Variable Neighborhood Search for solving the DNA Fragment Assembly Problem

Gabriela Minetti
Laboratorio de Investigacin en Sistemas Inteligentes
Universidad Nacional de La Pampa
Argentina
minettig@ing.unlpam.edu.ar

and

Enrique Alba - Gabriel Luque
Dpto. de Lenguajes y Ciencias de la Computación
Universidad de Málaga
Spain
{eat,gabriel}@lcc.uma.es

Abstract

The fragment assembly problem consists in the building of the DNA sequence from several hundreds (or even, thousands) of fragments obtained by biologists in the laboratory. This is an important task in any genome project, since the accuracy of the rest of the phases depends of the result of this stage. In addition, real instances are very large and therefore, the efficiency is also a very important issue in the design of fragment assemblers. In this paper, we propose two Variable Neighborhood Search variants for solving the DNA fragment assembly problem. These algorithms are specifically adapted for the problem being the difference between them the optimization orientation (fitness function). One of them maximizes the Parsons's fitness function (which only considers the overlapping among the fragments) and the other estimates the variation in the number of contigs during a local search movement, in order to minimize the number of contigs. The results show that doesn't exist a direct relation between these functions (even in several cases opposite values are generated) although for the tested instances, both variants allow to find similar and very good results but the second option reduces significatively the consumed-time.

Keywords: DNA Fragment Assembly Problem, Variable Neighborhood Search, 2-opt heuristic

1 INTRODUCTION

DNA fragment assembly is a technique that attempts to reconstruct the original DNA sequence from a large number of fragments. For that, the assembling DNA fragments is divided into three different phases: overlap phase (finding the overlapping fragments), layout phase (finding the order of fragments based on computed similarity scores), and consensus phase (deriving the DNA sequence from the layout). Most of sequence assembly algorithms are based on some variation of the greedy algorithm: Phrap [8], CAP3 [10], Celera assembler [17], TIGR Assembler

[23], STROLL [5]. In the greedy approach, the fragments are assembled by repeatedly merging the pair of fragments with highest overlap according a specific and complex criterion. These methods obtain good results for small-medium sequences but it has some problems in many large genome sequencing projects. Metaheuristic techniques on the contrary are being used with very accurate results even for large problems. Some of them are evolutionary algorithms (EAs) [11, 12, 13, 19], ant colony systems (ACS) [16], simulated annealing (SA) [3, 4, 6, 14, 15], variable neighborhood search (VNS) [9], among others.

Now, we summarize the main features of the most important metaheuristic approaches applied to this problem. Kim and Mohan [11] proposed a parallel version of the Hierarchical Adaptive Genetic Algorithm (HAGA), which adapts its parameters according to the input data, to addresses many specific issues of the DNA fragment assembly. HAGA freezes high confidence subsequences for future iterations of the evolutionary algorithm. Thus the algorithm steps levels, successively reducing the search space size, and working with the blocks identified earlier. Li and Khuri in [12] designed four different techniques to tackle this problem, obtaining the best results with structured pattern matching algorithm (PMA), which is based on the biologist technique called hybridization fingerprinting that deduces the overlap information among DNA clones from biological probes. DNA clones are exact copies of a particular part of a genome and are much longer than fragments. By other side, Alba *et al.* developed different versions to distribute and parallelize metaheuristics and had also created a new heuristic for solving this problem in a very accurate and efficient way. The reader could be obtain more information about those developments in [1, 2, 13, 18].

In [16], Meksangsouy and N. Chaiyaratana presented the use of an ant colony system algorithm in a DNA fragment assembly. They proposed an asymmetric ordering representation where a path co-operatively generated by all ants in the colony represents the search solution. The optimality of the fragment layout obtained is then determined from the sum of overlap scores calculated for each pair of consecutive fragments in the layout.

In the Simulated Annealing approach to the sequence assembly problem, the energy function is based on the overlaps of the fragments. This algorithm tries to minimize this energy function using stochastic reshuffling of the fragments. More details about SA solving FAP can be found in [3, 4, 6].

However, the objective function (fitness function) used in the previous algorithms is very consuming-time. This function evaluates the adaptation degree of found solution to the problem instance maximizing the results. Our objective is to propose a metaheuristic that increases the efficiency without losing quality. For that, we follow the idea presented in [2], we change the optimization orientation estimating the variation of contigs to decrement significatively the consumed-time. The optimal number of contigs for all the instances of this problem is *one*, consequently our new objective is to minimize the number of contigs. In order to do that, we design two versions of the Variable Neighborhood Search, one of them maximizes the overlapping among adjacent fragments in the layout and another minimizes the number of contigs. We show a behavior analysis of both VNS algorithms taking into account the result quality and the consumed CPU time. Finally, we compare our approaches with well-known assemblers in the literature.

The rest of this article is organized as follows. The next section introduces the DNA fragment assembly problem. Section 3 explains how the VNS solves the FAP. Section 4 shows the experiments performed and discusses the results of those experiments. Finally, the last section concludes and provides hints on further research.

2 THE DNA FRAGMENT ASSEMBLY PROBLEM

For studying the functional and structural information of an unknown DNA sequence, biologists compare this sequence with well known ones. If they are similar, they would have the same function. The process of sequencing is called *shotgun sequencing* and was introduced in [21]:

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

The last point, the assembling DNA fragments, is divided into three different phases: overlap phase (finding the overlapping fragments), layout phase (finding the order of fragments based on computed similarity scores), and consensus phase (deriving the DNA sequence from the layout). At the assembly stage, the only information available is the sequences of bases, and thus the ordering of the fragments must rely primarily on the similarity of fragments and how they overlap. An important aspect of the general sequencing problem is to determine the relationship and orientation of the fragments. Another important aspect is the incomplete coverage which happens when the algorithm is not able to assemble a given set of fragments into a single contig. A contig is a layout consisting of contiguous overlapping fragments.

Once the fragments have been ordered, the final consensus is generated from the layout. This process includes a detailed alignment step that must considerer the insertion and deletion errors potentially present in the data. 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 [22]:

$$Coverage = \frac{\sum_{i=0}^{n} length\ \ of\ \ the\ \ fragment\ \ i}{target\ \ sequence\ \ length} \quad (1)$$

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

Particularly, the assembly of DNA fragments into a consensus sequence corresponding to the parent sequence constitutes the "fragment assembly problem" [22]. It is a permutation and NP-hard problem [20]. Therefore, it is not possible to find an exact algorithm that solves this problem and runs in polynomial time (unless P = NP). In [4, 6], the reader could be find a more detailed information from this process.

3 VARIABLE NEIGHBORHOOD SEARCH FOR SOLVING FAP

The Variable Neighborhood Search (VNS) is a recent metaheuristic presented by Hansen *et al.* in [9]. VNS solves optimization problems by doing systematic changes of neighborhood within a local search. VNS is a descendent method which does not follow a trajectory since explores different predefined neighborhoods of the current solution using a local search (LS). The current solution is changed by a new one if and only if an improvement have been made. The basic idea is to change the neighborhood structure when the local search is trapped on

a local optimum. A neighborhood structure in a solution space S is a mapping $N : S \rightarrow 2^2$, $x \rightarrow N(x)$, where $N(x)$ constitutes the neighborhood of x. The steps of a basic VNS are shown in the Algorithm 1.

Algorithm 1 Basic VNS Algorithm

```
Initialization:
Select the set of neighborhood structures N_k, k = 1, ..., k_max;
Find an initial solution x;
Choose an end condition;
while end condition is not met do
  k = 1;
  while k <= k_max do
    Shaking:
    Generate randomly x' ∈ N_k(x);
    Local Search:
    Obtain the local optimum x'' by applying some local search to x';
    Move or not:
    if x'' is better than x then
      x = x'';
      k = 1;
    else
      k = k + 1;
    end if
  end while
end while
```

Variable Neighborhood Descendent (VND), General VNS (GVNS), and Reduced VNS (RVNS) are extensions of the Basic Variable Neighborhood Search Algorithm. The reader can be find detailed explanations of these extensions in [9, 1].

3.1 Our Proposal

Particularly, we implement two search algorithms based on Basic VNS version according to the permutation representation and the fragment assembly problem. The Algorithm 4 presents this version of VNS.

- *Solution Representation.* We use the permutation representation with integer number encoding. This permutation represents a sequence of fragment numbers, where successive fragments overlap. Consequently each fragment is represented by an unique integer ID. 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 all fragments must be presented in the ordering, and no duplicate fragments are allowed in the ordering.

- *Initial Solution* The solution used to init this VNS algorithm, is obtained by a method which modifies iteratively the solution applying movements in a structured way as it can see in the Algorithm 2.

- *Neighborhood Structure.* The neighbors in each neighborhood are generated by swapping between two positions from an initial solution. The first position is randomly chosen when

Algorithm 2 Algorithm to create the VNS initial solution

```
for i = 0 to number of fragments - 1 do
  aux_i = random(0, number of fragments * 2);
  x_i = i; {x is the initial solution}
end for
for i = 0 to number of fragments - 2 do
  for j = 0 to number of fragments - 1 do
    if aux_i > aux_j then
      temp = aux_i;
      aux_i = aux_j;
      aux_j = temp;
      temp = x_i;
      x_i = x_j;
      x_j = temp;
    end if
  end for
end for
return x;
```

the neighborhood structure is determined, and it stays fix during the whole execution since it represents to a neighborhood structure. The second position is randomly chosen in the shaking part and it changes in each iteration.

- *Number of neighborhoods*, k_{max}. The number of neighborhoods varies according with the instance. This value is proportional to number of fragments from each instance. In this way, the VNS process creates more search subspaces (neighborhoods) when the number of fragments grows. This feature allows to adequate the VNS computational effort and its efficacy to the problem complexity, intensifying the search when this complexity is increased. This idea arises after many trials where we have tested different ways to establish the number of neighborhoods. We could use the total chromosome size as number of neighborhood but, the execution time in the larger instances grows in a disproportionate way and it is not reflected in the quality of results. We have checked different percentages of instance size (10, 20, 25, 50 and 100%) as number of neighborhoods, and finally we have decided that a 10% is a good compromise value between quality and time.

- *Shaking*. For generating a solution from N_i two positions, of an initial solution copy, are swapped. The first one represents the neighborhood structure and the second one is randomly selected. We use a non consumed-time and simple process (swap) to generate a solution since we only need a solution belonging to a particular neighborhood, which can be modified and improved by a LS.

- *Local Search*. We have worked with a modified version of 2-opt heuristic (see Algorithm 3). We have modified the 2-opt algorithm for reducing the total number of iterations; that results necessary due to the complexity of this method and its application in VNS.

- *Fitness Function*, $F(l)$. Parsons, Forrest, and Burks proposed two different fitness functions which include errors in the sequence information, repeated sequences among other factors [19]. This fitness function sums the overlap score for adjacent fragments in a given solution. When this fitness function is used, the objective is to maximize such score. It

Algorithm 3 Modified 2-opt(x)

```
i = 0;
j = i + 2;
while i < number of fragments do
  x'=x; {x is the initial solution}
  x'_i=x_j;
  x'_j=x_i;
  if x' is better than x then
    x=x';
  end if
  j = j + 1;
  if j > (number of fragments - 1) then
    i = i + 1;
    j = i + 2;
  end if
end while
return initSolution;
```

means that the best individual will have the highest score.

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

where $w_{i,j}$ is the pairwise overlap strength of fragments i and j. The overlap score in F is computed using the semiglobal alignment algorithm.

$F(l)$ is used to evaluate and compare solutions. In this case a solution a is better than another b if $F(l)_a > F(l)_b$. In our first version of VNS, we consider FAP as a maximization problem with $F(l)$ as objective function. This version is called *FVNS*.

- *Contig Estimation*, Δ_c. Alba and Luque proposed to evaluate the candidate solution considering if the number of contigs is incremented or decremented when the local search is applied [2]. In this way only a part of permutation, which is modified by LS, is considered in the evaluation. This evaluation, Δ_c, is very simple since it sums 1 if a contig is broken or rests 1 if two contigs are merged. Therefore, we only need to take care of the concrete positions modified by the variant operator. Considering this, a solution is better than other if its number of contigs is lesser. In other words, FAP is considered as a minimization problem, where minimizing the number of contigs is the objective function. Therefore, we propose the second version of VNS, called *CVNS*, which uses this estimation as fitness function.

- *End condition.* We have established two points in the algorithm where two different stop criterions are necessary. The first point is related with the quantity of times the whole algorithm iterates (first *while* sentence in Algorithm 1); and the second one is related with the stop criterion used in the second *while* sentence in Algorithm 1.

  Since the problem features, we have set the number of complete iterations in only one to reduce the consumed-time. For the second case, we need to limit the number of times that the neighborhoods are explored given that the combination between the neighborhood number (proportional to permutation size) and some instance sizes can produce an

extremely long loop. For that, we have established a maximal iteration number which is calculated as follows:

$$iter_{max} = k_{max} * ((k_{max}/t) + 1) \quad (3)$$

where k_{max} is the neighborhood number and t, is an integer $\in$ [1..10] range.

Algorithm 4 VNS Algorithm for FAP

Initialization:
Set the initial solution x;
Set max as a percentage of the number of fragments;
Select the set of neighborhood structures $N_k, k = 1, ..., k_{max}$;
$k = 0$;
$iter = 0$;
while $((k < k_{max})$ and $(iter < iter_{max}))$ **do**
  Shaking:
  Generate randomly $x' \in N_k(x)$;
  Local Search:
  Obtain the local optimum x'' by applying 2−opt heuristic to x';
  Move or not:
  if x'' is better than x **then**
    $x = x''$;
    $k = 1$;
  else
    $k = k + 1$;
  end if
  $iter = iter + 1$;
end while

4 EXPERIMENTAL RESULTS

In this section we analyze the behavior of our proposed methods. We have chosen three sequences from the NCBI web site[1]: a human MHC class II region DNA with fibronectin type II repeats HUMMHCFIB, with accession number X60189, which is 3,835 bases long; a human apolopoprotein HUMAPOBF, with accession number M15421, which is 10,089 bases long; and the complete genome of bacteriophage lambda, with accession number J02459, which is 20k bases long. We used GenFrag [7] to generate the different data sets shown in Table 1. GenFrag is a UNIX/C application created to accept a DNA sequence as input and to generate a set of overlapping fragments as output, in order to test any assembly application.

Now, we will summarize the results obtained for executing proposed VNS algorithms, which use a 10% of chromosome size as k_{max}. In Table 1 k_{max} and $iter_{max}$ are specified per instance. For each algorithm, we have performed 30 independent runs per instance. We have used a Pentium IV at 2.4 GHz and 1 GB RAM. The operating system used is SuSE Linux with 2.4.19-4GB kernel version. Our aim is to offer meaningful results from a statistical point of view. In Section 4.1, we make an intra comparison on the two algorithms proposed here. After that we compare their results with the obtained ones by another assemblers.

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

Table 1: Information of datasets (Accession numbers are used as instance names) and VNS parameters

Parameters	Instance							
	X60189				M15421			J02459
Coverage	4	5	6	7	5	6	7	7
Fragment Length	395	386	343	387	398	350	383	405
Number of Fragment	39	48	66	68	127	173	177	352
k_{max}	5	6	8	8	14	18	19	36
$iter_{max}$	7	9	13	14	32	52	54	167

4.1 Comparing VNS algorithms between them

First of all, we have considered necessary to analyze the two proposed algorithmic approaches: *FVNS* which optimizes the overlaping among adjacent fragments in the layout and *CVNS* that optimizes the number of contigs. For that, we compare the results from different points of view and perform statistical tests to corroborate our inferences. In the Table 2, we present the best fitness and number of contigs obtained by FVNS and CVNS algorithms in the 30 runs per instance. In Table 3 we show a data resume of FVNS and CVNS for all executions in each instance. They are: the fitness value average, the percentage of the optimal number of contigs, the average of total consumed-time (in seconds) when the optimal number of contigs was found. From each data set, the results of Mann-Whitney U Test are presented, where a '+' symbol indicates the difference between both algorithms is significative; while a '–' symbol indicates that FVNS and CVNS have a similar behavior. Besides we show the percentage of reduced time of CVNS with respect to FVNS.

Table 2: Best fitness and their respective number of contigs, and best number of contigs and their respective fitness obtained by FVNS and CVNS algorithms in all the instances

Instances	*Best (maximal) fitness*		*Best (minimal) number of contigs*	
	FVNS	CVNS	FVNS	CVNS
x60189_4	9920 / 1	2988 / 1	1 / 8629	1 / 1138
x60189_5	12714 / 1	3034 / 1	1 / 12065	1 / 3034
x60189_6	15757 / 2	3343 / 1	1 / 15306	1 / 970
x60189_7	18749 / 1	4892 / 1	1 / 17832	1 / 2305
m15421_5	33720 / 1	3418 / 2	1 / 33495	1 / 2090
m15421_6	40018 / 2	3946 / 2	2 / 38573	2 / 2862
m15421_7	46241 / 2	4724 / 2	2 / 45375	2 / 4724
j02459_7	96816 / 2	5716 / 2	1 / 96517	1 / 5337

Looking at Table 2, we can infer that a good fitness value is not necessarily related with the optimum number of contigs. The reasons for that are the following:

- For some instances, like *x60189* set, *m15421_5* and *j02459_7*, both algorithms find the optimal number of contigs independently if their fitness quality is high or not.
- When FVNS algorithm finds its higher fitness value in *x60189_6* instance, its corresponding number of contigs is greater than one.

- In other instances, as *m15421_5*, CVNS optimizes the number of contigs and their respective fitness values are lesser than the best one. This is clearly visible in Figure 1, where the bars show the fitness value obtained by CVNS in each run and the number on the bar is its corresponding final number of contigs. Particularly, in this figure we can see that some of higher fitness values, are associated with a number of contigs greater than one. By other side, the optimal contigs are related with some of lower fitness values.

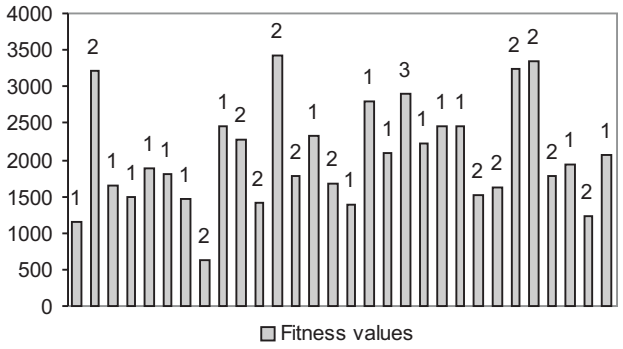

Figure 1: Fitness and number of contigs obtained by CVNS for *m15421_5* instance

Furthermore, from both tables (2 and 3), we can see FVNS outperforms significatively CVNS in all cases when the fitness quality is evaluated. This difference is corroborated using a Mann-Whitney U Test with $\alpha = 0.05$, which is applied to compare the fitness quality obtained by both algorithms in each instance; where the $p-$value is lesser than α. Even although, when we analysis the optimal number of contigs obtained by both algorithms (Table 3), a very similar behavior is observed. For example, for *x60189_4* instance both approaches obtain 1 contig (the optimum) in each execution, but for *x60189_6* instance FVNS reaches the optimum only 93% of times while CVNS finds it every time while for *j02459_7* instance FVNS obtains 1 contig in more executions than CVNS (46.67% against 33% respectively). Those are not significant differences and it is demonstrated by statistical tests where the $p-$values are greater than α. On the other hand, CVNS consumes much lesser time than FVNS to optimize the final number of contigs, in average the percentage of reduced time by CVNS reaches a 98%, being the mean time to obtain the optimal contig 56 seconds. This difference is corroborated using Mann-Whitney U tests with $p-$values lesser than α. The reason for that particularity is directly related with the number of times that fitness function, $F(l)$, is computed by each algorithmic approach. That is, CVNS only calculates this fitness function twice, at the beginning and the end of the algorithm; while the number of evaluations realized by FVNS is equal to iteration number of LS by iteration number of VNS. Thus, in the best cases FVNS surpasses a thousands evaluations of $F(l)$ and this evaluation number depends on the instance size.

Note that we have used a non parametric statistical test, Mann-Whitney U Test, since in many cases we can not prove neither normality or variance homogeneity for using a t-test. The significance level (α) is equal to 0.05 for all tests realized and the probability value ($p-$value) of statistical test is the probability of wrongly rejecting the null hypothesis if it is in fact true. The $p-$value is compared with the significance level and, if it is smaller, the result indicates that compared methods are significatively different.

Table 3: Average of found fitness values, percentage of the optimal number of contigs obtained, average of total time-consumed and time reduction of CVNS with respect to FVNS. The best values are marked in bold.

Instances	*Average Fitness*			*Opt. Contig %*			*Average Total Time*			*% of Reduced time*
	FVNS	CVNS	Mann-Whitney U Test	FVNS	CVNS	Mann-Whitney U Test	FVNS	CVNS	Mann-Whitney U Test	
x60189_4	**9290,47**	1549,80	+	**100,00%**	**100,00%**	–	0,047	**0,002**	+	94.98 %
x60189_5	**11994,07**	1970,73	+	**100,00%**	90,00%	–	0,167	**0,005**	+	96.69 %
x60189_6	**15211,20**	2074,90	+	93,33%	**100,00%**	–	0,846	**0,020**	+	97.65 %
x60189_7	**17966,80**	2770,70	+	96,67%	**100,00%**	–	1,018	**0,022**	+	97.86 %
m15421_5	**32904,00**	2055,20	+	**66,67%**	53,33%	–	46,752	**0,472**	+	98.99 %
m15421_6	**39035,37**	2362,50	+	0,00%	0,00%	–	-	-		
m15421_7	**45596,20**	2832,20	+	0,00%	0,00%	–	-	-		
j02459_7	**95034,50**	3144,80	+	**46,67%**	33,33%	–	11949,845	**55,759**	+	99.53 %
Average	**33379,08**	2345,10		**62,92%**	59,59%		1999,779	**9,380**		97.2 %

4.2 Comparison against other assemblers

In this section we compare the performance of approaches presented here against other assembler algorithms proposed in the literature: Problem Aware Local Search (PALS) [2], a genetic algorithm (GA) [19], a pattern matching algorithms (PMA) [12], and commercially available packages: CAP3 [10] and Phrap [8].

Firstly, we compare CVNS against PALS and we can conclude that these two algorithms present the same behavior for almost instances except for *m15421_6* and *m15421_7* instances.

Now, we compare our approaches with the rest of the above mentioned assemblers, in terms of the final number of contigs assembled (Table 4). In this sense both VNS algorithms have a better or equal behavior than the rest ones. PMA y Phrap obtain the same final number of contigs than our approaches. We can not make an execution time comparison since, in general, the authors does not provide this information.

Table 4: Best final number of contigs for FVNS and CVNS algorithms and for other specialized systems. - symbol indicates that this information is not provided

	FVNS	*CVNS*	*PALS* [2]	*GA* [19]	*PMA* [12]	*CAP3* [10]	*Phrap* [8]
x60189_4	1	1	1	1	1	1	1
x60189_5	1	1	1	1	1	1	1
x60189_6	1	1	1	-	1	1	1
x60189_7	1	1	1	1	1	1	1
m15421_5	1	1	1	6	1	2	1
m15421_6	2	2	-	-	2	2	2
m15421_7	2	2	1	1	2	2	2
j02459_7	1	1	1	13	1	1	1

5 CONCLUSIONS AND FUTURE WORK

In order to design an assembler which finds quickly good solutions for the fragment assembly problem, we propose two search algorithms based on the canonical VNS version. These algorithms are specifically adapted to the problem but they have a different optimization orientation. One of them, *FVNS*, maximizes the Parsons's fitness function which sums the overlap

score for adjacent fragments in a given solution. When this fitness function is used, the objective is to maximize such score. It means that the best individual will have the highest score. Another variant, *CVNS*, evaluates the candidate solution considering if the number of contigs is incremented or decremented when a local search movement is applied. In this way the objective is to minimize the final number of contigs. The result quality of both algorithms is very high since they found the optimal number of contigs in almost instances, whose number of fragments varies in a [39..352] range. Although CVNS reaches these results consuming a 97% less time than FVNS.

Our future work will be to incorporate this new minimization objective in our genetic algorithm versions for the fragment assembly problem.

REFERENCES

[1] E. Alba. *Parallel Metaheuristics A New Class of Algorithms*. WILEY Series on Parallel and Distributed Computing. Wiley, 2005.

[2] E. Alba and G. Luque. A New Local Search Algorithm for the DNA Fragment Assembly Problem. In *Evolutionary Computation in Combinatorial Optimization, EvoCOP'07*, volume 4446 of *Lecture Notes in Computer Science*, pages 1–12, Valencia, Spain, 2007. Springer.

[3] C. Burks, M.L. Engle, S. Forrest, R.J. Parsons, C.A. Soderlund, and P.E. Stolorz. Stochastic Optimization Tools for Genomic Sequence Assembly. In M.D. Adams, C. Fields, and J.C. Venter, editors, *Automated DNA Sequencing and Analysis*, pages 249–259. Academic Press, 1994.

[4] C. Burks, R.J. Parsons, and M.L. Engle. Integration of competing ancillary assertions in genome assembly. In R. Altman, D. Brutlag, P. Karp, R. Lathrop, and D. Searls, editors, *Proceedings Second International Conference on Intelligent Systems for Molecular Biology*, pages 62–69, Menlo Park, CA, 1994. AAA1 Press.

[5] T. Chen and S.S. Skiena. A case study in genome-level fragment assembly. *The Eighth Symposium on Combinatorial Pattern Matching*, pages 206–223, 1997.

[6] G. Churchill, C. Burks, M. Eggert, M.L. Engle, and M.S. Waterman. Assembling DNA Sequence Fragments by Shuffling and Simulated Annealing. Technical Report LA-UR-93-2287, Los Alamos National Laboratory, Los Alamos, NM, 1993.

[7] M.L. Engle and C. Burks. Artificially generated data sets for testing DNA fragment assembly algorithms. *Genomics*, 16, 1996.

[8] P. Green. Phrap. http://www.mbt.washington.edu/phrap.docs/phrap.html, 1996.

[9] P. Hansen, N. Mladenovic, and J.A. Moreno Prez. Variable neighbourhood search. *Revista Iberoamericana de Inteligencia Artificial*, (19):77–92, 2003. ISSN: 1137-3601.

[10] W. Huang and A. Madan. CAP3: A DNA Sequence Assembly Program. *Genome Research*, 9(9):868–877, 1999.

[11] K. Kim and C.K. Mohan. Parallel hierarchical adaptive genetic algorithm for fragment assembly. In IEEE, editor, *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, volume 1, pages 600–607, 2003.

[12] L. Li and 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*, pages 329–335, Las Vegas, 2004.

[13] G. Luque, E. Alba Torres, and S. Khuri. *Parallel Algorithms for Bioinformatics*, chapter Chapter 16: Assembling DNA Fragments with a Distributed Genetic Algorithm. Wiley, New York, 2005.

[14] A.P. Lyubartsev, A.A. Martsinovski, and P.N. Vorontosov-Veuaminov. New Approach to Monte Carlo Calculation of the Free Energy: Method of Expanded Ensembles. *Journal of Chemical Physics*, 96:1776–1783, 1992.

[15] E. Marinari and G. Parisi. Simulated Tempering: A new Monte Carlo Scheme. *Europhys. Lett.*, 19:451–458, 1992.

[16] P. Meksangsouy and N. Chaiyaratana. DNA fragment assembly using an ant colony system algorithm. In *The 2003 Congress on Evolutionary Computation, 2003. CEC '03*, volume 3, pages 1756– 1763. IEEE. ISBN: 0-7803-7804-0, 2003.

[17] E. W. Myers. A whole-genome assembly of drosophila. *Science*, 287:219–2204, 2000.

[18] A. J. Nebro, G. Luque, F. Luna, and E. Alba. DNA Fragment Assembly Using a Grid Based Genetic Algorithm. *Computers and Operations Research (to appear)*, 2007.

[19] R. Parsons, S. Forrest, and C. Burks. Genetic Algorithms, Operators, and DNA Fragment Assembly, 1993.

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

[21] F. Sanger, A.R. Coulson, G.F. Hong, D.F. Hill, and G.B. Petersen. Nucleotide Sequence of Bacteriophage Lambda DNA. *Journal of Molecular Biology*, 162(4):729–773, 1982.

[22] J. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology*. International Thomson Publishing, 20 park plaza, Boston, MA02116, 1999.

[23] 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 and Technology*, pages 9–19, 1995.