and randomly picks the second letter (F): $\boxed{A}\boxed{\mathbf{F}}\boxed{R}\boxed{A}$
- The fourth agent goes the 20th entry and randomly picks the third letter (B): $\boxed{E}\boxed{M}\boxed{\mathbf{B}}\boxed{L}$
- The fifth agent refers to the 4th entry in the search space and randomly picks the second letter (S): $\boxed{I}\boxed{\mathbf{S}}\boxed{D}\boxed{N}$

The letters picked are compared to the corresponding letters in the model that is D-N-A-F (see Table 1). In this case:

- The fourth letter from the first agent (E) is compared against the fourth letter from the model (F) and because they are not the same, the agent is set inactive.
- For the second agent, the first letter (E) is compared with the first letter from the model (D) and because they are not the same, the agent is set inactive.
- For the third, fourth and fifth agents, letters 'F', 'B' and 'S' are compared against 'N', 'A' and 'N' from the model. Since none of the letters correspond to the letters in the model, the status of the agents are set inactive.

In the next step, each inactive agent chooses another agent and gets the same hypothesis if the selected agent is active. If the selected agent is inactive, the choosing agent generates a random hypothesis. Assume that the first agent selects the third one; since the third agent is inactive, the first agent chooses a new random hypothesis from the search space (e.g. 6). Figure 1 shows communication between agents.

The process is repeated for the other four agents. When the agents are inactive, they all choose new random hypotheses (see Table 4).

In Table 4, the first, third, fourth and fifth agents do not refer to their corresponding letter in the model, therefore they become inactive. The second agent, with hypothesis '6', chooses the second letter (N) and compares it with the second letter of the model (N). Since the letters are the same, the agent becomes active.

In this case, consider the following communication between the agents: (see Fig. 2)

- The third and fourth agents choose the second one
- The first agent chooses the third one

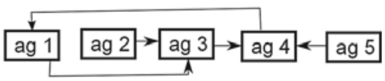


Fig. 1 Agent Communication 1

Table 4 Iteration 2

Agent no	1	2	3	4	5
Hypothesis position	1	6	22	12	17
	HISI	D**N**AF	BLY**P**	GM**E**N	AS**S**E
Letter picked	1st	2nd	4th	3rd	3rd
Status	×	√	×	×	×

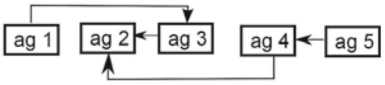


Fig. 2 Agents Communication 2

- The fifth agent chooses the fourth one

At this stage, the first and fifth agents, which chose the inactive third and fourth agents, have to choose other random hypotheses from the search space. However, agents three and four use the hypothesis of the active agent, two.

This process is repeated until all agents are active pointing to the location of the model inside the search. Depending on the problem, there are alternative termination strategies; for instance, in some cases, SDS algorithm is set to terminates only if all agents are active and refer to the same hypothesis.

The next section, provides a brief introduction to DNA assembly problem, stating the main phases and the major challenges faced by researchers in this field. This is followed by an overview of some of the algorithms that aimed to address the problem. Afterwards the experiments and results are reported.

3 Understanding DNA Assembly

There is no single solution available for NP-hard problems [22] and it is often not possible to find an extremely good algorithm that solves such problems [18].

In DNA assembly, a process is required to join the relevant fragments together. In other words, the overlapping fragments are to be assembled back into the original DNA sequence. Therefore, the goal of genome projects is to reconstruct the original genome sequence of an organism. To achieve the goal, DNA fragment assembly process is divided into three phases [6, 9]:

1. **Overlap Phase** is tasked to find the common sequence among the prefix of one sequence and suffix of another.
2. **Layout Phase** uses alignment strategies to determine the order of fragments based on high overlap scores and according to the level of similarity.
3. **Consensus Phase** assembles all fragments into the consensus sequence and omits the similar parts.

The quality of a consensus sequence is measured by the term coverage [6, 19]. Coverage is evaluated according to the following equation:

$$\text{Coverge} = \frac{\sum_{i=1}^{n} \text{length of fragment } i}{\text{target sequence length}} \tag{1}$$

where n is the number of fragments.

The higher the coverage, the higher the probability of covering original genome, the higher the correctness of the assembled parts, the fewer the number of the gaps, and the better the result [6, 11].

The Layout Phase is the most complex step due to the difficulty of finding the best overlap. This difficulty is caused by the following challenges [6, 11]:

- **Unknown orientation**: After the original sequence is divided into many fragments, the direction may change.
- **Base call errors:** substitution, insertion, and deletion errors are types of base call error. The errors happen because of experimental errors in the electrophoresis procedure that affects the finding of fragment overlaps.
- **Incomplete coverage:** It occurs when the algorithm cannot assemble a given fragments into one contig.
- **Repeated regions:** the problem occurs when some sequences are repeated two or more times in the DNA. None of the current assembly programs can solve the problem without an error [18].
- **Chimeras and contamination:** Chimeras arise when two fragments that are not adjacent, or overlapping on the target molecule, join together into one fragment. Contamination occurs due to the incomplete purification of the fragment from the vector DNA.

3.1 DNA Sequence Assembly and Swarm Intelligence

DNA Assembly problem is still open to a large extent because of the principal issue of “scaling up to real organism”. Some of the swarm intelligence and evolutionary algorithms, such as genetic algorithms and ant colony optimisation have been used for the fragment assembly problem focusing on the overlap, layout and consensus approach [14].

In 1995, Rebecca Parsons and Johnson created performance improvements for a genetic algorithm applied to the DNA sequence assembly problem [17]. In 2003 Kim and Mohan used a new parallel hierarchical adaptive genetic algorithm. The method is reported as accurate and noise-tolerant compared to previous methods [10]. In the same year, Meksangsouy and Chaiyaratana proposed ant colony optimisation. The goal of the search was to find the right order and orientation of each fragment to create a consensus sequence [15]. In 2005 Fang proposed approach speeded up the searching process and maximised the similarity or overlaps between given fragments [7]. Alba and Luque presented several methods, including genetic algorithm, a CHC method, scatter search algorithm, and simulated annealing to solve accurately DNA Assembly problem in 2005 [12]. They also proposed a local search method named PALS in 2007 [2]. In 2008, Luque and Alba studied the behaviour of a hybrid heuristic algorithm that combines a heuristic, PALS, with a meta-heuristic, a genetic algorithm, achieving an assembler to find optimal solutions for large instances of this DNA assembly problem [3]. In 2010 Kubalik presented a method called Prototype Optimisation with Evolved Improvement Steps (POEMS). Also in the same year Minetti and Alba presented a paper about how noiseless and noisy instances of this problem are handled by three algorithms: problem aware local search, simulated annealing and genetic algorithms [16].

There are some other solutions that are proposed in 2011 for DNA sequence assembly problem using Particle Swarm Optimisation (PSO) with Shortest Position Value (SPV) rule [22]. In 2012 Firoz analysed and discussed the performance of two swarm intelligence based algorithms namely Artificial Bee Colony (ABC), and Queen Bee Evolution Based on Genetic Algorithm (QEGA) to solve the fragment assembly problem [8]. In 2013 Fernandez-Anaya et al. designed a nature inspired algorithm (PPSO + DE) based on Particle Swarm Optimisation and Differential Evolution [13].

4 Experiments and Results I

In order to understand the process through which SDS is adopted and adapted for DNA sequence assembly problem, a number of fragments are used in the experiments. The fragments are the input of the program and the program is responsible to assemble the fragments and create one long sequence. This is achieved by taking a fixed number of characters from the end of the first fragments and trying to find those characters in the other fragments using SDS algorithm. Once the other fragment is found, the two fragments are joined and the repeated part is deleted from one of the fragments. This will create a longer fragment. This process is repeated until all fragments are joined. The steps required for SDS to assemble a set of fragments are detailed in Algorithm 2.

In the experiments reported in this paper the agent size is empirically set to 100 and the model size for SDS is set to 50. Table 5, as proposed by Mallén-Fullerton et al. [14], shows the benchmarks used by SDS for DNA assembly.

Using the benchmarks provided, SDS algorithm assembles the entire sequences correctly. Table 6 shows the performance of SDS when assembling the nine aforementioned benchmarks. Each benchmark is assembled 50 times. As the table shows, the larger the coverage, the more SDS iterations it takes to fully assemble the datasets. While the number of overall algorithm cycles needed follow the same structure, there are some exception caused by the order of the fragments. Observing the sum of active agents over all the iterations and their consistent proximity (check the negligible dif-

Algorithm 2 DNA sequence assembly using SDS

```
Choose a model from the end of 1st fragment in the search space
While (true)
• Use SDS to search the model in the fragments
  – If no matching fragment is found
    · Choose the model from the beginning of the first fragment
    · Use SDS to search the model in the fragments
    · If no matching fragment is found
        Break
• Compile a list of fragments where the model is found
• Pick the fragment ($j^{th}$) with the maximum similarity (based on agents activity)
  – assemble fragments $i$ and $j$.
  – Delete the $j^{th}$ fragment
  – Choose a new model from the end of assembled fragment
End While
```

Table 5 Benchmark datasets

Benchmark	Coverage	Mean fragment length	Number of fragments	Original sequence length
x60189 4	4	395	39	3,835
x60189 5	5	286	48	
x60189 6	6	343	66	
x60189 7	7	387	68	
m15421 5	5	398	127	10,089
m15421 6	6	350	173	
m15421 7	7	383	177	
j02459 7	7	405	352	20,000
bx842596 7	7	703	773	77,292

ference between the median and the mean, as well as the value of the standard deviation) shows the robustness of the technique.

The results shown in Table 6 indicate that three of the benchmarks (m15421 6, m15421 7 and bx842596 7) are not assembled fully into one sequence. SDS has been able to assemble two large, accurate sequences from the fragments of each of these datasets which make up the whole dataset. However up to this point, given there were no similarities between the two resulting sequences, they are returned separately. Therefore, caution is taken and they are reported as not completely assembled.

Next, one of the benchmarks is chosen (x60189 4) and the analysis are reported based on this benchmark. The results are compatible with the ones generated from the other benchmarks. Figure 3-left shows the level of agents activity at various stages of SDS assembling process, including both when a match is found and when a match is

Table 6 Summary of assembling four datasets

			Sum of active agents				
	Cycles	SDS Itrs	Median	Mean	Stdev	Max	Min
x60189 4	23	46,899	384,075	384,746	4,007	392,361	374,717
x60189 5	17	53,649	418,777	418,744	4,119	430,935	409,683
x60189 6	28	114,799	752,539	752,176	4,382	763,060	740,365
x60189 7	26	124,249	1,095,673	1,096,029	5,928	1,109,507	1,083,767
m15421 5	57	476,149	2,175,028	2,176,785	9,722	2,220,938	2,150,915
m15421 6	–	–	–	–	–	–	–
m15421 7	–	–	–	–	–	–	–
j02459 7	129	3,174,449	12,092,968	12,087,911	21,613	12,127,170	12,021,776
bx842596 7	–	–	–	–	–	–	–

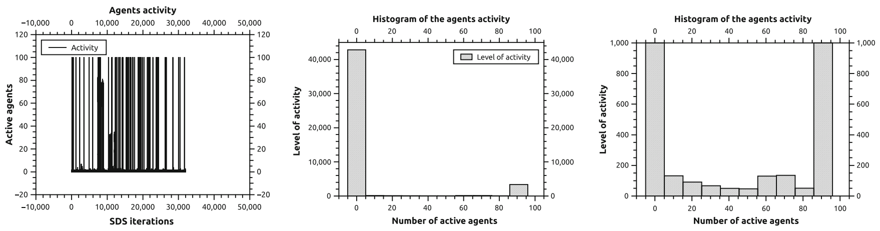


Fig. 3 *Left* activity of the agents in the fragments of $x60189$ 4; *middle* the histogram of the activity of the agents; *right* zooming to show the activity of agents between 0 and 100

not found in any given fragment. The activity of the agents is in the range [0, 100], however if less than the entire agent population (i.e. 100) are active, the agents' hypotheses are not taken into account for the assembling purpose; this ensures the presence of a full match. Reducing the 100 % accuracy would cater for a noisy environment which is one of the strengths of SDS algorithm.

To provide a better understanding, the histogram of the agents' activity is presented in Fig. 3-middle. This graph clearly shows that in most cases there is no high similarity between fragments (note that the similarity between fragments is evaluated by comparing the model to the fragments). However when there is a match (i.e. 100 % activity), the fragments are joined on the fly.

Figure 3-right provides a close-up view of the graph on its left and demonstrates that when there are no exact matches, some of the SDS agents could be activated; however if there are no full match, the activated agents eventually lose their active status in the consequent iterations when they choose a different micro-feature. This feature is particularly useful in a noisy environment whose complete analysis will be provided in an expanded future publication.

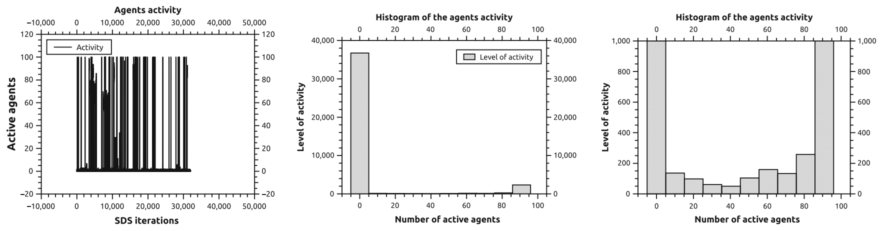


Fig. 4 *Left* activity of the agents in noisy set of fragments; *middle* histogram of the activity of the agents in noisy set of fragments; *right* zooming to show the activity of agents between 0 and 100

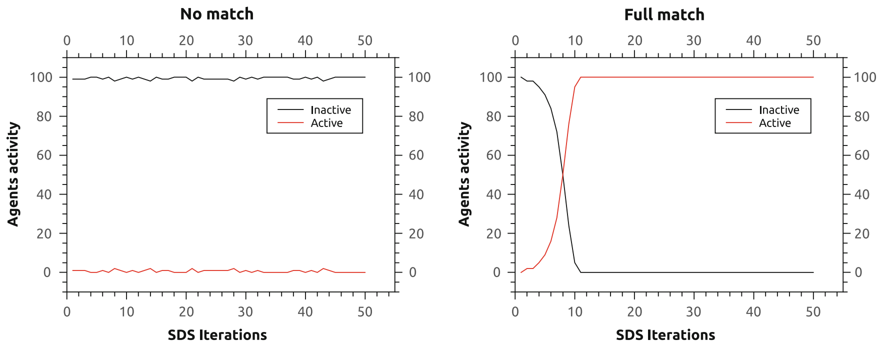


Fig. 5 *Left* absence of a match; *right* presence of a full match

In a similar experiment and in order to analyse the behaviour of the agents when some of the fragments are contaminated with noise, some noise (i.e. of type substitution) is added to all the fragments. Figure 4-left shows the activity of the agents and Fig. 4-middle and right illustrate the frequency of activity level at various iterations. Note that there are fewer number of iterations needed before SDS terminates (as at some point during the process, no match is found from either end of the growing sequence). However the proportion of agents activity between 0 and 100 is increasing with the presence of noise. Despite the fact that the entire fragment is contaminated with noise, SDS is able to produce more than half the length of the target sequence. Further research is required to improve this rate.

In order to illustrate the activity of the agents at each SDS iterations, the graphs in Fig. 5 are presented. In Fig. 5-left the activity of SDS agents are displayed when there is no match. As shown, most of the agents are inactive and very a few flicker from being active and then back to being inactive.

However, on the contrary to the lack of a match, when there is a full match, as shown in Fig. 5-right, soon after the start of the SDS iteration and through agents' communication and information exchange, the entire population becomes active and points to the right position, which is the position of the model within the fragment.

Table 7 Comparison with other techniques

	SDS	PALS	GA	PMA	CAPS	Phrap
x60189 4	1	1	1	1	1	1
x60189 5	1	1	1	1	1	1
x60189 6	1	1	–	1	1	1
x60189 7	1	1	1	1	1	1
m15421 5	1	1	6	1	2	1
m15421 6	2	NA	NA	NA	NA	NA
m15421 7	2	1	1	2	2	2
j02459 7	1	1	13	1	1	1
bx8425696 7	2	2	–	2	2	2

4.1 Comparison with Other Techniques

In another analysis, the performance of SDS is compared against a few other algorithms tasked with assembling the benchmarks. These algorithms, which are used in this context in the literature, are genetic algorithm (GA), a pattern matching algorithm (PMA), Problem Aware Local Search (PALS) and commercially available packages: CAP3 and Phrap. The algorithms are compared in terms of the final number of contigs assembled. Despite being in the early stages of its application in DNA assembly problem, SDS shows a competitive performance (see Table 7[2]). Other than an isolated case (m15421 7), where PALS and GA outperform SDS, in the rest of the cases (89 %), SDS either presents similar or better outcome. SDS is also tried on a benchmark (m15421 6) that is not attempted by the rest of the techniques. The accuracy of the assembled sequences is 100 %; in other words, whenever the accuracy is less than 100 %, the results are considered unsuccessful.

In these experiments, SDS deals with various issues common in DNA sequence assembly,[3] including but not limited to fragments with varying lengths, unknown orientation, incomplete coverage, repeated regions, chimeras and contamination, etc.

4.2 SDS Versus Smith–Waterman Algorithm I

Many DNA sequence assembly techniques use Smith–Waterman algorithm [20], which is a pairwise alignment method to create a similarity matrix between the fragments, therefore generating a complete picture of the entire available data before setting off to the overlapping and assembling stage. While Smith–Waterman

[2]The results of these algorithms, other than SDS, are borrowed from [2].

[3]These issues are explained in Sect. 3.

algorithm provides a precise and detailed account of the input data, it comes at the expense of being time consuming and computationally expensive [23].

Assuming there are n fragments, once the similarity between each pair is calculated using Smith–Waterman algorithm, an $n \times n$ matrix is created. The matrix is then used by other optimising algorithm to conduct the overlapping phase. The results of many of these algorithms are reported in [14].

In the experiments reported earlier in the paper, instead of using Smith–Waterman algorithm to calculate the similarities between fragments, SDS picks a model from a given fragment and aims to find the model in the rest of the fragments. Among the fragments containing the model, the one with the highest similarity is picked and assembled on the fly and then removed from the search space, thus reducing the subsequent computational cost.

On the contrary to many other swarm intelligence and evolutionary computation, SDS has been successful in assembling the benchmarks without using Smith–Waterman algorithm, therefore avoiding its time consuming and computational expensive nature. To understand the full picture of the process, further analysis is needed, among other things, to verify the impact left on the assembling process without accessing the very detailed information provided by Smith–Waterman algorithm.

5 Experiments and Results II

In the second set of experiments of this paper, SDS is tasked to generate overlap score matrices and then the same algorithm is used to optimise the generated matrices. In the next subsections, the process through which the overlap score matrices are generated are described, then the constrains in creating these matrices as well as choosing SDS hypothesis are presented. Afterwards SDS is applied to the benchmarks in the dataset and the results are reported.

5.1 Generating Matrices of Overlap Score by SDS

In this section, SDS is shown to be generating the overlap score matrices. As in the previous set of experiments, an already commonly used dataset [14] is used. More details about the reason behind using SDS (instead of the commonly used Smith–Waterman algorithm) for generating the overlap score matrices are provided in Sect. 5.5.

In the meantime, the steps through which the overlap scores between fragments are computed using SDS are described below:

1. Take a model from the end of the first fragment, and search each fragment one by one to check if other fragments have overlaps with the first fragment. For instance,

if fragment 1 and fragment 5 have 100 bases in common, this value is stored in the matrix as $(1, 5) = 100$.

2. After going through all the fragments, another model is taken from the beginning of the first fragment and again the program searches all the fragments. If the program finds a matching fragment, it stores the value of the overlap score in the matrix.
3. Then the orientation and bases of the first fragment is changed (i.e. A replaces T, T replaces A, C replaces G and G replaces C. Then the orientation is reversed).
4. Steps 1–3 are repeated for all other fragments (in the second cycle, the second fragment will be verified, and in the third the third fragment, until all fragments are checked, their matches are found and their overlap scores are stored in the correct entry of the matrix).

5.2 *Constraints in Overlap Score Matrices and Hypotheses Choice*

Before preceding to the experiments, some important points about the input matrices, overlap scores and the constraints and rules are listed below:

- The matrix always has symmetric overlap score.
- The matrix always has the overlap score of zero on the diagonal line.
- The dimension of the matrix is equal to the number of fragments in the dataset.
- For computing the overlap score of a dataset (that for instance) has 40 fragments, the program should select 40 indices from the matrix. In this work, each one of these elements is called the index hypothesis.
- For choosing the hypothesis from the matrix, the program should follow some rules and consider some constraints.

The following constraints should be considered when choosing the hypotheses (also see Fig. 6):

- Assume each hypothesis has one row and one column. The row and column should not be equal. In other words, hypothesis should not be chosen from the diagonal line of the matrix as the values on the diagonal line are always zero.

Fig. 6 Matrix Constraints (The entries highlighted in green are the valid elements (coordinates) of the hypothesis, and the entries in *red* are the invalid ones)

	0	1	2	3	4	5
0	0	🟥 10	🟩 50	300	65	0
1	🟩 10	0	44	72	100	0
2	50	44	🟥 0	35	🟥 84	🟩 0
3	300	🟩 72	35	0	26	0
4	65	100	84	🟩 26	0	0
5	0	0	0	0	🟩 0	0

- Two different hypotheses should not be on the same row.
- Two different hypotheses should not be on the same column.

Having mentioned the rules and constraints, the next section presents the experiments designed for this paper, demonstrating the performance of the proposed algorithm. Then the results are reported along with comparisons against other techniques.

5.3 *Applying SDS on Overlap Score Matrices*

As mentioned before, SDS has the three phases of initialisation, test and diffusion. In this part, these phases are explained in detail and it is shown how SDS is applied to the matrices in order to find the optimum solution. Thus, on the contrary to the previous set of experiments where the inputs were DNA bases, in this section the inputs data are overlap score matrix.

5.3.1 Initialisation Phase

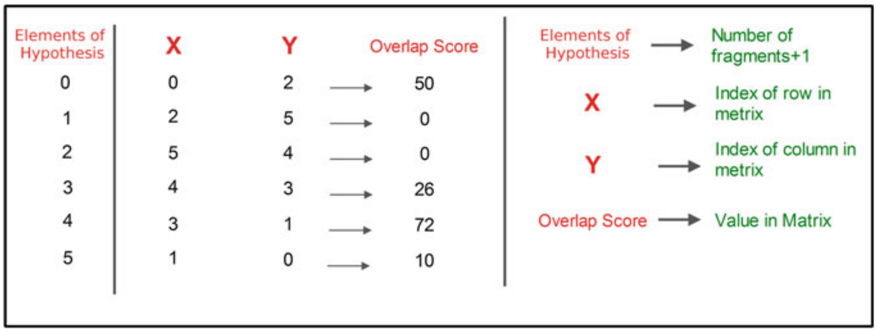
The initialisation phase of SDS algorithm should adhere to the rules and constraints described above. Also it is important to note the following problem-dependant issues:

- Every fragment (except the first and last one) is a prefix of one fragment and suffix of another fragment.
- The first fragment in a contig has no prefix, and the last fragment has no suffix.
- The fragments cannot be appended to itself.

During the implementation stage of the initialisation phase, a hypothesis (i.e. member of matrix) is generated for every agent with respect to the constraints.

For generating the hypothesis, the following steps should be taken:

1. Create a list and populate the cells with values from $[0, n - 1]$ where n is the size of any sides of the matrix.
2. Initialise the first value in X with zero (see Fig. 7).
3. Randomly choose the Y value from the list created. According to Fig. 7, 2 is chosen randomly. Now the row and column (X, Y) of the first hypothesis are assigned.
4. In the next step, the Y value of the previous hypothesis will be assigned as the X value of the second hypothesis. Then that number will be removed from the array (i.e. 2) which was created in the first step. Again the Y value of the second hypothesis will be selected randomly.
5. Finally when all the elements of the array are removed, zero is assigned as the Y value of the last hypothesis. Note that the algorithm is prevented from choosing zero as the Y value during generating the hypotheses until it reaches to the last hypothesis. As stated earlier, the reason behind picking the value of '0' is that



Elements of Hypothesis	X	Y		Overlap Score
0	0	2	⟶	50
1	2	5	⟶	0
2	5	4	⟶	0
3	4	3	⟶	26
4	3	1	⟶	72
5	1	0	⟶	10

Elements of Hypothesis	⟶	Number of fragments+1
X	⟶	Index of row in metrix
Y	⟶	Index of column in metrix
Overlap Score	⟶	Value in Matrix

Fig. 7 Sample Hypothesis of an agent (based on the matrix in Fig. 6)

the first and the last fragments in every sequence only have suffix and prefix respectively.

6. All these elements form one SDS hypothesis for the first agent.

After the initialisation phase, every agent has a SDS hypothesis which consists of n coordinates from the overlap score matrix. Therefore there will be an overlap score associated with every agent (e.g. the overlap score of the first agent is $50 + 0 + 0 + 26 + 72 + 10 = 158$). This will pave the way for comparing the agents against one another.

5.3.2 Test Phase

The test phase is explained by taking agent ‘0’ as the starting point to determine whether it should be active or inactive.

- Agent 0 is compared with a random agent
- If agent 0 has a higher overlap score than the randomly selected agent, it will be active
- If agent 0 has a lower overlap score than the randomly selected agent, it will be inactive
- This continues until all agents are labelled as either active or inactive

5.3.3 Diffusion Phase

This phase is similar to the original description of the SDS’s diffusion search and works as follows:

- Each inactive agent selects an agent randomly
- If the selected agent is active, the hypothesis (which contains all the x and y coordinates) will be copied from the active to the inactive agent.

- If the selected agent is not active, again, another *valid* hypothesis will have to be generated for that inactive agent.

This technique shows a good initial outcome however it does not return an optimum solution since the agents communicate with each other with limited information (the hypotheses that are chosen randomly) and there is not sufficient comparison with the "outside worlds" (all other possible hypotheses that can be generated from the matrix are not considered). Therefore, to improve the program, in the next section, another important phase (i.e. Reform Phase) is added. This phase complements the three previously discussed phases.

Prior to explaining the Reform phase, it is important to discuss what happens if one of the elements of the hypothesis moves from one position to another.

5.3.4 Hypothesis Sliding and the Reform Phase

In the following part, the steps are shown as to how to exchange the values of the coordinates of hypothesis in each agent.

Figure 8 shows a sample matrix representing a dataset with 8 fragments. Therefore each agent has a set of 8 coordinates belonging to the hypothesis. The details of the figure are explained below:

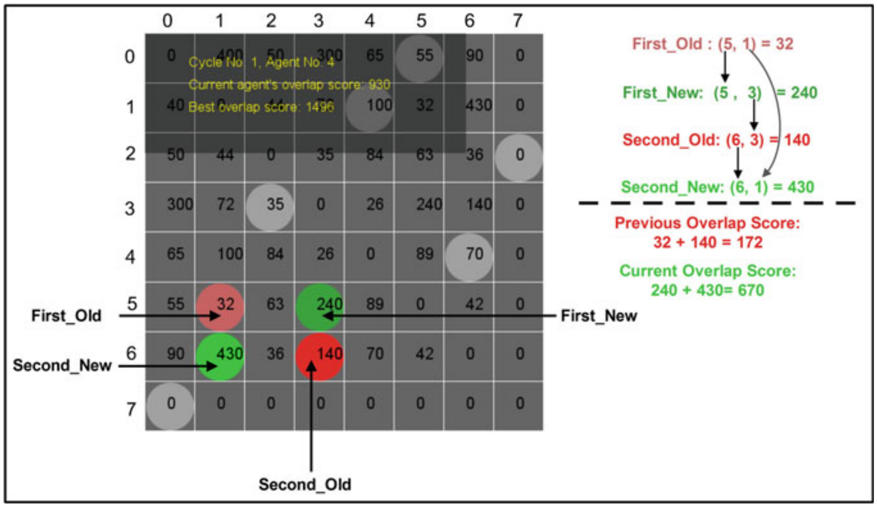


Fig. 8 Hypothesis Sliding (This figure shows the elements of a sample agent hypothesis. The hypothesis' elements are highlighted in *grey* as well as the one in *red*. When the *red* elements of the hypothesis are updated and the new coordinates are shown in *green*. Note that the number of hypothesis elements are 8)

- The grey circles are the hypotheses in the current agent.
- The red circles are the old hypotheses that are replaced by the new ones
- The green circles are the new hypotheses that are replaced with the old ones.

The below steps, describe how the Reform Phase works on the input matrix:

1. Initially the minimum hypothesis (with the smallest overlap score) other than zero, which is called ‘First Old’ is picked. Assume that the hypothesis (5, 1) which has the minimum value is picked.
2. The aim is to have the minimum hypothesis exchanged with an index in the matrix which has a value (overlap score) more than the minimum value. This index is called ‘First New’.
   - The position of this index (i.e. First New) should be selected from the same row of where the minimum overlap score (First Old) is located. In other words, ‘First Old’ and ‘First New’ are in the same row. In order to do that, a column is chosen randomly. Assuming the fourth column is selected, the index of ‘First New’ will be (5, 3).
3. Next, the program should find a hypothesis that is called ‘Second Old’. The steps below show how this is accomplished:
   - The ‘Second Old’ has the same column as the ‘First New’. According to the previous stage, the column of ‘First New’ was 3, therefore the column of ‘Second Old’ will become 3.
   - In order to find the row, the algorithm checks which hypothesis in the current agent has column 3. Then it picks the row of that hypothesis. Suppose the row of that hypothesis is 6. Thus the index of ‘Second Old’ hypothesis becomes (6, 3).
   - As mentioned before, in every agents hypothesis, every column is used once, and every row is used once too. In other words, no two items of the hypothesis should be in the same column or the same row.
4. The next tasks is to find the position of the new coordinate in the original matrix. This coordinate is called the ‘Second New’, which takes the row of the ‘Second Old’ and the column of the ‘First Old’.

5.4 Results

This section presents the results of optimising the overlap score on the real dataset. These experiments are conducted on the following datasets from Table 5: x60189_4, x60189_5, x60189_6, x60189_7.

In the experiments reported in this paper the agent size is empirically set to 100 and the model size for SDS is set to 50. Table 5, as proposed by Mallén-Fullerton et al. [14], shows the benchmarks used by SDS for DNA assembly.

Table 8 Collection of results for the commonly used 4 benchmark datasets along with SDS results

Benchmark	SDS	LKH	PPSO + DE	QEGA	PALS	SAX	POEMS
x60189 4	11,322	11,478	11,478	11,478	11,478	11,478	11,478
x60189 5	13,930	14,161	13,642	14,027	14,021	14,027	–
x60189 6	15,160	18,301	18,301	18,266	18,301	18,301	–
x60189 7	19,728	21,271	20,921	21,208	21,210	21,268	21,261

As part of the results reported, the performance of SDS is compared against a few other algorithms used for computing the overlap scores. These algorithms, which are used in this context in the literature, are Prototype Optimisation with Evolved Improvement Steps (POEMS), Problem aware local Search (PALS), Queen Bee Evolution Based on Genetic Algorithm (QEGA), Particle Swarm Optimisation and Differential Evolution, which is called (PPSO + DE), SAX and Lin-Kernighan (LKH). The algorithms are compared in terms of the total overlap score (see Table 8[4]) and the results show that the preliminary investigation of SDS behaviour demonstrates a competitive performance.

As explained before, the starting point in the reform phase is the minimum (smallest) element of the hypothesis. The algorithm aims to find a larger value to replace the minimum element. The question is: should the program search the entire matrix environment to find a bigger value (global search) or should it search just, for instance, the row or the column where the minimum elements are sitting in (local or neighbourhood search). At the moment the paper has explored the neighbourhood search and global search is the topic of an ongoing research.

5.5 SDS Versus Smith–Waterman Algorithm II

In the dataset used [14] for the experiments reported, the file 'matrix conservative' contains matrices that store the overlap score amongst all fragments for each benchmark in the dataset. The overlap scores are computed by the Smith–Waterman algorithm. Smith–Waterman algorithm takes into account the overlap of each fragment with every other fragments individually, therefore, while providing a comprehensive picture, it is computationally expensive. In other words, Smith and Waterman which was proposed in 1981, is a dynamic programming local sequence alignment algorithm that is used for this purpose. This algorithm's most common setting are 1 for a match, 3 for a mismatch, and 2 for a gap. This algorithm must of run on the entire possible combination of fragment pairs taking both orientations into account (regular and reverse compliment); this is necessary due to the unknown orientation

[4]The results of these algorithms, other than SDS, are borrowed from [14].

of every input fragment. Upon finding the maximum overlap by the algorithm, the actual overlap needs to be calculated. Once complete, the final values are inserted into the overlap score matrix, which details the overlap between any two pair of fragments. As stated before, this is a time consuming and computational expensive task, with the computational complexity of $O(n^2)$. As stated in [14], even if there are only 500 fragments of approximately 100 bases long, around 2,500,000,000 Smith–Waterman elements would be required to calculate the overlap matrix.

In order to explore an alternative less time consuming approach, SDS algorithm is tasked to accomplish the creation of the overlap score matrices.

SDS ignores the non-significant overlap. This is achieved by assigning the minimum overlap score by setting the model size. When the overlap between two fragments is less than the model size, the overlap score value of the fragments is set to zero in matrix. For instance, if model size is 50 and the overlap score between fragments 1 and 4 is 40, the value in entry $(1, 4)$ in the matrix will be zero. The advantage of using this technique (i.e. ignoring non-significant overlaps) is speeding up the search. Additionally the matrix could be compressed easily by removing the zero values.

6 Conclusions

Since DNA fragment assembly problem is NP-hard, it is difficult to find optimal solutions. The increasing presence of biological data and the requirements to study and understand them closely lead researchers and scientists in this field to use computational approaches. This work has shown how DNA fragment assembly problem can be addressed with meta-heuristics. This paper presents Stochastic Diffusion Search (SDS), which belongs to the extended family of swarm intelligence algorithms, in the context of DNA fragment assembly problem. An initial study into the behaviour of SDS is provided, offering an analysis into the agents' activity using several benchmarks. The results are promising as they demonstrate how the activity of the agents shed light into the way agents interact and eventually finalise the assembling process. Additionally it is shown that the level of agents' activity provides a measure of similarity between fragments, thus allowing more similar fragments to be joined in the assembling process.

Subsequently, SDS algorithm is used to optimise the overlap score of the input overlap score matrices. In this optimisation task, input matrices with overlap scores of fragments is given as input to the system and the adapted SDS algorithm is responsible for finding the optimum overlap score in order to assembly the fragments. Taking into account the initial attempt of using this algorithm, the results are close to the those of other researchers.

As part of the future research, this algorithm will be compared against other evolutionary computation techniques used in this field; also larger datasets with more complex features are to be used, and more research is needed in order to theoretically determine the two values (population size and model size) of the SDS parameters.

Additionally, CPU time and memory usage will be taken into account for all the comparisons to provide a more comprehensive account on the performance of the proposed algorithm.

References

1. al-Rifaie, M.M., Bishop, M.: Stochastic diffusion search review. In: Paladyn, Journal of Behavioral Robotics, vol. 4(3), pp. 155–173. Springer, Heidelberg (2013)
2. Alba, E., Luque, G.: A new local search algorithm for the dna fragment assembly problem. In: Evolutionary Computation in Combinatorial Optimization, pp. 1–12. Springer, Heidelberg (2007)
3. Alba, E., Luque, G.: A hybrid genetic algorithm for the dna fragment assembly problem. In: Recent Advances in Evolutionary Computation for Combinatorial Optimization, pp. 101–112. Springer, Heidelberg (2008)
4. Bishop, J.: Stochastic searching networks. In: Proceedings of the 1st IEEE Conference on Artificial Neural Networks. pp. 329–331. London, UK (1989)
5. Blum, C., Li, X.: Swarm Intelligence in Optimization. Springer, Berlin (2008)
6. Cotta, C., Fernández, A., Gallardo, J., Luque, G., Alba, E.: Metaheuristics in bioinformatics: DNA sequencing and reconstruction. In: Optimization Techniques for Solving Complex Problems, pp. 265–286 (2009)
7. Fang, S.C., Wang, Y., Zhong, J.: A genetic algorithm approach to solving dna fragment assembly problem. J. Comput. Theor. Nanosci. **2**(4), 499–505 (2005)
8. Firoz, J.S., Rahman, M.S., Saha, T.K.: 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. pp. 201–208. ACM, New York (2012)
9. Huang, K.W., Chen, J.L., Yang, C.S., Tsai, C.W.: A memetic particle swarm optimization algorithm for solving the dna fragment assembly problem. Neural Comput. Appl. pp. 1–12 (2014)
10. Kim, K., Mohan, C.K.: Parallel hierarchical adaptive genetic algorithm for fragment assembly. In: The 2003 Congress on IEEE Evolutionary Computation CEC'03, vol. 1, pp. 600–607. (2003)
11. Li, L., Khuri, S.: A comparison of dna fragment assembly algorithms. METMBS **4**, 329–335 (2004)
12. Luque, G., Alba, E.: Metaheuristics for the DNA fragment assembly problem. Int. J. Comput. Intell. Res. **1**, 98–108 (2005)
13. Mallén-Fullerton, G.M., Fernández-Anaya, G.: DNA fragment assembly using optimization. In: IEEE Congress on Evolutionary Computation (CEC), pp. 1570–1577 (2013)
14. Mallén-Fullerton, G.M., Hughes, J.A., Houghten, S., Fernández-Anaya, G.: Benchmark datasets for the dna fragment assembly problem. Int. J. Bio-Inspir. Comput. **5**(6), 384–394 (2013)
15. Meksangsouy, P., Chaiyaratana, N.: DNA fragment assembly using an ant colony system algorithm. In: The 2003 Congress on IEEE Evolutionary Computation CEC'03, vol. 3, pp. 1756–1763 (2003)
16. Minetti, G., Alba, E.: Metaheuristic assemblers of dna strands: noiseless and noisy cases. In: IEEE Congress on Evolutionary Computation (CEC), pp. 1–8 (2010)
17. Parsons, R., Johnson, M.E.: DNA sequence assembly and genetic algorithms-new results and puzzling insights. In: ISMB, pp. 277–284 (1995)
18. Pevzner, P.: Computational molecular biology: an algorithmic approach. MIT press, Cambridge (2000)
19. Setubal, J.C., Meidanis, J., Setubal-Meidanis.: Introduction to Computational Molecular Biology. PWS, Boston (1997)

20. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. J. Mol. Biol. **147**(1), 195–197 (1981)
21. U.S. National Library of Medicine: Cells and DNA. what is DNA? http://ghr.nlm.nih.gov/handbook/basics/dna. Accessed 06 Jan 2015
22. Verma, R.S., Singh, V., Kumar, S.: DNA sequence assembly using particle swarm optimization. Int. J. Comput. Appl. **28** (2011)
23. Wieds, G.: Bioinformatics explained: Blast versus smith-waterman. CLCBio. http://www.clcbio.com/index.php. (2007)