DNA Fragment Assembly Using Optimization

From nature inspired algorithms to formal methods

Guillermo M. Mallén-Fullerton
Engineering Dept., Universidad Iberoamericana
México D. F., México
guillermo.mallen@ibero.mx

Guillermo Fernández-Anaya
Physics and Mathematics Dept., Universidad Iberoamericana
México D. F., México
guillermo.fernandez@ibero.mx

Abstract - The DNA fragment assembly is an important phase required to obtain complete genomes. Optimization using nature inspired algorithms has been proposed by several authors. We present another nature inspired algorithm based on Particle Swarm Optimization and Differential Evolution. These algorithms are compared using a set of common benchmarks and show that our proposed algorithm has some advantages. We also applied the Traveling Salesman Problem (TSP) with better results than the nature inspired algorithms as we could obtain the true optima for 16 commonly used benchmarks for the first time to the best of our knowledge. The benchmarks are much smaller than the real organism assembly problems and scaling up from the benchmarks to real organisms presents important challenges. We propose a way to solve the scale up problems and test them using the Staphylococcus aureus COL Main Chromosome with the TSP approach.

Keywords— DNA, Fragment Assembly, Particle Swarm Optimization, Traveling Salesman Problem

I. INTRODUCTION

The “Shotgun Sequencing Method” is commonly used to sequence DNA. First, DNA is cloned to have enough material. The next step is to cut the DNA in random places to obtain a large number of fragments small enough to be sequenced by specialized machinery and finally reconstruct the original DNA sequence from the sequenced fragments. The coverage is the sum of all the fragments lengths divided by the size of the original DNA. Usually the coverage is around 10.

To reconstruct the original sequence we need to have enough fragments to cover several times the original DNA. This way, for each base in the original sequence on average there will be a number of fragments containing it, implying that the fragments will be overlapped. These overlaps are used to know the order of the fragments and reconstruct the original sequence.

There are many practical problems when reconstructing the sequence. Many overlaps are just random coincidences. As the sequencing machines make errors, the overlaps cannot be obtained by direct fragment comparison and special algorithms have to be used to calculate the real overlaps. DNA has also repeated sections causing more apparent overlaps. There are many different possible orders for the fragments and we wish to have the best one to obtain a sequence as close as possible to the original one.

The ordinary result from the fragment assembly methods is a set of contigs, where a contig is a partial sequence derived from several fragments with high overlap scores between them and separated from the other contigs by a low overlap score, below a predefined value. Anyway, a specialized biologist has to check the proposed sequence and make the necessary corrections to end with a sequence that makes sense biologically and is as close as possible to the unknown real sequence.

Several methods have been proposed to assemble the DNA fragments, from greedy algorithms to optimization. Despite the efforts the field is still open. One of the main problems is scaling up some of these methods to the size real organisms.

It is frequent to use nature inspired algorithms like Genetic Algorithms (GA) or Ant Colony Optimization (ACO). In 1995 Rebecca Parsons developed a genetic algorithm to solve the Fragment Assembly Problem (FAP) [1][2]. Later, other nature inspired algorithms were developed. Some GAs were developed by Kieun and Mohan [3], Fang, Wang and Zhong [4], Luque and Alba [5], Kikuchi and Chakraborty [6][7], Nebro, Luque, Luna, and Alba [8], Alba and Luque [9]. The Ant Colony algorithm has also been proposed by Meksangsoy and Chaiyaratana [10], Zhao, Yidan, Ping Ma, Jie Lan, Chun Liang, and Guoli Ji [11]. A variety of other similar algorithms have been proposed by Kubalik, Buryan and Wagner (POEMS) [12], Minetti, Leguizamón, and Alba (SAX) [13]. Alba and Luque [14] proposed a local search method (PALS).

The possibility of a Traveling Salesman Problem (TSP) approach was discarded by Parsons in 1995 with some objections we could solve allowing us to use the TSP solving methods in both benchmarks and real organisms.

Some approaches not based in nature inspired algorithms are greedy algorithms and de Bruijn graphs [15] among others. De Bruijn graphs do not attempt any optimization. The greedy algorithm rarely outputs optimal solutions and can even obtain the worst possible one [16].

The commonly used benchmarks for the FAP provide a useful tool to test algorithms and new ideas without the complications derived from the real world problems. Once a good method has been found a second challenge is to scale it up to the living organisms.
1.- Within each thread generate a random initial population within a predefined hyper-box that should contain the optimum. Determine the global best particle position using the fitness function. Consider this position as the best position obtained for each particle.

2.- For each particle i in each thread subpopulation

2.1.- If there is a new global best position, copy it from the global shared memory to local variables

2.2.- Calculate the new velocity and position vectors:

\[ v_{i+1} = w v_i + r_1 c_1 (x_{i} - x_{gb}) + r_2 c_2 (x_{i} - x_{lb}) \]  

(1)

\[ x_{i+1} = x_i + v_{i+1} \]  

(2)

2.3.- Calculate a DE perturbation vector \( p \)

\[ p_i = x_{i} + (x_{s2} - x_{s3}) \]  

(3)

2.4.- Cross the perturbation vector with the particle i substituting each element of the particle position vector \( x_i \) with the perturbation one with a probability \( 1/|x_i| \) giving a temporary particle

2.5.- Substitute the particle position with the temporary particle position if the temporary particle fitness is better than the original particle fitness

2.6.- Substitute the particle position with the temporary particle position if the temporary particle fitness is better than the original particle fitness

2.7.- Copy \( x_i \) as \( x_{gb} \) if \( f(x_i) \) is better that \( f(x_{gb}) \)

2.8.- Copy \( x_i \) as \( x_{gb} \) if \( f(x_i) \) is better that \( f(x_{gb}) \)

3.- Go to step 2 if the termination criteria has not been met.

Where:

\[ v_{i} \] – particle i velocity vector at time \( t \)

\[ x_{i} \] – particle i position vector at time \( t \)

\[ x_{gb} \] – particle i best position vector at time \( t \)

\[ x_{lb} \] – global best position vector at time \( t \)

\( r_1, r_2 \) - random numbers in the interval [0,1]

\( c_1, c_2 \) – constants in the interval [0, 2]

s1, s2 and s3 are random particles chosen from the same thread subpopulation.

In the FAP we need a permutation that can be obtained from the continuous variables used in PSO and DE using the Random Key Encoding or Smallest Position Value rule (SPV) that has been used in other combinatorial problems [20] [21]. In this rule the values of the variables are sorted in ascending order and the indexes of the variables form a permutation. The inverse operation can be performed distributing the continuous values at uniform intervals and assigning each continuous variable the value indicated by the permutation.

PPSO+DE has been modified to solve the FAP. Two new operators were added. Both work directly on the permutation obtained from the continuous variables using the SPV rule.

The first operator takes a random fragment and looks for its best position in the permutation. Instead of one fragment, two contiguous fragments can be used.

The second operator works on \( contigs \), taking one \( contig \) and locating the best possible position of that \( contig \) among the rest of the \( contigs \).
The first operator is applied after the DE operator for each particle and the second operator is applied at the end of each iteration in each thread.

As recommended by [1] our fitness function is:

\[ f(p) = \sum_{i=1}^{n-1} W_{p(i),p(i+1)} \]  

(4)

Where \( W_{p(i),p(i+1)} \) is the number of bases common to fragments in the positions \( i \) and \( i+1 \) of the permutation.

Many of the heuristics mentioned before use the result of a greedy algorithm as the initial position that is later improved. We followed the same idea and modified our PPSO+DE heuristic to initialize each particle position with a random instance of a greedy algorithm instead of a simple random position. The greedy instance is generated taking one random fragment and adding as the next element of the permutation, the fragment with the highest overlap score that has not been used earlier. The greedy algorithm keeps adding fragments until all fragments are incorporated in the permutation. Once a permutation is obtained the corresponding continuous variables values are calculated using an inverse SPV rule.

We tried our heuristic with 16 common benchmarks and obtained better results than the best results published in the six most difficult benchmarks. In other two benchmarks we obtained the same result obtained by some of the other heuristics we wanted to compare with. All the details will be presented in Sections III and IV.

To improve our heuristic we looked for better initial positions than the ones obtained with the greedy algorithm. We observed the similarity between the FAP and the TSP. The cities are fragments and the distances are overlap scores. This similarity was noticed by Parsons [1] before us but was discarded because a number of reasons, the main one:

"First, the solution to TSP is a circular tour of the cities; the endpoints of the tour are therefore irrelevant. In the fragment assembly problem, however, the endpoints represent fragments on opposite ends of the parent sequence. Many solutions which are equivalent for TSP are thus inequivalent in our context."

Fortunately, this objection is easy to overcome: just add a dummy city with the same distance to all the other cities (usually zero). This way the rest of the cities form an open Hamiltonian path instead of a Hamiltonian circuit and the dummy city closes the circuit without any preference to the other cities, leaving them free to accommodate the best open path. We also multiplied by -1 the objective function to convert the maximization we required to solve the FAP into the TSP minimization.

As the biggest benchmark is over 1000 fragments we choose the Lin-Kernighan heuristic as enhanced by Helsgaun [22] to solve the problem. The Lin-Kernighan heuristic involves swapping pairs of sub-tours to make a new tour. It is a generalization of 2-opt and 3-opt. 2-opt and 3-opt work by switching two or three sub tours to make the tour shorter. Lin–Kernighan chains a series of 2-opt operations to build a k-opt operation as long as adding another 2-opt operation keeps the objective function with a better value. It uses backtracking to try other solutions. Fig. 1 shows a Hamiltonian cycle before (a) and after (3) a 3-opt transformation. Fig. 2 shows a 2-opt (a) and a 4-opt made chaining 2-ops (3).

After a number of tests we discovered that our heuristic could not improve the result obtained with Lin-Kernighan and we suspected that those results were optimal and could not be improved by any means.

To confirm our suspicion we tried formal methods that can always reach an optimum in the TSP. There are a number of formal methods and some of them can be combined. The concorde program [23] is probably the best tool to solve the TSP and demonstrate that a solution is optimal. It uses an interesting mixture of algorithms clearly explained in [23].

Real organisms represent problems much larger than the presented benchmarks. Bacteria usually have a few million bases and require tens of thousands of fragments using long reads. Mammals, including man, can have up to hundreds of millions of bases. The largest benchmark used in this article is only 1049 fragments. In our scale up experiments we used 50,036 fragments from a bacterium.

Our first goal was to scale up at least to the level of bacteria and other microorganisms. There are two main problems in the scale up process: the calculation of the overlaps matrix and the fragment ordering. Overlaps matrix calculation is a slow process due to the need of an algorithm capable of finding overlaps despite some sequencing errors in the fragments. The method used to perform fragment comparisons is the Smith-Waterman (SW) algorithm [24]. It is a dynamic programming application that requires the calculation of an \((n_1+1)(n_2+1)\) matrix where \(n_1\) and \(n_2\) are the number of bases of the fragments. If we compare two 1000 bases fragments we have to calculate more than a million elements. Overlaps matrix calculation is of \(O(n^3)\) and grows rapidly with the size of the organism. The combination of these characteristics represents a serious scale up difficulty.

When we compare two fragments there will be at least a small common section even when the fragments come from different parts of the original DNA. When the common section is too small we can say that it is not a real overlap or a non

![Fig. 1. A 3-opt:(a) initial tour, (b) final tour](Image: 310x127 to 549x286)
significant overlap. The number of real overlaps is small compared with the size of the overlap matrix. On average, the number of real overlaps for a fragment is equal to the coverage, usually around 10. There are also many spurious overlaps due to random coincidences and repeated sequences in the original DNA that we can not distinguish from the real overlaps. Most of the calculated elements of an overlap matrix made comparing each fragment with the rest of the fragments are not significant. To have a good candidate for a significant overlap some conditions have to be fulfilled: the overlapped region should be at the end of one fragment and at the beginning of the other fragment and the percentage of bases coincidences should be high. A small overlap section in the middle of both fragments is not significant. The result of the FAP is a set of contigs where we expect only significant overlaps. It is possible to consider the non significant overlaps as inexistent or zero overlaps.

With these ideas on mind we designed an algorithm that can discard very rapidly most of the non significant overlaps, leaving only a small number of fragment pairs to be compared using the Smith-Waterman algorithm. We define a k-tuple as a sequence of k consecutive bases in a fragment. Our method requires a preprocessing of the fragments to obtain for each fragment and its inverse complement, all the possible k-tuples ordered alphabetically.

This is done the following way:

1.- Form the first k-tuple with the first k bases of the fragment
2.- Form the next k-tuple removing the first base from the last k-tuple formed and appending the next base in the fragment at the end of the k-tuple
3.- Repeat step 2 until there are no more bases in the fragment
4.- Repeat steps 1 to 3 with the inverse complement of the fragment
5.- Order alphabetically all the k-tuples obtained in steps 3 and 4

Once the preprocessing is done, we select the candidates to real overlaps eliminating the pairs that have not enough equal k-tuples. A central part of our algorithm is the application of the matching record algorithm used in the 1960s. The matching record algorithm is still used today in file sorting and merging [25].

The matching record algorithm applied to detect the number of equal k-tuples between the fragments i and j requires the following steps:

1.- Set \( k=l=1, m=0 \)
2.- If there are no more tuples in either fragment, finish
3.- if \( t_{ik} = t_{jl} \), increase \( k, l, m \) and go to step 2
4.- if \( t_{ik} < t_{jl} \), increase \( k \) and go to step 2
5.- increase \( l \) and go to step 2

Where \( t_{ik} \) is the tuple in the position \( k \) of the ordered set of tuples of the fragment \( i \)

At the end \( m \) will contain the number of equal tuples.

The selection of significant fragment pairs requires the following steps:

1.- Using the matching record algorithm, select the candidates with enough equal k-tuples according to a predefined threshold
2.- For each pair where the number of common k-tuples is above the threshold
   2.1.- Apply the SW algorithm
   2.2.- Obtain the position of the common section in each fragment
   2.2.1.- Remove the missing bases in each common section
   2.2.2.- Locate the initial base of the common section in each fragment
2.3.- Calculate the overlap length between the fragments based in the common section position in both fragments and fragments lengths
3.- Calculate the quality of the overlap dividing the common section length by the overlap length
4.- Accept the pair if the quality of the overlap is good enough

The space required to store the k-tuples is of \( O(2m(l – k)) \) where \( m \) is the number of fragments, \( l \) is the average length of each fragment and \( k \) is the length of a k-tuple. Step 1 of our method is of \( O(m^2) \) in time. Step 2.1 is of \( O(m(l_1 l_2)) \) but is performed only when a good candidate is found, an event that occurs with a low probability.

Low \( k \) values increase computation time as there could be a large number of random k-tuple coincidences that are discarded after the Smith-Waterman calculation. High values of \( k \) are also undesirable as the probability of missing a significant overlap increases because the probability of a mistake in the bases of a k-tuple is higher. Choosing \( k \) is an act of balance between speed and accuracy.
It is very important to have a good implementation. As we will see in a moment, 16 and 32 are good values for k. In our case we chose k=16 as we preferred to spend more CPU time instead of loosing accuracy. Representing each of the four possible bases with two bits it is possible to store one 16-tuple in a 32 bit unsigned integer instead of the 16 bytes required using the four letters, saving 75% of the space. With k=32 we could use a 64 bit unsigned integer. For 50,000 fragments of 500 bases in average and k=16, 242 megabytes of RAM will be required and this is typical of bacteria using long reads. For short reads, with 10 million fragments with a length of 100 bases, 8.5 gigabytes will be required, available today in some PCs. More important, time is also reduced when comparing k-tuples stored in an unsigned integer because a single machine language instruction is used to compare two unsigned integers instead of the 16 or 32 comparisons required using the four letters, a speed advantage of 16:1 or 32:1 depending on the k value used.

It is also possible to parallelize the matrix calculation. In SMP parallel computing the preprocessing could be done serially as it takes a short time, and then, a number of threads equal to the number of cores or processors can be launched to calculate a different section of the matrix in each thread.

An overlaps matrix allows also to remove redundant and useless fragments. A useless fragment, that could came from chimeras produced during the cloning process or from contamination, is detected when it has no significant overlaps with a part of another larger fragment and supplies no new information. The elimination of useless and redundant fragments can reduce significantly the optimization problem size.

### III. EXPERIMENTS PERFORMED

Experiment number 1 was designed to compare our PPSO+DE algorithm with the algorithms that reported the best results in the sixteen common benchmarks. The algorithms with the best objective functions we will be comparing are: Queen-bee Evaluation Based On Genetic Algorithm (QEGA) [26], Simulated Annealing (SA) [27], Problem Aware Local Search (PALS) [28], SAX [28] and Prototype Optimization with Evolved Improvement Steps (POEMS) [12].

In the experiment number 2 we applied the TSP approach using the Lin-Kernighan algorithm to each of the sixteen benchmark problems. To confirm that the results from the Lin-Kernighan algorithm were optimal, in the experiment number 3 we used the concorde program.

In experiment 4 we tested the TSP approach to a real organism. We choose the Staphylococcus aureus COL Main Chromosome because it is a moderate size bacterium and the test data is easily available allowing other researchers to independently replicate our results. A set of fragments from the Staphylococcus aureus COL Main Chromosome test data was obtained from the University of Maryland [29] to be used in our real organism experiments. It consists of 50,036 fragments including 1022 unrelated fragments. Once trimmed to eliminate the low quality ends of the fragments according to the specifications included in the data set, the average length of a fragment is 540.12 bases and the total length of the fragments is 27,025,250 bases. The Staphylococcus aureus COL DNA length is 2,813,940 bases and the coverage using this set of fragments is 9.6.

This experiment was performed in two phases. In the first one we calculated the overlaps matrix according to our algorithm and eliminated non significant and redundant fragments. In the second phase we solved the resulting TSP problem using the Lin-Kernighan algorithm and then used the concorde program to verify that the results from Lin-Kernighan were optimal.

### IV. RESULTS

Experiment 1 showed that our proposed algorithm is competitive with the best algorithms published. We obtained the best result for the six more difficult problems of the sixteen benchmarks and the same result as other algorithms in two other benchmarks.

Experiment 2, the TSP approach with the Lin-Kernighan algorithm, produced the best solution to all the sixteen benchmarks. In experiment 3, using the concorde program we obtained the optimal solutions to all the benchmark problems and confirmed our suspicion: all the solutions found by the Lin-Kernighan algorithm were optimal and it is not possible to obtain better values. Table II shows the optimal solution along with the solutions found with our PPSO+DE algorithm and the other methods. Values in **bold** indicate the best solution (optimal) and values in *italic bold* indicate the second best solution.

In the first phase of experiment 4, to obtain the overlaps matrix we used 16 bases k-tuples stored in a 32 bit unsigned integer. The calculation of the 1,251,775,639 elements of the overlaps matrix we used 16 bases k-tuples stored in a 32 bit unsigned integer. The calculation of the 1,251,775,639 elements of the overlaps matrix took 6:19 in a four core 2.8 GHz Intel processor using a four threads parallel program. The

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Optimal</th>
<th>PPSO+DE</th>
<th>QEGA (26)</th>
<th>SA(27)</th>
<th>PALS (28)</th>
<th>SAX(28)</th>
<th>POEMS (12)</th>
</tr>
</thead>
<tbody>
<tr>
<td>acm1</td>
<td>47616</td>
<td>47616</td>
<td>47715</td>
<td>46995</td>
<td>46995</td>
<td>46995</td>
<td>46995</td>
</tr>
<tr>
<td>acm2</td>
<td>151551</td>
<td>147429</td>
<td>144333</td>
<td>143705</td>
<td>144333</td>
<td>144333</td>
<td>144333</td>
</tr>
<tr>
<td>acm3</td>
<td>167877</td>
<td>163965</td>
<td>156138</td>
<td>156039</td>
<td>156776</td>
<td>156776</td>
<td>156776</td>
</tr>
<tr>
<td>acm4</td>
<td>163965</td>
<td>161571</td>
<td>146541</td>
<td>146607</td>
<td>145591</td>
<td>145591</td>
<td>145591</td>
</tr>
<tr>
<td>acm5</td>
<td>188968</td>
<td>180048</td>
<td>175528</td>
<td>167984</td>
<td>166004</td>
<td>166004</td>
<td>166004</td>
</tr>
<tr>
<td>acm6</td>
<td>344107</td>
<td>335222</td>
<td>322682</td>
<td>342656</td>
<td>329360</td>
<td>329360</td>
<td>329360</td>
</tr>
</tbody>
</table>

**TABLE II. BENCHMARK RESULTS**
program could calculate 55,018 overlaps per second using our algorithm. In some previous tests we made using only the Smith-Waterman algorithm in a serial program, we could calculate 158.73 overlaps per second indicating that our parallel algorithm is 346.6 times faster than the serial implementation of the Smith-Waterman algorithm using four threads/cores.

In the first overlaps matrix obtained we required that at least 50% of the bases in the overlapped section of each pair of fragments matched. We obtained 2,119,380 significant overlaps, 0.17% of all the possible pairs. A plot of the cumulative number of fragments against overlap quality showed that a 90% equal bases in the overlapped section could be a better criteria than the 50% used before as can be seen in Fig. 3. The fragments without at least one overlap with 90% of equal bases were eliminated leaving 49,180 fragments. This number is consistent with the number of unrelated fragments included in the data set (1022).

A large number of redundant fragments was found in the overlaps matrix and were eliminated. At the end, only 18,021 fragments with 104,011 significant overlaps were used.

The total number of bases in the fragments used is 11,223,524 for a coverage of 4. The final overlap matrix showed that each fragment had 11.54 significant overlaps in average, more than the 4 expected from the coverage, indicating that there were spurious overlaps that could not be eliminated.

The number of significant overlaps per fragment represents 0.064% of the 162,369,210 elements of the final overlaps matrix. A TSP with this characteristic should be easy to solve.

In the second phase, the Lin-Kernighan algorithm was used to solve this problem. To avoid a cyclic solution, a dummy city with zero distance to the rest of the cities was introduced in the data set. The overlaps from the overlaps matrix were multiplied by -1 to convert the maximization fragment assembly problem into a TSP minimization.

In the Staphylococcus aureus experiment 30 Lin-Kernighan runs were made (Table III). In 18 runs the maximum value of the objective function was found (bold in Table III) and we suspected this was the optimal value. To prove this is the optimal value, a long concorde run was made. After 3:17:08, using a linear programming relaxation of the problem with additional cut constraints, a maximum bound equal to the best value obtained in the Lin-Kernighan runs was found proving we had the real optimum.

It is important to notice that using the formal methods of the concorde program, after more than one million seconds, the optimum could not be found in spite of a very large number of iterations performed after the right bound was found. The Lin-Kernighan heuristic can not prove that the answer is optimal, but is very efficient and often finds the optimum in a short time. Formal methods sometimes can provide enough information to prove that a Lin-Kernighan solution is optimal even before the formal methods can find the optimal solution.

We found 1,315 contigs, the largest with 286 fragments.

V. CONCLUSIONS

We presented a parallel heuristic based in the PSO and DE using the SPV rule to convert the continuous variables used in

<table>
<thead>
<tr>
<th>Run number</th>
<th>Objective function</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8345783</td>
<td>63</td>
</tr>
<tr>
<td>2</td>
<td>8345836</td>
<td>64</td>
</tr>
<tr>
<td>3</td>
<td>8345826</td>
<td>64</td>
</tr>
<tr>
<td>4</td>
<td>8345841</td>
<td>64</td>
</tr>
<tr>
<td>5</td>
<td>8345841</td>
<td>63</td>
</tr>
<tr>
<td>6</td>
<td>8345841</td>
<td>66</td>
</tr>
<tr>
<td>7</td>
<td>8345836</td>
<td>65</td>
</tr>
<tr>
<td>8</td>
<td>8345841</td>
<td>63</td>
</tr>
<tr>
<td>9</td>
<td>8345841</td>
<td>64</td>
</tr>
<tr>
<td>10</td>
<td>8345836</td>
<td>65</td>
</tr>
<tr>
<td>11</td>
<td>8345744</td>
<td>72</td>
</tr>
<tr>
<td>12</td>
<td>8345841</td>
<td>69</td>
</tr>
<tr>
<td>13</td>
<td>8345836</td>
<td>67</td>
</tr>
<tr>
<td>14</td>
<td>8345841</td>
<td>68</td>
</tr>
<tr>
<td>15</td>
<td>8345841</td>
<td>68</td>
</tr>
<tr>
<td>16</td>
<td>8345841</td>
<td>67</td>
</tr>
<tr>
<td>17</td>
<td>8345841</td>
<td>66</td>
</tr>
<tr>
<td>18</td>
<td>8345841</td>
<td>65</td>
</tr>
<tr>
<td>19</td>
<td>8345790</td>
<td>67</td>
</tr>
<tr>
<td>20</td>
<td>8345836</td>
<td>67</td>
</tr>
<tr>
<td>21</td>
<td>8345841</td>
<td>64</td>
</tr>
<tr>
<td>22</td>
<td>8345841</td>
<td>62</td>
</tr>
<tr>
<td>23</td>
<td>8345841</td>
<td>65</td>
</tr>
<tr>
<td>24</td>
<td>8345841</td>
<td>66</td>
</tr>
<tr>
<td>25</td>
<td>8345836</td>
<td>65</td>
</tr>
<tr>
<td>26</td>
<td>8345744</td>
<td>63</td>
</tr>
<tr>
<td>27</td>
<td>8345841</td>
<td>64</td>
</tr>
<tr>
<td>28</td>
<td>8345841</td>
<td>64</td>
</tr>
<tr>
<td>29</td>
<td>8345841</td>
<td>66</td>
</tr>
<tr>
<td>30</td>
<td>8345739</td>
<td>64</td>
</tr>
</tbody>
</table>

Fig. 3 Cumulative % of overlaps against overlap quality
PSO and DE to the permutation required to solve the FAP. Two additional operators were necessary to improve this heuristic performance to a competitive level. Our heuristic could obtain better results in the six most complex benchmarks and the same value as other algorithms in other two benchmarks. During our improvement work we applied a variation of the TSP to solve the benchmark problems and got the global optima of all the 16 benchmarks for the first time as far as we know.

In the scale up process to the size of living organisms we had to develop a very efficient algorithm to obtain the overlaps matrix that resulted 346.6 times faster than the ordinary approach in a four core processor in our particular test problem. To test our ideas we used the Staphylococcus aureus COL Main Chromosome test data obtained from the University of Maryland. The overlaps matrix could be calculated in 6:19.

The optimization runs were made with the Lin-Kernighan heuristic. In 18 of the 30 runs the global optimum, verified by formal methods, was obtained. Our results show that optimization methods using a variant of the TSP problem and coupled with a very fast overlaps matrix calculation, can be used to assemble DNA fragments of real organisms and represent an alternative to the greedy methods usually used today.

The upper limit of the problem size depends on the amount of hardware and time available. Under reasonable conditions (i. e. an eight core processor and a few days calculation), it would be possible to assemble problems with hundreds of thousands of fragments or tens of millions of bases. This includes all prokaryotes and most unicellular eukaryotes [30]. Large genomes are usually cut at specific places yielding smaller assembly problems. With a faster overlap matrix calculation it should be possible to apply the TSP approach to solve these problems.

It should be noticed that all the tools needed to solve the FAP using the TSP approach were developed some time ago. Sometimes we rely too much in the nature inspired algorithms leaving other possibilities untouched. Evolutionary computation approaches are not always best.

VI. NEXT STEPS

An even faster method to calculate the overlaps matrix is required to scale up to complex organisms even if their DNA is divided to obtain smaller problems. Probably the best approach is to only calculate the significant overlaps without the need of a previous test. We think this is achievable. It is likely that the Lin-Kernighan algorithm can find a good answer for the larger problems, but it is unlikely that it could be proved optimal using formal methods. The most we can expect from formal methods in this case is to obtain a bound to the optimum and we would expect that the Lin-Kernighan can obtain results very close to that bound.

Some ideas presented in this work could be applied to other relevant but difficult problems, like the Political Districting Problem. In particular, the transformation of a continuous optimization method to a combinatorial optimization one, has a good potential. Hybrid algorithms can frequently achieve better results than pure ones.

References


