SOLVING TRAVELLING SALESMAN PROBLEM IN A SIMULATION OF GENETIC ALGORITHMS WITH DNA

Angel Goñi Moreno

Abstract: In this paper it is explained how to solve a fully connected N-City travelling salesman problem (TSP) using a genetic algorithm. A crossover operator to use in the simulation of a genetic algorithm (GA) with DNA is presented. The aim of the paper is to follow the path of creating a new computational model based on DNA molecules and genetic operations. This paper solves the problem of exponentially size algorithms in DNA computing by using biological methods and techniques. After individual encoding and fitness evaluation, a protocol of the next step in a GA, crossover, is needed. This paper also shows how to make the GA faster via different populations of possible solutions.

Keywords: DNA Computing, Evolutionary Computing, Genetic Algorithms.

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

Introduction

In a short period of time DNA based computations have shown lots of advantages compared with electronic computers. DNA computers can 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]. However, 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.

On the other hand there are genetic algorithms (or short GA) which are categorized as global search heuristics and use techniques inspired by evolutionary biology [Holland, 1975]. It seems to be perfect to combine DNA computing and GAs.

Previous work on molecular computation for genetic algorithms [J.Castellanos, 1998] show the possibility of solving optimization problems without generating or exploring the complete search space and give a solution to the first step to be done in a GA, the coding of the population and the evaluation of individuals (fitness). A recent work [M.Calviño, 2006] produced a new approach to the problem of fitness evaluation saying 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].

In this paper the crossover (also called recombination) of DNA-strands has been resolved satisfactorily by making a crossover operator suitable for DNA computing and its primitive operations. This crossover operator is used in the simulation of the travelling salesman problem (TSP) with both genetic algorithm and DNA computing continuing the work previously done about the coding of information. The Lipton [Lipton, 1995] encoding is used to obtain each individual coded by a sequence of zeros and ones, and when using DNA strands this information is translated into the four different bases that are presented in DNA – adenine (A), thymine (T), cytosine (C), and guanine (G).

Molecular Computing

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.

Here a short description of the tool box of techniques for manipulating DNA is provided so that the reader can have a clear intuition about the nature of the techniques involved.

- Strands separation:
  - Denaturation of DNA strands. Denaturation of DNA is usually achieved by heat treatment or high pH, which causes the double-stranded helix to dissociate into single strands.
  - According to their length using gel electrophoresis. This technique is used to push or pull the molecules through a gel matrix by applying an electric current. The molecules will move through the matrix at different rates depending on their size.
  - According to a determinated subchain using complementary probes anchored to magnetic beads.
- Strands fusing:
  - Renaturation. If the soup is cooled down again, the separated strands fuse again.
  - Hybridization. Originally it was used for describing the complementary base pairing of single strands of different origin (e.g., DNA with RNA).
- Cutting DNA. Using restriction enzymes which destroy internal phosphodiester bonds in the DNA.
- Linking (pasting) DNA. Molecules can be linked together by certain enzymes called ligases.
- PCR mutagenesis. To incorporate the primer as the new (mutant) sequence.

Genetic Algorithms

Genetic Algorithms 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 [Holland, 1975] was the first one to study an algorithm based on an analogy with the genetic structure and behavior of chromosomes. Genetic algorithms has been widely studied and experimented. 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 individuals, (3) cross and mutate selected individuals, (4) evaluate and introduce the new created individuals in the initial population. In that way, the successive generation will become more suited to their environment.

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.

Get the solution faster: island model

The next method (island model) is followed in order to get the solution faster. It represents an upgrade of a simple genetic algorithm. In this model, the initial population is duplicated as many times as we want creating a fully connected graph. Each of the populations exchanges a portion of individuals (*m*) with the others. The graph is presented on fig.1.

Each of the populations exchanges a portion of individuals (*m*) with the others. This process is repeated every generation until a common equilibrium frequency is reached. By this method the speed of the GA is increased exponentially. Next figure shows how quickly populations converge on the same allele frequency when 10% (m = 0,1) of each population is made up of immigrants from the other populations.

In the figure 2 we see the change of allele frequency when using five subpopulations exchanging migrants at the rate m= 0,1 per generation. It is reached a final frequency (p) for all subpopulations which depart from the initial frequency (p_0) [Hartl and Clark, 1989].

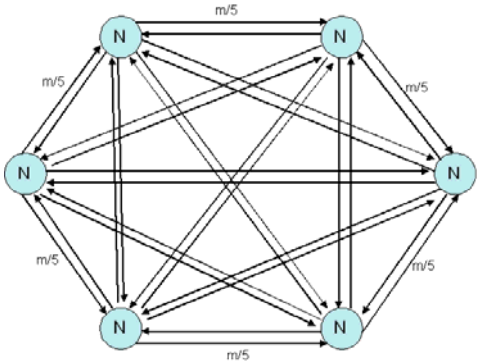

Fig. 1. Fully connected graph
for an initial population of N individuals.
The initial population is cloned six times.

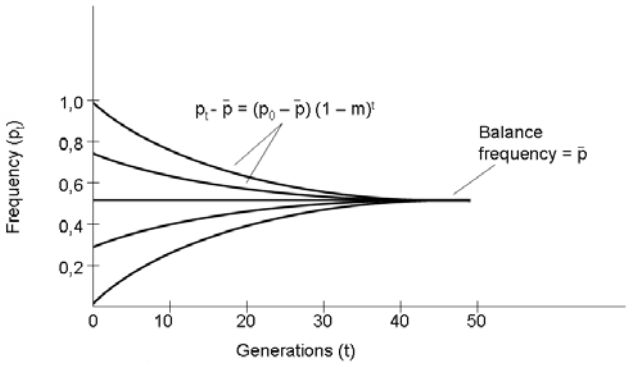

Fig. 2. Change of allele frequency using 5 subpopulations exchanging migrants at the rate m= 0,1 per generation

Fitness and selection

When solving TSP, each possible solution to the problem (each individual of the initial population) is represented in a single DNA strand [M.Calviño, 2006]. Their form is the next:

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

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 greater 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. See figure 3 to understand how centrifugation works.

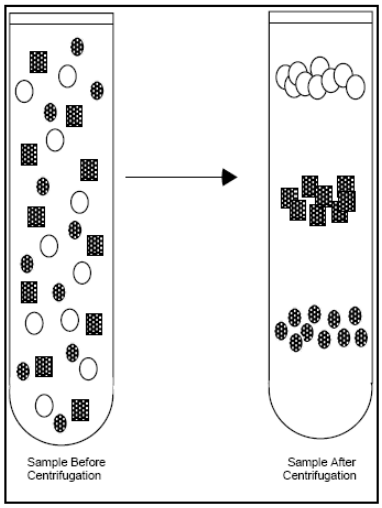

Fig. 3. Isopycnic centrifugation.

Crossover

As it has already been said the first thing to do when a problem is presented is the codification of individuals. In our case the problem is TSP. How can we code the paths? A possible solution was provided in a previous work [J.Castellanos, 1998] giving a representation of individuals like a DNA-strand for each path. This encoding is based on a sequence of genes each one represents an arch between two cities. Here the fitness would be an extra field placed at the beginning of the DNA-strand and its length is proportional to the value it represents (in fact it depends on the problem. For example in the travelling salesman problem, TSP, the length should be inversely proportional to the value of the path). Between the DNA code belonging to the genes a cutting site for a restriction enzyme will be inserted. The final encoding for a path is:

PCR Primer Np REp Fitness RE1 gene REn-1 …… RE0 gene REp Np PCR Primer

A recent work approached individual encoding by eliminating the field fitness. In that case, the fitness is embedded in the genes. The advantage of this work is that when all the individuals of the population are generated, there is no need to evaluate them because they have already been evaluated by themselves. After solving the problem of the selection by adding a specific field in each gene which tells the distance between both cities, it is necessary to see if the same format of the strands is valid in the next step of de GA, crossover.

First of all let's try to solve this step using the technique "cut and splice" like it is done in vivo. A single cut point is selected and after cut we splice both ends. An example is shown in figure 4 with two different chromosomes.

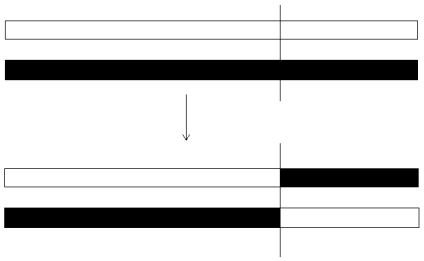

Fig 4

Solving TSP crossover with cut and splice:

	Example 1	Example 2
Parent 1 (P1)	AB BC CD | DE	AB BC | CD DE
Parent 2 (P2)	AD DB BC | CE	AD DB | BC CE
	↓	↓
Son 1	AB BC CD CE	AB BC BC CE
Son 2	AD DB BC DE	AD DB CD DE

Table 1

The results show that this method must be discarded because all the sons it produces are invalid. Obviously, it has no sense to create a son that contains a specific city twice.

I proceed then to apply a different protocol called "order crossover" (OX). Adjacency information in the total ordering is important and this crossover preserves relative ordering. Two parents are selected between the population then a random mask is selected (with 0's and 1's which are chosen randomly with the same probability both bits). During the first step, the sons are filled with the genes of the parents which the mask allows. To complete the sons, we put the genes missing in S1 in the order they appear in P2 and the same with S2. An example is shown in Table 2.

Suitable mask for order crossover

Let's try to apply OX to TSP. It is remembered that each gene represents an arch between two different cities. Like a first attempt it is used the mask: 1001 (one bit for gene).

As we see in Table 3, the result of the computation of OX using that mask is invalid because the genes CD and DB do not even exist in P2 so S1 cannot be completed. By using this mask we will never get valid individuals. Now, OX is computed with the mask 10 01 01 11, using two bits for each gene. Each bit represents a city. As usually, the mask is chosen randomly (every bit).

Once again the mask is not correct. In the example (Table 4) we can see how in the third gene of P1 (DB) there is missing only one city, city D. That has no sense at all, because the second city of the second gene must be the same as the first city of the third gene. That give us the idea of how the definitive mask should be. Let's try now with the mask 10 01 10 01. In this mask we choose randomly the pair of bits that represents the same city, for example we choose if the second bit of the first gene (10) and the first bit of the second gene (01) are 0 or 1 both of them but not different.

In this example (Table 5) the sons are correct. So that is the suitable mask. In order to find less invalid individuals we force the mask to one last rule: the first bit and the last must be 1 because in TSP the first city we visit and the last one must be always the same.

P1	A C D B E
P2	A D B C E
Mask	1 0 1 0 1
S1 (C, B missing)	A - D - E
S2 (D, C missing)	A - B - E

S1 (B before C in P2)	A B D C E
S2 (C before D in P1)	A C B D E

Table 2

P1	AC CD DB BE
P2	AD DB BC CE
Mask	1 0 0 1
S1 (missing CD, DB)	AC - - BE

Table 3

P1	AC CD DB BE
P2	AD DB BC CE
Mask	10 01 01 11
S1 (missing C(twice),D)	A- -D -B BE

Table 4

P1	AC CD DB BE
P2	AD DB BC CE
Mask	10 01 10 01
S1 (C, B missing)	A - -D D- -E
S2 (D, C missing)	A - -B B- -E

S1 (B before C in P2)	AB BD DC CE
S2 (C before D in P1)	AC CB BD DE

Table 5

Translating order crossover to DNA computing

How can be translated into DNA computing the previous crossover operator? Firstly, imagine that we have in a test tube the individuals that we had before but in the encoding which is explained above, representing each individual like a sequence of genes in a DNA strand. That is shown in Fig. 5.

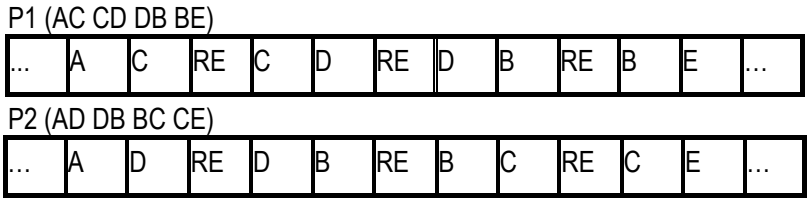

Fig. 5

When the problem of representing the mask is tackled a different view of the mask is given. This mask is suitable for our problem (TSP) using DNA computing and is obtained by following these steps:

1. Imagine that in our problem we have 5 cities: A B C D E.
2. To be discarded are the initial city and the final city. Then we have: B C D.
3. Randomly we choose one or several cities. For example: C D
4. Introduce in the soup (into the test tube) the following strands:

This represents the complementary bases of the cities C and D so that when introduced in the soup they can match the original strands there were in the soup before. See Fig. 6.

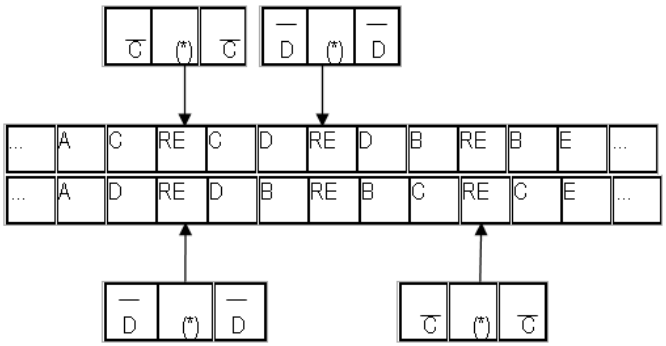

Fig 6

Computing crossover of Fig. 6:

P1	AC CD DB BE
P2	AD DB BC CE
Mask	C and D
S1 (C, D missing)	A - -- -B BE
S2 (D, C missing)	A - -B B- -E

S1 (D before C in P2)	AD DC CB BE
S2 (C before D in P1)	AC CB BD DE

Table 6

As a result of all this steps the crossover operator has changed a lot. Now it is still order crossover but with a very particular way of choosing the genes that must be changed. Instead of the initial mask we saw in Table 2, the mask it is used now consists on a selection of which cities (not genes) of P1 must be changed in the order they are found in P2. In the example shown in Table 5 the mask (10 01 10 01) showed the position of the cities that should be changed by the crossing operator. Now the mask for the same example (C, D) doesn't tell the position but de name of the cities to change.

However, if we try to apply this crossover operator in a genetic algorithm which uses the individual encoding that M.Calviño presented [M.Calviño, 2006] there is a big problem found. Spouse we have the gene AFC, in which F means the fitness between cities A and C. If we try to carry out the crossover operation of Fig 3, city C must be changed by D and then the gene would be AFD. Obviously F is not the fitness between A and D so this crossover operator only works with the strand-format proposed by J.Castellanos [J.Castellanos, 1998] though the other one works much better in the previous step of the GA, evaluation and selection.

Conclusion

The most important problem of DNA computers is resolved in this paper. This problem is the exponentially size algorithms the first DNA computer had. Genetic algorithms allow us to solve NP-Complete problems without exploring all the possible solutions. In GA a population of candidate solutions (individuals) to an optimization problem, like TSP, evolves toward better solutions. The "island model" of population dynamics can make the process faster meanwhile protocols of selection and crossover are presented in the paper.

The problem of crossover in a genetic algorithm using DNA has been resolved satisfactorily. Although the crossover technique might be different depending on the problem to be solved, it has been proved that it is possible to find a suitable crossover for NP-Complete problems such as TSP. This represents a new approach to the simulation of genetic algorithms with DNA.

Since the beginning of DNA computing, the lack of algorithms to be applied to this scientific area has been very large. Until recently, molecular computation has used "brute force" to solve NP-Complete problems. That is why the simulation of concepts of genetic evolution with DNA will help DNA computing to resolve hard computations. The crossover operator I have presented here gives an idea of how important and useful genetic algorithms are for DNA computing.

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.

[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.

[Gerald Karp, 2005] Gerald Karp.Cell and molecular biology: Concepts and experiments, 2005, Von Hoffman press

[Ayala, 1984] F.J. Ayala, J.A. Kiger. Modern genetics (2nd edition).

[Crow, 1986] J.F. Crow. Basic concepts in population, quantitative, and evolutionary Genetics. W.H. Freeman and Co. New York.

[Hartl, 1989] D.L. Hartl, A.G. Clark. Genetics of populations. Science Books International. Boston.

[Ford, 1991] T.C. Ford, J.M. Graham. An introduction to centrifugation. Bios Scientific Publishers. Oxford.

[Wilson, 1986] K. Wilson, K.H. Goulding. Principles and Techniques of Practical Biochemistry. Arnold LTD. Suffolk.

Authors' Information

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

A FRAMEWORK FOR FAST CLASSIFICATION ALGORITHMS

Thakur Ghanshyam, Ramesh Chandra Jain

Abstract: Today, due to globalization of the world the size of data set is increasing, it is necessary to discover the knowledge. The discovery of knowledge can be typically in the form of association rules, classification rules, clustering, discovery of frequent episodes and deviation detection. Fast and accurate classifiers for large databases are an important task in data mining. There is growing evidence that integrating classification and association rules mining, classification approaches based on heuristic, greedy search like decision tree induction. Emerging associative classification algorithms have shown good promises on producing accurate classifiers. In this paper we focus on performance of associative classification and present a parallel model for classifier building. For classifier building some parallel-distributed algorithms have been proposed for decision tree induction but so far no such work has been reported for associative classification.

Keywords: classification, association, and data mining.

1. Introduction

Data mining algorithms task is discovering knowledge from massive data sets. Building classifiers is one of the core tasks of data mining. Classification generally involves two phases, training and test. In the training phase the rule set is generated from the training data where each rule associates a pattern to a class. In the test phase the