Information Processing Letters 108 (2008) 94–100

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Seeding strategies and recombination operators for solving the DNA fragment assembly problem

Gabriela Minetti [a], Enrique Alba [b], Gabriel Luque [b,*]

[a] Laboratorio de Investigación en Sistemas Inteligentes, Univ. Nacional de La Pampa, Argentina
[b] Dpto. de Lenguajes y Ciencias de la Computación, Univ. Málaga, Spain

ARTICLE INFO

Article history:
Received 1 February 2008
Received in revised form 4 April 2008
Available online 11 April 2008
Communicated by L. Boasson

Keywords:
DNA
Fragment assembly problem
Genetic algorithm
Permutation operators
2-opt heuristic
Combinatorial problems

ABSTRACT

The fragment assembly problem consists in building 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 rest of the phases depend on the accuracy of the results of this stage. Therefore, accurate and efficient methods for handling this problem are needed. Genetic Algorithms (GAs) have been proposed to solve this problem in the past but a detailed analysis of their components is needed if we aim to create a GA capable of working in industrial applications. In this paper, we take a first step in this direction, and focus on two components of the GA: the initialization of the population and the recombination operator. We propose several alternatives for each one and analyze the behavior of the different variants. Results indicate that using a heuristically generated initial population and the Edge Recombination (ER) operator is the best approach for constructing accurate and efficient GAs to solve this problem.

1. Introduction

DNA fragment assembly is a technique that attempts to reconstruct the original DNA sequence from a large number of fragments. Most of sequence assembly algorithms are based on some variation of a greedy algorithm: Phrap [1], CAP3 [2], Celera assembler [3], TIGR Assembler [4], STROLL [5]. In these greedy approaches, the fragments are assembled by repeatedly merging the pair of fragments with the highest overlap (similarity score) according to a specific and complex criterion. These methods obtain good results for small to medium sequences but they have some problems in dealing with large genome sequencing projects. Metaheuristic techniques are being used with very accurate results even for large problems. The most popular techniques are evolutionary algorithms [6–9], ant colony systems [10], and simulated annealing [11–15].

Particularly, genetic algorithms show promising results for solving the DNA Fragment Assembly Problem (FAP). However, the resolution of real and large instances still represents a challenge to this technique. Therefore, a detailed analysis of its components is necessary to improve its performance. In [9] Parsons, Forrest and Burks studied and compared the sort-order and permutation representations and their associated operators. In this paper, we study a GA that uses a permutation representation to solve this problem with different recombination operators. These are Edge Recombination (ER) and Order crossover (OX), which were used by Parsons et al., but we also study other permutation operators which have not been applied to this problem, such as Cycle crossover (CX) and Partial Mapped crossover (PMX). In addition, we analyze different seeding strategies to generate the initial population, an important issue in building successful GAs. To do so, we incorporate solutions generated by a 2-opt heuristic and a greedy approach to the initial population in order to improve the final accuracy and efficiency of the algorithm. All those options have been developed, studied, and compared in this work.

* Corresponding author.
E-mail addresses: minettig@ing.unlpam.edu.ar (G. Minetti), eat@lcc.uma.es (E. Alba), gabriel@lcc.uma.es (G. Luque).

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2008.04.005

The rest of this article is organized as follows. The next section introduces the Fragment Assembly Problem. Section 3 explains how our GAs solve the FAP. Section 4 shows the experiments performed and discusses them. Finally, the last section concludes and provides suggestions for further research.

2. The DNA fragment assembly problem

We consider the *shotgun sequencing* method which consists in [16]:

(i) Several copies of the DNA are produced and each copy is broken into millions of random fragments.
(ii) Those fragments are read by a DNA sequencing machine.
(iii) An assembler pieces together the many overlapping reads and reconstructs the original sequence.

The assembling of DNA fragments is divided into three different phases: the overlap phase (finding the overlapping among fragments), the layout phase (finding the order of fragments based on computed similarity scores), and the 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 on how they overlap. An important challenge of the general sequencing problem is to determine the relationship and orientation of the fragments. Another important issue is the incomplete coverage which results 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 (layout), the final consensus is generated. This process includes a detailed alignment step that must consider 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. The 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 [17]:

$$\text{Coverage} = \frac{\sum_{i=0}^{n} \text{length of the fragment } i}{\text{target sequence length}}, \tag{1}$$

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

Specifically, the assembly of DNA fragments into a consensus sequence corresponding to the parent sequence constitutes the "fragment assembly problem" [17]. It is a permutation NP-hard problem [18]. In [12,13], the reader can find more detailed information about this process.

```
t = 0; {t is the generation number}
initialize P(t); {P(t) is the population at generation t}
evaluate individuals in P(t);
while not condition do
  t = t + 1;
  select C(t) from P(t − 1)
  apply variation operators (recombine and/or mutate)
    to individuals in C(t) building C′(t);
  evaluate individuals in C′(t);
  replace some individuals in P(t − 1) with C′(t)
    to build P(t);
end while
```

Algorithm 1. Genetic algorithm.

3. Solving the DNA fragment assembly problem with genetic algorithms

Genetic Algorithms (GAs) [19], a special class of Evolutionary Algorithms (EAs), are computer-based solving systems, which use evolutionary computational models as a key element in their design. They have a conceptual base of simulating the evolution of individual structures via the Darwinian natural selection process [20]. As is shown in Algorithm 1, a GA maintains a population of multiple tentative solutions (individuals) which evolve throughout generations by reproduction of the fittest ones. Selection, recombination, and mutation are the main operators used for modifying individual features. So, it is expected that evolved generations provide better and better individuals (tentative solutions in the problem space).

In this work, we propose different GAs with the following features:

- *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 a unique integer ID. The permutation representation requires special operators to make sure that we always obtain legal (feasible) solutions. In order to maintain a feasible solution, two conditions must be satisfied: all fragments must be present in the ordering, and no duplicate fragments are allowed in the ordering.
- *Fitness function*. Parsons, Forrest, and Burks [9] proposed two different fitness functions which can deal with errors in the sequence information and repeated sequences, among other factors. We use the first one proposed because of its low complexity ($O(n)$ versus $O(n^2)$ of the second one) and because preliminary experiments on our problem instances did not show any significant difference in the solution quality between both functions. 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 this score.

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

where $w(i, j)$ is the pairwise overlap strength of fragments i and j. The overlap score (w) in F is computed using the semiglobal alignment algorithm.

- *Population initialization*. This is an important step usually ignored in the literature. An initial population with a high diversity is important to explore the search space and to avoid local optimal solutions. However, having "good" solutions (seeds) in the first steps of the method can help exploit promising regions of the search space. Therefore, the classical intensification–diversification problem is also found in this phase and an accurate balance can improve the quality of the final results. In this study we use three different seeding strategies to generate a single individual in the initial population:
  (i) Random method. That is the classical way to generate populations, where the fragments are added to the solution randomly.
  (ii) 2-opt heuristic. This is a simple local search method. It randomly selects two non-adjacent fragments and inverts the subpermutation among them. It iteratively repeats this process while the termination criterion is not met. Specifically, in our experimentation, we apply the 2-opt heuristic to improve a randomly generated solution.
  (iii) Greedy approach. We propose a constructive heuristic to generate an initial solution for FAP. The idea behind this greedy approach is to generate solutions by adding appropriately selected solution components (fragments) to an initially empty partial solution. More specifically we follow these steps:
    (a) It starts generating a partial solution with a single randomly selected fragment.
    (b) The next fragment, which will be inserted, must satisfy the following conditions: (i) it must not have been inserted yet, (ii) it must form a contig with the last fragment inserted and, (iii) it will have the highest overlap score.
    (c) If we can not insert any fragment according to the previously mentioned conditions, we randomly select one from the rest of the fragments.
    (d) Steps (b) or (c), respectively, are repeated until all fragments are inserted into the chromosome.

  Based on these ideas to create an individual, we propose five different seeding strategies to generate the initial population:
  (i) The whole population is randomly generated. We called this variant GA.
  (ii) Half of the population is randomly created and the remaining solutions are generated using the 2-opt heuristic (version $GA2o_{50}$).
  (iii) The whole population is created using the 2-opt heuristic (version $GA2o_{100}$).
  (iv) Fifty percent of the population is randomly created and the rest is generated using the greedy approach (version GAG_{50}).
  (v) The total population is generated by means of a greedy approach (version GAG_{100}).
- *Recombination operators*. We work with specially designed recombination operators for permutation representation to avoid infeasible solutions. The operators considered in this study are: Cycle crossover (CX) [21], Edge Recombination (ER) [22], Order crossover (OX) [23], and Partial Mapped crossover (PMX) [24].
  - The cycle crossover takes some fragment IDs from one parent and selects the remaining fragment IDs from the other parent. The fragments from the first parent are selected to define a cycle according to the corresponding positions between parents.
  - The edge recombination transfers more than 95 percent of the edges from the parents to the single offspring. This is done with the help of an edge list created from both parents; this list provides for each fragment ID f, all other IDs connected to f. The construction of the offspring starts with a selection of an initial fragment from one of the parents. Then the fragment with the smallest number of edges in the edge list is selected, and so on.
  - The order-based crossover operator first copies the fragment IDs between two random positions of the first parent into the offspring's corresponding positions. It then copies the rest of the fragments from the second parent into the offspring in the relative order presented in the second parent. If the fragment ID is already present in the offspring, then it is skipped.
  - The partially mapped crossover is an extension of the two-point recombination of binary strings to permutation representations. It uses a special repairing method to resolve the constraint violations caused by the simple two-point recombination.
- *Mutation operator*. In this work, we use the swap mutation operator. This operator randomly selects two positions from a permutation and then swaps the two resulting fragments.
- *Selection operators*. We use two different selection methods. The binary tournament selection is used to choose the individuals to be recombined. This technique randomly selects two individuals and finally chooses the best one. However, for building the population for the next generation we select the best μ individuals from the μ parents and the λ offspring.
- *Stop criterion*. We use a fixed number of generations (1000) as stop condition. This value and the remaining ones (see Table 2) have been "optimized" through a previous hand-tuning process comparing it versus other different values.

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 3835 bases long; a human apolopoprotein HUMAPOBF, with accession number

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

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

Parameters	Instance						
	X60189				M15421		J02459
Coverage	4	5	6	7	5	7	7
Frag. length	395	386	343	387	398	383	405
Number of frag.	39	48	66	68	127	177	352

Table 2
Parametric values used for the different GAs

Parameter	Value
μ	512 (Population size)
λ	512 (Offspring size)
Recombination op. and probability	CX, ER, OX and PMX 0.7
Mutation op. and probability	Swap mutation 0.2
Parent selection	Binary tournament
Replacement	The best μ solutions from $(\mu + \lambda)$
Stop criterion	1000 generations

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 [25] 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. We have selected these instances because their fragment size is very close to the size of fragments that were used in the Human Genome Project (a real-world challenge), and because they have been used also in previous works [26].

Now, we will summarize the results obtained by the five proposed GAs (GA, $GA2o_{50}$, $GA2o_{100}$, GAG_{50}, GAG_{100}) with their variants (recombination operators) on all problem instances. Our aim is to offer meaningful results from a statistical point of view. Therefore, for each algorithm, we have performed 30 independent runs per instance using the parameters shown in Table 2. We have used a cluster composed of 16 machines with the following characteristics: Pentium 4 at 2.4 GHz and 1 GB RAM. The operating system used is SuSE Linux with 2.4.19-4 GB kernel version.

The main results are shown in Table 3 (solution quality) and Table 4 (execution time in seconds). From the solution quality point of view we can draw several conclusions. First, the utilization of seeding strategies, as the 2-opt heuristic and a greedy method, for the initialization of the population is beneficial for the search process. In particular, this can be observed in the largest instance (J02459(7)) where these seeding strategies allow an important improvement of the solution quality (18% when the 2-opt heuristic is applied and a 20% when the greedy approach is used). If we consider the seeding application percentage, $GA2o_{50}$ allows better results found in moderate complexity instances. We can observe that applying the seeding (greedy or 2-opt) to the 100% of the initial solutions is more desirable than any other case. In general we can observe that the 2-opt heuristic obtains the best quality results among all the seeding options.

Another conclusion is that the ER operator is the most appropriate operator for GA, $GA2o_{50}$ and $GA2o_{100}$ algorithms. However for GAG_{50} and GAG_{50} algorithms, the two operators which show a slightly better behavior are CX and OX.

With respect to the execution time (Table 4), it can be observed that the use of the seeding (2-opt and greedy heuristics) to generate the population does not increase the execution time. This result was expected since this step is only executed once and its execution time is negligible with respect to the execution time of the 1000 generations of a GA. Despite its negligible cost, the benefits of 2-opt in quality are very high, which is a remarkable finding. Another (expected) result is that the application of the ER recombination causes a large increase in the execution time. This is due to the fact that the complexity of this operator is $O(n^2)$ while the other operators are $O(n)$.

In order to further support the previous conclusion and to study whether there are significant differences in the quality (best fitness) and computational time, we have made an analysis of variance for non-parametric classification. Since the assumption on the normality and the homogeneity of the variance was impossible to verify, we use the Kruskal–Wallis test. We use $\alpha = 0.10$ for all instances (*alpha* is the confidence level). Since the average quality and time of these sets is different, a multiple comparison was done, concluding:

(i) The execution applying ER is statistically slower than the ones using the rest of the recombination operators.
(ii) From the solution quality point of view, statistical analysis shows that the results of the ER are similar to those of the other ones. On the contrary it indicates that ER works better when the seeding is obtained randomly or a 2-opt heuristic. However, the Cycle Crossover and the Order Crossover are the best option when the greedy approach is used.
(iii) The solution quality of $GA2o_{50}$ and $GA2o_{100}$ is similar. They too are different and better than GA, GAG_{50} and GAG_{100}. When the 2-opt seeding is used the quality of the solutions is improved without a significant increase of the overall execution time, independently of the recombination operator applied. This is quite an interesting result since it is a simple method to increase the numerical performance of the algorithm.
(iv) The execution time of algorithms which use 2-opt and greedy seedings are similar. But they are better than GA (version with a random seeding). The reason of this is that the GAs evolve a few generations to find their best solution (see Fig. 1) when the 2-opt heuristic and greedy approach are used. On the other hand, when these options are not applied at the beginning, the evolutionary process needs a higher number of generations to find its best results.

In order to analyze the phenotypic diversity, we present Fig. 2 where the diversity for each seeding strategy, considering the Edge Recombination as a crossover operator and M15421(5) as a representative instance, is shown. From this figure, we can see that GA keeps a larger amount of different solutions during the execution. Meanwhile the rest of the algorithms lose almost completely their diversity in the first hundred generations. The phenotypic di-

Table 3
Mean best fitness obtained for each proposed algorithm in all instances

Algorithm	Recom. op.	X60189(5)	X60189(6)	X60189(7)	M15421(5)	M15421(7)	J02459(7)
GA	CX	11693.09	14556.17	17596.10	**37366.71**	**52555.28**	67455.03
	ER	**13556.93**	**17379.13**	**20447.31**	35981.27	49307.00	**92296.20**
	OX	12238.34	15178.07	18076.34	35981.27	42957.07	80015.28
	PMX	11873.83	14600.28	17306.10	30738.00	41608.82	78935.00
$GA2o_{50}$	CX	13820.63	17978.77	20832.30	37414.75	52761.08	67455.03
	ER	**13988.20**	**18293.03**	**21221.00**	**37967.13**	**53041.89**	109274.69
	OX	13777.69	18044.68	20934.59	37491.07	52647.86	**109513.62**
	PMX	13798.79	18012.31	20882.07	37339.28	52530.07	109190.69
$GA2o_{100}$	CX	13821.53	18011.63	20865.03	37834.75	52880.00	109345.92
	ER	**13986.24**	**18286.44**	**21213.20**	**37931.58**	**53148.27**	109477.07
	OX	13827.34	17986.17	20858.41	37585.34	52917.38	**109816.82**
	PMX	13861.28	17972.55	20856.59	37324.69	52679.10	109139.17
GAG_{50}	CX	**13369.30**	17338.13	**20788.67**	37370.90	**52334.37**	110102.87
	ER	13271.23	17359.87	20722.80	37399.77	52272.37	109945.73
	OX	13275.50	**17414.90**	20737.23	**37506.83**	52277.93	**110223.87**
	PMX	13293.97	17379.20	20684.33	37360.07	52218.37	109975.10
GAG_{100}	CX	**13349.43**	17350.53	**20804.37**	37396.13	**52296.90**	110101.20
	ER	13304.53	17373.07	20707.60	37417.57	52252.03	110034.13
	OX	13326.70	**17473.75**	20668.77	**37528.27**	52285.27	**110303.47**
	PMX	13283.60	17457.13	20649.30	37345.57	52254.53	110064.57

Table 4
Mean time (in seconds) to find the best fitness obtained for each proposed algorithm in all instances

Algorithm	Recom. op.	X60189(5)	X60189(6)	X60189(7)	M15421(5)	M15421(7)	J02459(7)
GA_{50}	CX	1.86	3.25	3.33	**2.32**	**5.60**	**21.92**
	ER	2.45	8.87	7.04	64.57	122.26	337.12
	OX	**1.40**	2.53	2.89	7.60	12.84	27.43
	PMX	1.42	**2.24**	**2.47**	8.45	13.62	31.32
$GA2o_{50}$	CX	0.47	0.69	0.78	**2.45**	**5.87**	21.92
	ER	1.12	1.94	2.60	21.93	64.27	270.61
	OX	**0.33**	**0.54**	0.72	2.51	6.25	**20.63**
	PMX	0.35	0.63	**0.71**	2.85	7.49	24.25
$GA2o_{100}$	CX	0.37	0.59	0.72	3.00	6.15	**13.38**
	ER	0.77	1.56	2.77	**0.76**	64.27	450.12
	OX	**0.36**	**0.58**	0.69	2.84	**6.11**	20.41
	PMX	0.37	0.64	**0.68**	7.66	7.66	23.94
GAG_{50}	CX	0.22	0.44	0.82	0.73	2.77	**5.32**
	ER	0.49	0.95	2.57	2.97	27.27	88.77
	OX	0.22	**0.41**	0.69	0.62	**2.57**	6.08
	PMX	**0.21**	0.42	**0.66**	**0.62**	3.18	6.75
GAG_{100}	CX	0.25	0.45	**0.78**	0.85	**3.12**	**6.11**
	ER	0.52	1.12	3.09	2.74	29.82	85.40
	OX	0.25	0.47	0.85	0.76	3.20	6.84
	PMX	**0.25**	**0.44**	0.82	**0.74**	4.04	8.07

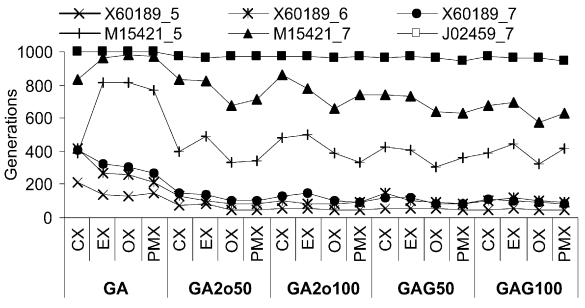

Fig. 1. Mean number of generations to find the best fitness obtained for each proposed algorithm in all instances.

versity has been calculated as a population fitness average each hundred generations.

Once we have studied our proposed algorithm and seen how the different alternative methods influence its performance, we turn to compare its results with other assemblers found in the literature: Parson's GA [9], a pattern matching algorithm (PMA) [7], a heuristic technique (PALS) [27] and commercially available packages: CAP3 [2] and Phrap [1]. We compare them in terms of the final number of contigs assembled. Table 5 gives a summary of the results.

From Table 5, we can observe that the use of a more intelligent seeding (heuristics ones) benefits the search process, obtaining better results (a lower number of contigs) than a random seeding. Also, those results are competitive

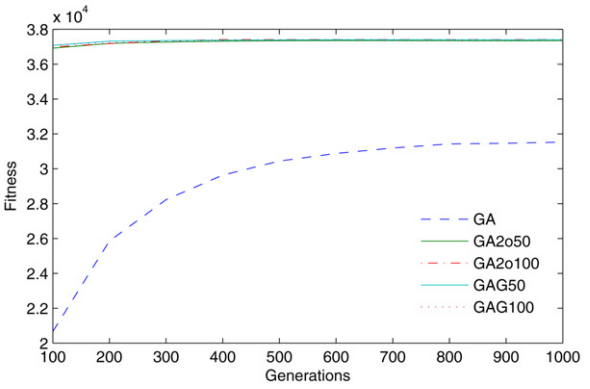

Fig. 2. Phenotypic diversity obtained for each proposed seeding strategy considering ER and M15421(5) instance.

Table 5
Best final number of Contig for the proposed algorithms and for other specialized systems

Algorithm	X60189(5-7)	M15421(5)	M15421(7)	J02459(7)
GA (random seeding)	1	2	2	2
GA (heuristic seeding)	1	1	2	1
PALS	1	1	1	1
Parson's GA	1	6	1	13
PMA	1	1	2	1
CAP3	1	2	2	1
Phrap	1	1	2	1

with respect to other approaches presented in the literature. In fact, the methods using a heuristically generated population achieve the optimal solution (a single contig) for 5 out of 6 FAP instances. In the instance where they do not find the optimal solution, their results are similar to very well-known commercial packages such as CAP3 or Phrap.

5. Conclusions

In this paper, we have proposed different approaches based on genetic algorithms to assemble DNA fragments to construct a single genome sequence. We studied the effects induced by the use of different seeding strategies to generate the initial population. Then, to improve the efficiency of the GA for the fragment assembly problem, we test several recombination operators for the permutation representation. The seeding strategies studied are: the random strategy, the 2-opt heuristic and a greedy method which was specially designed for the FAP. The operators studied are: cycle crossover, edge recombination, order crossover and partial mapped crossover.

From the study and analysis of all these components of the genetic algorithm we can observe that the 2-opt heuristic to generate the population is very beneficial since it allows us to improve the fitness quality, to obtain the number optimum of contigs (one) and it does not increase the execution time. From the recombination operator point of view, the edge recombination achieved the best results, although it resulted in a longer execution time for random and 2-opt seeding strategies. On the other hand, cycle and order crossover operators were the best options when the greedy strategy was used.

In the future we plan to study other good evolutionary algorithms for permutation: inver-over, augmented with the 2-opt heuristic for creating the initial population.

Acknowledgements

The last two authors are partially supported by the Spanish Ministry of Education and Science, and by European FEDER under contract TIN2005-08818-C04-01 (the OPLINK project, http://oplink.lcc.uma.es).

References

[1] P. Green, Phrap, http://www.phrap.org, 1996.
[2] W. Huang, A. Madan, CAP3: A DNA sequence assembly program, Genome Research 9 (9) (1999) 868–877.
[3] E.W. Myers, A whole-genome assembly of Drosophila, Science 287 (2000) 2196–2204.
[4] G.G. Sutton, O. White, M.D. Adams, A.R. Kerlavage, TIGR Assembler: A new tool for assembling large shotgun sequencing projects, Genome Science and Technology 1 (1) (1995) 9–19.
[5] T. Chen, S. Skiena, A case study in genome-level fragment assembly, in: The Eighth Symposium on Combinatorial Pattern Matching, 1997, pp. 206–223.
[6] K. Kim, C. Mohan, Parallel hierarchical adaptive genetic algorithm for fragment assembly, in: The 2003 Congress on Evolutionary Computation, 2003, in: CEC '03, vol. 1, IEEE Press, 2003, pp. 600–607.
[7] 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.
[8] G. Luque, E. Alba, S. Khuri, in: Assembling DNA Fragments with a Distributed Genetic Algorithm, Parallel Algorithms for Bioinformatics, Wiley, New York, 2005, pp. 285–302 (Ch. 16).
[9] R. Parsons, S. Forrest, C. Burks, Genetic algorithms, operators, and DNA fragment assembly, Machine Learning 21 (1) (1995) 11–33.
[10] P. Meksangsouy, N. Chaiyaratana, DNA fragment assembly using an ant colony system algorithm, in: The 2003 Congress on Evolutionary Computation, 2003, in: CEC '03, vol. 3, IEEE Press, 2003, pp. 1756–1763.
[11] C. Burks, M. Engle, S. Forrest, R. Parsons, C. Soderlund, P. Stolorz, Stochastic optimization tools for genomic sequence assembly, in: M. Adams, C. Fields, J. Venter (Eds.), Automated DNA Sequencing and Analysis, Academic Press, 1994, pp. 249–259.
[12] C. Burks, R. Parsons, M. Engle, Integration of competing ancillary assertions in genome assembly, in: R. Altman, D. Brutlag, P. Karp, R. Lathrop, D. Searls (Eds.), Proceedings Second International Conference on Intelligent Systems for Molecular Biology, AAA1 Press, Menlo Park, CA, 1994, pp. 62–69.
[13] G. Churchill, C. Burks, M. Eggert, M. Engle, M. Waterman, Assembling DNA sequence fragments by shuffling and simulated annealing, Tech. Rep. LA-UR-93-2287, Los Alamos National Laboratory, Los Alamos, NM, 1993.
[14] A. Lyubartsev, A. Martsinovski, P. Vorontosov-Veñuaminov, New approach to Monte Carlo calculation of the free energy: Method of expanded ensembles, Journal of Chemical Physics 96 (1992) 1776–1783.
[15] E. Marinari, G. Parisi, Simulated tempering: A new Monte Carlo scheme, Europhys. Lett. 19 (1992) 451–458.
[16] F. Sanger, A. Coulson, G. Hong, D. Hill, G. Petersen, Nucleotide sequence of bacteriophage lambda DNA, Journal of Molecular Biology 162 (4) (1982) 729–773.
[17] J. Setubal, J. Meidanis, Introduction to Computational Molecular Biology, Thomson, Boston, 1999.
[18] P. Pevzner, Computational Molecular Biology: An Algorithmic Approach, The MIT Press, 2000.
[19] J.H. Holland, Adaptation in Natural and Artificial Systems, first ed., The MIT Press, Cambridge, MA, 1975.

[20] M. Gen, R. Chen, Genetic Algorithms and Engineering Design, John Wiley & Sons, Inc., 1997.
[21] I. Oliver, D. Smith, J. Holland, A study of permutation crossover operators on the travelling salesman problem, in: Proceedings of the First International Conference on Genetic Algorithms, 1986, pp. 224–230.
[22] J. Grefenstette, Incorporating problem specific knowledge into genetic algorithms, in: L. Davis (Ed.), Genetic Algorithms and Simulated Annealing, Morgan Kaufman, 1987, pp. 42–60.
[23] L. Davis, Applying adaptive algorithms to domains, in: Proceedings of the International Joint Conference on Artificial Intelligence, 1985, pp. 162–164.
[24] D. Golberg, R. Lingle, Alleles, loci and the traveling salesman problem, in: Proceedings of the First International Conference on Genetic Algorithms, 1986, pp. 154–159.
[25] M.L. Engle, C. Burks, Artificially generated data sets for testing DNA fragment assembly algorithms, Genomics 16 (1993) 286–288.
[26] Y. Jing, S. Khuri, Exact and heuristic algorithms for the DNA fragment assembly problem, in: Bioinformatics Conference, 2003. CSB 2003, Proceedings of the 2003 IEEE, 2003, pp. 581–582.
[27] E. Alba, G. Luque, A new local search algorithm for the DNA fragment assembly problem, in: Proceedings of EvoCOP'07, Springer, Valencia, Spain, 2007, pp. 1–12.