A DNA CODIFICATION FOR GENETIC ALGORITHMS SIMULATION

Ángel Goñi, Francisco José Cisneros, Paula Cordero, Juan Castellanos

Abstract: In this paper we propose a model of encoding data into DNA strands so that this data can be used in the simulation of a genetic algorithm based on molecular operations. DNA computing is an impressive computational model that needs algorithms to work properly and efficiently. The first problem when trying to apply an algorithm in DNA computing must be how to codify the data that the algorithm will use. In a genetic algorithm the first objective must be to codify the genes, which are the main data. A concrete encoding of the genes in a single DNA strand is presented and we discuss what this codification is suitable for. Previous work on DNA coding defined bond-free languages which several properties assuring the stability of any DNA word of such a language. We prove that a bond-free language is necessary but not sufficient to codify a gene giving the correct codification

Keywords: DNA Computing, Bond-Free Languages, Genetic Algorithms.

ACM Classification Keywords: I.6. Simulation and Modelling, B.7.1 Advanced Technologies, J.3 Biology and Genetics

Conference: The paper is selected from Sixth International Conference on Information Research and Applications – i.Tech 2008, Varna, Bulgaria, June-July 2008

Introduction

Since the beginning of computation, John Von Neumann held that the different machine models should try to imitate the functions which take place in living beings. Recently, two paradigms of biological inspiration are being applied very satisfactorily to the resolution of problems: neural nets and genetic algorithms. Nowadays, computer scientist try to go a little bit further by working with the same row material the nature does. That is the case of Leonard Adleman who is the pioneer in this field and solved a problem using real DNA strands. In a short period of time DNA based computations have shown lots of advantages compared with electronic computers. DNA computers could solve combinatorial problems that an electronic computer cannot like the well known class of NP complete problems. That is due to the fact that DNA computers are massively parallel [Adleman, 1994].

Despite all the impressive benefits that DNA computations have, they also have several drawbacks. The biggest disadvantage is that until now molecular computation has been used with exact and “brute force” algorithms. It is necessary for DNA computation to expand its algorithmic techniques to incorporate aproximative and probabilistic algorithms and heuristics so the resolution of large instances of NP complete problems will be possible. Without algorithms DNA computing has linear time solving NP-Complete problems but exponential space.

DNA, deoxyribonucleic acid, is the main motor which moves this new computer paradigm. A DNA molecule consists of two single strands twisted. Each strand is a long polymer of bases. Four different bases are presented in DNA: adenine (A), thymine (T), cytosine (C) and guanine (G). It is the sequence of these four bases that encodes information

Leonard Adleman [Adleman, 1994], an inspired mathematician, began the research in this area by an experiment using the tools of molecular biology to solve a hard computational problem in a laboratory. That was the world’s first DNA computer. A year later Richard J.Lipton [Lipton, 1995] wrote a paper in which he discusses, in detail, many operations that are useful in working with a molecular computer. After this moment many others followed them and started working on this new way of computing.

Adleman’s experiment solved the travelling salesman problem (TSP). The problem consists on a salesman who wants to find, starting from a city, the shortest possible trip through a given set of customer cities and to return to its home town, visiting exactly once each city. TSP is NP-Complete (these kinds of problems are generally

believed cannot be solved exactly in polynomial time. Lipton [Lipton, 1995] showed how to use some primitive DNA operations to solve any SAT problem (satisfiability problem) with N binary inputs and G gates (AND, OR, or NOT gates). This is also a NP-Complete problem.

Genetic Algorithms (GA's) are adaptive search techniques which simulate an evolutionary process like it is seen in nature based on the ideas of selection of the fittest, crossing and mutation. GAs follow the principles of Darwin's theory to find the solution of a problem. The input of a GA is a group of individuals called initial population. The GA following Darwin's theory must evaluate all of them and select the individuals who are better adapted to the environment. The initial population will develop thanks to crossover and mutation.

John Holland in 1975 was the first one to study an algorithm based on an analogy with the genetic structure and behaviour of chromosomes. The structure of a basic genetic algorithm includes the following steps. (1) Generate the initial population and evaluate the fitness for each individual, (2) select the best individuals, (3) cross and mutate selected individuals, (4) evaluate and introduce the new created individuals in the initial population. All those steps together are called a generation.

Before generating the initial population, individuals need to be coded. That is the first thing to be done when deal with a problem so that it can be made combinations, duplications, copies, quick fitness evaluation and selection. Nature is a big genetic algorithm in which we are the individuals of the problem. Each of us is coded as a base sequence. We are all different form each other thanks to that sequence. A concrete code must have some characteristics that identify the individual to be more or less qualified.

A GA receives an initial population and after several generations, some codes will disappear and others will appear more often. That is how we can get the solution of the problem without exploring the complete search space.

Previous work on molecular computation for genetic algorithms [J.Castellanos, 1998] shows the possibility of solving optimization problems without generating or exploring the complete search space. A recent work [M.Calviño, 2006] produced a new approach to the problem of fitness evaluation declaring that the fitness of the individual should be embedded in his genes (in the case of the travelling salesman problem in each arch of the path). In both cases the fitness will be determined by the content in G+C (cytosine + guanine) which implies that the fitness of an individual will be directly related with the fusion temperature and hence would be identifiable by spectophotometry and separable by electrophoresis techniques [Macek 1997] or centrifugations.

Gene Characterization

In DNA computing is very important to know that instability of DNA strands can cause undesirable reactions. When facing a problem of any kind, one of the most important things to do is assuring that the data the problem will use is stable. It does not matter whether we are working with DNA or not to carry out this task. During the next sections we will tackle the problem of data encoding in DNA computing problems, concretely, in a simulation of a genetic algorithm with DNA.

The input data for DNA computing must be encoded into single or double DNA strands. Many conditions can cause loss of DNA bases or strand breakage and due to the Watson-Crick complementarity's parts of single DNA strands can bind together forming a double-stranded DNA sequence. Also, several DNA operations like electrophoresis or isopycnic centrifugation, which are absolutely essential for a correct DNA computation, are based on certain characteristics of the DNA strands. Those characteristics can not be altered if we want carry out a problem with DNA. For example, in the case of genetic algorithms, electrophoresis helps us to select the better adapted individuals. That operation is essential for the process of selection of the fittest and we have to take it into account we generating our data: the genes.

We must take care of all these conditions and characteristics so that we can assure the stability of every data of our problem. We can not choose our data randomly making long sequences of bases (A, C, G, and T) because as bigger is our initial data set, more mistakes we will find during the computation.

Taking all into account, we can distinguish two different problems: codification of a stable DNA language and codification of the genes.

Codification of a DNA language.

First of all we have to recall a list of known properties of DNA languages which are free of certain types of undesirable bonds and give a solution as a uniform formal language inequation [Lila Kari, 2004]. That is to create a language from which you can choose any word and be sure of the stability of that DNA molecule.

Codification of the genes.

We want to highlight the possibilities that offer the storage of the information in genes, one word is saved in a different gene, and these genes possess numerous properties (weight, size, ability). Some of the most precise operations that we can realize with the DNA are based on these properties. In our simulation each gene has a different amount of C+G bases. That condition identifies each gene.

When simulating a genetic algorithm, the individuals are formed by several genes and each gene has its own information or characteristic. This characteristic is the content of Cytosine and Guanine they have. An individual would have this aspect:

PCR-primer Np Rep XY RE0 XY RE1 … REn-1 XY Rep Np-1 PCR-primer

XY (gene) is better evaluated as more C+G content

Were the beginning of the individual and the end of it is the almost the same sequence of bases (PCR-primer Np Rep) and the different genes are separated by restriction enzymes (RE).

In this way the individuals are already evaluated. Once they are evaluated we must select them. By isopycnic centrifugation we can select the best suited to their environment. This technique is used to isolate DNA strands basing on the concentration of Cytosine and Guanine they have. The relationship between this concentration and the density (θ) of the strand is:

$$\Theta = 0{,}100[\%(G+C)] + 1{,}658$$

To begin the analysis, the DNA is placed in a centrifuge for several hours at high speed to generate certain force. The DNA molecules will then be separated based primarily on the relative proportions of AT (adenine and thymine base pairs) to GC (guanine and cytosine base pairs), using θ to know that proportion [Gerald Karp, 2005]. The molecule with greater proportion of GC base pairs will have a higher density while the molecule with grater proportion of AT base pairs will have a lower density. In this way the different individuals (different paths or solutions of TSP) are separated and can be easily selected.

With this technique we can identify one gene from the rest. But this work would be useless unless we codify the rest of the gene using a stable DNA language. If not, the genes we define in a genetic algorithm could be altered due to DNA instability.

The first problem can be solved using bond-free languages [Bo Cui, 2007] but those kind of languages do not allow us to codify the genes. We can not choose a word of that language, a sequence of nucleotides, to make a gene because it won't be different from the rest. These methods give a language which assures that any word w_1 of such a language is stable and won't bind together another word w_2 of the same language , but does not assign a proportional weight to the different words of the language. This implies that the genes could not be arranged by weight, preventing from realizing operations with DNA of that properties of the language could take advantage directly.

Every genetic algorithm will need different genes. The genes are the initial data and they must be well defined in order to obtain the correct solution to the problem. However, all the genes would have a similar format. This format must solve both of the problems, codification of a DNA language and codification of the genes. We will use a bond-free language to define most part of the gene but we will add some other characteristic (the fitness of the gene) that must be suitable for a concrete problem. This would be the aspect of a gene:

Bond-Free DNA language (w1) –- Fitness(w3) –- Bond-Free DNA language(w2)

Gene Encoding Language

Words w1, w2, w3: nucleotide sequences

For the language used to surround the fitness we will use a bond-free language. This language assure stability taking care of several conditions like temperature, complementarity, sequences of the same base, concentration of G+C, etc. Any word we take from a bond-free language can be optimal for this task, taking into account that all the words must be different to create different genes. A concrete problem will tell us how long this word must be and how many nucleotides we will use to make this words.

That kind of language could be useful in the resolution of DNA problems that uses 'brute force'. That is the case of Adleman's experiment. But if we try to go further in exploring the possibilities of DNA computing we must use algorithms like, for example, genetic algorithms. To complete the Gene Encoding Language we use a certain characteristic (fitness) which is outside 'stable' languages but are necessary to give a weight to each gene.

This characteristic which makes a gene different from the rest of the genes of the problem is based on the concentration of Cytosine and Guanine they have. Here also the problem will tell us how long this word must be. Anyway, this word should be much smaller than the other parts preserving the stability.

Example

Now we are going to establishing the notation that would allow us to describe the formalizations. Specifically, we define the terms *node, fitness, gene and language of genes*. For this example we will use the well known Travelling Salesman Problem (TSP), see figure 1. If the salesman starts at city X1, and it he distances between every city are known, what is the shortest path which visits all cities and returns to city X1?

We define *Xi* as a node of the graph (a city), and *Wij* as the length or fitness of a archers in the graph (the distance of the road).

We consider a gene as the minimal data unit of our problem and it is denoted by *Yi*. *Yi* is based on three parameters *{ Xi + Wij + Xj}* which are three different nucleotides sequence. The number of different genes in a problem is always the same as the number of arches in the graph.

$$Yi = \{ Xi + Wij + Xj \} \qquad I,j = 1..n$$

Once we codify all the genes, we have an alphabet of genes which can be used to form paths. A path is composed by a sequence of genes and they are possible solutions of the problem. We represent a path by Zi.

We codify *Xi* using a bond-free language, and *Wij* using a special language that preserves the weight of the gene. In this case, that special language gives each gene a different concentration of C+G.

More concentration of guanine and cytosine represent a shorter way. On the other hand, the lack of these nucleotides means that the gene represents a long way between two cities.

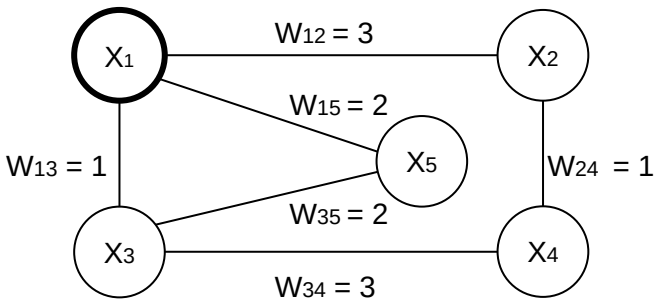

Fig. 1

Fig. 1 represents a map with five cities. Several cities are connected by roads of length W_{ij}. The city X_1 is the initial city. Five nodes are represented $\{X_1, X_2, X_3, X_4, X_5\}$ and only six roads $\{W_{12}, W_{13}, W_{24}, W_{34}, W_{45}, W_{15}\}$.

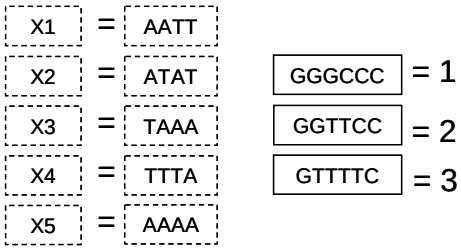

Fig. 2

In figure 2 we assign each city a different nucleotide sequence based on a bond-free language in order to make our data set stable. Also we assign sequences to the roads. The sequences of the roads are not chosen randomly from a certain language. They are chosen by the rule explained before: as shorter a road is, more concentration of C+G that gene would have. Only three values are possible for the roads as they are one, two or three kilometres long.

To represent the cities we use a 4-base sequence and to represent the roads we use a 6-base sequence. A gene is the conjunction of a departure city, a road and the arrival city. In this case there are only six different genes. Those genes are:

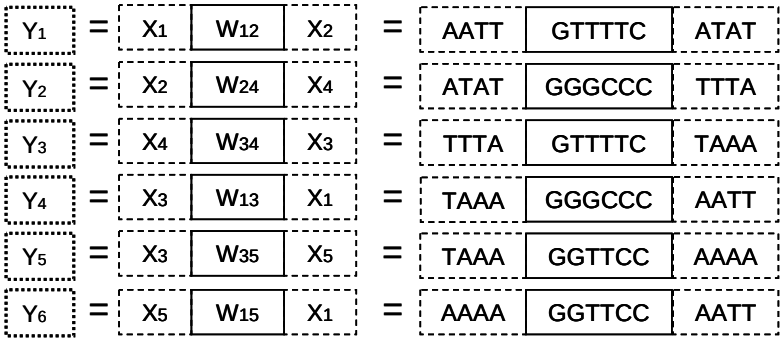

Fig. 3

The possible solutions of the problem are represented by a sequence of genes which we call a path. A possible path for the TSP shown in figure 1 is the next;

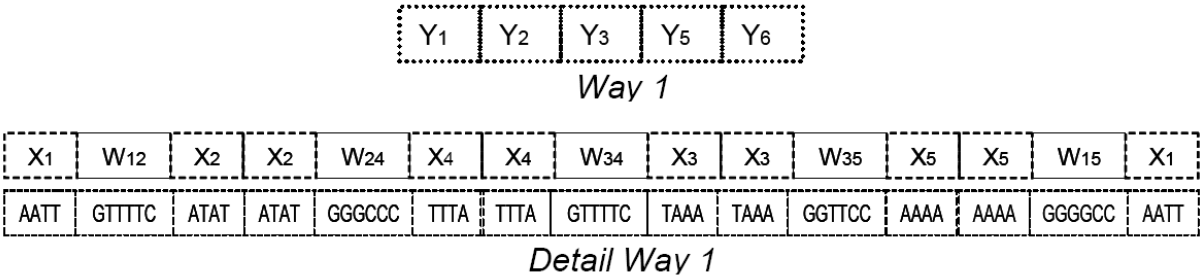

Fig. 4

When all the paths are formed in a soup they do not bind together and they do not disappear because of strand breakage. That is due to the fact that this gene encoding uses a bond-free language. Also, they can be easily identified by several DNA operations like gel electrophoresis which select those paths with more C+G. Using this technique will give us the shortest path of the problem

Conclusion

One of the problems DNA computing has is the instability of the DNA strands. Many conditions can cause loss of DNA bases or strand breakage and because of that the problem will probably be doomed to failure. We propose a codification of data when solving a genetic algorithm with DNA. That is, a codification of the genes. In this codification we take care of the conditions that can cause DNA instability and, at the same time, it is preserved the identity of each gene. As a result of such a process we have a DNA codification forming genes that assures stability of DNA and give a concrete property to each gene.

That property (fitness) which makes a gene different from the rest of the genes of the problem is based on the concentration of Cytosine and Guanine they have. However, the base sequence which forms the fitness is a small part of the DNA strand which represents a gene. For assuring DNA stability we use a bond-free language to complete the gene.

Bibliography

[Adleman, 1994] Leonard M. Adleman. Molecular Computation of Solutions to Combinatorial Problems. Science (journal) 266 (11): 1021-1024. 1994.

[Adleman, 1998] Leonard M. Adleman. Computing with DNA. Scientific American 279: 54-61. 1998

[Lipton, 1995] Richard J.Lipton. Using DNA to solve NP-Complete Problems. Science, 268:542-545. April 1995

[Holland, 1975] J.H.Holland. Adaptation in Natural and Artificial Systems. MIT Press. 1975.

[J.Castellanos, 1998] J.Castellanos, S.Leiva, J.Rodrigo, A.Rodríguez Patón. Molecular computation for genetic algorithms. First International Conference, RSCTC'98.

[M.Calviño, 2006] María Calviño, Nuria Gómez, Luis F.Mingo. DNA simulation of genetic algorithms: fitness computation. iTech 2006. Varna, Bulgary.

[Macek , 1997] Milan Macek M.D. Denaturing gradient gel electrophoresis (DGDE) protocol. Hum Mutation 9: 136 1997.

[Dove, 1998] Alan Dove. From bits to bases; Computing with DNA. Nature Biotechnology. 16(9):830-832; September 1998.

[Mitchell, 1990] Melanie Mitchell. An Introduction to Genetic Algorithms. MIT Press, Boston. 1998.

[Lee, 2005] S.Lee, E. Kim. DNA Computing for efficient encoding of weights in the travelling salesman problem. ICNN&B'05. 2005.

[SY Shin, 2005] SY Shin, IH Lee, D Kim, BT Zhang. Multiobjective evolutionary optimization of DNA sequences for reliable DNA computing. IEEE Transactions, 2005.

[Bo Cui, 2007] Bo cui, Stavros Konstantinidis. DNA Coding using the Subword Closure Operation. DNA 13. 13th Internacional Meeting on DNA Computing

[Kari, 2005] Lila Kari, Stavros Konstantinidis, and Petr Sosík. Bond-Free Languages: Formalizations, Maximality and Construction Methods. DNA10, LNCS 3384, pp. 169–181, 2005

[Konstantinidis, 2007] Bo Cui and Stavros Konstantinidis. DNA Coding using the Subword Closure Operation, DNA13, pp. 65–74, 2007.

[Kari, 2005] Kari, L., Konstantinidis, S., and Sosík, P. (2005) Preventing undesirable bonds between DNA codewords. Lect. Notes Comput. Sc., 3384, 182-191.

[Jonoska] Jonoska, N., Mahalingam, K.: Languages of DNA based code words. In: [4], 58–68

Authors' Information

Ángel Goñi Moreno – Natural Computing Group. Universidad Politécnica de Madrid, Boadilla del Monte, 28660 Madrid, Spain: e-mail: ago@alumnos.upm.es

Fco. Jose Cisneros de los Rios – Natural Computing Group. Universidad Politécnica de Madrid, Boadilla del Monte, 28660 Madrid, Spain: e-mail: kikocisneros@gmail.com

Paula Cordero – Natural Computing Group. Universidad Politécnica de Madrid, Boadilla del Monte, 28660 Madrid, Spain: e-mail: paula.cormo@gmail.com

Juan Castellanos – Natural Computing Group. Universidad Politécnica de Madrid, Boadilla del Monte, 28660. Madrid, Spain. e-mail: jcastellanos@fi.upm.es