I T H E A

International Journal

INFORMATION THEORIES & APPLICATIONS

2009 Volume 16 Number 3

International Journal
INFORMATION THEORIES & APPLICATIONS
Volume 16 / 2009, Number 3

IJ ITA is official publisher of the scientific papers of the members of the ITHEA® International Scientific Society

IJ ITA welcomes scientific papers connected with any information theory or its application.

IJ ITA rules for preparing the manuscripts are compulsory.
The rules for the papers for IJ ITA as well as the subscription fees are given on *www.ithea.org* .
The camera-ready copy of the paper should be received by http://ij.ithea.org.
Responsibility for papers published in IJ ITA belongs to authors.

General Sponsor of IJ ITA is the Consortium FOI Bulgaria (www.foibg.com).

International Journal "INFORMATION THEORIES & APPLICATIONS" Vol.16, Number 3, 2009

Printed in Bulgaria

Edited by the Institute of Information Theories and Applications FOI ITHEA®, Bulgaria,
in collaboration with the V.M.Glushkov Institute of Cybernetics of NAS, Ukraine,
and the Institute of Mathematics and Informatics, BAS, Bulgaria.

Publisher: ITHEA®
Sofia, 1000, P.O.B. 775, Bulgaria. www.ithea.org, e-mail: info@foibg.com

ISSN 1310-0513 (printed) ISSN 1313-0463 (online) ISSN 1313-0498 (CD/DVD)

GENE CODIFICATION FOR NOVEL DNA COMPUTING PROCEDURES

Angel Goni Moreno, Paula Cordero, Juan Castellanos

Abstract: The aim of the paper is to show how the suitable codification of genes can help to the correct resolution of a problem using DNA computations. Genes are the income data of the problem to solve so the first task to carry out is the definition of the genes in order to perform a complete computation in the best way possible. 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. The first problem when trying to apply an algorithm in DNA computing must be how to codify the data that the algorithm will use. With preciseness, the gene formation exposed in this paper allows us to join the codification and evaluation steps in one single stage. Furthermore, these genes turn out to be stable in a DNA soup because we use bond-free languages in their definition. 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. That is due to the fact that selection must be done based on a concrete gene characterization. This characterization can be developed in many different ways codifying what we call the fitness field of the gene. It is shown how to use several DNA computing procedures based on genes from single and double stranded molecules to more complex DNA structures like plasmids.

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

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

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].

Computing using a DNA molecule is a modern approach to a massive parallel paradigm. This concept is based on the work made by Leonard Adleman [Adleman, 1994], where the first implementation of a computer based on DNA operations solved a hard combinatorial problem using deoxyribonucleic acid molecules. The problem which Adleman solved was the well known Hamiltonian Path Problem (HPP). This problem consists of finding, given an undirected graph, weather there is a Hamiltonian Path or not in the graph. A Hamiltonian Path is a path which visits each vertex exactly once. Adleman showed the results of solving a HPP of seven vertexes. This problem belongs to the class of the problems named NP-complete. These kinds of problems present a great complexity and there is no polynomial algorithm known that solves them. 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.

Molecular computing consists of representing the information of the problem with organic molecules and to make them react within a test tube in order to solve a problem. The fundamental characteristics of this type of computations are, mainly, the massive parallelism of DNA strands and the Watson-Crick [Watson-Crick, 1953] complementarity. The speed of calculation, the small consumption of energy and the big amount of information which DNA strands are able to store are the best advantages that DNA computing has. Nevertheless one of the problems is the massive calculation space needed, which limits the size of the problems.

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.

Genetic Algorithms (GA's) are adaptive heuristic search algorithm premised on the evolutionary ideas of natural selection and genetic. The basic concept of GAs is designed to simulate processes in natural system necessary for evolution, specifically those that follow the principles first laid down by Charles Darwin of survival of the fittest. As such they represent an intelligent exploitation of a random search within a defined search space to solve a problem.

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.

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 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.

Physico-Chemical Properties of DNA

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.

In 1950, Erwin Chargaff analyzed the base composition of DNA composition in a number of organisms. He reported that DNA composition varies from one species to another. Such evidence of molecular diversity, which had been presumed absent from DNA, made DNA a more credible candidate for the genetic material than proteins are. [Chargaff, 1950]

Chargaff found that a peculiar regularity in the ratios of nucleotide bases. In the DNA of each species he studies, the number of adenines approximately equaled the number of thymine, and the number of guanines approximately equaled the number of cytosine. In human DNA, for example, the four bases are present in these percentages: A=30.9% and T=29.4%; G=19.9% and C=19.8%. The A=T and G=C equalities, later known as Chargaff's rules, helped Watson and Crick to discover the structure of DNA.

Chargaff's Rules:

(a) [A+G]=[T+C] The sum of the purines (A+G) equals that of the pyrimidines (T+C). The (C+G) and (A+T) ratio equals 1, (A+G)/(T+C)=1.

(b) The molar ratio of adenine to thymine equals 1, (A/T = 1)

(c) The molar ratio of guanine to cytosine equals 1, (G/C=1).And, as a direct consequence of these relationships.

(d) The (C+G) and (A+T) ratio varies from organism to organism.

Different variations can be observed in this relation for each type of organism, from 0.37 to 3,16. This causes that this relation is no good to express the composition of bases of a DNA molecule. The composition of bases of a DNA molecule, usually is expressed by the fraction of bases that are in pairs G.C, is [G] + [C]/[total bases]. This fraction is known like the content in G+C.

There is a linear relation between the content of G+C and the density of the DNA determined in a density gradient. A bigger content of G+C will make stronger the density of the DNA. Due to multiple studies about the density of DNA molecules taken from different organisms and their composition on nitrogenous bases, it has been established an empiric formula that relates the density of flotation (p) to the content of G+C expressed in mole percentage. This formula is the following:

$$p = 1{,}660 + 0{,}00098(G+C)$$

Watson and Crick proposed a structural model for the DNA known as the “Model of the double helix” based on Chargaff work. The structure of a DNA molecule is a double-helix, in which one strand runs from the 5' to the 3' end, and the other one goes in the opposite direction. The two strands interweave, giving the whole molecule a right-handed helical twist. The twist of the helix makes the whole molecules to have a new turn every 34 Amstrong.

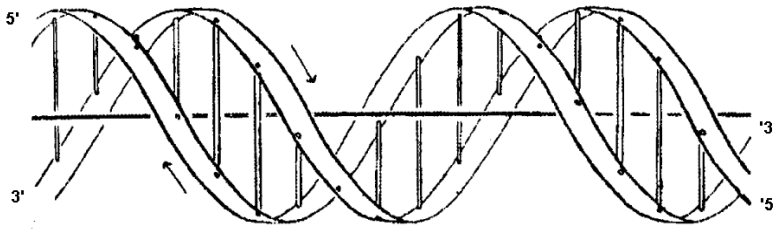


Fig. 1

In a laboratory, the set of DNA molecules is within a test tube, like a 'soup'. Single nucleotides are linked together end-to-end to form DNA strands. Under favorable physical tube conditions two single DNA strand composed by complementary nucleotides pairs, AT and G-C, are join together to form the double helix in a process called hybridization.

The Adenine of a helix matches the Thymine of the complementary helix by creating two hydrogen bonds. Also, the Guanine of a helix matches the Cytosine of the complementary three hydrogen bonds. Therefore, the bases of one strand are united by hydrogen bonds to the bases of the other strand, forming the base pairs A-T and G-C.

Fig. 2. Vertical lines represent bonds (2 bonds A-T and 3 bonds G-C) between complementary nucleotides of two single DNA sequences: 5'-AACCTTACGTTAGG-'3 and 3'-TTGGAATGCAATCC-'5.

In Figure2, the bonds formed by the two single DNA strands are complete due to perfect complementarity of the nucleotides that compose each DNA molecule; nevertheless this does not approach closer to reality of a laboratory. In many cases (Figure 3) it is possible that two parts of molecules will bind together even though some of their corresponding nucleotides are not complementary to each other.

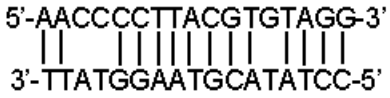


Fig. 3. Some of the corresponding nucleotides of the DNA sequence are not complementary to each other.

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. Such a language is called a bond-free language. The main variables to take into account when describing a bond-free language are the length of the words of such a language (d) and the number of words that compose the language (w). The second variable is very important if we try to give the alphabet the most stability possible. Obviously, if we want to create an alphabet of twenty words (w=20) of five nucleotides each (d=5) we will be able to give them more stability (no complementarities between them) than if the alphabet has forty words (w=40) of the same length. That is due to the fact that there are only 4 nucleotides to combine in sequences (a, c, g, t) and the lower the number w is the bigger Hamming Distance we can get among them. Here, we understand the Hamming Distance between two different words as how complement the words are. If the words are not complement at all, the Hamming Distance would be rather high. There must be the highest distance possible among the words w of the stable alphabet.

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 4. 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.

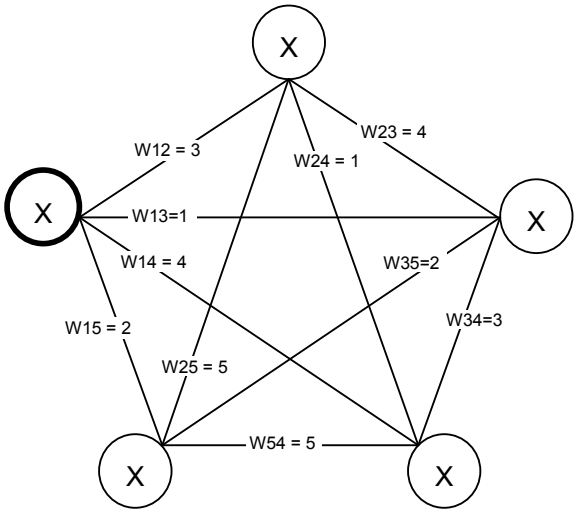


Fig. 4

The graph shown in figure 4 represents a map with five cities. All the 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, as the graph is fully connected, ten roads are defined $\{W_{12}, W_{13}, W_{14}, W_{15}, W_{25}, W_{24}, W_{23}, W_{35}, W_{34}, W_{45}\}$.In this example of TSP the roads (edges of the graph) have values between 1 and 5. The first step when trying to solve the problem with DNA is to assign a concrete DNA word to each data we have.

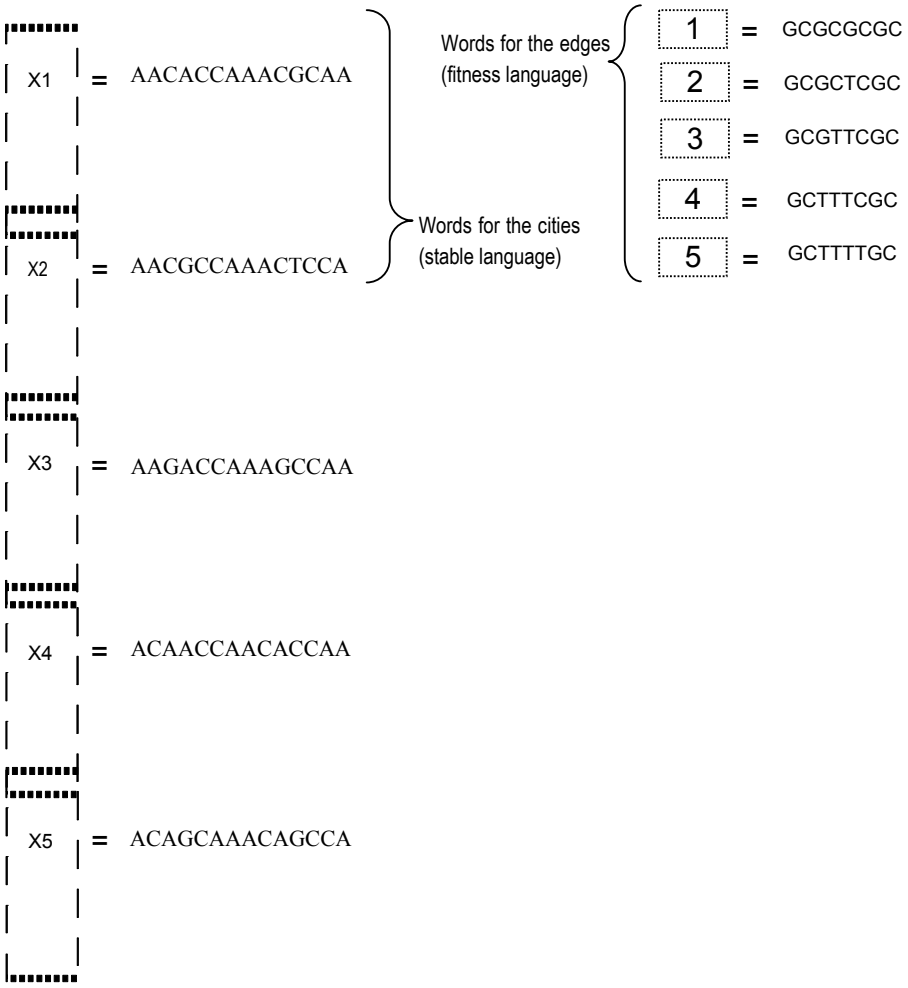


Fig. 5

As it is shown in figure 5 we assign each city a different nucleotide sequence based on a bond-free language in order to make our data set stable. The words have been composed by the software presented by Bo Cui and Stavros Konstantinidis [Bo Cui, 2007] using a constant GC Ratio and a Hamming distance equal to zero. 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 five values are possible for the roads as they are one, two, three, four or five kilometres long. To sum up, we can distinguish between the stable language (vertexes) and the fitness language (edges).

To represent the cities we use a 14-base sequence and to represent the roads we use an 8-base sequence. As long as this is an example developed in-info the sequences result to be long enough but if we carry out the experiment in-vitro is important to notice that with longer chains the results obtained in stability are better. A gene is the conjunction of a departure city, a road and the arrival city. In this case there are ten different genes. Those genes are shown in figure 6.

Y1	=	X1 +	W12 +	X2	=	AACACCAAACGCAA +	GCGTTCGC +	AACGCCAAACTCCA
Y2	=	X1 +	W13 +	X3	=	AACACCAAACGCAA +	GCGCGCGC +	AAGACCAAAGCCAA
Y3	=	X1 +	W14 +	X4	=	AACACCAAACGCAA +	GCTTTCGC +	ACAACCAACACCAA
Y4	=	X1 +	W15 +	X5	=	AACACCAAACGCAA +	GCGCTCGC +	ACAGCAAACAGCCA
Y5	=	X2 +	W23 +	X3	=	AACGCCAAACTCCA +	GCTTTCGC +	AAGACCAAAGCCAA
Y6	=	X2 +	W24 +	X4	=	AACGCCAAACTCCA +	GCGCGCGC +	ACAACCAACACCAA
Y7	=	X2 +	W25 +	X5	=	AACGCCAAACTCCA +	GCTTTTGC +	ACAGCAAACAGCCA
Y8	=	X3 +	W35 +	X5	=	AAGACCAAAGCCAA +	GCGCTCGC +	ACAGCAAACAGCCA
Y9	=	X3 +	W34 +	X4	=	AAGACCAAAGCCAA +	GCGTTCGC +	ACAACCAACACCAA
Y10	=	X4 +	W45 +	X5	=	ACAACCAACACCAA +	GCTTTTGC +	ACAGCAAACAGCCA

Fig. 6

In figure 6 there are described all the possible genes in the TSP of figure 4. These genes are denoted by Y_i where i is a natural number between 1 and 5. These sequences are defined before starting the resolution of the problem as single stranded sequences. The possible solutions of the problem are represented by a sequence of genes which we call a path. The paths are double stranded DNA molecules so it is necessary a process to form those molecules from the single stranded sequences of genes. Such a process is based on the complementarity of bases in order to reach long sequences of genes. This process is shown in figure 7.

In figure 7A) and figure 7B) the process of how two different genes are joined together is detailed in high and low level. The main factor which allows this formation is the presence of linker sequences. We call linker sequences to those complementary sequences of the second part of a gene and the first part of other gene. In the example of figure 7A) genes Y_1 and Y_6 form a double helix thanks to one of this linker sequences. Now we will explain how to form these linkers. Gene Y_1 has a fitness field with value W_{12} so there will be called W_{12}^1 to the first part of that fitness and W_{12}^2 to the second part. The complementary chains of W_{12}^1 and W_{12}^2 are denoted by W'_{12}^1 and W'_{12}^2. It is necessary to conclude the formation of the linker to add the complementary chain of city X_2 (X'_2). The result of pouring those three chains into a DNA soup can be seen in figure 7B).

In the way explained above the different paths *Z* of the problem are generated spontaneously like the one of figure 7C) but it is before this generation when a suitable encoding of the genes can help us very much with the stability of our problem. It is when single stranded molecules are free when we can achieve the goal of making our data stable.

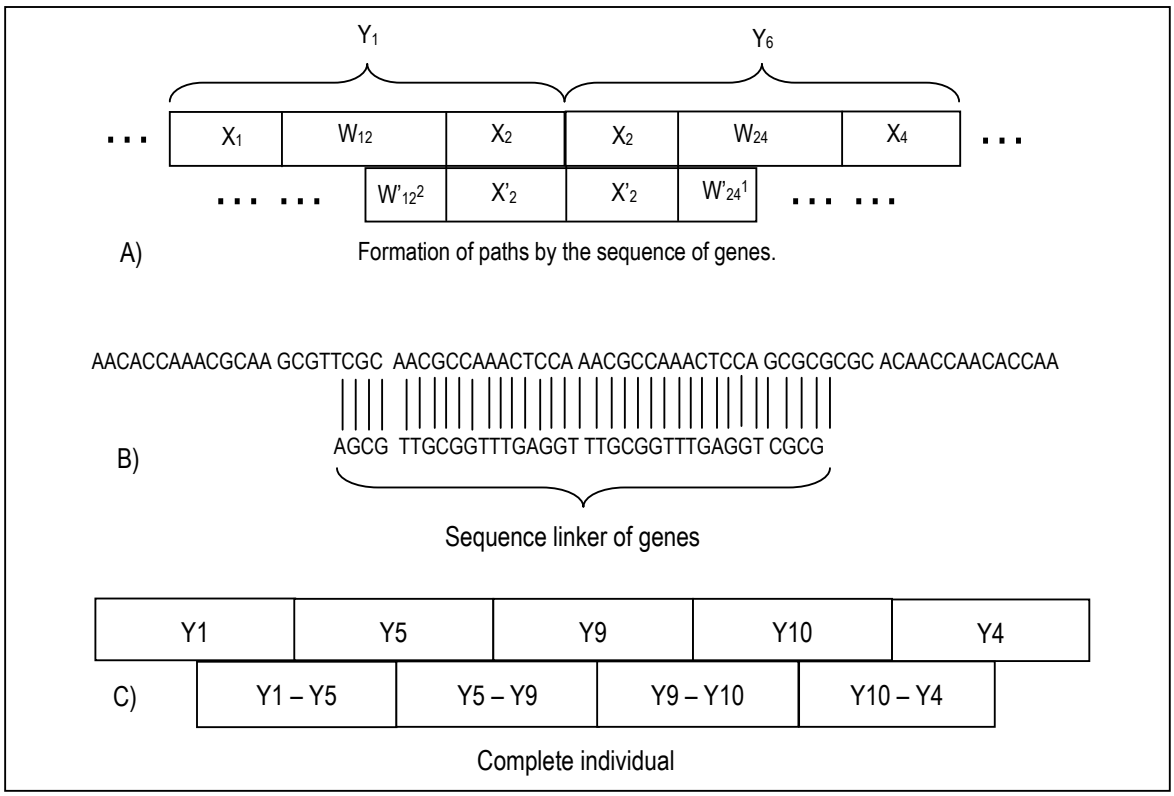


Fig. 7

Irregularities Due to Instability

There must be considered several instability factors before codifying the data (like those explained in previous sections). When we have all our genes ready to be mixed in a single test tube in-vitro in order to begin the computation a big advantage of a good codification can be seen: the uniform resistance to temperature. One of the clues that allow DNA computations is that the average amount of every data is about the same. As we do not have any power over the processes that happen in the test tube we should make sure before starting the experiment that this average value will be uniform during the computation. One of the main characteristics to take into account is the variation of temperature. A bad codification could lead us to the situation shown in the figure 8 where the initial set of data (several copies of two genes) suffer from a variation of temperature giving as a result a different set of data in which the second gene has lost most of its copies.

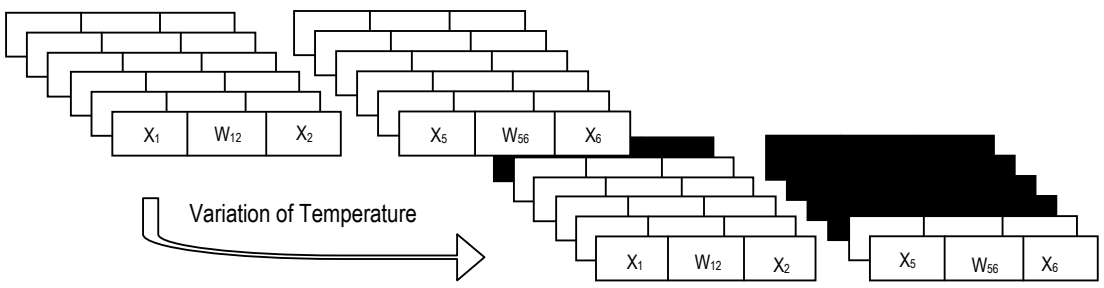


Fig. 8

The other big advantage of the codification using bond-free languages is to preserve the genes of our problem to make unwanted bonds and avoid the renaturalization process between the initial data. There are different forms of renaturalization if we do not use a bond-free language to define the genes.

First of all it is important to notice that a single gene can bond to itself in two different ways. The first one is shown in figure 9A) where both of the cities that surround the fitness of one gene are complementary. This complementarity does not have to be necessarily between the whole chain (X_2 or X_1). A high percentage of complementary bases can be enough to coil the gene and transform it into an invalid gene. The second way of losing a gene is exposed in figure 9B) where the chain of a single city (X_1 in the figure) is long enough to contain complementary sequences inside it. The result of making arbitrary sequences instead of using a bond-free language to define the gene is the loss of data. If this happens frequently the experiment will be doomed to failure.

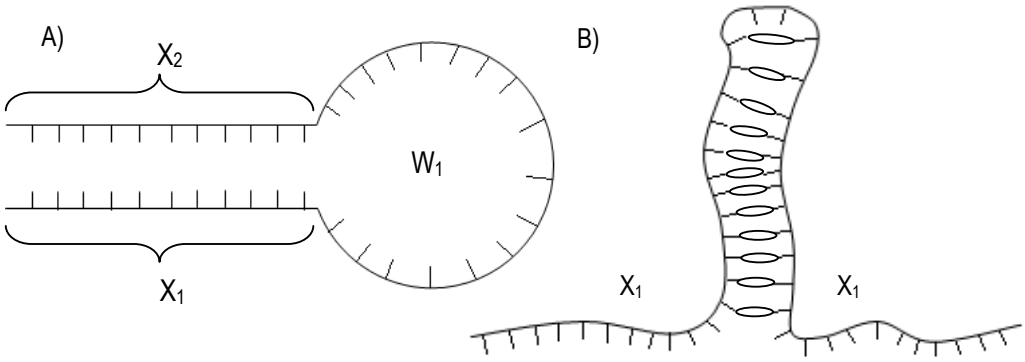


Fig. 9

Apart from the self-complementarity in genes, there also can be complementarities between two different genes like the illustration of figure 10. In this concrete example cities X_2 and X_5 show a partial complementarity between them. The result would be a double helix composed by both of the gens. That is not a correct structure for the resolution of the problem

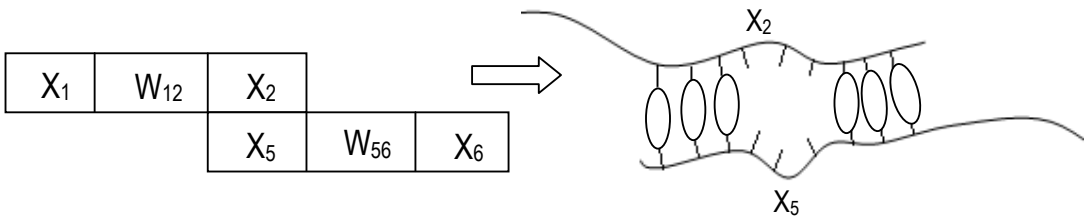


Fig. 10

As it has been explained and illustrated sufficiently, the codification of the initial data is a vital process for DNA computation and bond-free languages are necessary in this task. A probabilistic demonstration of this fact would be very illustrative. Let's call A to the phenomenon of renaturalization between randomly defined sequences and B to the phenomenon of renaturalization between sequences defined by bond-free languages. It is important that gene codification is used in a genetic algorithm simulation (GA) instead of using the brute force (F) applied in previous experiments. We can verify the following statements:

- P (A|GA) << P (A|F). The probability of the phenomenon A in a GA is much lower than using F due to de lower amount of DNA used.
- P (A) >> P (B) as explained in this section.
- P (B|GA) << P (A|GA). The probability of the phenomenon B in a GA is much lower than the probability of the phenomenon A in a GA. The benefits of a good codification are worthy.

Computations Based on Genes

The formation of genes like the ones used in solving the TSP shown in previous sections can lead us to a novel computation paradigm based on the computations of genes. It is important to emphasize that the DNA sequence which is between the bond-free sequences is the main field of the gene in order to complete the computations needed. That field is called the "fitness field" and it is used to carry out the selection step of a genetic algorithm. As we have seen, the condition of selection must be taken into account when the codification of the initial population is done. The reason for such a necessity is the time saved when the codification and evaluation steps of a genetic algorithm are done in a single step.

In the example showed above we considered this fitness field to consist of a specific sequence where a concrete number of G-C base pairs where codified. This codification allowed us to identify a concrete gene by using different techniques like isopycnic centrifugation which can separate molecules based primarily on the relative proportions of AT to GC base pairs. But this fitness field can be codified using a lot of different strategies to perform de selection step. For instance, there are several molecular procedures which use a specific sequence of antibiotic resistance to select genes. Imagine we have only three kinds of genes, those with an antibiotic resistance 1 (class 1), those with an antibiotic resistance 2 (class 2) and finally those with an antibiotic resistance 3 (class 3). If we ant to select the genes of class 1 then we only have to pour a solution of antibiotic 1 to the soup. The result would be that those genes without the resistance 1 will die and we will get in the soup only the genes wanted.

Gene oriented computations lead us to a different computation algorithm from the algorithms showed in the example of TSP above. This algorithm would be reflected in the illustration of figure 11.

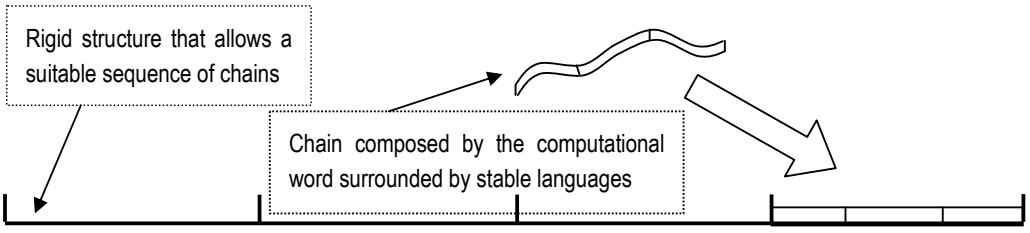


Fig. 11

This computation based on a rigid structure tries to give an answer to different problems (also NP-problems) like the Hamiltonian Path Problem (HPP). Instead of pouring all the genes in a soup and let them react guided by their codification, we codify a concrete sequence called rigid structure which represents the solution of the problem. This method is oriented to those kinds of problems in which we have to search for the solution in order to give a positive answer (YES) if the solution is achieved or a negative (NOT) if it is not. The positive answer would mean that the rigid structure is completed with genes.

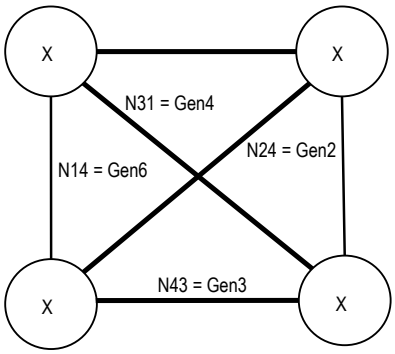
In the next section we will explain how to use more difficult DNA structures to perform these computations.

Computation by Using Plasmid

In this section it will be explained, in general, a possible computational method of one step included in the resolution of a complete algorithm that solves the well-known Hamiltonian Path Problem (NP-Complete Problem).

In graphs theory, an instance of the Hamiltonian Path Problem tries to determine if there is a cycle in a concrete graph that fulfills certain conditions. A Hamiltonian cycle is that cycle which passes through all the nodes or vertices of the graph exactly once. When we talk about directed graphs with a different weight on each edge, the Hamiltonian cycle (Cy) with a smaller cost (Co = $\sum$ edge_weigh $\forall$ edges of Cy) is also known as the solution to the Traveling Salesman Problem.

As this investigation advance we part from a codified population of ways using plasmids or vectors, as we will see next. This population must have been created following successive steps of a determined algorithm in such a form that finally we could get all the possible combinations of edges of the graph (figure 4) in order to obtain all the possible solutions of the problem. Concretely, we will focus the problem on the graph illustrated in the previous section (figure 4). In the initial set of paths we could find valid, invalid, complete and incomplete solutions. That is why the procedure explained next is so important. It is based on making possible the detection of an existent Hamiltonian way.



Each problem is represented on a certain graph. Not all the graphs will have a Hamiltonian Path (figure 13B). The purpose of final algorithm to solve HPP is to tell us if there is a solution in our graph or not.

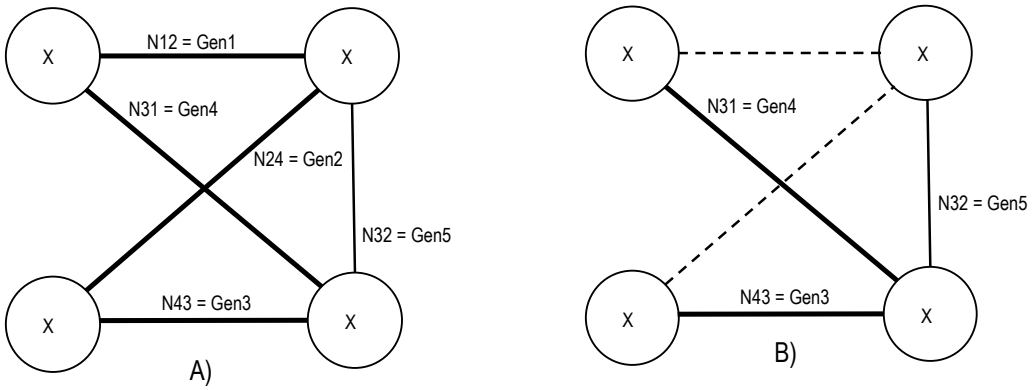


Fig. 13. The graph 13A would have a positive solution. A negative solution would be given under graph 13B.

The theoretical development of the presented proposal is based on a specific codification of the existing edges in the graph of the problem using nucleotides sequences which codify concrete genes. These genes will be expressed in fluorescents proteins when transcribed. Each edge of the graph will be codified as shown in figure 14. The fitness field representing the fluorescent gene is surrounded by bond-free sequences which represents half of the vertex.

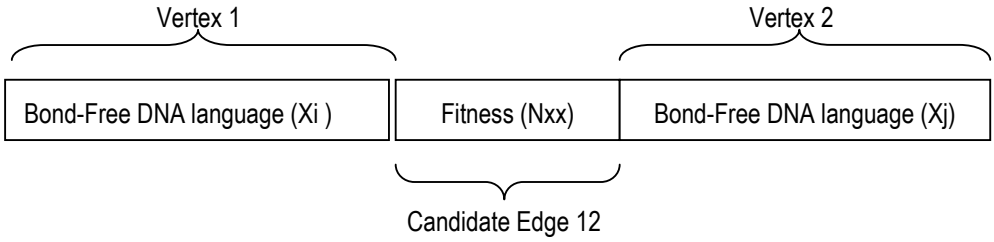


Fig. 14

Next there is a table which represents the correspondence between the existing edges in the graph (figure 13A) and the genes associated. The genes represented in the explanatory example are normal nucleotide sequences chosen randomly in order to simplify and clarify the method applied. However, there could be taken into account several possibilities to select the family of genes whose expression result to be fluorescent proteins to carry out the experiment. In this case we could design the experiment with proteins of the type GFP (the green fluorescent protein) and work with the variants that GFP produces like the RFP protein (red fluorescent protein) or YFP (yellow fluorescent protein).

An example of edge codification:

EDGE CODIFICATION	*FITNESS GENES*
N12: B-F DNA language (X1) - AAT-TGG-*CGA*-TTA-AAC - B-F DNA language (X2)	TTAACCG*C*TAATTTG AATTGGC*G*ATTAAAC
N24: B-F DNA language (X2) - TTA-CCA-*TGC*-TGA-CCC - B-F DNA language (X4)	AATGGTA*G*GACTGGG TTACCAT*C*CTGACCC
N43: B-F DNA language (X4) - GGT-CAG-*CTG*-ACG-TCA - B-F DNA language (X3)	CCAGTCG*A*CTGCAGT GGTCAGC*T*GACGTCA
N35: B-F DNA language (X3) - AGT-CGA-*TTC*-GAA-GGC - B-F DNA language (X5)	TCAGCTA*A*GCTTCCG AGTCGAT*T*CGAAGGC
N51: B-F DNA language (X5) - CGT-AGC-*TGA*-TCGA-TCT - B-F DNA language (X1)	GCATCGA*C*TAGCTAGA CGTAGCT*G*ATCGATCT
N45: B-F DNA language (X4) - GGC-TGA-*TCG*-TAA-AGT - B-F DNA language (X5)	CCGACTA*G*CATTTCA GGCTGAT*C*GTAAAGT
N14: B-F DNA language (X1) - CCG-TAG-*CTG*-ATC-GTC - B-F DNA language (X4)	GGCATCG*A*CTAGCAG CCGTAGC*T*GATCGTC

The possible solutions of the problem come expressed into vectors or plasmids as it is illustrated in figure 14.

Example of edges codification after the formation of the solution-set of the problem:

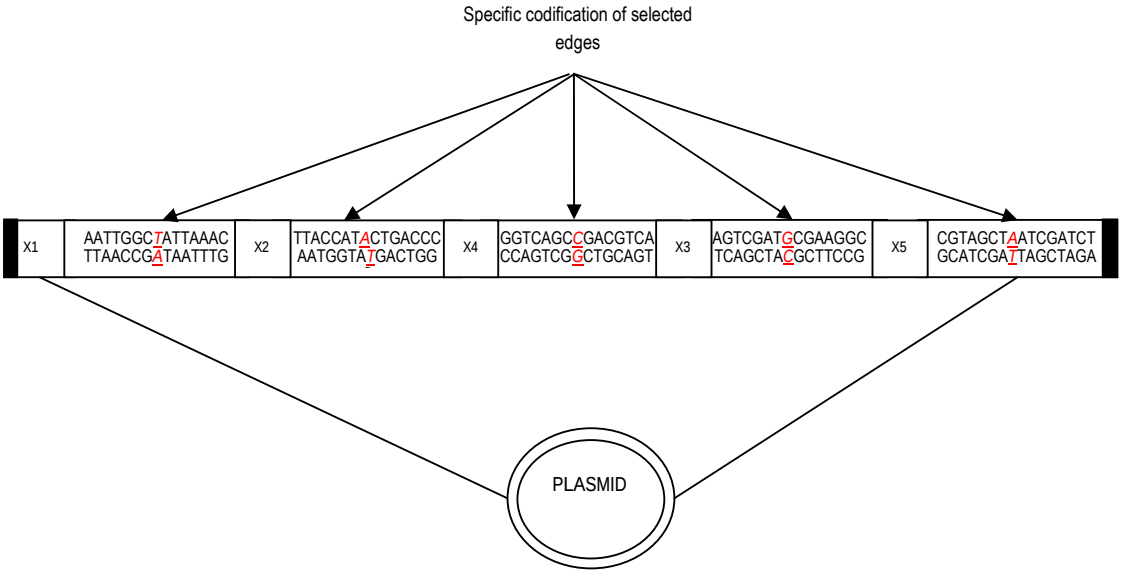


Fig. 14

Each plasmid contains sequentially the possible vertexes which form the Hamiltonian path we are looking for. As it has been mentioned before, within the set of plasmids there will be incomplete sequences and also nonvalid chains. For that reason the valid solution must have been searched. The proposal presented in this paper is based on this situation. In figure 14 is shown the plasmid of the initial set which contains the solution sequence to our problem in order to facilitate the later understanding of the theoretical development.

It is necessary to emphasize an important detail in the codification of the plasmid candidates. By observing figure 14 it can be seen that between each pair of vertexes the candidate edges are codified. These edges are codified carefully with a constant length of N-nucleotides and for the case of the existing edges in the initial graph a modification of the central nucleotide of the sequence which corresponds to the fitness gene take place. This modification consists on:

- Central sequence of the original edge N12: AATTGGCGATTAAAC
- Central sequence of the edge codified in the plasmid N12: AATTGGCTATTAAAC
  TTAACCGATAATTTG

It has been already explained the formation of the candidates which could be a solution of the problem. Next it is detailed the rest of the method which detects a correct solution. This technique is based on running n-cycles of mutagenesis in such a way that the solutions that contain the sequence of vertexes corresponding to the Hamiltonian Path will mutate until reach the moment in which the union between their vertexes turns into the fitness gene. This procedure is shown in figure 15 where a cycle of mutagenesis for the N12 edge is done. In the case that the edges do not exist in the initial graph this operation will not be carried out so the plasmid won't get the corresponding fitness field.

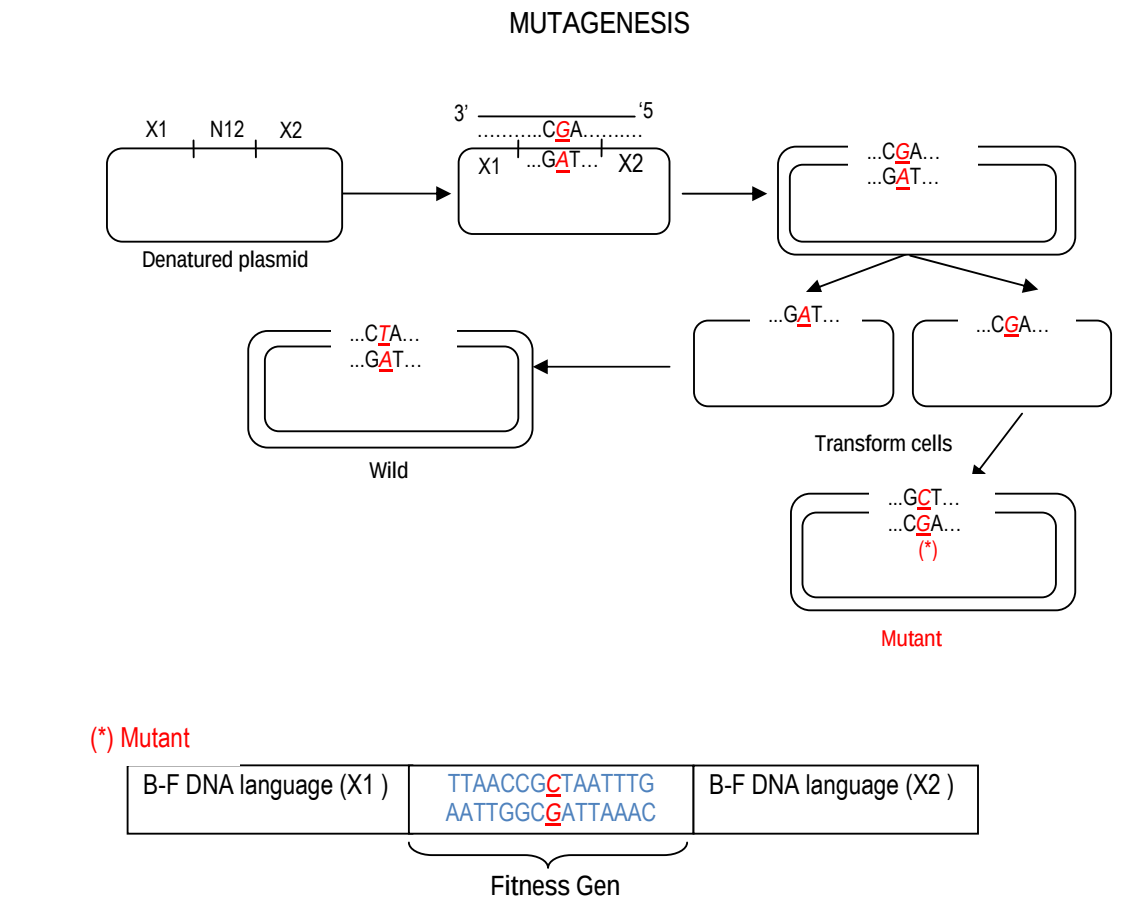


Fig. 16

When the last step has already been done, if we selected only the plasmids with the fluorescence conferred by the five genes of specific representations corresponding to the edges of the graph, we could affirm that the HPP proposed has a positive solution. Remember we are facing the problem of figure 13A.

In the same way, for that case in which a Hamiltonian Path does not exist in the given graph we would get a negative solution. That is due to the fact that we could notice after putting sequentially those bacteria which contain the vectors (once for each edge or protein) under fluorescent light of the absorption wavelength of the protein at issue that these ones do not emit.

In order to carry out these processes efficiently it is especially important to emphasize the use of bond-free languages for the codification of vertexes and edges providing stability and assuring the expected results during the formation and mutation steps of possible solutions of our problem.

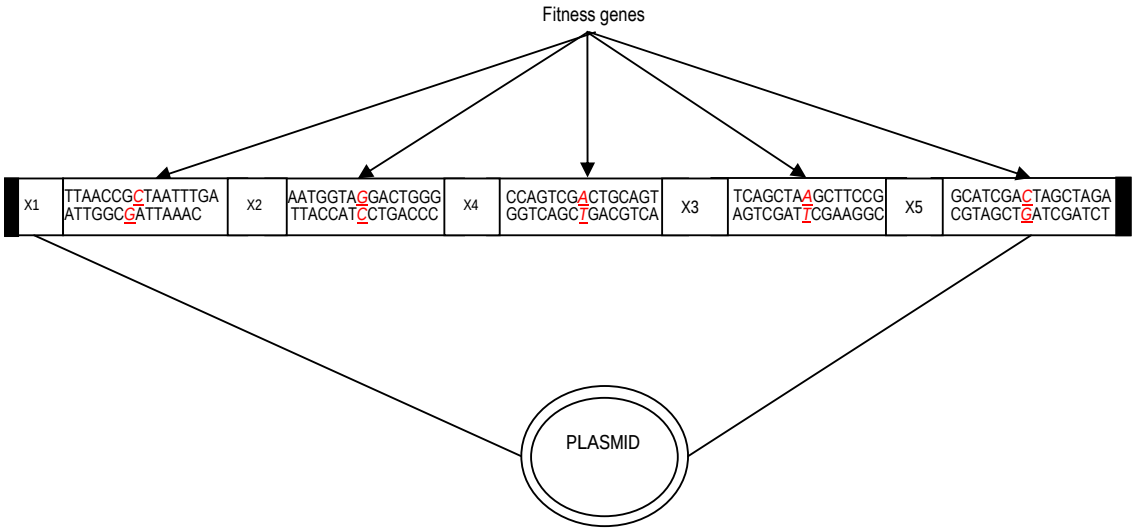


Fig. 17

Conclusion

In conclusion, we hope to have given a clarification of what gene-oriented computations are and how important is to code the gen in an appropriate way related to the problem to solve. There are as many gen codifications as problems are, but they have all the same structure which has been described in this paper. The general fields that all possible gens have in common are those that represent the stable sequences of the gen. 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. In the codification of the stable sequences of the gen we take care of the conditions that can cause DNA instability by means of bond-free languages.

However, the most important field in the gen is what we call the fitness field which is used to preserve the identity of each gene. As a result we propose a DNA codification forming genes that assures stability of DNA and give a concrete property to each gene. That property, 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 in the resolution of the TSP proposed. Furthermore, it is explained how to perform different DNA computations based on other kinds of fitness fields like antibiotic resistances. To sum up, the codification of the minimal computational unit (the gen) presented in this paper represents a strong structure that allow us to solve problems in a powerful and stable way.

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.

[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

[Yurke, B.], Turberfield, A.J., Mills, A.P., Simmel, F.C., and Neumann, J.L. 2000 August 10. A DNA-fuelled molecular machine made of DNA. Nature 406: 605-608

[Head, 2000] T. Head, G. Rozenberg, R.S. Bladergroen, C.K.D. Breek, P.H.M. Lommerse, H.P. Spaink. Computing with DNA by operating on plasmids. BioSystems 57 (2000) 87-93.

[Wakabayashi, 2005] Kenichi Wakabayashi, Masayuki Yamamura .Natural Computing, A Design for Cellular Evolutionary Computation by using Bacteria Vol. 4, No. 3. (September 2005), pp. 275-292.

[Lui, 2005] Wenbin Liu, Xiangou Zhu, Guandong Xu, Quiang Zhang and Lin Gao. A DNA based evolutionary algorithm for the minimal set cover problem. Volume 3645/2005, Advances in Intelligent Computing, 2005

Authors' Information

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

Paula Cordero – Natural Computing Group. Universidad Politécnica de Madrid, Boadilla del Monte, 28660 Madrid, Spain: e-mail: p.cordero@alumnos.upm.es

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