A CASE STUDY IN EXPERIMENTAL DESIGN APPLIED TO GENETIC ALGORITHMS WITH APPLICATIONS TO DNA SEQUENCE ASSEMBLY

Rebecca Parsons
Department of Computer Science
rebecca@cs.ucf.edu

Mark E. Johnson
Statistics Department
mejohnso@pegasus.cc.ucf.edu

University of Central Florida
Orlando, Florida USA

Synoptic Abstract

Experimental design and response surface methodology is applied to tuning the parameters of an optimization program employing genetic algorithms. Attention is directed to the combinatorially challenging DNA sequence assembly problem. Fine tuning of a 10K size test problem leads to a considerably improved solution to a 35K problem of sequence assembly that is of significant biological interest.

Key Words and Phrases: genetic algorithms; design of experiments; response surface methods; DNA sequence assembly.

1. Introduction

Design and analysis of experiments and response surface methods have prospered in this century through successful applications in agriculture initially and in industry in more recent decades. Great strides in the development of these methods have taken place owing to improvements in computing capability. Although the semiconductor industry upon which our computing platforms depend has particularly benefited from design of experiments and response surface methods, computer algorithms themselves are rarely subjected to "quality improvements" via the statistical paradigms embodied in design of experiments and response surface methods. The likely explanation is that "experiments" are cheap to execute and the tedium of methodical experimentation goes by the wayside in the eagerness to obtain instant results. This scenario is reminiscent of the arguments used to defend "one-factor-at-a-time" experiments and other seat-of-the-pants strategies. Although statisticians can sermonize about the sins of inefficient experimentation, practitioners are more commonly swayed by positive approaches such as successful applications that may be adaptable to their own situation. Genetic algorithms are our platform and DNA sequence assembly is our application area in espousing this philosophy. Genetic algorithms provide a class of global optimization algorithms, inspired by the principles of survival of the fittest and natural selection, surveyed recently by [Collins, Eglese, and Golden, (1988)]. The DNA sequence assembly process is responsible for taking many smaller, unoriented fragments of DNA from a parent string of DNA and ordering them, based on a similarity metric, to reproduce the sequence of the parent fragment.

GENETIC ALGORITHMS FOR DNA SEQUENCING

The purpose of this paper is to describe our efforts in improving the performance of a genetic algorithm applied to the DNA sequence assembly problem. The DNA Sequence Assembly problem, one of the problems highlighted by DIMACS, the Center for Discrete Mathematics and Theoretical Computer Science, 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. Genetic algorithms have several "tuning" parameters (mutation rate, crossover rate) that can effect its performance (convergence rate, quality of solution). In contrast, simulated annealing typically uses a single parameter tied to the cooling schedule (see either [Bohachevsky, Johnson, and Stein, (1995)] or [Bohachevsky, Johnson, and Stein, (1986)]. Convergence theorems dictate the value of the parameters in theory, although in applications, empirical adjustments are necessary [Bohachevsky, Johnson, and Stein, (1986)]. Employing the conventional wisdom associated with the settings of tuning parameters for the genetic algorithm indicates that realistically-sized problems are unapproachable [Parsons, Forrest, and Burks, (1995)]. We started with a simple DNA sequence assembly problem addressed with a genetic algorithm and realized a ten-fold improvement in performance by merely adjusting the tuning parameters (mutation rate, crossover rate). Moreover, the high-performance area appeared to resemble a mesa, suggesting that the new settings in this region would be robust. We then ran the optimized genetic algorithm code on a real problem and were delighted to discover a one "contig" solution (the pieces assembled into one long contiguous string) that appears competitive with the

current, published solution. Moreover, in the course of assessing this solution, we discovered several anomalies in the DNA segments themselves that have not been addressed by our predecessors.

Section 3 presents a brief introduction to genetic algorithms. Although books on genetic algorithms and complexity have appeared in the popular press [Holland, (1975), Goldberg, (1989), Holland, (1995)], results and methods are generally unknown to most statisticians. Section 5 outlines the steps taken in the design of experiments - response surface methods endeavor seeking "optimal" parameters in the smaller sequence assembly problem. Although this effort is a somewhat standard application of these techniques, it is helpful to recognize the benefits accrued when the experimentation can take place on a work station. Section 6 focuses on the 35K assembly problem, yielding a solution to an interesting biological problem that previously required considerable "human intervention" to arrive at a consensus sequence. We argue on a number of grounds that the published consensus sequence is suspect.

2. The Basics of DNA Sequence Assembly

The goal of the Human Genome Project is to identify the exact sequence and function of the base pairs for the entire human genome, which consists of approximately 3 billion base pairs for each person. There are many components to this project; the work discussed here deals with the sequencing step – combining information about the sequence of small fragments of DNA into a coherent sequence for a much larger sequence. Waterman [Waterman, (1995)] includes an approachable introduction to many of the terms involved with

DNA sequencing and other computational challenges stemming from the Human Genome Project.

DNA consists of two anti-parallel, complementary strands of nucleic acids. The accuracy of the various sequencing processes constrain laboratory approaches to DNA sequencing (see [Howe and Ward, (1989)] or alternatively [Hunkapiller, Kaiser, Koop, and Hood, (1991)]. Currently, strands of DNA longer than approximately 500-1000 base pairs cannot routinely be sequenced accurately. However, the length of DNA strands of interest are on the order of 40,000 bases or even 400,000 bases in length. Consequently, these large strands of DNA are broken into smaller pieces for sequencing. 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 (parent) sequence is lost. The only information remaining about a fragment is its base sequence.

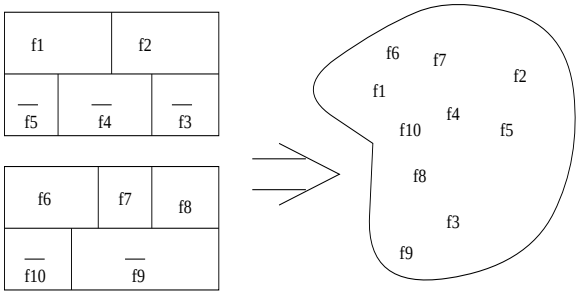

FIGURE 1: DNA Sequence Assembly with 2 Copies of the Parent

The assembly process is illustrated in Figure 1. The process begins with

a parent strand, that is copied numerous times, although the figure shows only two copies. Restriction enzymes are applied to the copies of the parent, creating a set of fragments. Since different enzymes look for different base sequences, the set of fragments for each parent differ. The bar over certain fragments indicate that they are on the opposite strand and therefore have a reverse orientation. This orientation is lost during the fragmentation process. The assembly process then compares the individual base pair sequences for the fragments and, using this information, assembles a consensus sequence for the parent DNA. Table 1 shows what the overlap comparison for the fragments in the figure would show. For example, fragment f_2 is wholly contained within the stretch of the parent DNA occupied by fragment f_9. Since f_2 is also a large fragment, this overlap is designated H for high-degree of overlap in the table. The other entries Table 1 can be determined from Figure 1 using the same approach.

Any empty locations in the table represent essentially random similarities, and hence are not necessarily zero. 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. One reasonable result ordering that accounts for all the high overlap scores would be the ordering of:

$$f_1, f_6, \bar{f}_5, \bar{f}_{10}, \bar{f}_4, \bar{f}_9, \bar{f}_2, f_7, \bar{f}_3, f_8$$

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 (see [Kececioglu and Myers, (1995)],

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
f_1				M	M	H			M	M
f_2			M	M			M	M	H	
f_3		M						M	M	
f_4	M	M				M	M		H	L
f_5	M					M				M
f_6	H			M	M				L	M
f_7		M		M					M	
f_8		M	M						M	
f_9	M	H	M	H		L	M	M		
f_{10}	M			L	M	M				

TABLE 1: Table of Relative Similarity Scores for Example Fragments. H – High degree of overlap; M – Medium degree; L – Low degree.

[Hunkapiller, Kaiser, Koop, and Hood, (1991)] or [Waterman, (1995)] for more information on the classic approach to DNA sequence assembly). The implications and validity of this hypothesis is addressed briefly in Myers [Myers, (1994)].

The DNA fragment assembly problem as it is formulated here is NP-hard. A problem is NP-complete (which stands for non-deterministic polynomial time complete) if it satisfies both of the following conditions:

1. The solution to the problem can be *verified* in an amount of time that grows as a polynomial in the size of the problem:

$$Time(n) = c_1 * n^k + c_2$$

Thus, the amount of computation time required to verify a solution to a problem of size n is a kth-order polynomial. This k is required to be independent of n. So, if one happened to guess the right answer, it would

take at worst a polynomial amount of time to verify that the guess was correct. The time to make the guess is assumed to be linear in the size of the problem.

2. It is possible to encode all other problems in the class of NP problems as an instance of this problem.

A problem is NP-hard if it is known only to satisfy the second of these two conditions.

The DNA sequence assembly problem resembles the better-known NP-hard problem, the Traveling Salesperson Problem (TSP). Briefly, an instance of TSP has a set of cities and distances between all pairs of cities. The objective is to find a tour of the cities that visits all cities exactly once and traverses the minimum total distance. By associating fragments with cities and the inverse of pairwise overlap strengths with the distances, the DNA fragment assembly problem can be seen as a TSP instance. There are crucial differences between TSP and fragment assembly that disallow many of the heuristic approaches to TSP. First, although the graph can be considered complete, many of the overlaps are zero. The overlap strengths do not satisfy the triangle inequality. Additional complications arise as inconsistent overlap information is possible due either to repeated sequences in the DNA itself or errors in the determination of the sequence of the fragment.

Since the sequence assembly problem is NP-hard, heuristic approaches are needed to solve an instance of the problem in reasonable time. Simulated annealing [Churchill, Burks, Eggert, Engle, and Waterman, (1993)]has been used for this problem as well as for the closely related physical map-

ping problem [Goldstein and Waterman, (1987)]. Here we describe a genetic algorithm approach to the fragment assembly problem. The genetic algorithm is implemented within the context of the Genesis genetic algorithm package [Grefenstette, (1984)]. Our modifications to Genesis to support the new operators are all written in the C programming language, as is the Genesis package itself. Genesis is available on the Internet; our modifications are available on request.

We use five realistically sized data sets, referred to as POBF, AMCG, Seto, MSeto and MSeto2 to demonstrate the performance of the approach. The first data set, POBF, is a human apolopoprotein with accession number M15421 in GenBank [Carlsson, Darnfors, Olofsson, and Bjursell, (1986)], 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, Coulson, Hill, and Petersen, (1982)]. The Seto data set is an experimental data set made available for testing sequencing algorithms [Seto, Koop, and 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 using only this similarity information. Initially, we only removed the 77 fragments with similarity less than 10. The data sets have 177, 352, 829, 752 and 743 fragments respectively.

3. The Basics of Genetic Algorithms

Genetic algorithms are a method of solving problems inspired by the principles of natural selection. 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 represents a potential solution to the problem or a setting of the function values. Each individual competes with the others in the population based on its fitness. Thus, the survival of a solution depends on its fitness relative to the fitness of other individuals currently in the population. Research on the behavior of genetic algorithms shows that this process exploits the presence of good "building blocks" – partial solutions that contribute positively to the fitness of an individual – by allocating an exponentially increasing number of trials to individuals with good building blocks.

Two primary search operators traditionally appear in genetic algorithms: crossover and mutation. The crossover operator combines two individuals in the population and creates two new individuals that each contain some portion of the genetic material of the parents. The mutation operator randomly alters some part of an individual to create a new individual. The selection mechanism determines which individuals move into the next generation and which die off. The basic selection mechanism uses the ratio of the fitness of the individual to the average fitness of the population to determine the expected value of the

number of copies of that individual that will appear in the next generation.

These general types of operators perform different roles in the evolution of solutions in a genetic algorithm. The crossover operator is considered to focus on exploiting the information existing in a population. In contrast, the mutation operator's role is exploring parts of the space not yet visited. The selection mechanism focuses the search to those areas that seem to have high fitness. Tuning a genetic algorithm requires balancing the forces of these operators to allow good solutions to evolve and flourish. The operators work together to combine good building blocks from different individuals to construct better and larger building blocks.

The genetic algorithm described here differs from the traditional genetic algorithm in several respects. First, 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 since the operators are not closed over the space of permutations. We chose to use specialized operators. Additionally, we use two specialized macro-operators that were designed to overcome deficiencies in the classes of solutions found. Thus, there are four operators: edge-recombination crossover; swap, which acts as our mutation operator; transposition; and inversion.

The fitness function used is analogous to that for TSP. We determine pairwise the similarity among all fragments to determine to what degree they overlap [Churchill, Burks, Eggert, Engle, and Waterman, (1993)]. This value

is their overlap score, $w[f_1, f_2]$. For individual $P = p_1, p_2, \ldots, p_n$, the fitness is computed as follows:

$$\sum_{i=1}^{n-1} w[f_{p_i}, f_{p_{i+1}}] \tag{1}$$

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. TSP is another problem in this class, while job shop scheduling is not. Edge recombination is a complicated operator; a detailed explanation of the operator implemented here appears in [Starkweather, McDaniel, Mathias, Whitley, and Whitley, (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.

The swap operator randomly selects two locations in the permutation and swaps the values. Transposition and inversion are a form of mutation operator in that they operate on one individual. These operators both affect individuals based on *contigs*. A contig is a region of an individual such that all adjacent fragments in the region have a non-zero overlap score. A contig represents a portion of the solution that forms a contiguous stretch of base pairs. The inversion operator randomly selects a contig and inverts the fragment ordering in the contig. The transposition operator selects two contigs at random and switches their position in the individual.

To help ameliorate the convergence problems, we use sigma scaling and an elitist strategy. Under the elitist strategy, the best individual encountered so far is always retained in the population. Sigma scaling controls convergence by limiting the number of new copies of an individual in the next generation.

The parameters required for a genetic algorithm with this configuration of operators include the rate of application of the four operators and the population size. We undertook this use of experimental design to discover how to improve the performance of the genetic algorithm.

4. 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. Problems 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, and Burks, (1995)]. 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.

Later in the text, we present Table 5, which 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.

5. Experimental Design for Genetic Algorithms

Genetic algorithm experiments have the same basic flavor as generic algorithms – namely, a set of independent variables to tweak the system and an outcome or performance measure to assess performance. The independent variables here are the algorithm tuning parameters (crossover, swap, transposition and inversion) as mentioned in the previous sections and include generally several operator application rates and population size. Previous work [17] had indicated that non-standard settings for the operator application rates would likely be required. Population size is a curious parameter in the mix in that it governs the rate of expenditures of computing resources in the investigation. Initially, we install this size at a fixed value and subsequent to fine-tuning the operator application rates, we decreased this characteristic for further savings. "Performance" has perhaps been inadvertently intimated to be total execution time to achieving the optimum for a given

problem. In fact, this could be a desirable yield/dependent variable definition – however, the characteristics of our problem suggest otherwise. Once the genetic algorithm approaches an optimal solution (as evidenced by diminished improvements of the fitness function), the "end-game" problem emerges (see [Bohachevsky, Johnson, and Stein, (1992)]). It is not productive to pursue improvements using a genetic algorithm at that stage, but rather one should use a problem specific strategy such as a hillclimbing algorithm. To finesse this difficulty, we opted to use the best value of the fitness function achieved for a specified expenditure of resources, measured by the total number of trials.

Initial interest focused on the four operators with two-level high and low rates given in Table 1. We used a fairly small population size of 500 and intended to report the performance measure as of one million trials. The problem chosen was the 10kb POBF data set, heretofore the largest problem solved under the conventional parameter settings. Intuitively, it seems quite conceivable that specification of one of the operator values could impact how another operator affects performance. Hence, we had concerns about potential interactions among the operator rates. Combining these concerns with our computing resource constraints (a Sparc workstation), we opted for a replicated 2^4 full factorial design with rates specified in Table 1. We had the ability to observe results of the full factorial design as they were produced so the run-order was set to be a half-fraction, the other half-fraction and then the replicates of the design. Although this constitutes what appears to be a blocking effect, the reality is that randomness arises solely from the random number generator so we could ignore this blocking contributor.

Our initial concerns about interactions turned out to be unfounded and in fact, only two of the four operator application rates (crossover and swap) had strong effects. Lower rates of each seemed to be desirable which suggested a steepest ascent search with these two variables with the other two rates effectively inert. However, the variance of the performance was lower with the higher rate of transposition, so that dictated its setting. For the time being, we focused attention solely on the rates for the crossover and swap operators. Table 3 gives the performance of the genetic algorithm for the replicated full factorial design. We then positioned a central composite design centered at a mutation rate of 3.2% and a crossover rate of 26.8%. We were very pleased to discover that there was a substantial region of crossover and swap values over which the genetic algorithm achieved good performance Figure 2 shows this mesa at a low rate for inversion and the high rate for transposition. The degree to which the performance plummets is striking.

The "optimal" region of good performance rates for crossover and swap are counter-intuitive. Historically, genetic algorithms were assumed to derive most of their power from the crossover operator, but set at a level much higher than that observed here. Of course, this preliminary result needed to be tempered until these rates were tested on larger problems. However, before doing so, we did some investigation of the population size. Population size is the resource driver in genetic algorithms affecting the number of evaluations of the fitness function. More specifically, the population size and the number of generations determine the number of fitness function evaluations for a given run – defined here as the number of trials. Large population sizes can reduce

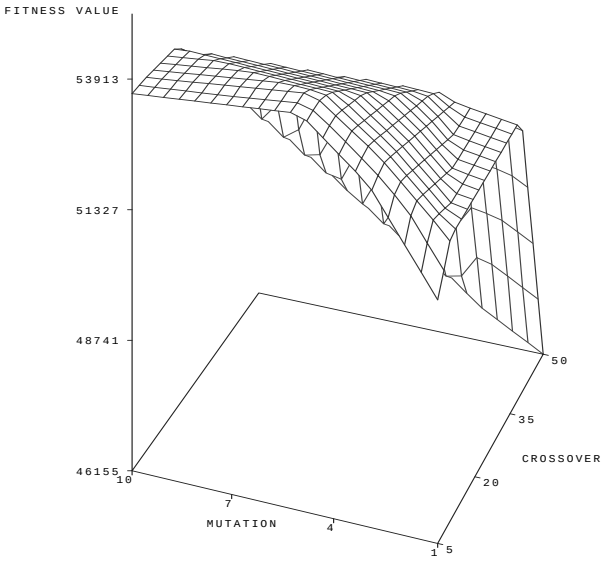

FIGURE 2: Fitness Response. Other parameters for all runs: Population size 500, 1 million trials, 2,100 generations, Sigma Scaling 2.0, and Elitist Strategy.

the number of generations required to produce convergence at the expense of increased memory requirements. Small population sizes could limit the diversity of the population and thus, could adversely affect the efficacy of 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 4. The table focuses on fitness function values only, as this is an appropriate

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

TABLE 2: Rates for Full Factorial Experimental Design on POBF data set. Parameters for all runs: Population size 500, 1 million trials, 2,100 generations, Sigma Scaling 2.0, and Elitist Strategy.

Crossover Rate	Transposition Rate	Mutation Rate	Inversion Rate	Fitness Value	
.5	.38	.14	.38	45,281	44,207
.5	.38	.14	.28	43,892	43,950
.5	.38	.04	.38	46,739	46,755
.5	.38	.04	.28	45,565	45,589
.5	.28	.14	.38	43,439	44,438
.5	.28	.14	.28	44,798	43,026
.5	.28	.04	.38	43,866	45,830
.5	.28	.04	.28	44,250	46,906
.3	.38	.14	.38	48,891	49,173
.3	.38	.14	.28	49,191	51,601
.3	.38	.04	.38	52,495	52,193
.3	.38	.04	.28	52,671	52,378
.3	.28	.14	.38	50,393	51,618
.3	.28	.14	.28	50,492	52,212
.3	.28	.04	.38	53,107	49,795
.3	.28	.04	.28	53,116	52,261

TABLE 3: Results for Full Factorial Experimental Design on POBF data set. Parameters for all runs: Population size 500, 1 million trials, 2,100 generations, Sigma Scaling 2.0, and Elitist Strategy.

measure of the quality of the optimization process itself. For each population size, several intermediate fitness function values and the final value after one million trials are recorded.

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

Pop Size	Fitness Value 20k	Fitness Value 240k	Fitness Value 500k	Fitness Value 740k	Fitness Value 1mil
1000	11,320	36,079	46,188	50,307	51,946
400	19,836	45,864	50,621	53,069	54,622
200	18,540	48,802	52,964	55,046	55,966
50	38,439	52,194	55,202	56,134	56,647
10	41,343	49,972	51,762	52,997	53,483

TABLE 4: 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.

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.

6. Results Generalization 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 5 summarizes the previous results for the three data sets described here and the results obtained using the new parameter settings.

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 similar-

DataSet Name	DataSet Size	Population Size	Number Generations	Number Trials	Number Contigs
POBF	177	1500	13,000	5,900,000	1
		200	3393	500,117	1
AMCG	352	2500	5,600	2,300,000	13
		400	6,786	2,000,021	1
Seto	829	2500	547	1,200,176	125
		900	9,045	6,000,265	27
		900	17,548	11,000,354	15
MSeto	752	N/A	N/A	N/A	N/A
		775	14,003	8,000,54	5
		775	37,623	21,500,393	1
MSeto2	743	N/A	N/A	N/A	N/A
		775	9,624	5,500,544	7

TABLE 5: Comparison of Genetic Algorithm Results. First line for a data set is result from prior work. Lines below use Crossover Rate .1, Swap Rate .04, Transposition Rate .38, Inversion Rate .28, Sigma Scaling 2.0, and Elitist Strategy.

ity metric [Churchill, Burks, Eggert, Engle, and Waterman, (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.

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, and Stein, (1992), Johnson, (1988)]. As in the case of simulated annealing, other algorithms, such as a hillclimbing algorithm, can be used to make these fine adjustments.

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 counter-productive) 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 5, 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.

7. Towards an Understanding of Genetic Algorithms

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 4 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 and 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.

8. 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 remain 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, and 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.

Acknowledgments

The authors wish to acknowledge C. Burks, M. Engle and S. Forrest for their help on this problem. Discussions with S. Forrest have been quite enlightening and have contributed to the progress reported on here. Computational assistance from D. Nickerson is gratefully acknowledged.

References

[Bohachevsky, Johnson, and Stein, (1986)] I.O. Bohachevsky, M.E. Johnson, and M.L. Stein. Generalized simulated annealing for function optimization. *Technometrics*, 28:209–217.

[Bohachevsky, Johnson, and Stein, (1992)] I.O. Bohachevsky, M.E. Johnson, and M.L. Stein. Stochastic optimization and the gambler's ruin problem. *Journal of Computational and Graphical Statistics*, 1(4):367–384.

[Bohachevsky, Johnson, and Stein, (1995)] I.O. Bohachevsky, M.E. Johnson, and M.L. Stein. Simulated annealing and generalizations. In J.H.Kalivar, editor, *Adaptation of Simulation Annealing to Chemical Optimization*, pages 3–24. Elsevier, Amsterdam, The Netherlands.

[Burks, Parsons, and Engle, (1994)] C. Burks, R.J. Parsons, and M.L. Engle. Integration of competing ancillary assertions in genome assembly. In *Proceedings of the Second International Conference on Intelligent Systems in Molecular Biology*, pages 62–69. AAAI Press, Menlo Park, California, USA.

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

[Churchill, Burks, Eggert, Engle, and Waterman, (1993)] G. Churchill, C. Burks, M. Eggert, M.L. Engle, and M.S. Waterman. Assembling DNA sequence fragments by shuffling and simulated annealing. Technical report, Los Alamos National Laboratory.

[Collins, Eglese, and Golden, (1988)] N.E. Collins, R.W. Eglese, and B.L. Golden. Simulated annealing – an annotated bibliography. *American Journal of Mathematical and Management Sciences*, 8:209–307.

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

[Goldberg, (1989)] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison Wesley Publishing Company, Reading, Massachusetts, USA.

[Goldstein and Waterman, (1987)] L. Goldstein and M.S. Waterman. Mapping DNA by stochastic relaxation. *Advances in Applied Mathematics*, 8:194–207.

[Grefenstette, (1984)] John J. Grefenstette. Genesis: A system for using genetic search procedures. In *Proceedings of a Conference on Intelligent Systems and Machines*, pages 161–165, Rochester, Minnesota, USA.

[Holland, (1975)] J.H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, Michigan, USA.

[Holland, (1995)] J.H. Holland. *Hidden Order: How Adaptation Builds Complexity.* Addison Wesley, Reading, Massachusetts, USA.

[Howe and Ward, (1989)] C.M. Howe and E.S. Ward, editors. *Nucleic Acids Sequencing: A Practical Approach.* The Practical Approach Series. IRL Press at Oxford University Press, Oxford, England.

[Hunkapiller, Kaiser, Koop, and Hood, (1991)] T. Hunkapiller, R.J. Kaiser, B.F. Koop, and L. Hood. Large-scale and automated DNA sequence determination. *Science*, 254:59–67.

[Johnson, (1988)] M.E. Johnson, editor. *Simulated Annealing (SA) and Optimization: Modern Algorithms with VLSI, Optimal Design and Missile Defense Applications.* Volume 17, American Series in Mathematical and Management Sciences. American Sciences Press, Columbus, Ohio, USA.

[Kececioglu and Myers, (1995)] J. D. Kececioglu and E.W. Myers. Combinatorial algorithms for dna sequence assembly. *Algorithmica*, 13(1/2):7–51.

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

[Parsons, Forrest, and Burks, (1995)] R.J. Parsons, S. Forrest, and C. Burks. Genetic algorithms, operators and DNA fragment assembly. *Machine Learning*, 21(1/2):11–33.

[Sanger, Coulson, Hill, and Petersen, (1982)] F. Sanger, A.R. Coulson, D.F. Hill, and G.B. Petersen. Nucleotide sequence of bacteriophage lambda DNA. *Journal of Molecular Biology*, 162:729–773.

[Seto, Koop, and Hood, (1993)] D. Seto, B.F. Koop, and L. Hood. An experimentally-derived data set constructed for testing large-scale DNA sequence assembly algorithms. *Genomics*, 15(3):673–673.

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

[Waterman, (1995)] M.S. Waterman. *Introduction to Computational Biology: Maps, Sequences and Genomes*. Interdisciplinary Statistics Series. Chapman and Hall, London, England.

[Whitley and Yoo, (1995)] D. Whitley and N. Yoo. Modeling simple genetic algorithms for permutation problems. In *Foundations of Genetic Algorithms 3*. Morgan Kaufman, San Mateo, California, USA.