DNA Computing Implementing Genetic Algorithms

Junghuei Chen, Eugene Antipov, Bertrand Lemieux, Walter Cedeño, and David Harlan Wood

(J. Chen) Dept. of Chemistry & Biochemistry, University of Delaware, Newark, DE
junghuei@udel.edu http://www.udel.edu/chem/chen/chen.html

(E. Antipov) Dept. of Chemical Engineering, University of Delaware, Newark, DE
ultra@udel.edu

(B. Lemieux) Dept. of Plant & Soil Sciences, University of Delaware, Newark, DE
blemieux@udel.edu http://bertrand.ags.udel.edu

(W. Cedeño) Hewlett-Packard Company, 2850 Centerville Road, Wilmington, DE
wcedeno@lf.hp.com http://www.personal.psu.edu/users/w/x/wxc28

(D. Wood) Department of Computer Science, University of Delaware, Newark, DE
wood@cis.udel.edu http://www.cis.udel.edu/~wood

Abstract. Genetic algorithms, DNA computing, and *in vitro* evolution are briefly discussed. Elements of these are combined into laboratory procedures, and preliminary results are shown. The traditional test problem for genetic algorithms called the MAX 1s problem is addressed.

1 Introduction

Evolution is a concept of obtaining adaptation through the interplay of selection and diversity. Analogies from evolution have been used in both computing and molecular biology. These two areas are called respectively "evolutionary computation" and "*in vitro* evolution." From the beginning of DNA based computing to the present there have been calls [15,3] to consider carrying out evolutionary computations using genetic materials *in vitro*. Almost two years ago, however, the first design for a DNA based genetic algorithm for a specific class of problems was proposed [2]. Unfortunately, this has not yet been carried out in the laboratory.

In this paper we identify elements of evolutionary computations and *in vitro* evolution that we recommend combining to address three simple problems. Specifically, we choose "genetic algorithms" because they manipulate bitstrings using operations of pointwise mutation and crossover. These operations can be performed by modifications of techniques from *in vitro* evolution. In particular, single point crossover extends results due to Stemmer [17,13].

We propose a laboratory implementation of one of these computations, and present our design. The crucial operation of physically separating DNA

strands by their "fitness" is demonstrated using 2-d denaturing gradient gel electrophoresis.

2 Genetic Algorithms and Other Evolutionary Computations

Genetic algorithms and other types of "evolutionary computation" come in many, many different varieties, but most are variations and/or elaborations on the following very loose outline. Genetic algorithms typically cycle electronic computers through the following steps, maintaining a population of bitstrings representing candidate solutions of a problem.

Genetic Algorithm
Begin with a diverse initial population, perhaps chosen randomly.

1. Evaluate fitness of candidates.
2. Select more fit candidates to breed and lesser candidates to be replaced.
3. Induce variation by breeding.

Repeat.

It should be mentioned that this loose outline fits several evolutionary computation paradigms having varying techniques for fitness evaluation, selection, and breeding.

2.1 Evolutionary Computation Has Controversial Aspects

We are careful to make the following point. We do not take any stance on the virtues of any particular method of evolutionary computation. Instead, we aspire to *provide the means for assessing* some evolutionary computations using population sizes larger than is practical with conventional computers.

Evolutionary computation makes few assumptions and is ostensibly applicable to very broad classes of problems. Naturally, this makes it difficult to establish any provable guidelines. Just to list a few debatable issues, we have

- How does one model fitness?
- Is crossover disruptive?
- What are the roles of transposition, inversions, and introns?
- Would pointwise mutation alone suffice?
- Which evolutionary computations predictably converge?
- How does one recognize convergence?

2.2 Varieties of Evolutionary Computation

Several different computing paradigms are inspired by biological evolution: genetic algorithms, classifier systems, genetic programming, evolutionary programming, and evolution strategies. A good initial orientation to all these paradigms is found at [8].

Indispensable books on genetic algorithms include the founding treatise of Holland [9], the popular textbook of Goldberg [5], and the books of Mitchell [11] and Forrest [4]. A bibliography of about 4,400 contributions to genetic algorithms is available [6].

3 DNA Is Suitable for Implementing Genetic Algorithms

We expect computing time using DNA to be proportional to the number of generations required. This motivates incorporating both pointwise mutation and crossover, and for that matter any evolutionary analogies that might reduce the number of required generations.

Modifications to current technology suffice to implement crossover and pointwise mutation. However, selecting DNA strands for "breeding" in genetic algorithms is moderately challenging because one must physically separate DNA strands according to their "fitness."

3.1 DNA Attributes Suit Genetic Algorithms

Of all evolutionarily inspired approaches, genetic algorithms seem particularly suited to implementation using DNA. This is because genetic algorithms are generally based on manipulating populations of bitstrings using both crossover and pointwise mutation.

DNA computing techniques are desirable for genetic algorithm computations. The first main advantage is that these techniques might process, in parallel, populations billions of times larger than is usual for conventional computers. The usual expectation is that larger populations can sustain larger ranges of genetic variation and thus can generate high-fitness individuals in fewer generations.

The second main advantage is the massive information storage available using DNA. For example, a gram of DNA contains about 10^{21} bases. The information content is approximately 2×10^{21} bits, greatly exceeding the 200 petabyte storage of all the digital magnetic tape produced in one year (1995). [24].

A third advantage is that methods for implementing crossover using DNA are possible as variations on "Sexual PCR" pioneered by Stemmer [17].

An additional issue is that biolaboratory operations on DNA inherently involve errors. These are more tolerable in executing genetic algorithms than

in executing deterministic algorithms. To some extent, errors may be regarded as contributing to desirable genetic diversity.

3.2 DNA Genetic Algorithms Compared to Supercomputers

The following oversimplified estimates indicate DNA computing techniques can compare favorably to supercomputers in some cases. These favorable cases include executing genetic algorithms having simple fitness evaluations and very large populations of candidate solutions.

Consider a population represented by total of p bits. The work involved in executing a genetic algorithm can be estimated as

$$T \approx g \times \mathcal{O}(p) \text{ fitness evaluations,} \tag{1}$$

where g is the number of generations used.

Assume the fitness evaluation of a candidate solution processes all the bits of the candidate solution. A state-of-the-art teraflop supercomputer performs 10^{12} operations per second. Thus, from Eq 1 we have the very rough estimate

$$T \approx g \times p \times 10^{-12} \text{ seconds} \approx g \times p \times 10^{-17} \text{ days.} \tag{2}$$

To compare this to DNA computing, let us assume the fitness evaluation of an entire population can be done in the laboratory in 24 hours. Thus, from Eq 1 the time complexity of a DNA approach to genetic algorithms is seen to be

$$T \approx g \text{ days.} \tag{3}$$

Essentially no new laboratory techniques or equipment would be needed to use gram quantities of DNA. This corresponds to populations represented by about $p = 10^{21}$ bits. For populations of this size, we see from Eq 2 and from Eq 3 that DNA implementation of genetic algorithms compares favorably to supercomputer time complexity.

Some caution is needed in interpreting this comparison. The comparison is based on unprecedented large populations. (Still miniscule compared to the size of the space of all possible solutions, of course!) As far as we know, it is unclear exactly how beneficial large population sizes might be. The classical "schema theorem" of Holland [9] says a population of N distinct candidates probes $\mathcal{O}(N^3)$ potential solutions. However the applicability of this result, like many others in evolutionary computation, is actively debated.

This may be an appropriate point to repeat our primary goal. We do not take any stance on the virtues of any particular method of evolutionary computation. Instead, we aspire to provide the means for assessing some evolutionary computations using population sizes larger than is practical with conventional computers.

3.3 DNA Genetic Algorithms Compared to *In Vitro* Evolution

Genetic algorithms are reminiscent of methods in molecular biology referred to as "*in vitro* evolution." Naturally, these methods use fitness criteria constrained to properties of biomolecular interest. Indeed, finding means to physically separate biological materials by "fitness" has determined which problems are addressed by *in vitro* evolution. *In vitro* methods suit some very important but relatively small classes of problems seeking ribozymes [21,12,7,19], binding sites [20,19], enzymes [14], etc.

In contrast to *in vitro* evolution, genetic algorithms use *unconstrained fitness criteria* on bitstrings. Consider computations using bitstrings of length 100. Such computations can, in principal, evolve a population of fixed size in such a way as to create any one of $2^{100} \approx 10^{30}$ possible outcomes. Thus, genetic algorithm methods using DNA can address larger, but less structured, classes of problems than does *in vitro* evolution.

Genetic algorithms using DNA would be similar to conventional computers in that (virtually) all 10^{30} possible inputs and outputs would be equally suitable. In contrast, *in vitro* evolution is suited to the very rare DNA sequences encoding biological functionality. For example, bitstrings might be realized as DNA strands having 100 As and Gs. But as far as we know very few, if any, such DNA bitstrings code for biologically active functions. Or to put it another way, virtually all sequences are equally "meaningful" for the purposes of genetic algorithms. In contrast, *in vitro* evolution focuses on variations of the very rare DNA sequences of biological or biochemical interest.

3.4 Simple Problems for DNA Genetic Algorithms

Here we present three problems that are relatively simple to address using DNA implementations of genetic algorithms. The common thread is that more fit candidate strands of DNA can be physically separated from less fit candidates according to how well they match (hybridize with) "target" DNA strands. More details are provided in Section 4.

The MAX 1s Problem This is a traditional test problem for genetic algorithms. It involves binary bitstrings of fixed length. An initial population (usually randomly generated) is given. The objective is to evolve some bitstrings to match a prespecified "target" (generally taken to be all 1s).

A design of laboratory techniques for solving this problem using DNA is given in Section 4 of this paper.

The Royal Road Problem The "Royal Road" family of problems are of particular interest because it is one of very few families of problems for which theoretical predictions are available [23].

Again, a fixed-length target is specified consisting of N blocks, each block consisting of K bits. Each block of a candidate bitstring makes no contribution to fitness unless it is a *perfect* match to the corresponding block on the target. Conventionally, the fitness of a candidate is taken to be the number of such perfectly matched blocks. The objective is to evolve some bitstrings to perfectly match the target.

This family of problems got its name from the fact that it was intended to be especially suitable for genetic algorithms using crossover. Distressingly, computer trials exhibit a wide variety of unpleasant convergence behaviors (see Figure 1). Confirming an earlier conjecture [22], a recent seminal pa-

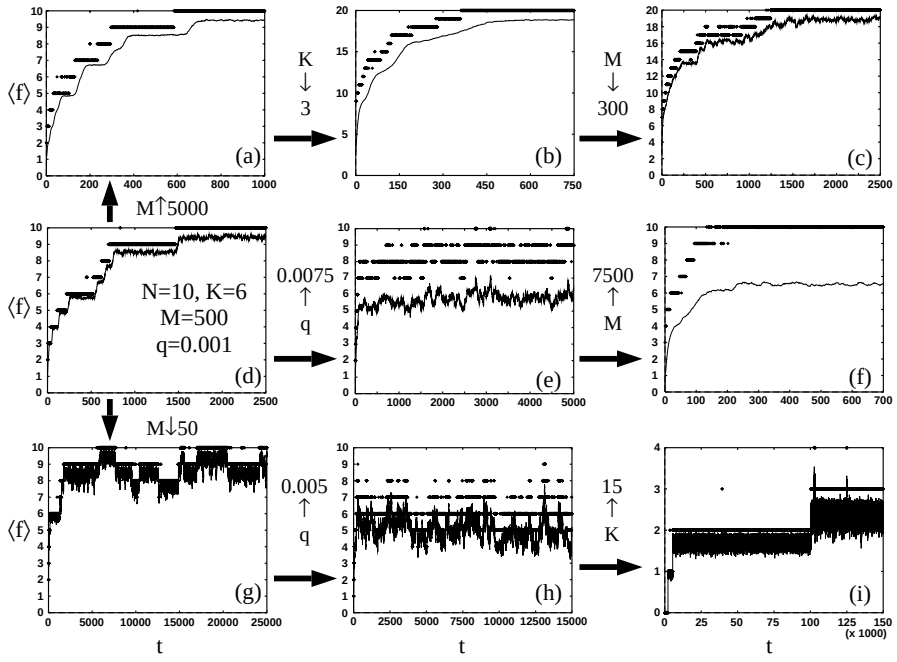

Fig. 1. Evolution of average fitness <f> for a genetic algorithm for the Royal Road problem varies greatly with population size M, mutation rate q, and the number of blocks N, each having K bits. Eight graphs show the effects of varying parameters from those used in graph (d). (From [23], with permission.)

per [23] from the Santa Fe Institute predicts the previously unanticipated behaviors for the Royal Road problem, attributing them to limitations on

population sizes. In future work on Royal Road problems we hope to test the predictions of the Santa Fe paper using population sizes which are too large to be practical for conventional computers.

The Cold War Problem The above problems are particularly suitable for theoretical analysis because the answer (target) is known in advance. We now mention a problem of another kind with many potential applications. In the Cold War Problem, two populations are simultaneously evolved. For simplicity, let us continue to refer to them as candidates and targets, and assume they are of fixed length.

Each of two parties is able to generate (breed) vast numbers of possible offers (candidates or targets) reflecting their own interests. The objective is to evolve good matches between candidates and targets. An outline of a genetic algorithm for this problem appears below.

Genetic Algorithm for the Cold War Problem

Begin with diverse initial populations of targets and candidates.

1. Evaluate fitness of target-candidate pairs.
2. Select more fit pairs to breed and lesser pairs to be replaced.
3. Using more fit pairs, separate candidates from targets.
   (a) Induce variation of candidates by breeding.
   (b) Induce variation of targets by breeding.
4. Combine offspring candidates and targets, obtaining a new generation.

Repeat.

This problem is particularly interesting because it further exploits the massive parallelism inherent in DNA computing. Here, fitness evaluation is to be done, not for individual candidates, but for *pairs* of candidates. This means that conventional computers would have a time complexity of $T \approx g \times p^2 \times 10^{-17}$ days instead of $T \approx g \times p \times 10^{-17}$ days, as in Eq 2. In contrast, the time complexity of the DNA implementation is unchanged, $T \approx g$ days.

4 The MAX 1s Problem Implemented in DNA

This section begins with an outline of a DNA implementation (illustrated in Figure 2). The remainder of the section gives details and preliminary laboratory results.

Throughout this section, information is grouped in the following categories: (1) candidate pool, (2) fitness evaluation, (3) selection and (4) breeding.

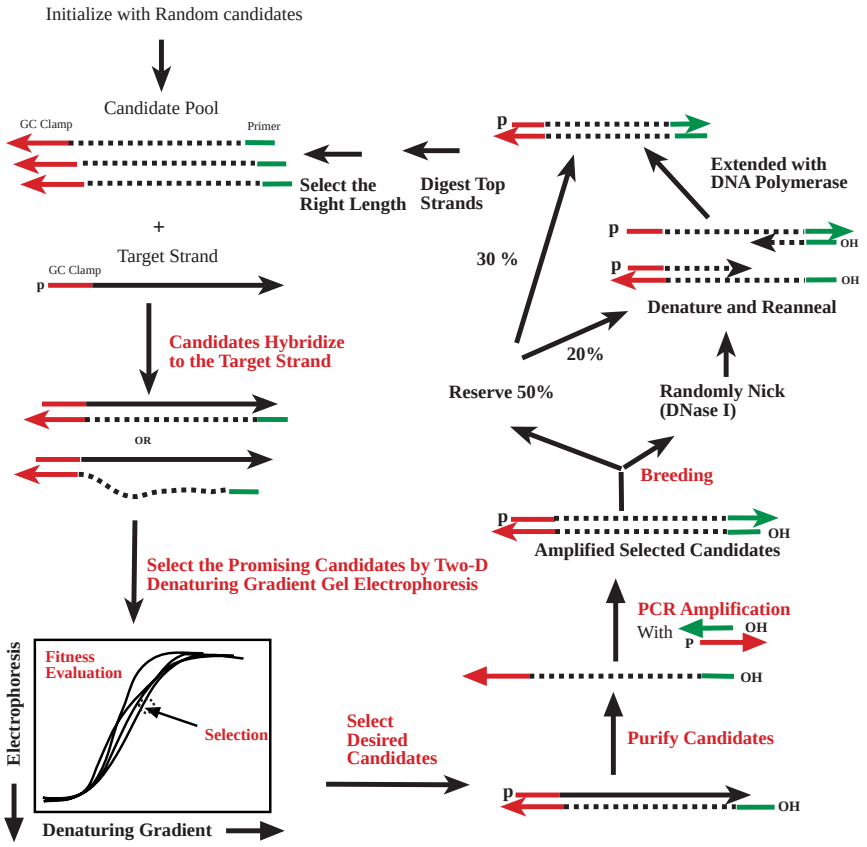

Fig. 2. Outline of DNA implementation of genetic algorithms for MAX 1s problem. The candidate pool appears in the upper left. Selection using 2-d DGGE appears at the lower left. Purification and amplification of the more fit candidate strands appears at the lower right. Breeding using crossover appears at the upper right.

4.1 Outline of DNA Implementation

The implementation is given by the following outline. The same information, with a few added details, is shown in Figure 2.

DNA Genetic Algorithm for MAX 1s Problem
Begin with a diverse initial population of candidates.

1. Evaluate fitness by hybridizing to target strands and physically separate on a 2-d gel.
2. Select and purify more fit candidates to breed.
3. Amplify fit candidates with pointwise mutation and reserve a portion.
4. Breed candidates, using crossover.
5. Combine reserved and bred candidates, obtaining a new generation.

Repeat.

4.2 Design of Candidate Solutions and Target DNA Strands

Figure 3 shows our design. A target strand and a perfect candidate strand are

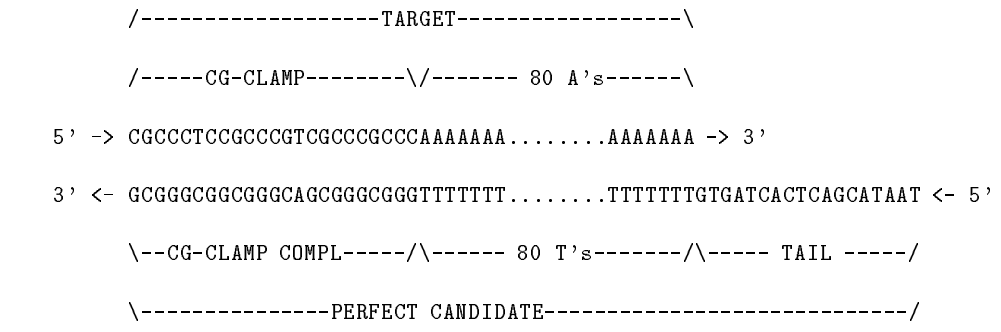

Fig. 3. Design of target and a perfect candidate. Imperfect candidates would have a mixture of 80 Ts and Cs in place of the 80 Ts in the perfect candidate.

shown in the figure. Imperfect candidate strands have a mixture of 80 Ts and Cs instead of 80 consecutive Ts. At the $3'$ end of all candidate strands there is a universal section complementary to the CG clamp section of the target strands. The CG clamp has been designed to encourage correct alignment and to avoid secondary structure (hybridizing to itself). The candidate strands are longer to facilitate eventual separation of target and candidate strands using denaturing gel electrophoresis. All $5'$ ends of the candidate strands are extended by a universal tail sequence. Since candidate strands have known primer sites at both ends, they can be amplified by PCR.

4.3 Fitness Evaluation by DGGE Physical Separation of DNA

Our fitness evaluation is carried out in the laboratory using two-dimensional (2-d) denaturing gradient gel electrophoresis (DGGE) [10]. Let us first review the nature of DGGE. Figure 4 shows DGGE from our laboratory having perfect candidates combined with target strands. The target strands hybridize (stick) to the perfect candidate strands, with a tail of unmatched bases at the $5'$ end. The mixture of hybridized strands is placed uniformly along the top of the gel. The hybridized strands travel vertically downward in the gel as a result of an applied electric field. However, their speed of migration is determined by their initial placement from left to right; that is, by how strongly they are denatured (pulled apart). On the left, where no denaturant is encountered, the strands move relatively quickly downward. In the center, they move more slowly because they encounter intermediate denaturing. At the extreme right, the stands are able to move only very slowly because the

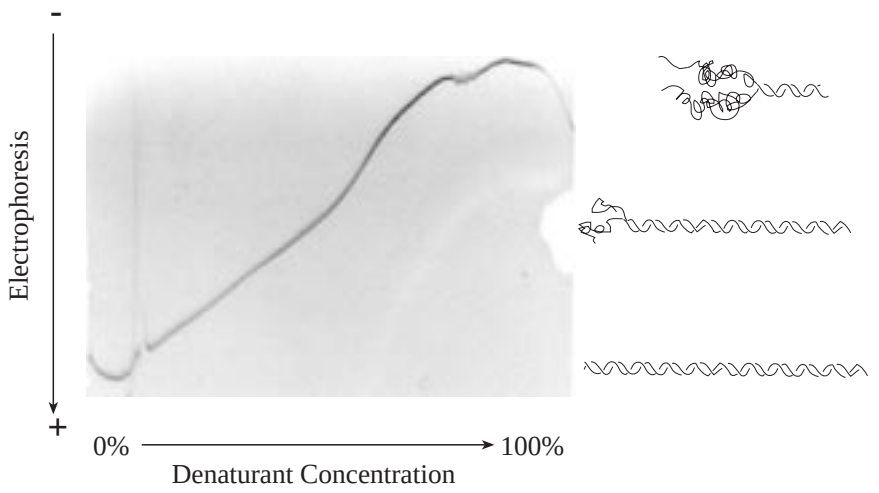

Fig. 4. DGGE using perfect candidates. DNA strands move downward from a reservoir at the top of the figure. The speed of vertical strand migration is retarded as strands come apart (denature) as shown schematically on the right edge of the figure.

strands are almost completely pulled apart except in the more resistant CG clamp region.

An important fact for DNA computing is that *DGGE can detect a single base mismatch.* Indeed, this is a common application of DGGE in molecular biology [10].

We heuristically reason that when we repeat the above experiment with a mixture of targets and imperfect candidates, we expect that everywhere across the gel the candidate strands that best match (hybridize to) the targets will migrate downward relatively faster. In fact, imperfect matches exhibit vertical spreading in our experiments. See Figure 5. In this figure the 80 variable positions of the imperfect candidates are chosen to be Ts with probability 0.8 and Cs with probability 0.2.

4.4 Selection of More Fit Candidate Solutions

Selection is done by excising a portion of the 2-d gel and extracting the DNA strands from it. This allows a wide latitude for selection criteria. The most fit candidates are presumably lowest on any vertical line. However, the nature of variation from left to right is not clear. Further experiments will be needed to optimize a selection strategy. Experience in genetic algorithm computing demonstrates the desirability of maintaining genetic diversity to prevent the loss of genetic information which may be needed in later stages of evolution.

The selected DNA is purified (for example by length, using conventional denaturing gel electrophoresis) to get rid of target strands. The purified candi-

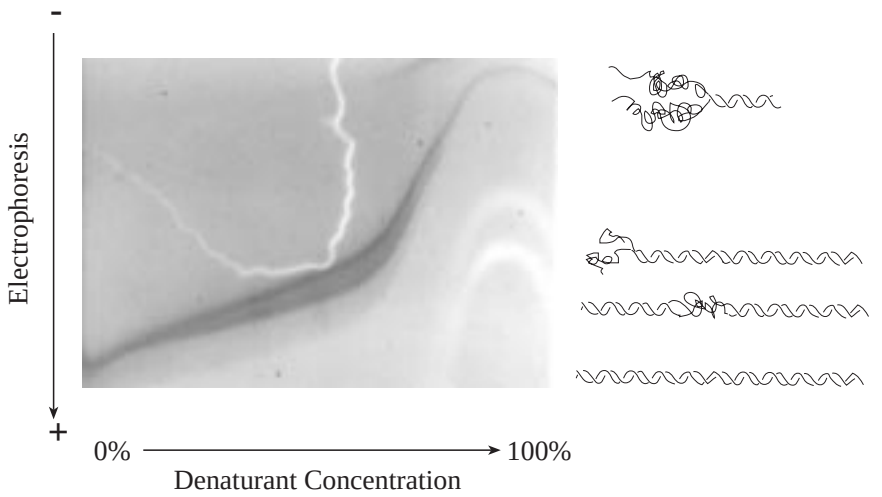

Fig. 5. DGGE vertically separates imperfect candidates. The speed of vertical strand migration is retarded as strands come apart (denature). This is due to two factors: increasing denaturant concentration and decreasing quality of target-candidate matching (hybridization).

date strands are amplified by PCR, which can also induce pointwise mutation at a rate of 10^{-3} to 10^{-4} [1]. One of the PCR primers, the one that makes strands complementary to the candidates, is phosphorylated on its $5'$ end so that these strands can later be digested with λ-exonuclease.

A portion of the resulting double stranded product is temporarily reserved; the remainder is used for breeding.

4.5 Breeding Using Single Point Crossover

The portion of double stranded product to be used for breeding is partially digested with *DNase I* to nick (cut only one strand) at random locations about once. The nicked strands are combined with a similar amount of reserved unnicked strands. The mixture is denatured (strands are melted apart) and allowed to reanneal forming new combinations. Many, many possible configurations could be formed. But among these, some will be intact complements of candidate strands annealed to a $5'$ end of a candidate strand including its CG clamp, which enforces alignment. These are featured in the upper right corner of Figure 2. By adding DNA polymerase, the partial candidate strand is extended to a full length legal candidate combining its genetic information with that encoded in the intact strand. The net result is single point crossover. The offspring candidate strand has a block of genetic information from one parent followed by another block from a different parent.

The Sexual PCR (gene shuffling) technique of Stemmer [17,13] is similar to our crossover operation. Sexual PCR would be limited to populations of candidate DNA strands which are very similar and nonuniformly structured. These two properties are needed to ensure alignment of the DNA fragments. We avoid these restrictions on the variety of candidate strands by adding the universal CG sequences at the ends of the candidate strands to enforce alignment. However, our present approach limits us to using single point crossover (which is usually used in genetic algorithms).

Finally, the reaction products are combined with the remainder of the reserved material and complementary strands are digested with λ-exonuclease. Our crossover reactions may produce many products besides offspring candidates, but they will be benign and rarely the same length as a candidate. Purification by length (using denaturing gel electrophoresis) completes the breeding operation.

The new generation is ready to be processed.

Acknowledgment

We want to thank our colleagues Steven Kimbrough, James Laing, and Harvey Rubin at the University of Pennsylvania for stimulating discussions of the Cold War Problem.

References

1. R. Craig Cadwell and Gerald F. Joyce. Mutagenic PCR. In Carl W. Dieffenbach and Gabriela S. Dveksler, editors, *PCR Primer: A Laboratory Manual*. Cold Spring Harbor Laboratory Press, Plainview, New York, 1995.
2. R. Deaton, R. C. Murphy, J. A. Rose, Max H. Garzon, Donald R. Franceschetti, and S. E. Stevens Jr. A DNA based implementation of an evolutionary search for good encodings for DNA computation. In *IEEE International Conference on Evolutionary Computation*, pages 267–271, Indianapolis, Illinois, April13–16, 1997.
3. Alan Dove. From bits to bases: Computing with DNA. *Nature Biotechnology*, 16(9):830–832, September 1998.
4. Stephanie Forrest. *Genetic Algorithms*. Morgan Kaufmann, San Francisco, 1998.
5. David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, 1989.
6. David E. Goldberg, Kerry Zakrzewski, Brad Sutton, Ross Gadient, Cecilia Chang, Pilar Gallego, Brad Miller, and Erick Cantú-Paz. Genetic algorithms: A bibliography. ftp://ftp-illigal.ge.uiuc.edu/pub/papers/IlliGALs/97011.ps.Z, December 1997. IlliGAL Report No. 97011.
7. Rachel Green, Andrew D. Ellington, David P. Bartel, and Jack W. Szostak. *In vitro* genetic analysis: Selection and amplification of rare functional nucleic acids. *Methods*, 2:75–86, 1991.

8. Jörg Heitkötter and David Beasley. The hitch-hiker's guide to evolutionary computation (faq for comp.ai.genetic). Web page at http://research.de.uu.net:8080/encore/www, September 1998.
9. John H. Holland. *Adaptation in natural and artificial systems.* MIT Press, Boston, 1992.
10. L. S. Lerman, K. Silverstein, and E. Grinfeld. Searching for gene defects by denaturing gradient gel electrophoresis. *Trends in Biochemical Sciences*, 172(3):89–93, 1992.
11. Melanie Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, Boston, 1998.
12. D. L. Robertson and F. G. Joyce. Selection *in vitro* of an RNA enzyme that specifically cleaves single-stranded DNA. *Nature*, 344(6265):467–468, March 29, 1990.
13. Willem P. C. Stemmer. DNA shuffling by random fragmentation and reassembly: *In vitro* recombination for molecular evolution. *Proceedings of the National Academy of Science, U.S.A.*, 91:389–391, 1994.
14. Willem P. C. Stemmer. Rapid evolution of a protein by DNA shuffling. *Nature*, 370:389–391, 1994.
15. Willem P. C. Stemmer. The evolution of molecular computation. *Science*, 270:1510–1510, December 1, 1995.
16. Willem P. C. Stemmer. Searching sequence space. *Bio/Technology*, 13:549–553, 1995.
17. Willem P. C. Stemmer. Sexual PCR and assembly PCR. In Robert M. Meyers, editor, *The Encyclopedia of Molecular Biology and Molecular Medicine*, volume 5, pages 447–457. VCH, New York, 1996.
18. Willem P. C. Stemmer, Andreas Crameri, Kim D. Ha, Thomas M. Brennan, and Herbert L. Heyneker. Single-step assembly of a gene and entire plasmid from large numbers of oligodeoxyribonucleotides. *Gene*, 164(1):49–53, 1995.
19. Jack W. Szostak. *In vitro* genetics. *Trends in Biochemical Sciences*, 172(3):89–93, 1992.
20. H. J. Thiesen and C. Bach. Target detection assay (TDA) — A versatile procedure to determine DNA-binding sites as demonstrated on SP1 protein. *Nucleic Acids Research*, 18(11):3203–3209, 1990.
21. C. Tuerk and L. Gold. Systematic evolution of ligands by exponential enrichment — RNA ligands to bacteriophage-T4 DNA-polymerase. *Science*, 249(4968):505–510, August 3, 1990.
22. Erik van Nimwegen, James P. Crutchfield, and Melanie Mitchel. Finite populations induce metastability in evolutionary search. *Physics Letters A*, 229(2):144–150.
23. Erik van Nimwegen, James P. Crutchfield, and Melanie Mitchel. Statistical dynamics of the Royal Road genetic algorithm. *Theoretical Computer Science*, To Appear.
24. Roy Williams. Data powers of ten. Web page at http://www.ccsf.caltech.edu/ roy/dataquan.