DNA-Like Genomes for Evolution *in silico*

Michael West, Max H. Garzon, and Derrel Blain

Computer Science, University of Memphis
373 Dunn Hall, Memphis, TN 38152
{mrwest1, mgarzon}@memphis.edu, derrelb@earthlink.net

Abstract. We explore the advantages of DNA-like genomes for evolutionary computation *in silico*. Coupled with simulations of chemical reactions, these genomes offer greater efficiency, reliability, scalability, new computationally feasible fitness functions, and more dynamic evolutionary algorithms. The prototype application is the decision problem of HPP (the Hamiltonian Path Problem.) Other applications include pre-processing of protocols for biomolecular computing and novel fitness functions for evolution *in silico*.

1 Introduction

The advantages of using DNA molecules for advances in computing, known as biomolecular computing (BMC), have been widely discussed [1], [3]. They range from increasing speed by using massively parallel computations to the potential storage of huge amounts of data fitting into minuscule spaces. Evolutionary algorithms have been used to find word designs to implement computational protocols [4]. More recently, driven by efficiency and reliability considerations, the ideas of BMC have been explored for computation *in silico* by using computational analogs of DNA and RNA molecules [5]. In this paper, a further step with this idea is taken by exploring the use of DNA-like genomes and online fitness for evolutionary computation.

The idea of using sexually split genomes (based on pair attraction) has hardly been explored in evolutionary computation and genetic algorithms. Overwhelming evidence from biology shows that "the [evolutionary] essence of sex is Mendelian recombination" [11]. DNA is the basic genomic representation of virtually all life forms on earth. The closest approach of this type is the DNA-based computing approach of Adleman [1]. We show that an interesting and intriguing interplay can exist between the ideas of biomolecular-based and silicon-based computation. By enriching Adleman's solution to the Hamiltonian Path Problem (HPP) with fitness-based selection in a population of potential solutions, we show how these algorithms can exploit biomolecular and traditional computing techniques for improving solutions to HPP on conventional computers. Furthermore, it is conceivable that these fitness functions may be implemented *in vitro* in the future, and so improve the efficiency and reliability of solutions to HPP with biomolecules as well.

E. Cantú-Paz et al. (Eds.): GECCO 2003, LNCS 2723, pp. 413–424, 2003.

In Section 2, we describe the experiments performed for this purpose, including the programming environment and the genetic algorithms based on DNA-like genomes. In Section 3, we discuss the results of the experiments. A preliminary analysis of some of these results has been presented in [5], but here we present further results and a more complete analysis. Finally, we summarize the results, discuss the implications of genetic computation, and envision further work.

2 Experimental Design

As our prototype we took the problem that was used by Adleman [1], the Hamiltonian Path Problem (HPP), for a proof-of-concept to establish the feasibility of DNA-based computation. An instance of the problem is a digraph and a given source and destination; the problem is to determine whether there exists a path from the source to the destination that passes through each vertex in the digraph exactly once. Solutions to this problem have a wide-ranging impact in combinatorial optimization areas such as route planning and network efficiency.

In Adleman's solution [1], the problem is solved by encoding vertices of the graph with unique strands of DNA and encoding edges so that their halves will hybridize with the end vertex molecules. Once massive numbers of these molecules are put in a test tube, they will hybridize in multiple ways and form longer molecules ultimately representing all possible paths in the digraph. To find a Hamiltonian path, various extraction steps are taken to filter out irrelevant paths, such as those not starting at the source vertex or ending at the destination. Good paths must also have exactly as many vertices as there are in the graph, and each vertex has to be unique within the final path. Any paths remaining represent desirable solution Hamiltonian paths.

There have been several improvements on this technique. In [10], the authors attempt to automate Adleman's solution so that the protocols more intelligently construct promising paths. Another improvement [2] uses reflective PCR to restrict or eliminate duplicated vertices in paths. In [8], the authors extend Adleman's solution, by adding weights associated with melting temperatures to solve another NP-complete problem, the Traveling Salesman Problem (TSP).

We further these genetic techniques by adding several on-line fitness functions for an implementation *in silico*. By rewriting these biomolecular techniques within the framework of traditional computing, we hope to begin the exploration of algorithms based on concepts inspired by BMC. In this case, a large population of possible solutions is evolved in a process that is also akin to a developmental process. Specifically, a population of partially formed solutions is maintained that could react (hybridize), in a pre-specified manner, with other partial solutions within the population to form a more complete (fitter) solution. Several fitness functions ensure that the new solution inherits the good traits of the mates in the hybridization. For potential future implementation *in vitro*, the fitness functions are kept consistent with biomolecular computing by placing the genomes within a simulation of a test tube to allow for random movement and interaction. Fitness evaluation is thus more attuned to developmental

and environmental conditions than customary fitness functions solely dependent on genome composition.

2.1 Virtual Test Tubes

The experimental runs were implemented using an electronic simulation of a test tube, the virtual test tube *Edna* of Garzon et al. [5], [7] which simulates BMC protocols *in silico.* As compared to a real test tube, Edna provides an environment where DNA analogs can be manipulated much more efficiently, can be programmed and controlled much more easily, cost much less, and produce results comparable to real test tubes [5]. Users simply need to create object-oriented programming classes (in C++) specifying the *objects* to be used and their *interactions*. The basic design of the entities that are put in *Edna* represents each nucleotide within DNA strands as a single character and the entire strand of DNA as a string, which may contain single- or double-stranded sections, bulges, and other secondary structures. An unhybridized strand represents a strand of DNA from the 5'-end to the 3'-end. In addition to the actual DNA strand composition, other statistics were also saved such as the vertices making up the strand and the number of encounters since extension.

The interactions among objects in *Edna* are chemical reactions through hybridizations and ligations resulting in longer paths. They can result in one or both reactants being destroyed and a new entity possibly being created. In our case, we wanted to allow the entities that matched to hybridize to each other's ends so that an edge could hybridize to its adjacent vertex. We called this reaction *extension* since the path, vertex, or edge represented by one entity is extended by the path, vertex, or edge represented by the other entity, in analogy with the PCR reaction used with DNA. Edna simulates the reactions in successive *iterations*. One iteration moves the objects randomly in the tube's container (the RAM really) and updates their status according to the specified interactions based on proximity parameters that can be varied within the interactions. The hybridization reactions between strands were controlled by the h-distance [6] of hybridization affinity. Roughly speaking, the h-distance between two strands provides the number of Watson-Crick mismatching pairs in a best alignment of the two strands; strands at distance 0 are complementary, while the hybridization affinity decreases as the h-distance increases. Extension was allowed if the h-distance was zero (which would happen any time the origin or destination of a path hybridized with one of its adjacent edges); or half the length of a single vertex or edge (such as when any vertex encountered an adjacent edge); or, more generally, when two paths, both already partially hybridized, encountered each other, and each had an unhybridized segment (of length equal to half the length of a vertex or edge) representing a matching vertex and edge. These requirements essentially ensured perfect matches along the sections of the DNA that were supposed to hybridize. Well-chosen DNA encodings make this perfectly possible in real test tubes [4].

The complexity of the test tube protocols can be measured by counting the number of iterations necessary to complete the reactions or achieve the desired objective. Alternatively, one can measure the wall clock time. The *number of iterations* taken be-

fore a correct path is found has the advantage of being indifferent to the speed of the machine(s) running the experiment. However, it cannot be a complete picture because each iteration will last longer as more entities are put in the test tube. For this reason, *processor time* (wall clock) was also measured.

2.2 Fitness Functions

Our genetic approach to solving HPP used fitness functions to be enforced online as the reactions proceeded. The first stage, which was used as a benchmark, included checks that vertices did not repeat themselves, called *promise* fitness. This original stage also enforced a constant number of the initial vertices and edges in the test tube in order to ensure an adequate supply of vertices and edges to form paths as needed.

Successive refinements improve on the original by using three types of fitnesses: *extension* fitness, *demand* fitness, and *repetition* fitness, as described below. The goal in adding these fitnesses was to improve the efficiency of path formation. The purpose of the fitnesses implemented here was to bring down the number of iterations it took to find a solution since Edna's speed, although parallel, decreases with more DNA. Toward this goal, we aimed at increasing the opportunity for an object to encounter another object that is likely to lead to a correct path. This entailed increasing the quantity of entities that seemed to lead to a good path (were more fit) and decreasing the concentration of those entities that were less fit. By removing the unlikely paths, we moved to improve the processor time by lowering the overall concentration in the test tube. At this point, the only method to regulate which of its adjacent neighbors an entity encounters is by adjusting the concentration and hence adjusting the probability that its neighbors are of a particular type.

Promise Fitness. As part of the initial design, we limited the type of extensions that were allowed to occur beyond the typical requirement of having matching nucleotides and an h-distance as described above. Any two entities that encountered each other could only hybridize if they did not contain any repeated vertices. It was checked during the encounter by comparing a list of vertices that were represented by each strand of DNA. A method similar to this was proposed in [2] to work *in vitro*. As a consequence, much of the final screening otherwise needed to find the correct path was eliminated. Searching for a path can stop once one is found that contains as many vertices as are in the graph. Since all of the vertices are guaranteed to be unique, this path is guaranteed to pass through all of the vertices in the graph. Because the origin and destination are encoded as half the length of any other vertex, the final path's strand can only have them on the two opposite ends and hence the path travels from the origin to the destination.

Constant Concentration Enhancement. The initial design also kept the concentration of the initial vertices and edges constant. Simply put, whenever vertices and edges encountered each other and were extended, neither of the entities was removed although the new entity was still put into the test tube. It is as if the two original entities were copied before they hybridized and all three were returned to the mixture. The same mechanism was used when the encountering objects were not single vertices or edges but instead were paths. This, however, did not guarantee that the concentration of any type of path remained constant since new paths could still be created. The motivation behind this enhancement was to allow all possible paths to be created without worrying about running out of some critical vertex or edge. It also removed some of the complications about different initial concentrations of certain vertices or edges and what paths may be more likely to be formed. However, this fitness, while desirable and enforceable *in silico* (although not easily *in vitro* just yet) creates a huge number of molecules that made the simulation slow and inefficient.

Extension Fitness. The most obvious paths to be removed are lazy paths that are not being extended. These paths could be stuck in dead-ends where no extension to a Hamiltonian path is possible. To make finding them easier, all paths were allowed to have the same, limited number of encounters without being extended (an initial lifespan) which, when met, would result in their being removed from the tube. If, however, a path was extended before meeting its lifespan then the lifespan of both reacting objects was increased by 50%. The new entity created during an extension received the larger lifespan of its two parents.

Demand Fitness. The concentration of vertices and edges in the tube can be tweaked based on the demand for each entity to participate in reactions. The edges that are used most often (e.g., bridge edges) have a high probability of being in a correct Hamiltonian path since they are likely to be a single or critical connection between sections of the graph. Hence we increase the concentration of edges that are used the most often. Since all vertices must be in a correct solution, those vertices that are not extended often have a disadvantage in that they are less likely to be put into the final solution. In order to remedy this, vertices that are not used often have their concentration increased. The number of encounters and the number of extensions for each entity was stored so a ratio of extensions to encounters was used to implement demand fitness. To prevent the population of vertices and edges from getting out of control, we set a maximum number of any individual vertex or edge to eight unless otherwise noted.

Repetition Fitness. To prevent the tube from getting too full with identical strands, repetition fitness was implemented. It filtered out low performing entities that were repeated often throughout the tube. Whenever an entity encountered another entity, the program checked to see if they encoded the same information. If they did, then they did not extend, and they increased their count of encounters with the same path. Once a path encountered a duplicate of itself too many times, it was removed if it was a low enough performer in terms of its ratio of extensions to encounters.

2.3 Test Graphs and Experimental Conditions

Graphs for the experiments were made using Model A of random graphs [12]. Given a number of vertices, an edge existed between two vertices with probability given by a parameter p= (0.2, 0.4, or 0.6) of including an edge (more precisely, an arc) from the set of all possibilities. For positive instances, one witness Hamiltonian path was placed randomly connecting source to destination. For negative instances, the vertices were divided into two random sets, one containing the origin and one containing the destination; no path was allowed to connect the origin set to the set containing the destination, although the reverse was allowed so that the graph may be connected.

The input to *Edna* was a set of non-crosshybridizing strands of size 64 consisting of 20-oligomers designed by a genetic algorithm using the h-distance as fitness criterion. One copy of each vertex and edge was placed initially in the tube. The quality of the encoding set is such that even under a mildly stringent hybridization criterion, two sticky ends will not hybridize unless they're perfect Watson-Crick complements. In the first set of experiments, the retrieval time was measured in a variety of conditions including variable library concentration, variable probe concentrations, and joint variable concentration. At first, we permitted only paths that were *promising* to become Hamiltonian. Later, other fitness constraints were added to make the path assembly process smarter as discussed below with the results.

Each experiment was broken down into many different runs of the application all with related configurations. All of the experiments went through several repetitions where one or two parameters were slightly changed so that we could evaluate the differences over these parameters (number of vertices and edge density), although we sometimes changed other parameters such as maximum concentration allowed, maximum number of repeated paths, or tube size. Unless otherwise noted, all repetitions were run 30 times with the same parameters, although a different randomly generated graph was used for each run. We report below the averages of the various performance measures. A run was considered unsuccessful if it went through 3000 iterations without finding a correct solution, in which case the run was not included within the averages. We began with the initial implementation as discussed above and added each fitness so that each could be studied without the other fitnesses interfering. Finally we investigated the scalability of our algorithms by adding a population control parameter and running the program on graphs with more vertices.

3 Analysis of Results

The initial implementation provided us with a benchmark from which to judge the fitness efficiency. In terms of iterations (Fig. 1, left) and processor time (Fig. 1, right), the results of this first experiment are not at all surprising. Both measures increase as the number of vertices increases. There is also a noticeable trend where the 40% edge densities take the most time. Edge density of 20% is faster because the graph contains fewer possible paths to search through whereas 60% edge density shows a decrease in time of search because the additional edges provide significantly more correct solu-

tions. It should be noted that altogether there were only two unsuccessful attempts, both with 9 vertices, one at 20% edge density and the other at 40% edge density. This places the probability of success with these randomized graphs above 99%.

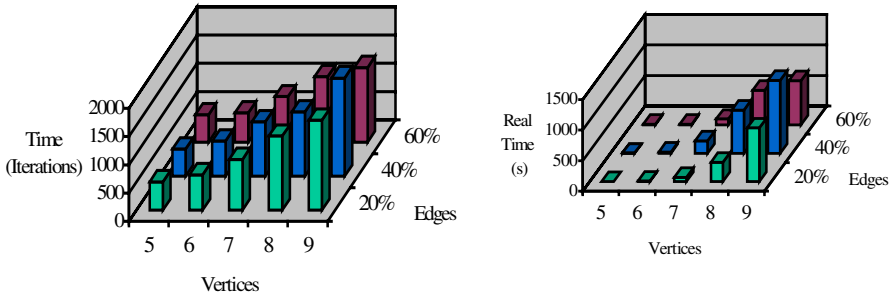

Fig. 1. Successful completion time for the baseline runs (only unique vertices and constant concentration restrictions in force) in number of iterations (left) and processor time (right)

The first comparison made was with extension fitness. The test was done with the initial lifespan set to 150 and the maximum lifespan also set to 150. As seen in Fig. 2, the result cut the number of iterations 54% for 514 fewer iterations on average.

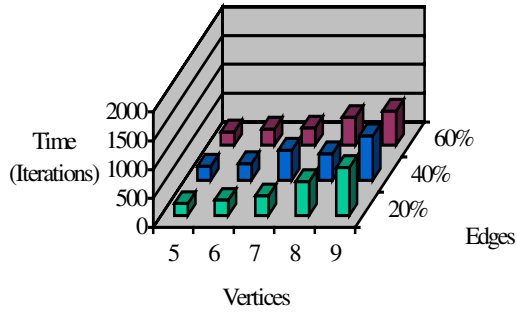

Fig. 2. Successful completion times with extension fitness

From what data is available at this time, demand fitness did not show as impressive an improvement as extension fitness although it still seemed to help. The greatest gain from this fitness is expected to be for graphs with larger numbers of vertices where small changes in the number of vertices and edges will have more time to have a large effect. The number of iterations recorded, on average, can be seen in Fig. 3. The minimum ratio of extensions to encounters before an edge was copied, the edge ratio, was set to .17. The maximum ratio of extensions to encounters below which a vertex

was copied, the vertex ratio, was set to .07. Although it was not measured, the processor time for this fitness seemed to be considerably greater then that of the other fitnesses.

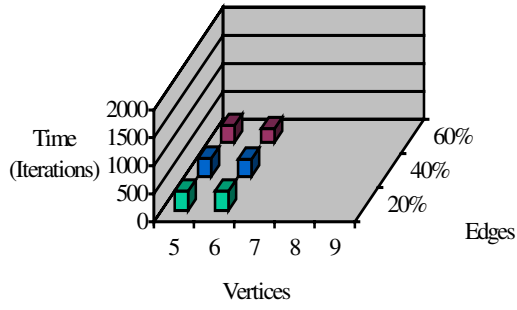

Fig. 3. Successful completion times with demand fitness

The last fitness to be implemented, repetition fitness, provided a 49% decrease in iterations resulting in 465 less iterations on average (Fig. 4). The effect seems to become especially pronounced as the number of vertices increases.

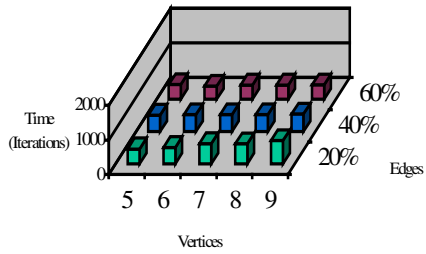

Fig. 4. Successful completion times with the addition of repetition fitness

Finally, we combined all of the fitnesses together. The results can be seen in Fig. 5 in terms of iterations (left) and in terms of processor time (right). Note that the scale for both graphs changed from the comparable ones above. We also increased the radius of each entity from one to two. The initial lifespan of entities was 140, and it was allowed to reach a maximum lifespan of 180. The edge ratio was set to .16, and the vertex ratio was set to .07. For demand fitness, the number of paths allowed was 20, and the removal ratio was .04. All of the fitnesses running together resulted in decreasing the number of iterations by 93% for 880 iterations less, on average. The processor time was cut by 69% saving, on average, 219.90 seconds per run.

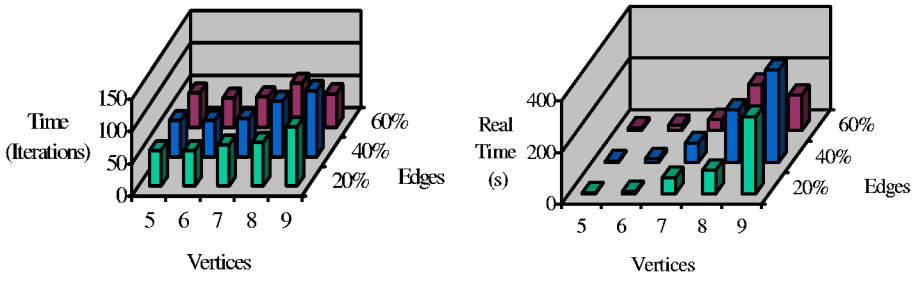

Fig. 5. Successful completion time with all fitnesses running in terms of number of iterations (left) and running time (right)

An important objective of these experiments is to explore the limits of Adleman's approach, at least *in silico*. What is the largest problem that could be solved? In order to allow the program to run on graphs with large numbers of vertices, we put an upper limit on the number of entities present in the tube at any time. Each entity, of course, takes up a certain amount of memory and processing time so this limitation would help keep the program's memory usage in check. Unfortunately, when the limit on the number of entities is reached, the fitnesses, if they are configured with reasonable settings, will not remove very many paths during each iteration meaning that many new paths cannot be added. The dark red line in Fig. 6 shows the results; as the number of entities in the tube reaches the maximum, only a small number of entities are removed, thus not allowing room for many new entities to be created and preventing new, possibly good paths, from forming.

It is necessary to not only limit the population but also to control it. The desired effect would be for the fitnesses to be aggressive as the entity count nears the maximum and reasonable as it falls back down to some minimum. Additionally it would be advantageous for the more aggressive settings to be applied to shorter paths and not longer ones since the shorter paths can be remade much faster then the longer ones. Longer paths have more "memory" of what may constitute a good solution. In order to achieve this, once the maximum number of vertices was reached a population control parameter was multiplied by the values of the extension and repetition fitnesses. The population control parameter is made up of two parts: the *vertex effect*, used on paths with less vertices so that they are more likely to be effected by the population control parameter, and the *entities effect*, used to change the population control parameter as the number of entities in the tube changes. The vertex effect is calculated by:

$$\alpha+(1-\alpha)*(\text{number of vertices in path} / \text{largest number of vertices in any path}) . \quad (1)$$

such that α is configurable. The entities effect is:

$$(\text{max entities} - \text{actual entities in the tube}) / (\text{max entities} - \text{min entities}) . \quad (2)$$

The population control parameter is then calculated using the vertex effect and entities effect with:

$$\text{Entities Effect} + (1 - \text{Entities Effect}) * \text{Vertex Effect} . \quad (3)$$

Using a population control parameter with an α =0.6, maximum vertices of 10,000, and minimum vertices of 6000, the dark blue line (population control parameter) in Fig. 6 shows the number of entities added over time. In order to show that the population control parameter also has the effect of improving the quality of the search, Fig. 6 also shows the length of the longest path, in terms of number of vertices times 100, for both the use of just a simple maximum (in light red) and when using the population control parameter (in light blue).

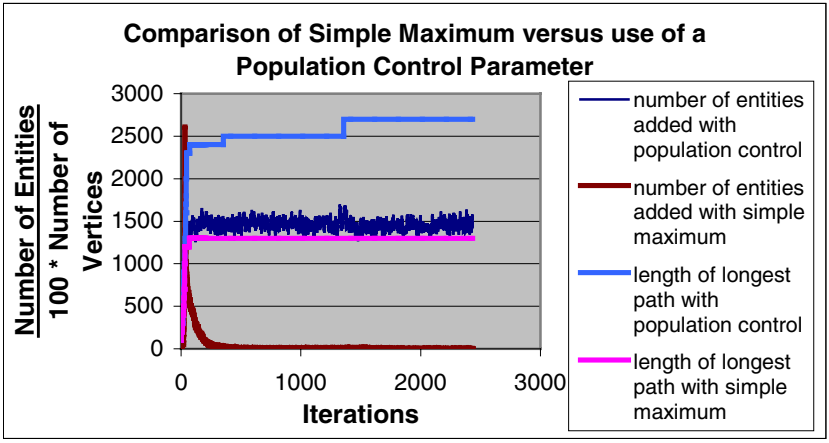

Fig. 6. Comparison of use of a simple maximum versus a population control parameter in terms of both the number of entities added over time and the length of the longest path

Under these conditions, random graphs under 10 vertices can be run with high reliability on a single processor in a matter of hours. The nature of the approach in this paper is instantly scalable to a cluster of processors. Experiments under way may test whether running on a cluster of *p* processors, *Edna* is really able to handle random graphs of about *10*p* vertices, the theoretical maximum.

4 Summary and Conclusions

The results of this paper provide a preliminary estimation of the improved effectiveness and reliability of evolutionary computations *in vitro* that DNA-like genomic representations and environmentally dependent online fitness functions may bring to evolutionary computation. DNA-like computation brings in advantages that biological molecules (DNA, RNA and the like) have gained in the course of millions of years of evolution [11], [7]. First, their operation is inherently parallel and distributable to any number of processors, with the consequent computational advantages. Further, their computational mode is asynchronous and includes massive communications over

noisy media, load balancing, and decentralized control. Second, it is equally clear that the savings in cost and perhaps even time, at least in the range of feasibility of small clusters of conventional sequential computers, is enormous. The equivalent biochemical protocols *in silico* can solve the same problems with a few hundred virtual molecules while requiring trillions of molecules in wet test tubes. Virtual DNA thus inherits the customary efficiency, reliability, and control now standard in electronic computing, hitherto only dreamed of in wet tube computations.

On the other hand, it is also interesting to contemplate the potential to scale these algorithms up to very large graphs when conducting these experiments, either in a real or in virtual test tubes. Biomolecules seem unbeatable by electronics in their ability to pack enormous amounts of information in tiny regions of space and to perform their computations with very high thermodynamical efficiency [13]. This paper also suggests that this efficiency can be brought to evolutionary algorithms *in silico* as well using the DNA-inspired architecture *Edna* used herein.

References

1. Adleman, L.M.: Molecular Computation of Solutions to Combinatorial Problems. In: Science, Vol. 266. (1994) 1021-1024. *http://citeseer.nj.nec.com/adleman94molecular.html*
2. Arita, M., Suyama, A., Hagiya, M.: A heuristic approach for Hamiltonian Path Problem with molecules. In: Proceedings of the Second Annual Genetic Programming Conference (GP-97), Morgan Kaufmann Publishers (1997) 457–461
3. Condon, A., Rozenburg, G. (eds.): DNA Computing (Revised Papers). In: Proc. of the 6th International Workshop on DNA-based Computers. Leiden University, The Netherlands (2000). Springer-Verlag Lecture Notes in Computer Science **2054**
4. Deaton, R., Murphy, R., Rose, J., Garzon, M., Franceschetti, D., Stevens Jr., S.E.: Good Encodings for DNA Solution to Combinatorial Problems. In Proc. IEEE Conference on Evolutionary Computation, IEEE/Computer Society Press. (1997) 267–271
5. Garzon, M., Blain, D., Bobba, K., Neel, A., West, M.: Self-Assembly of DNA-like structures *In Silico.* In Journal of Genetic Programming and Evolvable Machines **4**:2 (2003), in press
6. M. Garzon, P. Neathery, R. Deaton, R.C. Murphy, D.R. Franceschetti,S.E. Stevens, Jr.. A New Metric for DNA Computing. In: J.R. Koza, K. Deb, M. Dorigo, D.B. Fogel, M. Garzon, H. Iba, R.L. Riolo (eds.): Proc. 2nd Annual Genetic Programming Conference, San Mateo, CA: Morgan Kaufmann (1997) 472–478
7. Garzon, M., Oehmen, C.: Biomolecular Computation on Virtual Test Tubes, In: Proc. 7th Int. Meeting on DNA Based Computers, Springer-Verlag Lecture Notes in Computer Science **2340** (2001) 117–128
8. Lee, J., Shin, S., Augh, S.J., Park, T.H., Zhang, B.: Temperature Gradient-Based DNA Computing for Graph Problems with Weighted Edges. In: Hagiya, M. and Ohuchi, A. (eds): Proceedings of the 8th Int. Meeting on DNA Based Computers (DNA8), Hokkaido University, Springer-Verlag Lecture Notes in Computer Science **2568** (2002) 73–84
9. Lipton, R.: DNA Solutions of Hard Computational Problems. *Science* 268 (1995) 542-544.

10. Morimoto, N., Masanori, A., Suyama, A.: Solid Phase Solution to the Hamiltonian Path Problem. In: DNA Based Computers III, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. **48** (1999) 193–206
11. Sigmund, K: Games of Life. Oxford University Press (1993) 145
12. Spencer, J.: Ten Lectures on the Probabilistic Method. In: CMBS **52**, Society for Industrial and Applied Mathematics, Philadelphia (1987) 17–28
13. Wetmur, J.G.: Physical Chemistry of Nucleic Acid Hybridization. In: Rubin, H. and Wood, D.H. (eds.): Proc. DNA-Based Computers III, University of Pennsylvania, June 1997. DIMACS series in Discrete Mathematics and Theoretical Computer Science **48** (1999) 1–23
14. Wood, D.H., Chen, J., Lemieux, B., Cedeno, W.: A design for DNA computation of the OneMax problem. In: Garzon, M., Conrad, M. (eds.): Soft Computing in Biomolecules. Vol. 5:1. Springer-Verlag, Berlin Heidelberg New York (2001) 19–24