# DNA Sequence Assembly and Genetic Algorithms New Results and Puzzling Insights

**Rebecca Parsons**
Department of Computer Science
University of Central Florida
Orlando, FL USA
rebecca@cs.ucf.edu

**Mark E. Johnson**
Statistics Department
University of Central Florida
Orlando, FL USA
markj@cs.ucf.edu

**Abstract**

Applying genetic algorithms to DNA sequence assembly is not a straightforward process. Significantly improved results in terms of performance, quality of results, and the scaling of applicability have been realized through non-standard and even counter-intuitive parameter settings. Specifically, the solution time for a 10kb data set was reduced by an order of magnitude, and a 20kb data set that was previously unsolved by the genetic algorithm was solved in a time that represents only a linear increase from the 10kb data set. Additionally, significant progress has been made on a 35kb data set representing real biological data. A single contig solution was found for a 752 fragment subset of the data set, and a 15 contig solution was found for the full data set. This paper discusses the new results, the modifications to the previous genetic algorithm used in this study, the experimental design process by which the new results were obtained, the questions raised by these results, and some preliminary attempts to explain these results.

## Genetic Algorithms and DNA Sequence Assembly

The laboratory processes for DNA sequencing are still limited to relatively short stretches of DNA, necessitating the assembly of longer sequences based on the base configuration of the shorter sequences. This assembly process, described in more detail below and in (Parsons, Forrest, & Burks 1994), is a combinatorial problem that is closely related to the Traveling Salesman Problem (TSP). However, the DNA sequence assembly problem is sufficiently different from TSP that the heuristic methods developed for TSP are not applicable in this context. In prior work, we have described a genetic algorithm for the DNA Sequence Assembly problem which gave promising results on small sequences but did not scale well to the problem sizes being tackled in the large-scale sequencing laboratories (Parsons & Burks 1993; Parsons, Forrest, & Burks 1994).

We have been studying the genetic algorithm to understand its behavior and the limiting factors on its performance. Several critical issues affect the performance of a genetic algorithm: problem representation, operator selection, population size, and parameter selection. Our previous work indicated that the performance of the genetic algorithm was sensitive to the setting of the rate parameters. This issue seemed important to address first. We applied the techniques of experimental design and response surface analysis (Box & Draper 1987; Box, Hunter, & Hunter 1978) to the rate parameters controlling the application of the various operators in the genetic algorithm. This analysis prompted us to set the operator rates at significantly different levels than those we had previously tried and those typically used by the genetic algorithm community. The result, however, was significantly improved convergence times for small to medium-sized problems and the resolution of a 20kb sequence which had previously not been solved by the genetic algorithm.

We next explored the population sizes for the genetic algorithm and found that significantly smaller population sizes resulted in additional performance improvements. Combining these results, we returned to the problem of the 35kb data set consisting of real data from a sequencing laboratory, a common benchmark for sequencing algorithms. We saw significantly improved performance on this data set as well, although we also found that minor modifications are required to the operators to properly exploit the building blocks within the evolved solutions.

In the remainder of this section, we introduce both the DNA Sequence Assembly problem and genetic algorithms. Next, we briefly summarize our previous work and describe the application of experimental design techniques to the GA parameter selection and the resulting parameter settings. We then describe the results obtained on the larger data sets using the modified GA. Finally, we discuss the questions of parameter setting and population sizes posed by these performance improvements and how these results relate to the application of genetic algorithms to other permutation problems.

## The Basics of DNA Sequence Assembly

The laboratory sequencing processes for DNA are being aggressively tested and improved as a result of the Human Genome Project. The scale of this project is daunting, as one individual's genome contains approximately 3 billion bases (bonded pairs of nucleic acids). DNA consists of two anti-parallel, complementary strands of nucleic acids. The shotgun sequencing process replicates and separates these strands, and then breaks the strands into smaller fragments. These fragments are processed on the automated sequencing machines to determine the base sequence for the fragment. Unfortunately, in the process of making and sequencing the fragments, information about the location, orientation and strandedness of the fragments with respect to the original sequence is lost. The only information remaining about a fragment is its base sequence. The assembly process compares the individual base pair sequences for the fragments and, using this information, assembles a consensus sequence for the parent DNA. Consecutive overlapping fragments make up a *contig*, and the goal of the assembly process is to order the fragments such that the result is one contiguous sequence of overlapping fragments. The hypothesis followed in the assembly process is that fragments with a high degree of similarity of their sequences (a high overlap strength) likely come from the same area of the DNA. The implications and validity of this hypothesis is addressed briefly in the conclusions and in Myers (Myers 1994).

In this paper, we discuss results on five realistically sized data sets, referred to here as POBF, AMCG, Seto, MSeto and MSeto2. The first data set, POBF, is a human apolopoprotein (Carlsson *et al.* 1986), with accession number M15421 which is 10089 bases long. The data set AMCG is the initial 40% (20,100) of the bases from LAMCG, the complete genome of bacteriophage lambda, accession numbers J02459 and M17233 (Sanger *et al.* 1982). The Seto data set is an experimental data set made available for testing sequencing algorithms (Seto, Koop, & Hood 1993). This data set represents 34,475 bases of consensus sequence. The data sets referred to here as MSeto and MSeto2 respectively are 752 and 743 fragment subsets of the Seto data set. When we analyzed the Seto set, we found 86 fragments that did not have significant similarity to the published consensus sequence based on our similarity metric. We removed these fragments from the data set, since they can not be properly placed, relative to the published consensus sequence, using only this similarity information (and not expert information from sequencers). Initially, we only removed the 77 fragments with similarity less than 10. The data sets have 177, 352, 829, 752 and 743 fragments respectively.

## The Basics of Genetic Algorithms

Genetic algorithms are a method of solving problems inspired by the principles of natural selection and genetic inheritance. Originally proposed by Holland (Holland 1975) in the seventies, genetic algorithms have experienced a resurgence in popularity and use (Forrest 1993; Goldberg 1989). In its simplest form, a genetic algorithm operates over a population of binary strings, termed individuals. A fitness function, usually whatever function is to be optimized, assigns a fitness to each individual in the population. An individual competes with the others in the population based on its fitness.

Three primary operators are used in each generation of a GA: reproduction, crossover and mutation. The reproduction operator implements the survival competition; the expected number of copies of an individual in the new population is based on the individual's fitness relative to the fitness of the rest of the population. Poorly performing individuals die out, with their places taken in the next generation by new copies of more highly fit individuals. The crossover operator combines the information contained in two individuals in the population to create offspring. The mutation operator randomly alters an individual. Finally, each new individual has its fitness computed using the fitness function. This cycle of reproduction, crossover, mutation, and evaluation continues until some convergence criterion is satisfied.

The design of a genetic algorithm for a problem requires the identification of a fitness function, a representation for an individual solution, crossover and mutation operators that generate new solution individuals from existing ones, and a selection mechanism. The representation and operators should be selected to exploit the building blocks of a problem, where a building block is a component of solution that both has a positive effect on fitness by itself and is part of a much better solution when combined with other building blocks. The dynamics of the genetic algorithm attempt to construct increasingly larger good building blocks until a good solution is found.

## Summary of Previous Work

Genetic algorithms have been applied to many different optimization problems, including parameter/function optimization and combinatorial optimization problems. Permutation problems such as job shop scheduling and the traveling salesman problem raise difficult representation and operator design issues. The obvious representation for a permutation, an ordered list of elements in the permutation, is not readily manipulated by the traditional crossover and mutation operators. Problem occur as there are a significant number of infeasible solutions (combinations that are not permutations) and the operators are not closed over feasible solutions. In our prior work, we used both the sorted-order representation with the standard operators and the straightforward representation with specialized operators (Parsons, Forrest, & Burks 1994). This work indicated that the specialized operators were essential to achieve good performance for the DNA Sequence Assembly Problem. However, we had limited success scaling that algorithm to problems larger than 10kb.

## Representation, Fitness Function, and Operators

The DNA Sequence Assembly problem is a permutation problem, so solutions consist of a permutation of the fragments. The simplest way to represent a permutation is using the list of fragment identifiers; this is the representation used here. This representation, though, necessitates the use of either specialized operators or some mechanism to penalize illegal solutions. We chose to use specialized operators. For the reproduction and selection phase, we chose to use generational reproduction with sigma scaling and an elitist strategy.

The edge-recombination crossover operator was specifically designed for problems such as the DNA Sequence Assembly Problem that exploit adjacency information in the formation of high-quality solutions. Edge recombination is a complicated operator; a detailed explanation of the operator implemented here appears in (Starkweather *et al.* 1991). In general, this crossover attempts to preserve adjacencies in the parents, and in particular, those adjacencies that are common to both parents. When neither of those options is possible, a random selection is made. This operator is appropriate for the sequence assembly problem since the building blocks of a good fragment ordering consist of a set of fragments that are related to each other by the similarity metric and should therefore be adjacent to one another.

For a mutation operator, we chose the swap. This is the simplest operator that preserves the permutation property. Two positions in the ordering are selected at random, and the fragments in those positions are swapped to create the new individual (Churchill *et al.* 1993). This operator is a restricted form of a 4-opt transformation (Lin & Kernighan 1973).

We used two other operators, each of which rely on some domain-specific information. These two operators, inversion and transposition, move blocks of fragments, specifically contigs, in the ordering. Contigs are selected by choosing a fragment at random and moving out in both directions until the edge of the either contig or the ordering is encountered. An adjacent pair of fragments with an overlap strength of zero denotes the edge of a contig. Inversion reverses the order of the fragments in the selected contig (Goldberg 1989). Transposition moves a contig to a position between two adjacent contigs. Transposition and inversion are also a restricted form of 4-opt, with the restriction focusing on the selection of the edges to break and with what edges to replace them.

The final factor in designing a genetic algorithm for a particular problem is the selection of an appropriate fitness function. The basic information available for evaluating an ordering is the pairwise similarity of the fragments in the fragment set (Churchill *et al.* 1993). The pairwise similarity is computed using the base sequence and considering all possible strand and orientation combinations. The final score, called the overlap strength, is the maximum similarity measure for the pair. There are several ways to evaluate the fitness of a particular ordering using this pairwise information. In previous work, we had determined that using a simple fitness function, analogous to that used for the traveling salesman problem, provided the best results for the genetic algorithm. Specifically, the fitness of an ordering is the sum of the pairwise overlap strengths for the fragments that are adjacent in the ordering:

$$\mathtt{F1}(I) = \sum_{i=0}^{n-2} w_{f[i],f[i+1]}$$

There are obvious problems with this fitness function, but it has proved adequate and computational efficient in the past.

Table 3 summarizes the previous results of the genetic algorithm on the larger data sets considered here. The genetic algorithm was able to find the correct solution to the 10kb data set, although the number of trials required did not indicate that the process scaled well for larger data sets. The previous best results for the 20kb and Seto data sets supported that concern. While the 13 contig solution is not a terrible one, the solution of the Seto set is not much improved over a random solution. Clearly, if the genetic algorithm was to contribute to problems of realistic size, the performance on these data sets had to be understood.

## Experimental Design and Genetic Algorithm Parameters

Use of a genetic algorithm requires the proper setting of several parameters: operator application rates, selection pressure and population size. Here we considered the setting of the operator rates and the population size. Previously, we had determined that the genetic algorithm for the fragment assembly problem required non-standard settings for the operator application rates (Parsons, Forrest, & Burks 1994). Specifically, the crossover rate required to achieve good performance, even on the smaller problems was low. Since the performance was sensitive to the operator rates, we undertook a statistical analysis of the behavior of the genetic algorithm relative to the operator application rates (crossover, swap, transposition and inversion). There was, in addition, some indication that there was some interaction among the operators that might have an affect on the performance.

Specifically, we did an initial study to determine the effects of high or low rates for each of these four parameters. These tests were performed on the 10kb POBF data set, which was the largest data set solved under that previous method. A $2^4$ full factorial design (Box,

|  | Cross. Rate | Trans. Rate | Swap Rate | Inversion Rate |
|---|---|---|---|---|
| High | .5 | .38 | .14 | .38 |
| Low | .3 | .28 | .04 | .28 |

Table 1: Rates for Full Factorial Experimental Design on POBF data set. Parameters for all runs: Population size 500, 1Mil trials, 2.1k generations, Sigma Scaling 2.0, and Elitist Strategy

Hunter, & Hunter 1978) was used in the initial investigation of the four parameters, at the rates shown in Table 1. The results obtained for these 16 combinations of parameter settings showed that the performance improved with lower rates both of crossover and of swap (coefficients -.016 and -.004) respectively. The performance was independent of the choice of rate for transposition and inversion, although the variance among the runs was lower with the higher rate of transposition. In addition, no second or third order affects among the parameters was indicated in this study. Thus, we could concentrate on the rates for the two parameters, crossover and mutation.

The analysis of the full factorial design led to the method of steepest descent, using the coefficients determined above, to search for an optimal region. This descent located a mesa-like region in the parameter-performance space, which we delineated using a central composite design. Both these approaches are techniques of response surface methodology (see, for example, (Box & Draper 1987)). These subsequent runs were all performed using the same parameters as the initial runs, varying only the crossover and swap rates. The lower rate of inversion and higher rate of transposition were used for the remainder of the runs.

The central composite design indicated that there was a large region surrounding the resulting values within which the genetic algorithm achieved good performance. The values indicated by this design, however, are counter-intuitive. Specifically, genetic algorithms are frequently assumed to derive most of their power from the crossover operator. While previous work has shown that crossover should be used in the operator mix, the final value indicated by the central composite design is a rate of 10%, much lower than previously considered.

Once these rates were determined, we turned to the population size. The time requirements for the genetic algorithm are typically dominated by the evaluation of the fitness function. The population size and the number of generations together determine the number of fitness function evaluations for a given run, also referred to as the number of trials. Larger population sizes can decrease the number of generations required but also increase the memory requirements. Too small a population size limits the diversity available to the genetic algorithm and limits the potential for crossover to contribute to the optimization. Premature convergence is a typical problem when the population size is too small, since the rate at which the superior individuals in a population reproduce quickly reduces the representation in the population of competing but potentially beneficial individuals.

We studied the performance of the genetic algorithm on the data set used above, varying the population sizes. The results are shown in part in Table 2. The table focuses on fitness function values only, as this is an appropriate measure of the quality of the optimization process itself. Each entry in the table represents the progress of the optimization on a typical run.[1] For each population size, several intermediate fitness function values and the final value after one million trials are recorded. A gross measure of homogeneity of the population is also shown. This convergence estimate is the approximate percentage of the bits that are the same across all individuals in the population (lost bits, in genetic algorithm terminology). This is a gross measure since, as discussed briefly below, the traditional measure of convergence assumes characteristics of the operators that are not held by our suite of operators.

The tests on population size indicated, for this data set, that a population size comparable to the number of fragments in the data set was the most effective value when considering total number of function evaluations, reproducibility of results, and population diversity. This result is somewhat surprising, when one considers the growth rate of the space of permutations and the fact that the individuals grow at a rate of *nlogn*. Additionally, the data indicates that there is a minimum population size below which the genetic algorithm can not effectively function, due presumably to a loss of diversity in the population. The conventional wisdom of genetic algorithms indicates that this minimum level should be much larger for a problem of this size — 1416 bits per individual — based solely on the number of low level building blocks. The next challenge was to see if these results generalized to larger and different data sets.

## Generalization of the Results and the Seto Data Set

Previously, the genetic algorithm was unable to solve the data sets that were larger than 10kb. Using the same parameter settings found for the 10kb data set and a population size only slightly larger than the number of fragments, we ran the genetic algorithm on the other large data sets described above. Table 3 summarizes the previous results for the three data sets described here and the results obtained using the new parameter settings.

[1]Other runs with different random seeds gave comparable results.

| Pop Size | Num Gens | Num Trials | Fitness Function | Conv Est |
|---|---|---|---|---|
| 1000 | 27 | 20,280 | 11,320 | 0 |
| 1000 | 326 | 240,711 | 36,079 | 0 |
| 1000 | 678 | 500,459 | 46,188 | 0 |
| 1000 | 1003 | 740,481 | 50,307 | 20.3 |
| 1000 | 1355 | 1,000,469 | 51,946 | 20.2 |
| 400 | 68 | 20,212 | 19,836 | 0.2 |
| 400 | 815 | 240,191 | 45,864 | 0.1 |
| 400 | 1,697 | 500,065 | 50,621 | 1.0 |
| 400 | 2,512 | 740,127 | 53,069 | 5.8 |
| 400 | 3394 | 1,000,054 | 54,622 | 1,7 |
| 200 | 136 | 20,041 | 18,540 | .1 |
| 200 | 1,632 | 240,127 | 48,802 | 21.6 |
| 200 | 3,397 | 500,091 | 52,964 | 30.7 |
| 200 | 5,024 | 740,138 | 55,046 | 28.6 |
| 200 | 6,788 | 1,000,040 | 55,966 | 24.8 |
| 50 | 551 | 20,020 | 38,439 | .2 |
| 50 | 6,598 | 240,012 | 52,194 | 2.2 |
| 50 | 13,692 | 500,015 | 55,202 | 36.6 |
| 50 | 20,244 | 740,031 | 56,134 | 34.6 |
| 50 | 27,342 | 1,000,003 | 56,647 | 46.0 |
| 10 | 2986 | 20,005 | 41,343 | 3.9 |
| 10 | 36,043 | 240,006 | 49,972 | 2.1 |
| 10 | 74,968 | 500,002 | 51,762 | 5.5 |
| 10 | 110,547 | 740,000 | 52,997 | 4.7 |
| 10 | 149,172 | 1,000,003 | 53,483 | 3.4 |

Table 2: Population Size Tests for POBF data set. Parameters for all runs: Crossover rate .1, Swap Rate .04, Transposition Rate .38, Inversion Rate .28, Sigma Scaling 2.0, and Elitist Strategy

The results presented in this table represent improvements in several aspects of the performance of the genetic algorithm although they also raise some issues. The number of function evaluations (trials) required to find the correct solution for the POBF data was reduced by an order of magnitude and the number of generations by a factor of 4 through the use of appropriate parameter settings and the smaller population size. For the AMCG data set, a smaller number of trials resulted in the correct solution using the adjusted parameter settings.

The results for the Seto set are most striking. While 27 contigs is not a reasonable answer, it is a dramatic improvement over the previous best solution we had obtained. As we were analyzing this solution, we realized that there were a large number of fragments that, according to our similarity metric (Churchill *et al.* 1993), did not match the parent sequence to any significant degree. Since our fitness function is driven only by the overlap information from this similarity metric and these fragments did not have significant overlaps, we removed them from the data set. At this stage, we identified 77 fragments that had a similarity metric of less than 10 with the final parent consensus, giving us a data set with 752 fragments.

| Name | Pop Size | Num Gens | Num Trials | Num Contigs |
|---|---|---|---|---|
| POBF | 1500 | 13,000 | 5,900,000 | 1 |
|  | 200 | 3393 | 500,117 | 1 |
| AMCG | 2500 | 5,600 | 2,300,000 | 13 |
|  | 400 | 6,786 | 2,000,021 | 1 |
| Seto | 2500 | 547 | 1,200,176 | 125 |
|  | 900 | 9,045 | 6,000,265 | 27 |
|  | 900 | 17,548 | 11,000,354 | 15 |
| MSeto | N/A | N/A | N/A | N/A |
|  | 775 | 14,003 | 8,000,54 | 5 |
|  | 775 | 37,623 | 21,500,393 | 1 |
| MSeto2 | N/A | N/A | N/A | N/A |
|  | 775 | 9,624 | 5,500,544 | 7 |

Table 3: Comparison of Genetic Algorithm Results. First line for a data set is result from prior work (Parsons, Forrest, & Burks 1994). Lines below use Crossover Rate .1, Swap Rate .04, Transposition Rate .38, Inversion Rate .28, Sigma Scaling 2.0, and Elitist Strategy.

The MSeto data set is the Seto data set with these 77 fragments mentioned above removed. The 5 contig solution presented above is very close to a correct solution. In fact, a simple greedy algorithm could convert this solution to the correct one. Genetic algorithms are best at getting close to the right solution, but do not tend to make the fine adjustments necessary to move from a near-optimal to an optimal solution. This behavior is consistent with the behavior of simulated annealing, another stochastic optimization technique (Bohachevsky, Johnson, & Stein 1993).

However, the solutions represent another challenge for the genetic algorithm; they contain defects that are not correctable, except by the relatively unlikely event of a swap operation. The discrepancies occur because lower quality, yet statistically significant overlaps are chosen instead of higher quality overlaps (an overlap of 75 is significant, but an overlap of 250 may be the correct one). The specialized operators are currently designed to retain significant overlaps by only moving contigs. As a consequence of the above choices, the connections between contigs can not be properly made, and the contig isolation and stranding of fragments occurs.

There are several approaches to this problem. We combined two of them in an effort to get to the optimal solution: relaxing the constraint on contig boundaries for the transposition and inversion operators and the addition of a form of greedy swap. Each of these techniques is described below.

While an overlap of 75-100 is clearly different than random, it likely represents an inappropriate placement for the fragment. Specifically, one of these fragments should likely be closer to the end of the contig where it may serve as a bridge fragment joining contigs. The completely random application of transposition and inversion proved ineffective (and counterproductive) in our earlier studies, prompting us to restrict application of the operator to contigs. However, this restriction means that the only mechanism available for the movement of this kind of isolated fragment to its proper location is the swap. What we chose to do instead was to introduce an additional degree of randomness into the transposition and inversion operator. We added a threshold value for the contig boundary in conjunction with a random value. Overlaps above that threshold were still considered within the contig. Overlaps below that threshold had an increasing probability of being designated as the contig boundary. For these preliminary experiments, we used a threshold of 100, with a 10% probability of breaking the contig at an overlap of 90 and a 90% probability at an overlap of 10.

The existing swap operation randomly selects two fragments in the ordering and swapped their positions. We introduced a modification to this operator, that only takes affect late in the run. Swaps late in the run are either completely random as before or greedy. For the greedy swap, one fragment, $i$, is chosen at random, and fragment $j$, the fragment with the highest overlap strength with $i$, is identified. A cursory analysis of the neighborhood in the ordering of both $i$ and $j$ is made to determine whether to move $i$ next to $j$ in the ordering or to move $j$ next to $i$. There are a couple of issues with an operator like this. First, it is important to delay the application of this operator until later in the run, since otherwise the genetic algorithm may be led too early into local minima. Second, it proved critical to examine the neighborhood surrounding the fragments to determine which move to make. The selection of when to begin the greedy swap and how often to apply it are currently ad hoc. As with the previous modification, more parameters have been introduced into the optimization process.

These changes allowed us to find a single contig solution of the MSeto data set, building from the population which produced the 5 contig solution. With this new operator configuration, we returned to the Seto data set and produced a 15 contig solution, again building on our previously evolved population. In examining this solution, we realized that there were more fragments that had low similarity to the parent with our metric. Indeed, there are 86 fragments with an overlap score of 13 or less with the parent sequence; the other 743 have an overlap score of 86 or more. Additionally, there are 9 fragments that have no overlap of weight greater than 100 with more than 1 other fragment. Therefore, we generated another, slightly smaller, data set with these 86 fragments removed. This data set, as shown in Table 3, is proceeding at a faster rate of improvement using the whole suite of modified operators. The performance of the genetic algorithm with the modified operators is encouraging for the large, and therefore realistically sized, data sets.

## Related Work

Different techniques have been applied to the problem of DNA Sequence Assembly. Greedy algorithms are the most popular, as they are easily the most efficient. In many data sets, greedy solutions are also the correct ones. The most popular greedy algorithm is that of Staden (Staden 1980), although Huang has proposed a greedy system that relies on a different mechanism for computing the similarity of two fragments and exploits this information later in the process (Huang 1992). Kececioglu and Myers (Kececioglu 1991; Kececioglu & Myers 1989) have proposed both an exact graph algorithm and an approximate algorithm that is provably close to the correct alignment when no errors exist in the data. Churchill *et al.* have developed a simulated annealer that uses the swap operator discussed above and a modified fitness function (Churchill *et al.* 1993).

## Towards an Understanding of Genetic Algorithms for Permutation Problems

The study of the population sizes has led to several questions about the effects on population diversity by the various operators. In the standard genetic algorithm, once a given position is the same in all members of the population, only the mutation operator applied at that position will allow that value to change. This effect is the result of the preservation of positions by the crossover operator. In the operator suite designed for this problem, all operators have the potential of altering all bit positions. Therefore, the traditional measure of convergence is not applicable. Indeed, the figures in Table 2 demonstrate that the homogeneity metric can vary significantly over the course of the run.

Most of the conventional wisdom regarding population size selection derives from the problems with diversity and convergence. As the population becomes less diverse, the search narrows drastically, since the individuals generated by a homogeneous population do not differ radically from those individuals in that population. We hypothesize that the different effects of our operator suite and their impact on convergence may be what allows us to use smaller populations than would otherwise be anticipated. We are exploring this question to determine the kind of models and behavior that can be expected for genetic algorithms applied to permutation problems. Whitley and Yoo have recently developed exact models of genetic algorithm behavior for certain of the permutation operators (Whitley & Yoo 1995). These models are inapplicable to the question posed here, because they assume an infinite population. However, their models, assuming these infinite populations, still provide insight into the asymptotic behavior in the finite population case.

## Conclusions

This paper reports on significant performance improvements for a genetic algorithm applied to the problem of DNA Sequence Assembly. Specifically, an order of magnitude improvement was obtained on a medium-sized data set, and larger data sets have either been solved completely or have produced workable near-optimal solutions. These performance improvements are the result of applying techniques from experimental design and response surface analysis to the parameter settings. Additional changes in the operator suite resulted in the solution of the realistic data sets. A better understanding of the nature of the performance enhancements and the operation of the genetic algorithm in this setting is needed. We intend to explore such questions as we extend this work further.

There are still the problems associated with the fitness function and problem formulation. Specifically, it is easily shown that, in the presence of significantly conserved repeat sequences, this formulation of the problem leads to a consensus sequence that is shorter than the correct sequence. This compression occurs since the overlap among fragments from different repeated regions is extremely high, violating the hypothesis of the assembly. Myers (Myers 1994) has proposed an alternative formulation for the sequencing problem to address the problem of repeated DNA in the sequence. These issues, as well as the value judgements made by the sequencers evidenced in the 86 fragments with low similarity to the parent sequence, indicate that the information being used by the genetic algorithm is likely insufficient to adequately solve the problem. However, the genetic algorithm should be able to exploit additional information through combined fitness functions such as those proposed by Burks et al (Burks, Parsons, & Engle 1994). The problem of repeated DNA is specifically addressed in that work through the use of map information. Optimization of the modified fitness function separates the members of the different repeat regions, resulting in the proper consensus sequence.

The DNA Sequence Assembly problem, one of the problems highlighted by DIMACS during its Computational Biology Year, is a critical part of the Human Genome Project. It also provides a realistic test bed for the study of genetic algorithms applied to permutation problems in general.

## Acknowledgments

The authors wish to acknowledge C. Burks, M. Engle and S. Forrest for their past help on this problem. Discussions with S. Forrest and C. Kennedy have been quite enlightening and have contributed to the progress reported on here. The referees offered useful suggestions to improve the presentation.

## References

Bohachevsky, I. O.; Johnson, M. E.; and Stein, M. L. 1993. Stochastic optimization and the gambler's ruin problem. *Journal of Computational and Graphical Statistics* 1(4).

Box, G. E. P., and Draper, N. E. 1987. *Empirical Model-Building and Response Surfaces.* John Wiley and Sons.

Box, G. E. P; Hunter, W. G.; and Hunter, J. S. 1978. *Statistics for Experimenters, An Introduction to Design, Data Analysis and Model Building.* John Wiley and Sons.

Burks, C.; Parsons, R.; and Engle, M. 1994. Integration of competing ancillary assertions in genome assembly. In *Proceedings of the Second International Conference on Intelligent Systems in Molecular Biology.* AAAI Press.

Carlsson, P.; Darnfors, C.; Olofsson, S.-O.; and Bjursell, G. 1986. Analysis of the human apolipoprotein B gene; complete structure of the B-74 region. *Gene* 49:29–51.

Churchill, G.; Burks, C.; Eggert, M.; Engle, M.; and Waterman, M. 1993. Assembling DNA sequence fragments by shuffling and simulated annealing. Los Alamos Technical Report.

Forrest, S. 1993. Genetic algorithms: Principles of natural selection applied to computation. *Science* 261:872–878.

Goldberg, D. E. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison Wesley Publishing Company.

Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: The University of Michigan Press.

Huang, X. 1992. A contig assembly program based on sensitive detection of fragment overlaps. *Genomics* 14:18–25.

Kececioglu, J., and Myers, E. 1989. A procedural interface for a fragment assembly tool. Technical Report TR-89-5, Depart of Computer Science, University of Arizona, Tucson, AZ.

Kececioglu, J. 1991. *Exact and approximation algorithms for DNA sequence reconstruction.* Ph.D. Dissertation, University of Arizona, Tucson, AZ. TR 91-26, Department of Computer Science.

Lin, S., and Kernighan, H. W. 1973. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research* 21:498–516.

Myers, G. 1994. An alternative formulation of sequence assembly. DIMACS Workshop on Combinatorial Methods for DNA Mapping and Sequencing.

Parsons, R., and Burks, C. 1993. An analysis of the random keys representation for DNA sequence assembly. Manuscript in Preparation.

Parsons, R.; Forrest, S.; and Burks, C. 1994. Genetic algorithms, operators and DNA fragment assembly. *Machine Learning.* To appear.

Sanger, F.; Coulson, A.; Hill, D.; and Petersen, G. 1982. Nucleotide sequence of bacteriophage lambda DNA. *J. Mol. Biol.* 162:729–773.

Seto, D.; Koop, B.; and Hood, L. 1993. An experimentally-derived data set constructed for testing large-scale DNA sequence assembly algorithms. *Genomics.* In press.

Staden, R. 1980. A new computer method for the storage and manipulation of DNA gel reading data. *Nucl. Acids Res.* 8:3673–3694.

Starkweather, T.; McDaniel, S.; Mathias, K.; Whitley, D.; and Whitley, C. 1991. A comparison of genetic sequencing operators. In *4th International Conference on Genetic Algorithms*, 69–76.

Whitley, D., and Yoo, N.-W. 1995. Modeling simple genetic algorithms for permutation problems. To Appear: Foundations of Genetic Algorithms 3.