Málaga, 16 de noviembre de 2008

Informe Ejecutivo

Título: BIO-1.0: Un algoritmo híbrido para el problema del ensamblado de fragmentos de ADN

Resumen: En este documento se propone y estudia el comportamiento de un nuevo algoritmo híbrido para el problema del ensamblado de fragmentos de ADN. Este problema debe ser resuelto como fase inicial de cualquier proyecto relativo al genoma y por lo tanto es muy importante ya que las restantes fases dependen de la calidad que ofrezca como resultado. Este problema es de tipo NP por lo que no puede resolverse con técnicas convencionales. En este documento se analizará el algoritmo híbrido propuesto y se compará contra las técnicas "puras" a partir del que se forma, AG y PALS.

Objetivos:

1. Proponer un nuevo algoritmo que combine las características de algoritmos existentes.
2. Comparar su rendimiento respecto a los algoritmos "puros" que han sido utilizados para construir el híbrido.
3. Comparar el algoritmo resultante contra los algoritmos estado-del-arte para el problema analizado.

Conclusiones:

1. Los resultados de la propuesta híbrida superan en calidad a los algoritmos existentes en las instancias probadas.
2. En cambio, la eficiencia del nuevo algoritmo es bastante menor que la de algunos de los algoritmos existentes, en especial del PALS, por lo que se necesita mejorar este aspecto en las siguientes versiones (por ejemplo, proponiendo versiones paralelas).

Relación con entregables:

Málaga, November 22nd, 2008

Executive Summary

TITLE: BIO-1.0: A Hybrid Algorithm for the DNA Fragment Assembly Problem

ABSTRACT: In this document we propose and study the behavior of a new hybrid heuristic algorithm for the DNA fragment assembly problem. The DNA fragment assembly is a problem solved in the early phases of the genome project and thus very important, since the other steps depend on its accuracy. This is an NP-hard combinatorial optimization problem which is growing in importance and complexity as more research centers become involved on sequencing new genomes. In this document will be analyzed the hybrid version and it will be compared against the "pure" algorithms, GA and PALS.

GOALS:

1. Propose a new algorithm combining several existing ones.
2. Compare the proposed algorithm against the "pure" algorithms.
3. Compare the proposed method against the state-of-the-art techniques for this problem.

CONCLUSIONS:

1. The new hybrid algorithm outperforms the existing methods for the tested instances.
2. The efficiency of the hybrid algorithm is quite worse than some existing methods and the new mechanism for improve it should be proposed (such as parallel versions).

RELATION WITH DELIVERABLES:

BIO-1.0: A Hybrid Algorithm for the DNA Fragment Assembly Problem

DIRICOM

June 2009

1. Introduction

With the advance of computational science, bioinformatics has become more and more attractive to researchers in the field of computational biology. Genomic data analysis using computational approaches is very popular as well. The primary goal of a genomic project is to determine the complete sequence of the genome and its genetic contents. Thus, a genome project is accomplished in two steps; the first one is the genome sequencing and the second one is the genome annotation (i.e., the process of identifying the boundaries between genes and other features in raw DNA sequence).

In this work, we focus on the genome sequencing, which is also known as the DNA fragment assembly problem. The fragment assembly occurs in the very beginning of the process and therefore other steps depend on its accuracy. At present, DNA sequences that are longer than 600 base-pairs (bps) cannot routinely be sequenced accurately. For example, human DNA is about 3.2 billion nucleotides in length and cannot be read at once. Hence, long strands of DNA need to be broken into small fragments for sequencing in a process called *shotgun sequencing*. In this approach, several copies of a portion of DNA are each broken into many segments short enough to be sequenced automatically. But this process does not keep neither the ordering of the fragments nor the portion from which a particular fragment came. This leads to the DNA fragment assembly problem [1] in which these short sequences have to be reassembled to their (supposed) original form. The automation allows shotgun sequencing to proceed far faster than traditional methods. But comparing all the tiny pieces and matching up the overlaps requires massive computation or carefully designed programs.

The assembly problem is therefore a combinatorial optimization problem that, even in the absence of noise, is NP-hard [2]: given k fragments, there are $2^k k!$ possible combinations. Over the past decade a number of fragment assembly packages have been developed and used to sequence different organisms. The most popular packages are PHRAP [3], TIGR assembler [4], STROLL [5], CAP3 [6], Celera assembler [7], and EULER [8]. These packages deal with the previously described challenges to different extents, but none of them solves all of them. Each package automates fragment assembly using a variety of algorithms. The most popular techniques are greedy-based procedures while other approaches have tackled the problem with metaheuristics [9]. Recently, Alba and Luque [10] proposed a new promising heuristic for this problem, PALS, in which the search is guided by an estimation of the number of contigs. This estimation is one key point in reducing the computational demands of the technique. We here extend [10] and design a new hybrid method based on that heuristic. This method consists of a genetic algorithm which uses PALS as a mutation operator. We compare the results of our new approach with the "pure" methods, GA and PALS.

2. The Optimization Problem Addressed

In order to determine the function of specific genes, scientists have learned to read the sequence of nucleotides comprising a DNA sequence in a process called DNA sequencing. To do that, multiple exact copies of the original DNA sequence are made. Each copy is then cut into short fragments at random positions. These are the first three steps depicted in Fig. 1 and they take place in the laboratory. After the fragment set is obtained, a traditional assemble approach is followed in this order: overlap, layout, and then consensus. To ensure that enough fragments overlap, the reading of fragments continues until a coverage is satisfied. These steps are the last three ones in Fig. 1. In what follows, we give a brief description of each of the three phases, namely overlap, layout, and consensus.

Overlap Phase - Find the overlapping fragments. This phase consists in finding the best or longest match between the suffix of one sequence and the prefix of another. In this step, we compare all possible pairs of fragments to determine their similarity. Usually, a dynamic programming algorithm applied to semiglobal alignment is used in this step. The intuition behind finding the pairwise overlap is that fragments with a high overlap are very likely next to each other in the target sequence.

Layout Phase - Find the order of fragments based on the computed similarity score. This is the most difficult step because it is hard to tell the true overlap due to the following challenges:

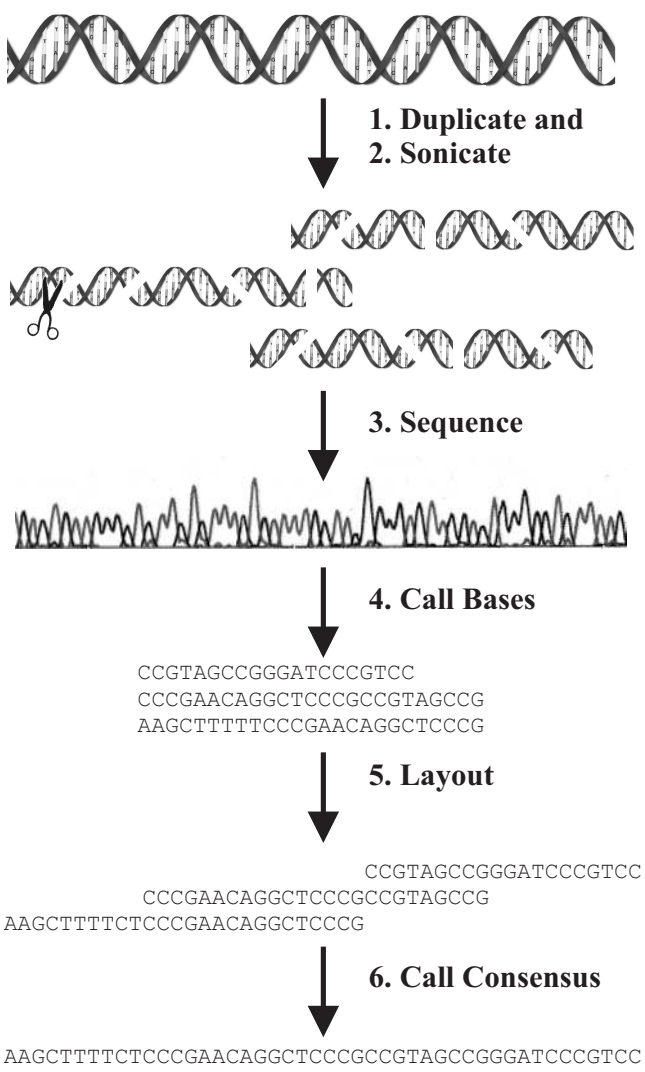

Figura 1: Graphical representation of DNA sequencing and assembly

1. Unknown orientation: After the original sequence is cut into many fragments, the orientation is lost. One does not know which strand should be selected. If one fragment does not have any overlap with another, it is still possible that its reverse complement might have such an overlap.

2. Base call errors: There are three types of base call errors: substitution, insertion, and deletion. They occur due to experimental errors in the electrophoresis procedure (the method used in the laboratories to read the ADN sequences). Errors affect the detection of fragment overlaps. Hence, the consensus determination requires multiple alignments in highly coverage regions.

3. Incomplete coverage: It happens when the algorithm is not able to assemble a given set of fragments into a single contig. A contig is a sequence in which the overlap between adjacent fragments is greater or equal to a predefined threshold (cutoff parameter).

4. Repeated regions: "Repeats" are sequences that appear two or more times in the target DNA. Repeated regions have caused problems in many genome-sequencing projects, and none of the current assembly programs can handle them perfectly.

5. 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.

After the order is determined, the progressive alignment algorithm is applied to combine all the pairwise alignments obtained in the overlap phase.

Consensus Phase - Derive the DNA sequence from the layout. The most common technique used in this phase is to apply the majority rule in building the consensus.

To measure the quality of a consensus, we can look at the distribution of the coverage. Coverage at a base position is defined as the number of fragments at that position. It is a measure of the redundancy of the fragment data, and it denotes the number of fragments, on average, in which a given nucleotide in the target DNA is expected to appear. It is computed as the number of bases read from fragments over the length of the target DNA [1].

$$Coverage = \frac{\sum_{i=1}^{n} length\ of\ the\ fragment\ i}{target\ sequence\ length} \quad (1)$$

where n is the number of fragments. The higher the coverage, the fewer number of the gaps, and the better the result.

3. Algorithms

In this section we detail the three proposed algorithms that we have designed to solve the DNA fragment assembly problem. Firstly, we explain the two cannonical methods, a GA and PALS (Problem Aware Local Search), and then we address the hybrid one (GA+PALS).

4. Genetic Algorithm

A genetic algorithm (GA) [11] is an iterative technique that applies stochastic operators on a pool of individuals (the population). Every individual in the population is the encoded version of a tentative solution. Initially, this population is randomly generated. An evaluation function associates a fitness value to every individual indicating its suitability to the problem.

The genetic algorithm that we have developed for the solution of the DNA fragment assembly follows the basic structured shown in Algorithm 1. In next paragraphs we give some details about the most important issues of our GA implementation.

Algorithm 1 Basic GA

```
Generate (P(0))
t := 0
while not Termination_Criterion(P(t)) do
  Evaluate(P(t))
  P'(t) := Selection(P(t))
  P'(t) := Recombination(P'(t))
  P'(t) := Mutation(P'(t))
  P(t+1) := Replacement(P(t), P'(t))
  t := t+1
end while
return Best_Solution_Found
```

For the indivuals, we use a permutation representation with integer number encoding. A permutation of integers represents a sequence of fragment numbers, where successive fragments are later assumed to overlap. The solution in this representation requires a list of fragments assigned with a unique integer ID. For example, 8 fragments will need eight identifiers: 0, 1, 2, 3, 4, 5, 6, 7. The permutation representation requires special operators to make sure that we always get legal (feasible) solutions. In order to maintain a legal solution, the two conditions that must be satisfied are (1) all fragments must be presented in the ordering, and (2) no duplicate fragments are allowed in the ordering. For example, one possible ordering for 4 fragments is 3 0 2 1. It means that fragment 3 is at the first position and fragment 0 is at the second position, and so on.

In the DNA fragment assembly problem, the fitness function measures the multiple sequences alignment quality and finds the best scoring alignment. Parsons, Forrest, and Burks mentioned two different fitness functions [12]. In this work, we use the first one. This function sums the overlap score for adjacent fragments in a given solution. When this fitness function is used, the objective is to maximize such a score. It means that the best individual will have the highest score.

$$F(l) = \sum_{i=0}^{n-2} w_{f[i],f[i+1]}, \quad (2)$$

where $w_{i,j}$ is the overlap between fragments i and j. This function have several drawbacks, but despite then and its apperent simplicity it is very difficult to find a better one (see next subsection).

For our experimental runs, we use the order-based crossover (OX) as recombination operator. The order-based crossover operator first copies the fragment ID between two random positions in Parent1 into the offspring's corresponding positions. We then copy the rest of the fragments from Parent2 into the offspring in the relative order

presented in Parent2. If the fragment ID is already present in the offspring, then we skip that fragment. The method preserves the feasibility of every string in the population.

Finally, the swap mutation operator is applied to maintain diversity in the population. This operator randomly selects two positions from a permutation and then swaps the two fragment positions. Since this operator does not introduce any duplicate number in the permutation, the solution it produces is always feasible.

4.1. Problem Aware Local Search (PALS)

Classical assemblers use fitness functions that favor solutions in which strong overlap occurs between adjacent fragments in the layouts, using equations like Eq. 2. However, the actual objective is to obtain an order of the fragments that minimizes the number of contigs, with the goal of reaching one single contig, i.e., a complete DNA sequence composed of all the overlapping fragments. Therefore, the number of contigs is always used as the high-level criterion to judge the whole quality of the results, since it is difficult to capture the dynamics of the problem into other mathematical functions. Contig values are computed by applying a final step of refinement with a greedy heuristic regularly used in this domain [13]. We have even found that in some (extreme) cases it is possible that a solution with a better fitness (using Eq. 2) than other one generates a larger number of contigs (worse solution). All this suggests that the fitness (overlapping) should be complemented with the actual number of contigs.

And here we arrive ti the true obstacle: the calculation of the number of contigs is a quite time-consuming operations, and this definitely precludes any algorithm to use it. A solution to this problem is the utilization of a method which should not need to know the exact number of contigs, thus being computationally light. Our key contribution is to indirectly estimate the number of contigs by measuring the number of contigs that are created or destroyed when tentative solutions are manipulated in the algorithm. We propose a variation of Lin's 2-opt [14] for the DNA field, which does not only use the overlap among the fragments, but that it also takes into account (in an intelligent manner) the number of contigs that have been created or destroyed. The pseudo-code of our proposed method is shown in Algorithm 2.

Algorithm 2 PALS

```
s ← GenerateInitialSolution() {Create the initial solution}
repeat
  L ← ∅
  for i = 0 to N do
    for j = 0 to N do
      Δc, Δf ← CalculaeDelta(s,i,j) {See Algorithm 3}
      if Δc >= 0 then
        L ← L∪ < i, j, Δf, Δc > {Add candidate movements to L}
      end if
    end for
  end for
  if L <> ∅ then
    < i, j, Δf, Δc >← Select(L) {Select a movement among the candidates}
    ApplyMovement(s,i,j) {Modify the solution}
  end if
until no changes
return: s
```

Our algorithm works on a single solution (an integer permutation) which is generated using the `GenerateInitialSolution` method, and it is iteratively modified by the application of movements in a structured manner. A movement is a perturbation (`ApplyMovement` method) that, given a solution s, and two positions i and j, reverses the subpermutation between the positions i and j.

The key step in PALS is the calculation of the variation in the overlap (Δ_f) and in the number of contigs (Δ_c) among the current solution and the resulting solution after applying a movement (see Algorithm 3). This calculation is computationally light since we do not calculate neither the fitness function nor the number of contigs, but instead we estimate the *v*ariation of these values. To do this, we only need to analyze the affected fragments by the tentative movement (i, j, $i-1$ and $j+1$), reducing the overlap score of affected fragments of the current solution and adding the one of the modified solution to Δ_f (equations of lines 4-5 of Algorithm 3) and testing if some current contig is broken (first two if statements of Algorithm 3) or two contigs are merged (last two if statements of Algorithm 3) by the movement operator.

In each iteration, PALS makes these calculations for all possible movements, storing the candidate movements in a list L. Our method only considers those candidate movements that do not increase the number of contigs ($\Delta_c \leq 0$). Once it has completed the previous calculations, the method selects a movement of the list L and applies it. The algorithm stops when no more candidate movements are generated.

Algorithm 3 `CalculateDelta(s,i,j)` function

$\Delta_c \leftarrow 0$
$\Delta_f \leftarrow 0$
{Calculate the variation in the overlap}
$\Delta_f = w_{s[i-1]s[j]} + w_{s[i]s[j+1)}$ {Add the overlap of the modified solution}
$\Delta_f = \Delta_f - w_{s[i-1],s[i]} - w_{s[j]s[j+1]}$ {Remove the overlap of the current solution}
{Test if a contig is broken, and if so, it increases the number of contigs}
if $w_{s[i-1]s[i]} > cutoff$ **then**
  $\Delta_c = \Delta_c + 1$
end if
if $w_{s[j]s[j+1]} > cutoff$ **then**
  $\Delta_c = \Delta_c + 1$
end if
{Test if two contig are merged, and if so, it decreases the number of contigs}
if $w_{s[i-1]s[j]} > cutoff$ **then**
  $\Delta_c = \Delta_c - 1$
end if
if $w_{s[i]s[j+1]} > cutoff$ **then**
  $\Delta_c = \Delta_c - 1$
end if
return: Δ_f, Δ_c

To complete the definition of our method we must decide how the initial solution is generated (`GenerationInitialSolution` method) and how a movement is selected among all possible candidates (`Select` method). In [10] we tested several alternatives for these operations, and we concluded that the best setting is to start from a random solution and to select the best movement found in each iteration.

4.2. Hybrid Approach

Finally, we define here the hybrid algorithm proposed in this wor. In its broadest sense, hybridization refers to the inclusion of problem-dependent knowledge in a general search algorithm [15] in one of two ways: *strong hybrids*, where problem-knowledge is included as problem-dependent representation and/or operators, and *weak hybrids*, where several algorithms are combined in some manner to yield the new hybrid algorithm. Therefore, we define a weak hybrid called GA+PALS, where a GA uses PALS as an additional evolutionary operator. The figure and the pseudo-code of this approach is shown in Algorithm 4. In the main loop of this method after the traditional recombination and mutation operators are applied, several solutions are randomly selected (according to a very low probability) from the current offspring and they are improved using the local search algorithm. The rationale for this sort of hybridization is that, while the GA locates "good" regions of the search space (exploration), PALS allows for exploitation in the best regions found by its partner (neighborhood intensification). Evidently, the motivation in this case is to see if, by taking the best of these two heuristics (i.e., the genetic algorithm and PALS), we could produce another heuristic which would perform better than any of the two approaches from which it was created (new emergent behavior).

Algorithm 4 Hybrid GA+PALS

Generate (P(0))
t := 0
while not Termination_Criterion(P(t)) **do**
  Evaluate(P(t))
  P'(t) := Selection(P(t))
  P'(t) := Recombination(P'(t))
  P'(t) := Mutation(P'(t))
  P'(t) := PALS(P'(t))
  P(t+1) := Replacement(P(t), P'(t))
  t := t+1
end while
return Best_Solution_Found

Cuadro 1: Information of datasets. Accession numbers are used as instance names

Parameters	BX842596	
Coverage	4	7
Mean fragment length	708	703
Number of fragments	442	773
Name	BX842596(4)	BX842596(7)

5. Experiments

In this section we analyze the behavior of our proposed GA+PALS method. First, the target problem instances used and the parameter setting are presented in Section 5.1. In the next subsection, we study the results of the different algorithms presented previously in Section 3, and finally, we compare our approaches with other assemblers.

The experiments have been executed on a Intel Pentium IV 2.8GHz with 1GB running SuSE Linux 8.1. Because of the stochastic nature of the algorithms, we performed 30 independent runs of each test to gather meaningful experimental data and apply statistical confidence metrics to validate our results and conclusions.

5.1. Target Problem Instances and Parameter Settings

To test and analyze the performance of our algorithm we generated several problem instances with GenFrag [16]. GenFrag takes a known DNA sequence and uses it as a parent strand from which random fragments are generated according to the criteria supplied by the user (mean fragment length and coverage of parent sequence).

We have chosen a large sequence from the NCBI web site[1]: the Neurospora crassa (common bread mold) BAC, with accession number BX842596, which is 77,292 bases long. The instances generated are free from errors of types 4 and 5 (see Section 2) and the remainder errors are considered and eliminated during the calculation of the overlap among the fragments.

We must remark that the instance is quite complex. It is often the case that researches use only one or two instances of low-medium sizes (15-30k bases long). We dare to include two large instances (up to 77k bases long) because the efficiency of our technique, that will be shown to be competitive to modern assemblers.

We experimented with coverage 4 and 7. The latter instances are very hard since they are generated from very long sequences using a small/medium value of coverage and a very restrictive cutoff (threshold to join adjacent fragments in the same contig). The combination of these parameters produces a very complex instance. For example, longer target sequences have been solved in the literature [6], however they have a higher coverage which makes then not so difficult. The reason is that the coverage measures the redundancy of the data, and the higher coverage, the easier the problem. The cutoff, which we have set to thirty (a very high value), provides one filter for spurious overlaps introduced by experimental error. Instances with these features have been solved in the past only when target sequences vary from 20k to 50k base pairs [12, 13, 17] while we solve instances up to 70k base pairs.

Table 1 presents information about the specific fragments sets we use to test our algorithm.

From these previous analyses, we conclude that the best settings for our problem instances of the fragment assembly problem are the following: the GA uses a population size of 1024 individuals, OX as crossover operator (with probability 0.8), and with an edge swap mutation operator (with probability 0.2). We use the same configuration for the hybrid one, including the PALS as additional evolutionary operator with a very low probability (2/population_size). A summary of the conditions for our experimentation is found in Table 2. Values in the tables are average results over 30 independent runs. Since we deal with stochastic algorithms, we have carried out an statistical analysis of the results which consists of the following steps. A Kolmogorov-Smirnov test is performed in order to check whether the variables are normal or not. All the Kolmogorov-Smirnov normality tests in this work were not successful, and therefore we use a non-parametric test: Kruskal-Wallis (with 95 % of confidence).

5.2. Performance Analysis

We thoroughly compare in this section the results obtained by the hybrid GA+PALS, a GA and the PALS heuristic itself. Additionally, these algorithms are compared versus some of the most popular assemblers in the literature in terms of the final number of assembled contigs in the last part of this section.

We show in tables 3 and 4 the results obtained by the studied algorithms for instance BX842596 with coverage 4 and 7, respectively. Specifically, we compare the algorithms in terms of the best and average solutions found, the best and average number of contigs in the solution, and the average elapsed wall clock time. The best results are **bolded**. All the obtained results are statistically significant (it is indicated by + symbol in **Test** row).

It can be seen in these two tables that the proposed hybrid method is the best of the three compared algorithms for the two studied instances in terms of best and average solution found, and the number of contigs in the solution. This means that GA+PALS is very good at finding accurate solutions, which is the main goal of biologists. Conversely,

[1] `http://www.ncbi.nlm.nih.gov/`

Cuadro 2: Parameters for GA and GA+PALS.

Paramter	**Value**
Independent runs	30
Cutoff	30
Max Evaluations	512000
Popsize	1024
P_c	0.8
P_m	0.2
Selection	Ranking
P_{PALS}	2/Popsize

Cuadro 3: Comparison of the studied algorithms. Instance BX842596(4)

	Best Solution	**Average Solution**	**Best Contigs**	**Average Contigs**	**Time (seconds)**
PALS	227130	226744,33	4	9,9	**3,87**
GA	92772	88996,6	6	14,29	32,62
GA+PALS	**228810**	**227331,24**	**2**	**4,71**	298,06
Test	•	+	•	+	+

it is much slower than PALS, as it was expected. The GA is the worst algorithm with statistically significant results in all the cases (**Test** row).

An interesting finding is that, even thought the GA is the worst algorithm, it becomes the more accurate after hybridizing it with PALS as a local search step. Also, although the run time is clearly penalized (population driven search), it is still highly competitive with the existing assemblers (see Table 5 in the literature (which require several hours). Indeed, the hybrid method is not only the most accurate of the compared algorithms, but also very robust in its accuracy (lowest average contigs, see Table 3). Additionally, since GAs are very easily parallelizable [18], the utilization of parallel platforms or even grid systems might allow a considerable reduction of the execution time.

Cuadro 4: Comparison of the studied algorithms. Instance BX842596(7)

	Best Solution	**Average Solution**	**Best Contigs**	**Average Contigs**	**Time (seconds)**
PALS	443512	440779,3	**2**	7,8	**24,84**
GA	308297	304330,6	4	14,29	185,77
GA+PALS	**445426**	**443234,81**	**2**	**3,69**	1531,31
Test	•	+	•	+	+

Cuadro 5: Best final number of contigs for the studied assemblers and for other specialized systems.

	PALS	**GA**	**GA+PALS**	**PMA [13]**	**CAP3 [6]**	**Pharp [3]**
BX842596(4)	4	6	**2**	7	6	6
BX842596(7)	**2**	4	**2**	**2**	**2**	**2**

The three studied algorithms are compared versus some of the most important assemblers existing in the literature in Table 5. These algorithms are a pattern matching algorithm (PMA) [13] and two commercially available packages: CAP3 [6] and Phrap [3]. From this table we can conclude that the proposed GA+PALS is a really competitive tool in the best trade-off for biologists and computer scientists, clearly outperforming all the compared algorithms, and thus representing a new state of the art.

6. Conclusions

The DNA fragment assembly is a very complex problem in computational biology. Since the problem is NP-hard, the optimal solution is at impossible to find for real cases, only for very small problem instances. Hence, computational techniques of affordable complexity such as heuristics are needed for it.

We proposed in this work a new hybrid algorithm for solving the DNA fragment assembly problem. The algorithm was obtained after hybridizing a GA with the PALS heuristic, a recently proposed algorithm that represents the state of the art for this problem. The resulting algorithm clearly improves the results of PALS, as well as those of

an equivalent GA without local search. The drawback is that, the resulting GA+PALS considerably increases the computational time with respect to PALS.

Referencias

[1] J. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology*, chapter 4 - Fragment Assembly of DNA, pages 105–139. University of Campinas, Brazil, 1997.

[2] M. Pop. Shotgun sequence assembly. *Advances in Computers*, 60:194–248, 2004.

[3] P. Green. Phrap. http://www.mbt.washington.edu/phrap.docs/phrap.html.

[4] G. G. Sutton, O. White, M. D. Adams, and A. R. Kerlavage. TIGR Assembler: A new tool for assembling large shotgun sequencing projects. *Genome Science & Technology*, pages 9–19, 1995.

[5] T. Chen and S. S. Skiena. Trie-based data structures for sequence assembly. In *The Eighth Symposium on Combinatorial Pattern Matching*, pages 206–223, 1998.

[6] X. Huang and A. Madan. CAP3: A DNA sequence assembly program. *Genome Research*, 9:868–877, 1999.

[7] E. W. Myers. Towards simplifying and accurately formulating fragment assembly. *Journal of Computational Biology*, 2(2):275–290, 2000.

[8] P. A. Pevzner. *Computational molecular biology: An algorithmic approach*. The MIT Press, London, 2000.

[9] G Luque and E. Alba. Metaheuristics for the DNA Fragment Assembly Problem. *International Journal of Computational Intelligence Research*, 1(2):98–108, January 2006.

[10] E. Alba and G. Luque. A new local search algorithm for the dna fragment assembly problem. In *Evolutionary Computation in Combinatorial Optimization, EvoCOP'07*, Lecture Notes in Computer Science 4446, pages 1–12, Valencia, Spain, April 2007. Springer.

[11] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[12] R. Parsons, S. Forrest, and C. Burks. Genetic algorithms, operators, and DNA fragment assembly. *Machine Learning*, 21:11–33, 1995.

[13] L. Li and S. Khuri. A comparison of DNA fragment assembly algorithms. In *International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences*, pages 329–335, 2004.

[14] S. Lin and B.W. Kernighan. An effective heuristic algorithm for TSP. *Operations Research*, 21:498–516, 1973.

[15] C. Cotta. A study of hybridisation techniques and their application to the design of evolutionary algorithms. *AI Communications*, 11(3-4):223–224, 1998.

[16] M. L. Engle and C. Burks. Artificially generated data sets for testing DNA fragment assembly algorithms. *Genomics*, 16, 1993.

[17] Y. Jing and S. Khuri. Exact and heuristic algorithms for the DNA fragment assembly problem. In *Proceedings of the IEEE Computer Society Bioinformatics Conference*, pages 581–2, Stanford Univeristy, August 2003. IEEE Press.

[18] E. Alba, editor. *Parallel Metaheuristics: A New Class of Algorithms*. Parallel and Distributed Computing. Wiley, 2005.