Information Sciences 277 (2014) 273–283

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

An improved trajectory-based hybrid metaheuristic applied to the noisy DNA Fragment Assembly Problem

Gabriela Minetti [a,*], Guillermo Leguizamón [b], Enrique Alba [c,d]

[a] Laboratorio de Investigación en Sistemas Inteligentes, National University of La Pampa, Argentina
[b] Laboratorio de Investigación y Desarrollo en Inteligencia Computacional, National University of San Luis, Argentina
[c] Dpto. de Lenguajes y Ciencias de la Computación, University of Málaga, Spain
[d] Technical University of Ostrava, Czech Republic

ARTICLE INFO

Article history:
Received 28 October 2011
Received in revised form 9 December 2013
Accepted 9 February 2014
Available online 18 February 2014

Keywords:
Metaheuristic
Simulated Annealing
Problem Aware Local Search
Parallelism
Noisy instance
DNA Fragment Assembly Problem

ABSTRACT

The DNA Fragment Assembly Problem (FAP) is an NP-complete that consists in reconstructing a DNA sequence from a set of fragments taken at random. The FAP has been successfully and efficiently solved through metaheuristics. But these methods usually face difficulties to succeed when noise appears in the input data or during the search, specially in large instances. In this regard, the design of more efficient techniques are indeed necessary. One example of these techniques found in literature is the Problem Aware Local Search (PALS) which represents a state-of-the-art and robust assembler to solve noisy instances. Although PALS performs better than other metaheuristics, the quality of the achieved solutions by this method can still be improved. Towards this aim, this work proposes a new hybrid and effective method that combines a local search technique specially designed for this problem (PALS) with Simulated Annealing (SA), which is further distributed following an island model. Our proposed hybrid approach showed an improved performance on the largest non-noisy and noisy instances when compared separately with Simulated Annealing and PALS.

1. Introduction

The Fragment Assembly Problem (FAP) consists in building a DNA sequence from several hundreds (or even thousands) of fragments which are obtained in the laboratory. The DNA fragment assembly is needed because current technology, such as gel electrophoresis, cannot sequence DNA molecules longer than 3000 bases in a direct and accurate way. Moreover, most genomes are much longer than 3000 bases. FAP is the primary goal in any genome project and the remaining phases strongly depend on the accuracy of the results at this stage. Therefore, we need accurate and efficient methods to solve this problem. Unfortunately, the length of the DNA sequence is not the only difficulty that researchers face. Noise is another important obstacle that can appear in the laboratory when the sequencing of long DNA strands is carried out in the laboratory. Thus, the data used for assembling the fragments may contain errors.

Most sequence assembly algorithms are based on variations of a greedy algorithm, e.g., Phrap [11], CAP3 [12], Celera assembler [25], TIGR Assembler [29], and STROLL [8]. Through such greedy approaches, the fragments are assembled by repeatedly merging a pair of them with the highest overlap, according to a specific and complex criterion. The methods mentioned before obtain high quality results for small–medium sequences, but they present some drawbacks with respect to many large genome sequencing projects. In contrast, metaheuristic techniques are being used with accurate results on

* Corresponding author. Tel.: +54 2302422780.
E-mail address: minettig@ing.unlpam.edu.ar (G. Minetti).

http://dx.doi.org/10.1016/j.ins.2014.02.020
0020-0255/

small–medium sequences and also in larger ones. What is more, many efficient assemblers are based on metaheuristics, such as Simulated Annealing [19], Variable Neighbourhood Search (VNS) [2,5], Genetic Algorithms (GAs) [2,14–16], Ant Colony Optimization (ACO) [19], and Problem Aware Local Search (PALS) [3,4].

It is worth noting that, on a general basis, the algorithms mentioned above have never dealt with problem instances containing any kind of noise in their data. Furthermore, few works about metaheuristics report basic studies dealing with errors or noise in some stages of FAP, for instance [13,20,21,31]. Specifically Minetti et al. [21], used SA, PALS, and GA to solve non-noisy and noisy instances of FAP. Moreover, they dealt with three different sources of noise: Noisy Nucleotide Base (NB), Noisy Score Matrix (NS), and Noisy Fitness (NF). The authors concluded that SA is the best of the three methods applied to solve non-noisy instances whereas PALS is the best and most robust assembler to solve the noisy ones. Unfortunately, for the largest instances, PALS found solutions with an undesirable number of contigs.

As obtaining a DNA sequence mainly depends on the accuracy and efficiency of the applied assemblers, our interest is twofold: analyze the PALS shortcomings and propose a more accurate and efficient technique based on this metaheuristic to solve noisy instances of FAP. Our proposal draws on the benefits of PALS strengths and mitigates their weaknesses. Thus, the new assembler (in the same way as PALS) uses an inexpensive way to evaluate each candidate movement of fragments. But, unlike PALS, it is designed to avoid a quick convergence to local optima by incorporating two useful design strategies: hybridization with SA and parallelization through an island model.

Over the last years, the interest in hybrid metaheuristics has notably increased. Combinations of trajectory-based metaheuristics, population-based metaheuristics, mathematical programming, constraint programming, and machine learning techniques have provided very powerful search algorithms. The most frequently implemented model of hybrid metaheuristics is the combination of trajectory-based (e.g., local search, Simulated Annealing, taboo search, etc.) with population-based metaheuristics (e.g., genetic algorithms, scatter search, swarm optimization, etc.). This kind of hybridization is known as *low-level teamwork hybrid* [30] and it was used by Chelouah and Siarry [7], Salto et al. [27], Minetti et al. [23], Duarte et al. [9], among many other researchers.

Alternatively, only few hybrid metaheuristic models use the strategy called *low-level relay hybrid* [30] which consists in embedding a given metaheuristic (such as SA) into a trajectory-based metaheuristic (such as PALS). For example, Martin et al. [17] incorporated deterministic local search techniques into SA in order to explore only local optima, and Mashinchi et al. [18] proposed a method based on taboo search and Nelder–Mead search strategy applied to global continuous optimization problems. Following this hybridization approach, we propose a low-level relay hybrid technique to embed SA into PALS in order to avoid local optima, i.e., to promote exploration when a stagnation point is detected.

Also, it is worth noting that several parallel models that have been designed for metaheuristics collaborate with the hybridization techniques to avoid local optima, since those parallel models usually increase the exploration. Furthermore, these kinds of models constitute an effective strategy to reduce the search time, improve the solution quality and robustness, and solve large-scale problems. For instance, the popular island model performs sparse exchanges of solutions among sets of solutions (islands). Consequently, time complexity is significantly reduced but keeping two desirable features: diversity of solutions and exploitation of promising regions of the search space. Based on the above considerations, we propose an assembler based on a hybrid metaheuristic which is implemented according to the widely known island model formerly proposed for parallel metaheuristics.

The rest of the paper begin describing in Section 2 the Fragment Assembly Problem. Section 3 introduces and explains our algorithmic proposal and Section 4 describes the experimental design. Then, an analysis of the algorithmic behavior for non-noisy and noisy instances is presented in Section 5. This analysis includes a detailed study of our proposal and a comparison with existing assemblers. Finally, conclusions and hints for further research are discussed.

2. The DNA Fragment Assembly Problem

The reconstructs an original DNA sequence from a set of separate fragments which are obtained by a sequencing procedure. One of the most widely used sequencing procedures is *shotgun sequencing* [28]. This procedure firstly cuts the DNA into small pieces, then identifies the sequences in each of these fragments, and lastly puzzles the fragments together to create the original contiguous sequence, i.e., *the contig*. The main advantages of the shotgun sequencing method are its high level of automation and its scalability. Specifically, the shotgun sequencing method involves three steps:

1. Several copies of the DNA are produced and each copy is broken into millions of random fragments.
2. Those fragments are read by a DNA sequencing machine.
3. An assembler pieces together many overlapping fragments and reconstructs the original sequence.

The DNA fragment assembly process is divided into three different phases (as shown in Fig. 1):

1. *Overlap*: finding the overlapping among fragments (score). This phase consists in finding the best or longest match between the suffix of the first sequence and the prefix of the second one for each pair of sequences. All possible pairs of fragments are compared to determine their similarity. Usually, the dynamic programming algorithm is used in this step to find semi-global alignments.

2. *Layout*: finding the fragment order based on computed similar scores. This is the most difficult step due to the fact that true overlaps are hard to determine. After the order is determined, an alignment algorithm is applied to combine all the pair-wise alignments obtained in the overlap phase.
3. *Consensus*: deriving the DNA sequence from the layout. The most common technique used in this phase is to apply the majority rule to build the consensus.

If no sequencing error is detected at the overlap phase, the process simply finds the longest suffix of one string that matches exactly the prefix of another one. However, when sequencing errors exist, the process searches for the best match but a small percentage of errors still remains (1–3%). During the assembly process the only information available is the sequence of bases; therefore, the ordering of the fragments must primarily rely on fragment similarity and on how much they overlap.

Once the fragments have been ordered (layout), the final consensus is generated. This means that a multiple alignment is computed to obtain a consensus sequence that will be used as the final genomic sequence. The quality of a consensus can be measured by measuring the coverage distribution. The coverage at a base position is defined as the number of fragments present in that position, i.e., it measures the redundancy of fragment data. Moreover, it represents on average, the number of fragments in which a given nucleotide in the target DNA is expected to appear. The coverage is then computed as the number of bases read from fragments over the length of the target DNA:

$$Coverage = \frac{\sum_{i=1}^{n} length\ of\ the\ fragment\ i}{target\ sequence\ length}, \tag{1}$$

where n stands for the number of fragments. Thus, the higher the coverage and the smaller the number of gaps, the better the results. A partial coverage is achieved provoked when the algorithm cannot assemble a given set of fragments into a single contig. More precisely, a contig is defined as a layout consisting of contiguous overlapping fragments. Overlapping between adjacent fragments must be greater than or equal to a predefined threshold (i.e., cutoff parameter).

Particularly, the assembly of DNA fragments into a consensus sequence corresponding to the parent sequence constitutes the "Fragment Assembly Problem", which is a permutation NP-hard problem [26].

3. Parallel and hybrid PALS-based metaheuristic

PALS [3] uses a specific local search to improve a single solution. In this way, this metaheuristic assembler performs a trajectory in the search space. PALS' main strength is the inexpensive calculation of the variation in the overlap and in the number of contigs between the current solution and the resulting solution after applying a movement. This calculation is not computationally expensive since neither the fitness function nor the number of contigs are calculated in each iteration; instead, PALS estimates the variation of these values. However, its most important weakness is the quick convergence to local optima caused by the local search. This drawback is observed when PALS is already applied to large instances, especially if those instances are noisy, as it has been reported [21].

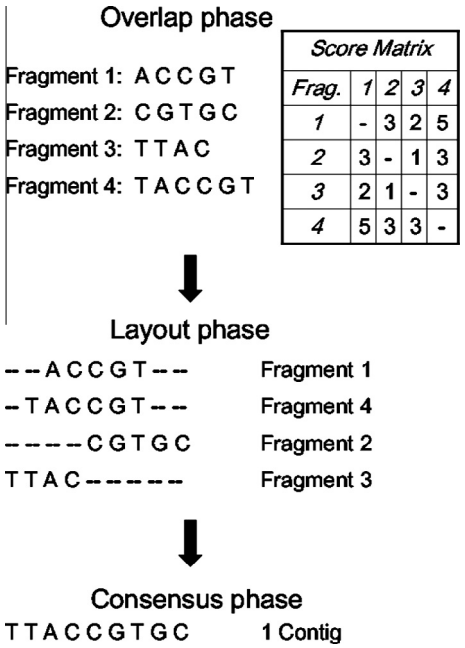

Fig. 1. DNA Fragment Assembly Process.

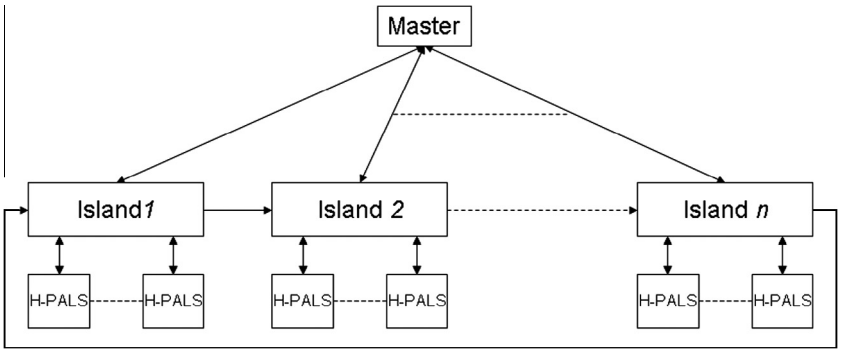

Fig. 2. PH-PALS.

In order to take advantage of PALS' strengths and improve the accuracy of results, we propose a new metaheuristic based on PALS. Our proposal uses three different mechanisms to prevent local optima by:

1. Extending the local search to all possible fragment movements in a solution.
2. Applying SA (refer to [21]) when the best movement does not produce any change after a specified number of tries.
3. Using the island model to promote diversification as well as intensification during the search.

The main aim of the first mechanism is to force screening through all the remaining movements; however, only the best one is kept. The second mechanism introduces an adaptive hybridization, since SA is applied if, after a certain number (threshold) of tried moves, PALS does not accept another best movement. These two mechanisms generate a new hybrid metaheuristic, called H-PALS (see Algorithm 1).

Algorithm 1. H-PALS.

Generate S; {generate the initial solution by using the greedy seeding strategy}
$k = 0$; {initialize the current iteration counter}
$threshold = 5 * N$; {N is the number of fragments}
repeat
  $t = 0$; {initialize the current movement counter}
  $L = \varnothing$;
  bestΔ_c = best$\Delta_f = 0$;
  for $i = 0$ to N **do**
    for $j = 0$ to N **do**
      $\Delta_c,\ \Delta_f$ = CalculateDelta(S, i, j); {see Algorithm 2}
      if $(\Delta_c \leqslant 0)$ and $(\Delta_c <$ best$\Delta_c)$ and $(\Delta_f >$ best$\Delta_f)$ **then**
        $L = L \cup < i, j, \Delta_c, \Delta_f >$; {Add candidate movements to L}
        best $\Delta_c = \Delta_c$;
        best $\Delta_f = \Delta_f$;
        $t = 0$;
      else
        $t = t + 1$;
      end if
    end for
    {K is the max. number of iterations per individual}
    if $(threshold \geqslant t)$ and $(k < K)$ **then**
      Apply SA(S, k); {Apply SA to the current solution increasing the number of iterations (k)}
      $L = \varnothing$;
      best Δ_c = best$\Delta_f = t = 0$;
    end if
  end for
  if $L <> \varnothing$ **then**
    $< i, j, \Delta_c, \Delta_f >$ = Extract(L); {Select the best movement from L}
    ApplyMovement(S, i, j); {Modify the current solution}

$k = k + 1$; {Increment the number of iterations}
end if
until $k >= K$
return S;

The strategy of embedding SA into PALS – a low-level relay hybridization [30] – reintroduces exploration in the search process without a negative impact on exploitation. In other words, as SA accepts with a high probability solutions of lower quality during the first stage of the search, the hybrid algorithm will be capable of avoiding local optima. During the annealing process, this probability decreases, thus intensifying the search and reducing the exploration in order to exploit a restricted area of a search space. Therefore, the algorithm can restart the search from another potential promising region towards avoiding stagnation.

Algorithm 2. CalculateDelta.

$\Delta_c = 0$;
$\Delta_f = 0$;
{Calculate the variation in the overlap}
$\Delta_f = w_{S[i-1],S[j]} + w_{S[i],S[j+1]}$; {Add the score of the new overlap}
$\Delta_f = \Delta_f - w_{S[i-1],S[i]} - w_{S[j],S[j+1]}$; {Remove the score of the current overlap}
{Test if a contig is broken, and if so, increase the number of contigs}
if $(w_{S[i-1],S[i]} >$ cutoff) or $(w_{S[j],S[j+1]} >$ cutoff)**then**
$\Delta_c = \Delta_c + 1$;
end if
if $(w_{S[i-1],S[j]} >$ cutoff) or $(w_{S[i],S[j+1]} >$ cutoff) **then**
$\Delta_c = \Delta_c - 1$;
end if
return Δ_c, Δ_f;

Finally, H-PALS is distributed into small islands to take advantage of a multicore processor architecture. Thus, the number of individuals (solutions) on each island matches the number of cores in the processor. Consequently, each core runs a thread where only one permutation is processed by this new hybrid metaheuristic. In this way, two levels of parallelism can be recognized: one is given by the island model that runs one island per machine, the other one is presented on each island. Therefore, a thread is created on each core where only one permutation is computed at a time. As a consequence of applying this third mechanism to H-PALS, a new parallel algorithm called PH-PALS is created.

In the island model, a set of solutions (population in the island) is processed independently from others. Moreover, PH-PALS tries to overcome premature convergence by preserving the diversity generated by the semi-isolation of the populations. With a given migration frequency, the multiple islands exchange solutions between them over a certain communication topology. Due to this, PH-PALS enables cooperation by exploiting promising areas found by other islands.

PH-PALS arranges the islands in a unidirectional ring with asynchronous communication and each island migrates a solution when k is multiple of 25,000. Besides, the source island chooses the best solution as a migrant, and the target island selects and replaces the worst individual if the incoming one is better. Hence, the algorithm jumps to another search area where better solutions have been found. Fig. 2 shows a working scheme of PH-PALS where the master process is in charge of:

1. generating n islands,
2. receiving a copy of the best individual from each island, and
3. selecting and returning the best individual received.

In addition, each island completes these tasks by:

1. generating m threads and one individual per thread,
2. running H-PALS over the individual in each thread,
3. selecting the best individual and sending a copy of it to the next island,
4. receiving an individual from the previous island,
5. replacing the worst individual if the incoming one is better or equal, and
6. selecting the best individual and sending a copy of it to the master when all threads end.

The main idea of the second level of parallelism is to take advantage of the current processor architecture. In order to do so, a thread is created to run H-PALS in each core. Therefore, m permutations are processed in parallel on each island. Besides,

Table 1
Information of data sets. Accession numbers are used as instance names.

Instances	Coverage	Mean fragment length	Number of fragments	Original sequence length
GenFrag instances				
x60189_4	4	395	39	3835
x60189_5	5	386	48	
x60189_6	6	343	66	
x60189_7	7	387	68	
m15421_5	5	398	127	10089
m15421_6	6	350	173	
m15421_7	7	383	177	
j02459_7	7	405	352	20000
bx842596_4	4	708	442	77292
bx842596_7	7	703	773	
DNAgen instances				
acin1	26	182	307	2170
acin2	3	1002	451	147200
acin3	3	1001	601	200741
acin5	2	1003	751	329958
acin7	2	1003	901	426840
acin9	7	1003	1049	156305

if the first level of parallelism is considered, $m \times n$ permutations are processed at the same time in the ring. Consequently, while PALS processes one solution at a particular time, PH-PALS processes $m \times n$ solutions simultaneously.

In previous works [21–24] we detected significant improvements in the performance of metaheuristic assemblers when a greedy seeding strategy was used to create FAP initial solutions. Therefore, PH-PALS is designed to use the greedy technique proposed by Minetti et al. [22] in order to create the respective initial solutions on each core.

We also considered a panmictic version of this algorithm, called PanH-PALS, where a population of $m \times n$ individuals is processed in only one computer. In this case, each individual is processed separately from others using H-PALS during 500,000 iterations.

4. Experimental methodology

In this section we describe the problem instances used in the different computational experiments designed for this work, as well as the execution environment.

We chose four sequences from the NCBI website[1]: a human MHC class II region DNA with fibronectin type II repeats HUMMHCFIB, with accession number x60189; a human apolopoprotein HUMAPOBF, with accession number M15421; the complete genome of bacteriophage lambda, with accession number J02459; and a sequence of Neurospora crassa BAC, with accession number BX842596 (GI38524243). As shown in Table 1, we used data generated by GenFrag [10] to generate the different data sets from these sequences. GenFrag is a UNIX/C application created to accept a DNA sequence as input, and to generate a set of overlapping fragments as output. Furthermore, we selected other sequences from the NCBI website, which correspond to a human microbion bacterium ATCC 49176 with accession numbers from ACIN02000001 to ACIN02000026. Particularly, we used the longest sequences from this genome, and we fragmented them with the DNAgen application[2]. These new instances are shown in the second part of Table 1. The cutoff, which we set to thirty (a very high value), provides a filter for spurious overlaps introduced by experimental error.

This set of instances only contains non-noisy data whereas our goal is to test the capacity of the proposed assembler to deal with instances with and without noise. Particularly in this work, noise comes from three different sources: the Nucleotide Bases (NB), the Score Matrix (NS), and Fitness (NF). Therefore, noise occurs in three different phases of FAP resolution: before the first phase – Overlap – if it occurs in the nucleotide bases; during the Overlap phase if errors appear in the score (or overlapping) matrix, and during the Layout phase, if the fitness is noisy. A noteworthy fact is that the NB and NS sources of noise lead to difficulties in aligning the overlapping portions of sequences and in determining the consensus sequence, while NF can misguide the metaheuristic search. In Minetti et al. [21], new sets of noisy instances were generated by these three sources of noise, using the instances described in Table 1. Consequently, we used the instances with noise in the nucleotide bases, *NB-originalInstanceName* (e.g., *NB-x60189_4*), the instances with 10% of noise in the score matrix, *NS10-originalInstanceName* (e.g., *NS10-acin1*), and instances with noisy fitness, *NF-originalInstanceName* (e.g., *NF-38524243_7*).

We used the MALLBA library [6] to implement H-PALS, PanH-PALS, and PH-PALS. The experimentation was carried out on a cluster of 12 computers with Intel Core 2 Duo at 3 GHz, linked by Fast Ethernet, under Linux with a 2.4.19–4 GB kernel version.

[1] http://www.ncbi.nlm.nih.gov/.
[2] http://mdk.ing.unlpam.edu.ar/lisi/principal.html.

Table 2
Parametric values used by H-PALS, PH-PALS, and PanH-PALS.

Parameter	Value
H-PALS	
Markov chain length	10
Temperature decay	0.99
Selection movement	The best movement
Threshold	$5\times$ *Number of fragments*
Max. iterations (K)	500,000
PH-PALS	
Number of individuals per Island (m)	2
Number of Islands (n)	3, 6, 9, and 12
Migration frequency	25,000 iterations
Migration policy	The best individual is sent The worst individual is replaced If the incoming one is better or equal
PanH-PALS	
Population size	$m \times n$

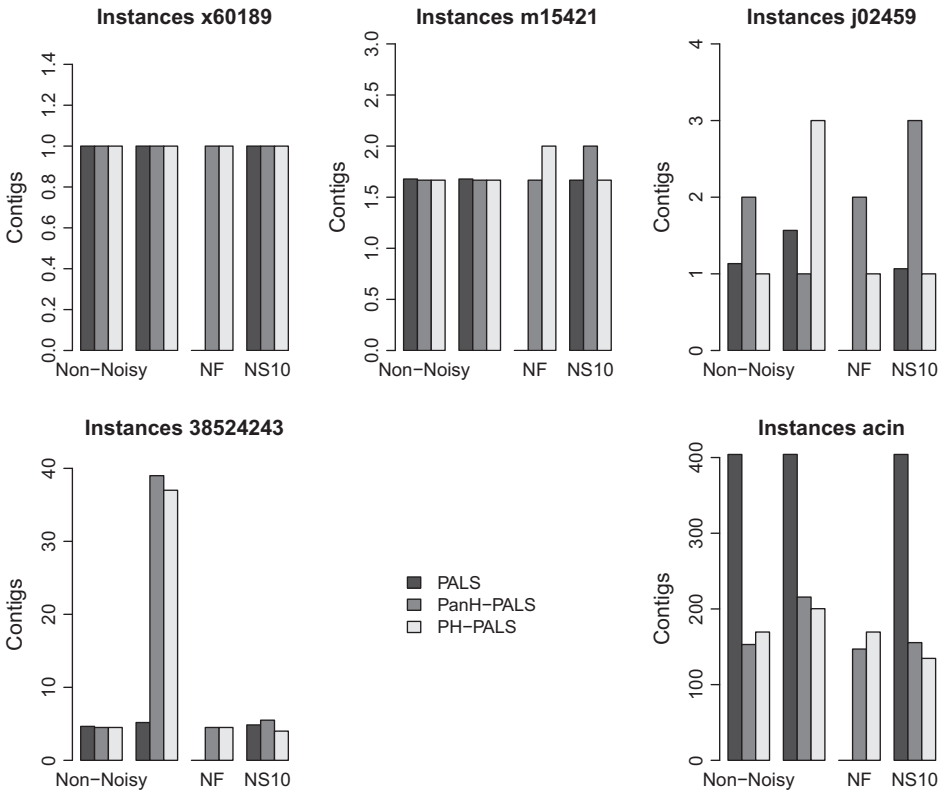

Fig. 3. Average number of contigs found by PALS, PanH-PALS, and PH-PALS under the panmictic and distributed versions for each group of non-noisy and noisy instances. (Solving NF instances with PALS makes non-sense).

Each island was physically run on a separate processor. In order to study the PH-PALS behavior on different numbers of processors, PH-PALS has been distributed on 3, 6, 9, and 12 islands. Two permutations were processed in parallel in each processor, one by core. Furthermore, the panmictic version was also tested. Because of the stochastic nature of the algorithms, 30 independent runs of each test were performed to gather meaningful experimental data and apply statistical confidence metrics to validate the results and conclusions. Some preliminary runs were performed and the best results were found by using the parameter values summarized in Table 2.

5. Experimental results

In this section we analyzed the behavior of PanH-PALS and PH-PALS to solve non-noisy and noisy FAP instances. We also compared their outcomes with the results obtained by PALS [21]. After that, we used the speedup measure to study the PH-PALS performance. Finally, we compared the quality of the results obtained by PanH-PALS and PH-PALS with other assemblers proposed in literature. It should be noted that each analysis was statistically corroborated by applying the Kruskal–Wallis test with a 95% of confidence level to the results obtained for each instance.

5.1. Behavior analysis

Fig. 3 shows the results obtained from PALS, PanH-PALS, and PH-PALS for the non-noisy and noisy cases. It must be noted that PALS does not evaluate the fitness function during the search; as a consequence, solving NF instances with PALS makes no sense.

Afterwards, we compared the behavior of PALS, PanH-PALS, and PH-PALS on non-noisy instances and all three sources of noise (NB, NF, and NS10) and, based on this comparison (see Fig. 3) we detected that:

- For the smallest instances (accession number *x60189*), all three assemblers found the optimal number of contigs on non-noisy, NB, NF, and NS10 cases.
- For medium instances (*m15421*), a good behavior is shown by all three assemblers since they find solutions with at most two contigs (optimal is always one contig). Furthermore, PALS, PanH-PALS, and PH-PALS showed a similar behavior

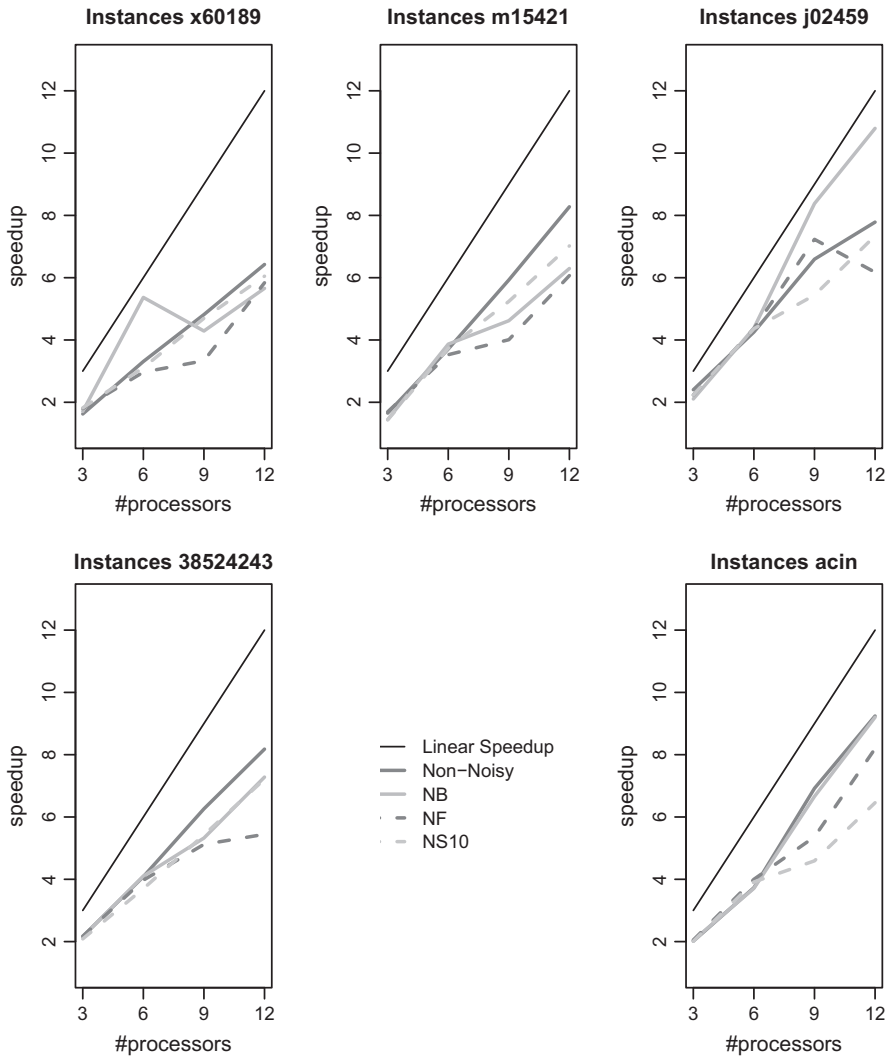

Fig. 4. PH-PALS speedup for different number of processors.

Table 3
Average number of contigs obtained by PH-PALS, SA, PALS, GA, CAP3, and PHRAP. Symbol – showed that this information cannot be computed. The best values are in boldface.

Instances	PH-PALS	SA	PALS	GA	CAP3	PHRAP
x60189	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
m15421	1.67	**1.00**	1.67	1.67	3.33	1.50
j02459	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
38524243	4.50	**1.00**	4.50	4.50	5.00	4.00
acin	169.5	**1.00**	404.00	404.17	407.33	–
NB-x60189	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	–
NB-m15421	**1.67**	**1.67**	**1.67**	**1.67**	3.33	–
NB-j02459	2.00	3.00	3.00	2.00	**1.00**	–
NB-38524243	37.00	6.00	**4.50**	5.00	6.50	–
NB-acin	**200.33**	404.00	556.33	404.00	412.33	–

pattern when solving non-noisy and NB instances. Instead, when the NF-*m15421* instances were solved, PanH-PALS obtained slightly better solutions than PH-PALS. Finally, for *NS10-m15421* instances PALS and PH-PALS found better solutions than PanH-PALS.

- For the instances with accession number *j02459* and 352 fragments we found more differences than similarities on the behavior of the three assemblers. For non-noisy, NF, and NS10 instances, only PALS and PH-PALS found optimal solutions. Instead, for the NB cases, PALS and PanH-PALS found the optimal number of contigs.
- For still instances (*38524243*), none of the three assemblers found the optimal solution. Specifically for the non-noisy, NF, and NS10 instances, all three assemblers behaved in a similar way since, on average, they obtain solutions with five contigs. However, for the NB cases, PALS was the best choice, since it obtained results with five contigs in contrast with PanH-PALS and PH-PALS that found solutions with 35 or more contigs.
- For the largest instances (*acin*), the number of contigs found by PanH-PALS and PH-PALS decreased more than 50% in comparison with the results found by PALS. In particular, for non-noisy and NF instances, PanH-PALS was the best choice, and for NB and NS10 instances the best one was PH-PALS.

To summarize, when PanH-PALS and PH-PALS solve the largest instances, they achieve the goal proposed in this paper. Thus, they are capable of avoiding local optima and decreasing the number of contigs in the solutions for larger noisy instances. Moreover, for the remaining instances, our proposal performs as well as PALS. Additionally, PH-PALS and PALS share an interesting feature; they are robust in the presence of noisy information, since no significant differences are detected among solutions found in non-noisy and noisy instances.

5.2. Performance analysis

In this section we analyzed (see Fig. 4) the PH-PALS performance taking into account the speedup measure. The speedup measure compares the serial time with respect to the parallel time to solve a particular instance of FAP. More specifically, we used the weak speedup proposed by Alba [1] because PH-PALS is a non-deterministic algorithm. In order to assess the speedup of PH-PALS properly, we ran this assembler on 3, 6, 9, and 12 processors using 3, 6, 9, and 12 islands, respectively.

We also studied the PH-PALS speedup on non-noisy instances and all three sources of noise (NB, NF, and NS10). After performing this study (Fig. 4), we concluded that:

- In general, the speedup grows as the number of processors increases (except for *NB-x60189* and *NF-j02459* instances).
- For all cases, we obtained a sublinear speedup, since it was less than the respective number of processors.
- In general, PH-PALS achieved a higher speedup when solving non-noisy instances, although the differences between non-noisy instances and each noisy scenario (NB, NF, and NS10) were not statically significant.

In general, we can stated that PH-PALS is able to scale with larger computational resources, what constitutes a desirable feature of all parallel algorithms.

5.3. Comparison with existing assemblers

In this section, we compared the behavior of our approach with other well-known assembler algorithms found in literature: CAP3 [12], PHRAP [11], SA [21], PALS [21], and GA [21]. The first two packages automate fragment assembly by using a variety of greedy-based techniques. On the other hand, the last three assemblers are metaheuristic algorithms adapted to solve FAP. It is worth mentioning that, regarding the computational effort, these assemblers apply the stopping criterion proposed by Minetti et al. [21] in order to make a similar effort to solve the same instance.

To compare the algorithms mentioned above, in Table 3 we show the average number of contigs that PH-PALS, SA, PALS, GA, and CAP3 found for non-noisy and noisy instances (NB). Due to the way in which noise is considered in the problem, CAP3 can only work with the NB instances and PHRAP with several non-noisy instances: *x60189*, *m15421*, *j02459*, and 38524243. On the other hand, given the absence of an "evaluation" concept in PALS and the huge sets of data in Bioinformatics, we decided to use real time as stopping criterion to establish a fair comparison among these metaheuristic assemblers. For this reason, PH-PALS, SA, PALS, and GA are given 60 s as execution time and then their attained accuracy is reported.

After analyzing Table 3, we found that SA provided the best solution for all non-noisy instances. Furthermore, PH-PALS outperformed PALS, GA, CAP3, and PHRAP when solving the non-noisy *acin* instances; since the number of contigs decreased between 30% and 75%. For the remaining non-noisy instances, PH-PALS behaved in a similar way to PALS, GA, CAP3, and PHRAP. In addition, for *NB-acin* instances PH-PALS outperformed all these assemblers, including SA, by reducing the number of contigs to between 50% and 65%. For the rest of NB instances PH-PALS generally equaled the quality achieved by the remaining assemblers.

To summarize, for the largest instances PH-PALS outperformed PALS, GA, CAP3, and PHRAP and, for the remaining instances, the behavior was similar among all of them.

6. Conclusions

In this paper we presented PH-PALS, a new parallel and hybrid metaheuristic based on PALS for solving noisy instances of FAP. PH-PALS is the result of the hybridization of PALS with SA, which is additionally distributed and parallelized according to an island model. The use of SA helps PH-PALS to escape from local optima and, as a result, the parallelization increases PH-PALS performance. The quality of the results found by the proposed PH-PALS algorithm clearly outperformed SA, PALS, GA, and CAP3, when it solved the largest noisy instances for all different sources of noise (NS, NB, and NF). PH-PALS also behaved better than PALS, GA, and CAP3 when the largest non-noisy were solved. For the remaining instances, PH-PALS showed the same behavior with respect to the other assemblers. In short, PH-PALS may constitute a new state-of-the-art technique for FAP, since it equals and outperforms the rest of the competitors. Indeed, PH-PALS can be considered a robust assembler, in view of its high accuracy and real time efficiency, both for non-noisy and noisy FAP instances.

As further research lines, we would first attempt to fine-tune the algorithmic parameters with the aim of improving the current results and, hopefully, reach the optimal solutions for noisy instances (ideally, one contig). As a second research line, we plan to increase the speedup of our algorithm in order to obtain, at least, a linear speedup. Another proposal would address the exchange of useful information among individuals belonging to the same island, by applying some crossover operator among them, as previously tested in [22]).

Acknowledgements

The authors wish to thank the following institutions for their financial support which funded this research: National University of La Pampa; ANPCYT, PICTO 2011-0278; National University of San Luis; Spanish Ministry of Sciences and Innovation European FEDER under contract TIN2011-28194 (road ME Project, available at URL http://roadme.lcc.uma.es); VSB-Technical University of Ostrava and IT4Innovation center (Czech Republic) through contract 806/5474142.

References

[1] E. Alba, Parallel evolutionary algorithms can achieve super-linear performance, Inf. Process. Lett. 82 (2002) 7–13.
[2] E. Alba, Parallel Metaheuristics: A New Class of Algorithms, WILEY Series on Parallel and Distributed Computing, Wiley, 2005.
[3] E. Alba, G. Luque, A new local search algorithm for the DNA fragment assembly problem, in: Proceedings of Evolutionary Computation in Combinatorial Optimization, EvoCOP'07, Lecture Notes in Computer Science, vol. 4446, Springer, Valencia, Spain, 2007, pp. 1–12.
[4] E. Alba, G. Luque, A hybrid genetic algorithm for the DNA fragment assembly problem, in: C. Cotta, J. van Hemert (Eds.), Recent Advances in Evolutionary Computation for Combinatorial Optimization, Studies in Computational Intelligence, vol. 153, Springer, Berlin/Heidelberg, 2008, pp. 101–112.
[5] E. Alba, G. Luque, G. Minetti, Variable neighborhood search for solving the DNA fragment assembly problem, in: Proceedings of XIII Congreso Argentino de Ciencias de la Computación (CACIC 2007), Corrientes, Argentina, pp. 1359–1370.
[6] E. Alba, G. Luque, J. Nieto, G. Ordóñez, G. Leguizamón, MALLBA: a software library to design efficient optimization algorithms, Int. J. Innov. Comput. Appl. 1 (2007) 74–85.
[7] R. Chelouah, P. Siarry, Genetic and Nelder–Mead algorithms hybridized for a more accurate global optimization of continuous multiminima functions, Euro. J. Operat. Res. (2003) 335–348.
[8] T. Chen, S. Skiena, A case study in genome-level fragment assembly, in: Proceedings of The Eighth Symposium on Combinatorial Pattern Matching (CPM 1997), pp. 206–223.
[9] A. Duarte, R. Martí, F. Glover, F. Gortázar, Hybrid scatter tabu search for unconstrained global optimization, Ann. OR 183 (2011) 95–123.
[10] M.L. Engle, C. Burks, Artificially generated data sets for testing DNA sequence assembly algorithms, Genomics 16 (1993) 286–288.
[11] P. Green, Phrap, version 1.090518, 2009. <http://www.phrap.org>.
[12] W. Huang, A. Madan, CAP3: a DNA sequence assembly program, Genome Res. 9 (1999) 868–877.
[13] K. Kim, C. Mohan, Parallel hierarchical adaptive genetic algorithm for fragment assembly, Proceedings of The 2003 Congress on Evolutionary Computation, CEC03, vol. 1, IEEE, 2003, pp. 600–607. ISBN: 0-7803-7804-0.
[14] L. Li, S. Khuri, A comparison of DNA fragment assembly algorithms, in: Proceedings of The 2004 International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences, Las Vegas, 2004, pp. 329–335.
[15] G. Luque, E. Alba, Metaheuristics for the DNA fragment assembly problem, Int. J. Comput. Intell. Res. 1 (2005) 98–108.

[16] G. Luque, E. Alba, S. Khuri, Chapter 16: assembling DNA fragments with a distributed genetic algorithm, in: Parallel Algorithms for Bioinformatics, Wiley, 2006.
[17] O. Martin, S. Otto, E. Felten, Large-step Markov chains for the TSP incorporating local search heuristics, Operat. Res. Lett. (1992) 219–224.
[18] M.H. Mashinchi, M.A. Orgun, W. Pedrycz, Hybrid optimization with improved tabu search, Appl. Soft Comput. 11 (2011) 1993–2006.
[19] P. Meksangsouy, N. Chaiyaratana, DNA fragment assembly using an ant colony system algorithm, Proceedings of The 2003 Congress on Evolutionary Computation, CEC03, vol. 3, IEEE, 2003, pp. 1756–1763. ISBN: 0-7803-7804-0.
[20] J.R. Miller, A.L. Delcher, S. Koren, E. Venter, B. Walenz, A. Brownley, J. Johnson, K. Li, C. Mobarry, G. Sutton, Aggressive assembly of pyrosequencing reads with mates, Bioinformatics 24 (2008) 2818–2824.
[21] G. Minetti, E. Alba, Metaheuristic assemblers of DNA strands: noiseless and noisy cases, in: Proceedings of IEEE Congress on Evolutionary Computation (CEC 2010), pp. 1–8.
[22] G. Minetti, E. Alba, G. Luque, Seeding strategies and recombination operators for solving the DNA fragment assembly problem, Inform. Process. Lett. 108 (2008) 94–100.
[23] G. Minetti, G. Luque, E. Alba, Variable neighborhood search as genetic algorithm operator for DNA fragment assembling problem, in: Proceedings of Eighth International Conference on Hybrid Intelligent Systems (HIS'08), Barcelona, Spain, pp. 714–719.
[24] G. Minetti, G. Luque, E. Alba, A new hybrid SA for solving the DNA fragment assembly problem, in: Proceedings of XXVIII Internacional Conference of the Chilean Computing Science Society, Chilean Computing Science Society (SCCC 2009), 2009, pp. 109–116.
[25] E.W. Myers, A whole-genome assembly of drosophila, Science 287 (2000) 219–2204.
[26] P. Pevzner, Computational Molecular Biology: An Algorithmic Approach, The MIT Press, 2000.
[27] C. Salto, G. Leguizamón, E. Alba, J.M. Molina, Hybrid ant colony system to solve a 2-dimensional strip packing problem, in: Eighth International Conference on Hybrid Intelligent Systems (HIS'08), pp. 708–713.
[28] F. Sanger, A. Coulson, G. Hong, D. Hill, G. Petersen, Nucleotide sequence of bacteriophage lambda DNA, J. Mol. Biol. 162 (1982) 729–773.
[29] G.G. Sutton, O. White, M.D. Adams, A.R. Kerlavage, TIGR assembler: a new tool for assembling large shotgun sequencing projects, Genome Sci. Technol. (1995) 9–19.
[30] E.G. Talbi, A taxonomy of hybrid metaheuristics, J. Heuristics 8 (2002) 541–564.
[31] A. Valouev, D.C. Schwartz, S. Zhou, M.S. Waterman, An algorithm for assembly of ordered restriction maps from single DNA molecules, Proc. Natl. Acad. Sci. 103 (2006) 15770–15775.