VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

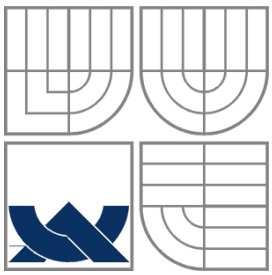
GRAPHICAL ENVIRONMENT FOR GENETIC ALGORITHM DEMONSTRATIONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

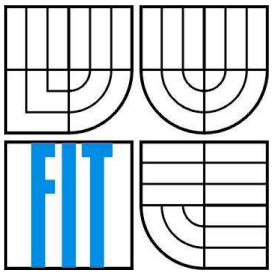
AUTOR PRÁCE
AUTHOR

JAIME SANTOS GONZÁLEZ

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

GRAPHICAL ENVIRONMENT FOR GENETIC ALGORITHM DEMONSTRATIONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

JAIME SANTOS GONZÁLEZ

VEDOUCÍ PRÁCE
SUPERVISOR

DR. ING. FRANTISEK ZBORIL

BRNO 2008

Abstract

This paper studies the implementation of six different user interfaces that helps to the user to solve 6 differents problems, explained in more detailed further, using genetic algorithms. These 6 problems are: “Greatest binary number“ : finding the greatest binary number of a given length;“TSP“: Travelling Salesman Problem;“Greatest sin in a defined interval“: finding the element with the biggest sin value;“Most ones followed by one zero“: finding the binary number with more couples "10";“Greatest real number“ : finding the biggest real number between 0 and 100;“Greatest binary real number“: finding the biggest real number specifying total and real lengths.

Keywords

Artificial Intelligence, Soft Computing, Genetic Algorithm, Java, User Interface.

Citation

SANTOS GONZÁLEZ, Jaime. *Graphical environment for genetic algorithm demonstrations.* Master thesis Brno University of Technology, Brno, 2007. 85p.

Graphical Environment for Genetic Algorithm Demonstrations

Declaration

I declare that i have solved this Master Thesis by myself.
I have mentioned all information resources used in the thesis.

……
Name
Date

Acknowledgement

I would like to thank my supervisor Ing. **Frantisek Zboril**, for his great supervision, help and patience with this MSc. Thesis.

Thanks to everybody in Czech Republic who made this experience unforgettable for me, specially to my new friends which made me enjoy every minute of this 9 months.

Thanks to all my old spanish friends, I mean, all the spanish friends that i met before this erasmus period, for supporting me and make me pass so good times. I want to thank specially to Álvaro Zubizarreta López and to Jorge Martín Martín for their help in critical moments.

Special thanks to **my family**, specially for my mother and my father, Mr. Fernando Santos López and Mrs. Carmen González Fuente. They always are there when i need them.

Finally i want to thanks to my grandparents all these lovely 24 years that i passed with them. It doesn´t matter where you are now: you will be always in my heart and present in my memory.

Brno, 2008

INDEX

1 Introduction and objectives

In this chapter we expose the initial ideas that have been the base or reason to develop the present project. At the end of this chapter we make a slight description about the structure and contents of the different chapters that compose the present memory.

Objectives

The actual project is included inside the investigation of the department of Intelligent Systems of Brno University of Technology.

In this case, the project is based in the books and the rest of references specified in the bibliography of this memory, but very specially based in:

- Aliev R.A., Aliev R.R., Soft Computing and its applications, World Scientific Co. Pte. Ltd., 2001, pages 251-283.

- Munakata Toshinori, Fundamentals of the New Artificial Intelligence, Springer, 1998, pages 65-101.

Having this knowledge base, the marked objectives for this project are:

- Develop a genetic algorithm as it is described in the references indicated previously, interpreting and resolving the gaps and incoherences that appear in them.

- The implementation of that algorithm will be realized using the programming language called Java.

- The previous objectives implies the knowledge of matters as:

  - Simple structures like List, arrays, Strings, etc.

  - General programming structure in Java.

  - User interface programming aspects in Java.

  - Knowing a development toolkit to develop the source code. In our case, it has been used JBuilder 2005.

- Application of the developed system to 6 different problems: "Greatest binary number", “Most ones followed by one cero", "TSP", "Greatest sin in a defined interval", “Greatest real number” and "Greatest binary real number", explained later.

- Showing over the interface the most important results that show the evolution of the algorithm in the different problems specified in this memory.

Memory organization

This memory is organized in 6 chapters, the bibliography and 2 appendices. Next a little summary of every one of them is made:

- Chapter 1: *Introduction and objectives*. In this chapter the initial ideas that have been the base or reason to develop the present project are exposed.

- Chapter 2: *Theorical fundamentals*. In this chapter the necessary theorical aspects to be able to understand the developed application will be shown.

- Chapter 3: *Environments and programming languages*. In this chapter the software tools used to develop the present project will be exposed.

- Chapter 4: *Developed application*. In this chapter it will be described the structure of the developed application and a summary of the behaviour of the system to make the understanding of the application easier.

- Chapter 5: *Experiments*. In this chapter the results obtained of the execution of the implemented application are exposed.

- Chapter 6: *Conclusions*. In this chapter the results obtained from the application of the genetic algorithm to 6 different problems are exposed.

- *Bibliography*: In this part of the project, there will be exposed books and several web pages used to develop it.

- Appendix A: *User's guide*. In this appendix it is exposed to the reader how to use the application that has been implemented.

- Appendix B: *CD content*. In this other appendix it is shown the content of the CD that is delivered with the memory.

2 Theorical fundamentals

In this chapter we are going to specify some theorical concepts that will let us understand the behaviour of the system to implement.

2.1 The softcomputing paradigm

In [7] it is said that a computer can work in a "more efficient" way than a human being can when the work implies intensive processing, like the inversion of a matrix of big dimensions. On the other hand, if the task requires any kind of cognitive perception or ability, the VonNeumann machine is very far from the humans. For example, the people can recognize forms of different sizes, orientations, and things like these more successfully than a VonNeumann machine can.

Therefore, a VonNeumann machine is suitable for good structured problems, while the human brain is better to solve vague formulated problems of the real life. To try to break the limitations of the traditional computation, the scientific has been looking for new computational ways that can imitate the human process of thinking and the operation of the brain, and be able to solve problems of the real world efficiently. As a consequence of those researches several computational areas have arised, some of them are known together as "soft computing".

In [5], it is said that in the traditional "hard computing", the main objectives are accuracy, certainty and rigor. However, in "soft computing" the main idea is that accuracy and certainty entail a cost; in soft computing the idea is to exploit the tolerance for imprecision, uncertainty and the rest of characteristic of the real world to obtain acceptable ("not necessarily perfect") solutions of low cost. This leads to remark the important human quality of making rational decisions in an environment of uncertainty and imprecision, like driving a vehicle in dense traffic.

The processing of the information in a natural environment is a natural phenomenon that let us survive making duties as making predictions, planning, and act in consequence. The processing of the human type information means logic-level and intuitive-level processing. The conventional computers are good in the first task, but in the second one are not as good as humans. The first request that a computer has to make intuitive processing, a computer has to be enough flexible to show the next characteristics: generality, to be robust, tolerance y real-time processing.

- Generality of a system is his skill to adapt to be able to face changes of the real world.
- With "be robust" it is referred to the stability of the system.
- Its tolerance to face incomplete or imprecise information.
- Real-time processing implies that the computer has to answer to an event in a reasonable time.

The information systems that have the former 4 characteristics are called "Real World Computing" (RWC). "Soft computing" can be seen as the key ingredient of the systems RWC.

2.1.1 Main Components of Soft computing

- **Fuzzy logic**: FL provides a simple way to treat elements that can be vague, ambiguous, imprecise, noisy, or presents missed input information. For example, in the reality the water doesn't have only two different states, "cold" or "hot", like the boolean logic would say. The water can be frozen, cold, warm, hot, boiling. FL is a problem-solving control system methodology that can implement systems from simple, small, embedded micro-controllers to

large, networked, multi-channel PC or workstation-based data acquisition and control systems. It can be implemented in hardware, software, or a combination of both.

- **Neural Computing** [WP6]: Artificial Neuronal Networks use a parallel, distributed and adaptive style to be able to learn from examples. These systems imitate hardware structure of the human brain to try to simulate some of its skills (that's the reason of its name). These networks consist of an interconnected group of artificial neurons and processes information using a connectionist approach to computation. In most cases an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network during the learning phase. They must be trained with a suitable number of examples.

- **Evolutionary computation** [WP7]: it is subfield of artificial intelligence born in 1993 that gets concepts about evolution and genetics to solve mainly optimization problems. It uses iterative progress, such as growth or development in a population. This population is then selected in a guided random search using parallel processing to achieve the desired end. The main components of this soft computing technique are

  - Evolutionary algorithms:

    - **Genetic algorithms** : It will be explained in the chapter 2.2.

    - Evolutionary programming: its basis is ideas of the evolution proposed by Gregor Charles Darwin and the discoverers realized by Gregor Mendel in genetics. Members of the population are viewed as part of a specific species rather than members of the same species therefore each parent generates an offspring, using a $(\mu + \mu)$ survivor selection.

    - Evolution strategy: It was created by Ingo Rechenberg and Hans-Paul Schwefel en the 1970s. Its main objective was to resolve problems relationated with optimization of parameters.

    - Genetic programming: methodology inspired by biological evolution to find computer programs that perform a user-defined task. It is a specialization of genetic algorithms where each individual is a computer program.

    - Learning classifier systems: is a machine learning system with close links to reinforcement learning and genetic algorithms

  - Swarm intelligence: is artificial intelligence based on the collective behaviour of decentralized, self-organized systems. The expression was introduced by Gerardo Beni and Jing Wang in 1989, in the context of cellular robotic systems. These systems are typically composed of a population of simple agents interacting locally with one another and with their environment. Although the agents follow very simple rules, the local interactions between such agents lead to the emergence of complex global behaviour. Its most important components are:

    - Ant colony optimization.

    - Particle swarm optimization.

  - Other components as:

    - Self-organization techniques.

- Artificial life.
- Harmony search algorithm.
- Artificial immune systems.
- Learnable Evolution Model.

➢ **Probabilistic reasoning**: The theory of probability is a theory developed to describe random event. In this section, Bayesian networks are the most important element [WP3, 2]:

o **A Bayesian network** [WP1] has two components:

- The first of them, more qualitative, is represented as a directed acyclic graph G = (V, E) where the nodes (finite set V) are the random variables of the problem, and the edges $E \subset V \times V$ show relations between variables.
- The second of them, quantitative, is probability distribution set (one per node) where the distribution in every node depends on all possible combinations of parental values.

In conclusion, it is a probabilistic model that shows the relation between a set of random variables with a directed graph, that shows explicitly the influence of cause. Thanks to the continue update rule of probabilities, Bayes rule, Bayesian networks are an extremely useful tool in the probabilities estimate with new facts.

➢ **Chaotic theory** [8]: is the popular denomination of the part of mathematics and physics that describes dynamics or random answers of certain nonlinear dynamical systems. Examples of this behaviour can be systems as the atmosphere, solar system or the population growth.

A very important characteristic of Soft computing is that its members complement each other, rather than compete (in the next section it will be studied some combinations between them). The combination of several of these components works better than the components separated of the combination, because the weaknesses of one of the methodologies are covered by other methodology that is in the combination.

Finally, it is important to say that neuronal and fuzzy technology are being used very frequently to solve tasks relationated with signal processing, pattern recognition and control.. For example, neuronal networks are being used to eliminate echo in telephone signals of very big distance, characters optical recognition, prediction of banking crisis or help to pilot aeroplanes. In the other hand, the fuzzy systems drive metropolitan trains, stabilize image of video cameras or control the most modern washing machines [3].

2.1.2 Alternatives in soft computing

As it was said in the previous section, better results are obtained making use of combination of some of the methodologies exposed than using one alone, because the defects or weaknesses of one of the methodologies can be corrected by other member methodology of that combination. In [1] are explained some of the possible combinations that can be used:

➢ Neuro-Fuzzy (Neuro computing + Fuzzy Logic):

Neural networks have a problem: its lack of interpretability. Fuzzy systems have another one: poor learning capability. The combination of both tries to solve both problems, due to

the learning capability of neural networks and the interpretability property of fuzzy systems. The essence of this approach is incorporate neural networks in fuzzy systems for fuzzification, construction of fuzzy rules, optimization and adaptation of fuzzy knowledge base, implementation of fuzzy reasoning and finally defuzzification.

- Fuzzy-Genetic (Fuzzy logic + genetic algorithms) :
  This combination enables creation of effective, robust and adaptive systems. The combination between FL and GA allows optimization of fuzzy knowledge base by defining optimal number of rules and optimal values for centers and shapes of membership functions. In the combination between FL and GA, theory of fuzzy systems can be used for improvising the behaviour of genetic operators or genetic algorithms on whole.

- Fuzzy-Chaos:
  This combination is very useful to build system's chaotic behaviour into rule structure.

- Neural-Genetic:
  One of the main problems in development of artificial neural systems is selection of a suitable learning method for tuning the parameters of a neural network (weights, thresholds and structure). The most known algorithm is the "error back propagation" algorithm, but this has some problems:

  - First: the quality of the learning depends on initial weights, which are generated randomly.
  - Second: the algorithm doesn't avoid local minima.
  - Finally: if the learning rate is too slow, finding a solution requires too much time.

  When the error back propagation fail, the application of genetic algorithms is a very good idea.

- Neural-Chaos:
  This combination is very can be very helpful for prediction and control.

- Fuzzy-Probabilistic:
  It is a very useful tool for modelling and reasoning under conditions of uncertainty, very typical in real-world problems.

- Other combinations very used are:
  - Fuzzy-Neural-Genetic
  - Neural-Fuzzy-Genetic

2.2 Genetic algorithms

2.2.1 Genetics in real life

In [4] it is said that before beginning to explain how a genetic algorithm works, I am going to review in a brief way how genetics works in real life. Every living being consist of cells, and each cell contains the same set of one or more **chromosomes**, strings of DNA, that serve as a "blueprint" for the organism. A chromosome can be divided into **genes**, each of which encodes a particular protein. A gene can be thought as encoding a trait, such as eye colour. The different possible "settings" for a

trait (e.g., blue, green, brown) are called **alleles**. Each gene is in a particular **locus** (position) on the chromosome.

Many organisms have multiple chromosomes in each cell. All the chromosomes of the organism, the complete collection of genetic material, form the organism's **genome**. The particular set of genes contained in a genome is called **genotype**. Two individuals have the same genotype if they have identical genomes. The genotype gives rise to the organism's **phenotype**, I mean, the physical and mental characteristics of the organism, such as eye colour, height, brain size, and intelligence.

In [5] it is explained too that if a pair of chromosomes for the child is looked closely, there are thousands or even millions of genes on each chromosome. It depends on the kind of organism of the owner of the chromosomes. In the case of the humans, the life of the body starts at fertilization of an egg by a sperm. Before conception, there are 23 chromosomes in an egg, and 23 in a sperm (a total of 46). In a diploid organism, for example a human, two chromosomes of the same number, one from egg and the other from the sperm, make a pair of chromosomes for the child (e.g., chromosomes mother No. 1 and father No. 1 make child pair No. 1).

As an illustration of all the previous paragraphs of this section 2.2.1, it is going to be exposed the case of the child's blood type. This blood can be one of the next possible types: O, A, B, or AB... Each gene for blood type can have a value of 0, 1, or 2, called **alleles**. Possible gene-pair combinations, called **genotypes**, and their corresponding **phenotypes** are shown in Table 1:

Genotype	Phenotype
00	O
10 or 11	A
20 or 22	B
21	AB

Table 1: possible genotypes and phenotypes for child's blood.

For example, if the gene value from mother is 1 and from father is 0, the child's genotype will be 10 and the phenotype will be blood type A. Note that the order of gene values in real life is immaterial (e.g., 10=01). Alleles differ for different kinds of genes (e.g., a specific gene may have either 0 or 1, another gene may have 0, 1, 2, or 3, and so on).

In biology, those individuals of any specie who better adapt to the environment have higher probabilities for survival. The individuals with more probability to survive will have higher probabilities for producing their offspring **(reproduction)** too. Over the different generations this process is repeated, and the result is that those individuals and genes that better adapt to the environment tend to remain, while those that don't adapt to the environment tend to disappear, i.e., become extinct. This theory of a natural screening process is called **(Darwinian) evolution**.

2.2.2 A brief history about genetic algorithms

In [1] it is explained that during the last three decades the interest in algorithms which rely on analogies to natural processes has grown. These algorithms have obtained a practical interest because of the emergence of massively parallel computers. Some of the algorithms that can be named in this group are evolutionary programming, genetic algorithms, evolution strategies, simulated annealing, etc.

It can be said that in the early 1950s genetic algorithms begin to appear when several biologists used computers for simulations of biological systems. However, the works done in late 1960s and early 1970s at the University of Michigan under the direction of John Holland led to genetic algorithms, as they are known today. In [4] it is said that Holland's original goal was not to design algorithms to solve specific problems, in contrast with evolution strategies and evolutionary programming, but rather to formally study the process of adaptation as it occurs in nature and to

develop different ways in which the mechanisms of natural adaptation may be imported into computer systems.

Holland presented in 1975 the genetic algorithm in the book “Adaptation in Natural and Artificial Systems” as an abstraction of biological evolution and gave a theoretical framework for adaptation under the GA. Holland's GA is a method to create a new population of "chromosomes" (e.g., strings of ones and zeros, or "bits") from another population of the same type using a kind of "natural selection" together with the genetics-inspired operators of crossover, mutation, and inversion. As it has been said in the section 2.2.1, each chromosome consists of "genes" (e.g., bits), each gene being an instance of a particular "allele" (e.g., 0 or 1). The selection operator chooses those chromosomes in the population that will be allowed to reproduce, and on average the fitter chromosomes produce more offspring than the less fit ones (this is the main idea of Darwin evolution, see last paragraph of section 2.2.1). Crossover exchanges subparts of two chromosomes, roughly mimicking biological recombination between two single-chromosome ("haploid") organisms; mutation randomly changes the allele values of some locations in the chromosome; and inversion reverses the order of a contiguous section of the chromosome, thus rearranging the order in which genes are arrayed. (Here, as in most of the GA literature, "crossover" and "recombination" will mean the same thing.) .

Holland's introduction of a population-based algorithm with crossover, inversion, and mutation was a major innovation. Moreover, Holland was the first to attempt to put computational evolution on a firm theoretical footing (see Holland 1975). Until recently this theoretical foundation, based on the notion of "schemas," was the basis of almost all subsequent theoretical work on genetic algorithms

In the last several years several researchers are working in group studying various evolutionary computation methods, and the boundaries between GAs, evolution strategies, evolutionary programming, and other evolutionary approaches have broken down to some extent. Today, researchers often use the term "genetic algorithm" to describe something very different of the original conception created by Holland.

2.2.3 Basic steps of a genetic algorithm

In [5], the mentioned steps of a genetic algorithm are the next:

- STEP 0: Initialization of the population.
  Create a set of solutions randomly. These solutions will be different depending the problem that is being solved.

- Repeat the following three steps until a terminal condition is satisfied. This condition depending what the user of the algorithms wants. One example of terminal condition can be “when the best solution of all the iterations does not improve over a certain number of iterations (e.g., 10), we terminate the iterations”.

  - STEP 1: Reproduction. The sub-steps to follow are the next:

    - Determine the fitness values (that will be different depending on the problem) and their corresponding probabilities for all the solutions in the population.

    - Create a mating pool. This pool is created selecting randomly solutions weighted by the fitness.

    It is clear that solutions with the higher fitness are more likely to be picked out than the unfit ones and tend to survive into the next generation. Here we see the influence of

the Darwinian evolution theory: the evolution concept based on the principle of natural selection.

- STEP 2: Crossover (or recombination) breeding. The sub-steps to carry out are the following:

  - Take two solutions randomly at a time. After this, with a fixed crossover probability pc (e.g., pc = 0.7), randomly determine whether crossover takes places.

    - ✓ If crossover does take place, go to the next sub step.

    - ✓ Otherwise, form two offsprings that are exact copies of the two solutions (parents), and go to STEP 3.

  - Select randomly internal points (crossing sites) of the solutions, and then swap the solution parts that follow these points. The number of crossing sites and the kind of crossover will depend on the implementation of the algorithm done.

  Perform this step 2 for all solutions obtained in Step 1, randomly selecting a pair at a time.

  What is the importance of this step? Parts of each solution may contain notions of importance or relevance. A genetic algorithm exploits this information by reproducing high quality notions, then crossing over these notions among high performers.

- STEP 3: Random mutation (called simply mutation too).

  With a certain fixed small mutation probability, pm (e.g., pm = 0.001 or the number of bits/1000) randomly select a small portion of the solutions and artificially change it (e.g., a bit of 1 to 0 or 0 to 1).

  The change produce by the mutation depends of the kind of problem to be solved. For example, if the value of a gene of a chromosome can be 0, 1 or 1, change 1 by 0 and vice versa is not enough. A possible mutation scheme can be: change 0 by 1, 1 by 2 and 2 by 0 (new value of gene can be the rest of the division between "(previous value of gene + 1)" and "2" (the maximum value of the gene)).

  What is the importance of this step? With this step, it can be created a new breed what would not be possible from the ordinary reproduction and crossover breeding processes. After a certain number of iterations, sometimes most solutions in the population become alike so that no further significant changes occur, yet they are far from original. The parts changed by mutation often shake the set of solutions out of such a static configuration (imagine that we are finding the global maximum of a function and we are "trapped" in the zone of a local maximum value).

2.2.4 Peculiarities

In [1], it is specified that Genetic algorithms (GA) have a number of specific peculiarities by which they differ from the other methods of optimization. These are the following ones:

- GA employ only the objective function, not the derivate one or some other information on the object. It is very convenient in case that the function is neither differentiable nor discrete.

- GA employ a parallel multipoint search strategy by maintaining a population of potential solutions, which provides wide information on the function behaviour and exclude the possibility of sticking in local extreme of the function. The traditional search methods, such as gradient, can not cope with this problem.

- GA use probability-transitive rules instead deterministic ones. Besides, GA is very simple for computer implementation.

2.2.5 Advantages

Taking information of [WP2, WP4], the main advantages obtained are:

- The power of GAs comes from the fact that the technique is robust and can deal successfully with a wide range of difficult problems.

- GAs are not guaranteed to find the global optimum solution to a problem, but they are generally good at finding "acceptably good" solutions "acceptably quickly".

- Even where existing techniques work well, the combination with GA improves the results.

- The basic mechanism of a GA is so robust that, within fairly wide margins, parameter settings are not critical.

- Genetic algorithms **are intrinsically parallel**. Most other algorithms are serial: this means that can only explore the solution space in only one direction at a time. That has a consequence: if the solution they discover turns out to be suboptimal, they have to abandon all work completed and start over.

However GAs can explore the solution space in multiple directions at once (because they have several individuals in every iteration). If one path turns out to be a dead end (its fitness is not good), they can easily eliminate it and continue work on more promising avenues, **giving them a greater chance each run of finding the optimal solution.**

However, the advantage of parallelism goes beyond this. Consider the following: All the 8-digit binary strings (strings of 0's and 1's) form a search space, which can be represented as ******** (where the * stands for "either 0 or 1"). The string 01101010 is one member of this space. However, it is also a member of other spaces, like the space 0******* or the space 01******. By evaluating the fitness of this one particular string, a genetic algorithm would be sampling each of these many spaces to which it belongs. Therefore, a GA that explicitly evaluates a small number of individuals is implicitly evaluating a much larger group of individuals (because evaluating a member, you are evaluating the different spaces that this member belongs to). It is just as a pollster who asks questions of a certain member of an ethnic, religious or social group hopes to learn something about the opinions of all members of that group, and therefore can reliably predict national opinion while sampling only a small percentage of the population. In the same way, the GA can "home in" on the space with the highest-fitness individuals and find the overall best one from that group.

In the context of evolutionary algorithms, this is known as the Schema Theorem, and is the "central advantage" of a GA over other problem-solving methods (Holland 1992, p. 68; Mitchell 1996, p.28-29; Goldberg 1989, p.20).

- Due to the parallelism, genetic algorithms are particularly **well-suited to solving problems where the space of all potential solutions is truly huge** - too big to make an exhaustive search within a reasonable time. Most problems that fall into this category are known as "nonlinear".

In a linear problem, the fitness of each component is independent. This makes that an improvement to any one part will produce an improvement in the system as a whole. The main problem is that the most part of the real-world problems aren't like this. Normally the problems are "nonlinear" problems and this kind of problems is not so simple. For example, a change in one component may have ripple effects on the entire system, or multiple changes that individually are detrimental may lead to much greater improvements in fitness when they are combined.

Besides, nonlinearity results in a combinatorial explosion of possibilities inside the space: for example, a problem of 1,000-digit binary strings:

- If it is linear, can be exhaustively searched by evaluating only 2,000 possibilities.
- Whereas if it is nonlinear, an exhaustive search requires evaluating 21000 possibilities - a number that would take over 300 digits to write out in full.

Fortunately, the implicit parallelism of a GA allows it to overcome this problem successfully finding optimal or very good results in a short period of time after directly sampling only small regions of the enormous space of solutions.

- Another notable strength of genetic algorithms is that **they perform well in problems for which the fitness landscape is complex - ones where the fitness function is discontinuous, noisy, changes over time, or has many local optima**. Most practical problems have a vast solution space, impossible to search exhaustively; the challenge then is to have an algorithm that avoids the optimum locals. Many search algorithms can't assure this and can become trapped by local optima. Making a "geographical" comparison, it is like sometimes they reach the top of a hill on the fitness landscape, they will not discover better solutions near the actual one and they will conclude that they have reached the best one, even though higher peaks exist in other part of the map.

  Evolutionary algorithms, on the other hand, are good escaping local optima and discovering the global optimum, even in a very complex fitness "landscape". All four of a GA's major components - parallelism, selection, mutation, and crossover - work together to accomplish this:

  - The first thing that a GA makes is to generate a diverse initial population, casting a "net" over the fitness landscape.
  - Selection focuses progress, selection the best individuals, making us "climb" in the "fitness landscape".
  - Small mutations enable each individual to explore its immediate neighbourhood.
  - However, crossover is the key element that distinguishes genetic algorithms from other methods such as hill-climbers and simulated annealing. Without crossover, each individual solution is on its own, exploring the search space in its immediate vicinity without reference to what other individuals may have discovered. However, due to crossover process, there is a transfer of information between successful candidates - individuals can benefit from what others have learned, and schemata can be mixed and combined, with the potential to produce an offspring that has the strengths of both its parents and the weaknesses of neither.

- Another area in which genetic algorithms excel is their **ability to manipulate many parameters simultaneously**. Many real-world problems cannot be stated in terms of a single

value, they must be expressed in terms of multiple objectives, usually with tradeoffs involved between them. GAs are very good at solving such kind of problems because their use of parallelism enables them to produce multiple equally good solutions to the same problem, possibly with one candidate solution optimizing one parameter and another candidate optimizing a different one (Haupt and Haupt 1998, p.17), and a human overseer can then select one of these candidates to use.

- Finally, one of the characteristics that in a first view can be a weakness it is one of the strengths of Gas: **GAs know nothing about the problems they are deployed to solve**: they make random changes to their candidate solutions and then use the fitness function to determine whether those changes produce an improvement. Since its decisions are based on randomness, **all possible search pathways are theoretically open to a GA**; by contrast, the strategies that use prior knowledge can't assure this. they rule out many pathways a priori, therefore missing any novel solutions that may exist there. Similarly, any technique that relies on prior knowledge will break down when such knowledge is not available, but again, GAs are not adversely affected by ignorance. Finally, it is important to say that this prior knowledge if not easy to obtain in some occasions.

2.2.6 Disadvantages

In [WP2] is explained that although genetic algorithms are a good strategy for solving one kind of problems, it is clear that they are not a panacea. GAs do have certain limitations; however, it will be shown that all of these can be overcome and none of them bear on the validity of biological evolution.

- The first difficulty is **defining a representation for the problem**. The language used to specify candidate solutions must be robust; i.e., it must be able to tolerate random changes such that fatal errors.

  This problem does not arise in nature, because the method of representation used by evolution, namely the genetic code, is inherently robust: there is no such thing as a sequence of DNA bases that cannot be translated into a protein. Therefore, virtually any change to an individual's genes will still produce an intelligible result. This is in contrast to human-created languages such as English, where the number of meaningful words is small compared to the total number of ways one can combine letters of the alphabet, and as a result of a mutation we can obtain a nonsense result.

- Other difficult to have in mind is **how to write the fitness function** : if this part is not made well, chosen the fitness function poorly or defining it imprecisely, the GA

  - may be unable to find a solution to the problem,
  - or may end up solving the wrong problem.

  This is not a problem in nature because only one fitness function exists: the drive to survive and reproduce, no matter what adaptations make this possible. Those organisms which reproduce more abundantly compared to their competitors are more fit.

- In addition **the other parameters of a GA - the size of the population, the rate of mutation and crossover, the type and strength of selection - must be also chosen with care**, for example:

- If the population size is too small, the genetic algorithm may not explore enough of the solution space and doesn't find good solutions.

- If the rate of genetic change is too high or the selection scheme is chosen poorly, the population may change too fast for selection to ever bring about convergence.

Living things do face similar difficulties, and evolution has dealt with them. It is true that, in an extreme drastic environmental change (population size falls too low, mutation rates are too high or the selection pressure is too strong) the species may go extinct. The solution that the nature has adopted is "the evolution of evolvability" -> adaptations that alter a species' adaptability. For example, most living things have evolved elaborate molecular machinery that checks for and corrects errors during the process of DNA replication, keeping their mutation rate down to acceptably low levels. Of course, not all catastrophes can be evaded, but in general, evolution is a successful strategy.

- Other weakness of GAs is **problems with "deceptive" fitness functions**, i.e., those where the locations of improved points give misleading information about where the global optimum is likely to be found.

The resolution to this problem for both genetic algorithms and biological evolution is the same: evolution is not a process that has to find the single global optimum every time.

- One well-known problem that can occur with a GA is known as **premature convergence**. If an individual that is more fit than most of its competitors emerges early on in the course of the run, it may reproduce so abundantly that it drives down the population's diversity too soon, leading the algorithm to converge on the local optimum that that individual represents rather than searching the fitness landscape thoroughly enough to find the global optimum. This is an especially common problem in small populations, where even chance variations in reproduction rate may cause this problem.

The most common methods implemented by GA researchers to deal with this problem all involve controlling the strength of selection, so as not to give excessively fit individuals too great of an advantage. Rank, scaling and tournament selection, discussed earlier, are three major means for accomplishing this.

In nature, premature convergence is less common since most beneficial mutations in living things produce only small, incremental fitness improvements; mutations that produce such a large fitness gain as to give their possessors dramatic reproductive advantage are rare.

- Finally, several researchers advise against **using genetic algorithms on analytically solvable problems**, because traditional analytic methods take much less time and computational effort and usually guaranteed mathematically to obtain the exact solution.

In nature, this issue does not arise since there is no perfect solution to any problem of biological adaptation.

2.2.7 Pecularities of the developed GA

In this section it is going to be explained some particular operations and parameters that appear in the developed genetic algorithm:

- **Crossover percentage**:

This parameter indicates the number of elements of the actual population that are not going to be part in the following cross operation. For example:

Imagine that we have a population of 10 elements and the crossover percentage is 20%. That means that the **worst** 2 elements (the 2 elements with the worst fitness) are not going to be part of the "cross operation", while the other 8 elements the 8 elements with the best fitness) are going to take part of the following cross operation...

- **Kinds of cross developed**:
  We have developed 2 kind of cross operation:

  - The first one, which can be called "bits cross", where we get the bits that represents of 2 elements and we mix them. We can have 1, 2 or 3 points of crossover. Examples:

    - With one point of crossover:
      Element 1: 1 0 1 0
      Element 2: 0 0 1 1
      Position of crossing: 2.
      Element 1 after crossing: 1 0 1 1.
      Element 2 after crossing: 0 0 1 0.

    - With two point of crossover:
      Element 1: 1 0 1 0 1 0 1 0 1 0 1
      Element 2: 0 0 1 1 0 0 1 1 0 0 1
      Positions of crossing: 2, 7.
      Element 1 after crossing: 1 0 1 1 0 0 1 1 1 0 1.
      Element 2 after crossing: 0 0 1 0 1 0 1 0 0 0 1.

    - With three points of crossover:
      Element 1: 1 0 1 0 1 0 1 0 1 0 1 0.
      Element 2: 0 1 0 1 0 1 0 1 0 1 0 1.
      Positions of crossing: 2, 6, 9.
      Element 1 after crossing: 1 0 0 1 0 1 1 0 1 1 0 1.
      Element 2 after crossing: 0 1 1 0 1 0 0 1 0 0 1 0.

    In all the previous crosses, the positions of crossing were generated randomly.

  - The second one, that can be called **"parameter cross"**, doesn't make a literal cross of the bits of the elements. In this "parameter cross", the new elements are obtained at a mix of the fitnesses of the fathers. The amount or percentage to take of the fitness of every parent depends of a parameter, "a", which belongs to interval [-0.25, 1.25]. This parameter is generated randomly. Example:

    - Element 1(e1): 42.82245635986328.
      Element 2(e2): 41.09468460083008
      a1: 0.458093
      a2: 0.27732277
      Element 1 after crossing (ce1):
      ce1= a1 * e1 + (1-a1) * e2 = 41.88616472770832
      Element 2 after crossing(ce2):
      ce2= a2* e1 + (1-a2) * e2 =41.314180061355046

- **Type of mutations**:
  We have developed 2 kinds of mutations:

- Type A: it can be called "mutation per bit". The mutation process affects to not every bit, I mean, one random number is generated for every bit that is in the population. If the generated number is less than the 1/1000, the bit considered is mutated.

- Type B: it can be called "global mutation". One random number is generated for all the population. If the generated number is less than the mutation probability (numberBits/1000), a number of mutations are going to be produced. The number of mutations produced is the integer part of the mutation probability + 1, I mean:

  - If probability of mutation is a number between [0, 1), one mutation is produced.

  - If probability of mutation is a number between [1, 2), two mutations are produced.

  The position of mutation (element and bit of that element) is produced randomly.

- **Iteration criteria**:

  We have developed 6 different iteration criteria:

- Iteration criterion 1: Selecting this option, the user will have to define the number of times than the algorithm has to be repeated until its end.

- Iteration criterion 2: Selecting this option, the user will define the number of iterations consecutive without change in the best element that the algorithm has to execute before its ending. For example:

  Imagine that the user indicates that 5 is the number of iterations without change and we will call iteration without change as "iterWCh". Then, a possible execution of the algorithm can be:

  Iteration 0: new best element and iteration without change =0.
  Iteration 1: appear a new best element, and then iterWCh is 0.
  Iteration 2: don't appear new element, and then iterWCh is 1.
  Iteration 3: Appear new element, and then iterWCh is 0.
  Iteration 4: don't appear new element, and then iterWCh is 1.
  …….
  Iteration number 8: don't appear new element, and then iterWCh is 5. Algorithm ends.

- Iteration criterion 3: Selecting this option, the user will have to define 2 things:
  - The number of iterations without change until force a mutation. For future explanations, this number is going to be called "A".

  - The number of forced mutations that the user wants to force. For future explanations, this number is going to be called "B".

  In this method, when the number of iterations without change until force a mutation appears, the algorithm forces a mutation in one of the elements of the actual population. The number of total mutations forced until the end of the algorithm is defined by the user too as it can be seen before. The algorithm ends when "A" iterations without change in the best algorithm have appeared since the forced mutation number "B".

- Iteration criterion 4: It is the iteration criterion number 1 where number of iterations is 5.

- Iteration criterion 5: It is the iteration criterion number 2 where number of iterations without change is 5.

- Iteration criterion 6: It is the iteration criterion number 1 where both parameters are 5.

3 Environments and programming languages

In this chapter it is going to be exposed a presentation of the different environments and programming languages used to develop the application proposed in the chapter 1, based on the theorical aspects developed in the chapter 2. This chapter will be divided in 1 main section, JAVA, that contains 4 sub-sections: brief history of Java, where it is going to be exposed the beginning and the expansion of this language; main characteristics, where it will be shown the main aspects that defines the actual language; advantages, which shows the strengths of the language; and disadvantages, where it will be exposed the weakest spot of the language.

3.1 JAVA

In [WP8], it is exposed that Java is a programming language created by Sun Microsystems and released in 1995 as one of the main components of Sun's Java platform. The language follows the steps of C and C++ in the syntax but has a simpler object model and fewer low-level facilities. Java applications are normally compiled to byte code. This byte code can run on any Java virtual machine (JVM) independently what is the computer architecture.

The reference and original implementation Java compilers, class libraries and virtual machines were developed by Sun from 1995. In May 2007, following the specifications of the Java Community Process, Sun made available most of their Java technologies as free software under the GNU General Public License. Other companies have developed alternative implementations, such as the GNU Compiler for Java and GNU Classpath.

3.1.1 Brief history of JAVA

In [WP8] it is specified that in June 1991, James Gosling created the Java language for use in a set top box project. The language was initially called Oak because James Gosling had an oak tree outside his office. After several names. including the name “Green”, it was decided to call the language “Java”, word selected from a list of random words. The goals that gosling wants to obtain were to implement a virtual machine and a language that similar in notation to C and C++.

In 1995 the first implementation of java that appeared in public. That implementation was Java 1.0. It promised "Write Once, Run Anywhere" (WORA), providing no-cost runtimes on popular platforms. It was quite secure and its security was configurable: the user was able to restrict network and file access. A short time after, major web browsers incorporated the ability to run secure Java applets within web pages.

In a short time of period, Java became popular. With the arrival of Java 2, new versions had multiple configurations created to answer the new needs of the different types of platforms:

- J2EE was for enterprise applications and the greatly stripped down version.
- J2ME was for mobile applications.
- J2SE was the designation for the Standard Edition.

On 13 November 2006, Sun released much of Java as free software under the terms of the GNU General Public License (GPL). On 8 May 2007 Sun finished this process, making all the code of

Java's core open source, except a small portion of code that Sun did not have the copyright.

3.1.2 Main characteristics

They have been extracted from [WP8].

- **Platform independence**:

  This characteristic lets Java users write programs that must run similarly on any supported hardware/operating-system platform. This target is reached by most Java compilers by compiling the Java language code halfway (**Java byte code**). This Java byte code can be defined as simplified machine instructions specific to the Java platform. After this process, the code is run on **a virtual machine** (VM). The virtual machine is a program written in native code that interprets and executes generic Java byte code. This code is written on the host hardware The Java bytecode is interpreted or converted to native machine code by the JIT compiler (look Figure 1: obtained from PDF file of Lecture1 of the course "IJE").

  An interpreted virtual machine was used in the first implementations of the language to achieve **portability**. The disadvantage of these implementations was that produced programs ran more slowly than programs compiled to native executables, like in C or C++ programs. This produced a reputation for poor performance to the language. More recent Java Virtual Machine implementations produce programs that run quite faster than before, using multiple techniques. These techniques are going to be exposed in the next paragraphs.

  The first technique that is going to be exposed is the technique known **as just-in-time compilation (JIT)** With this technique, the Java bytecode is translated into native code at the time that the program is run. This produces a program that executes faster than interpreted code but this has a bad aspect: there is compilation overhead during execution.

  To solve this aspect, more sophisticated virtual machines use **dynamic recompilation**. This technique is based on the idea that the VM can analyze the behaviour of the running program and selectively recompile and optimize critical parts of the program. Dynamic recompilation can obtain code more optimized than the one obtained using static compilation.

  The third technique that is going to be exposed is the technique commonly known as **static compilation**. With this technique, the code is directly compiled into native code. This is used by the more traditional compilers. Static Java compilers translate the Java language code to native object code, without the intermediate bytecode stage. One example of computer that uses this technique is GCJ. This method achieves good performance compared to interpretation, but has one weakness: the portability is minor.

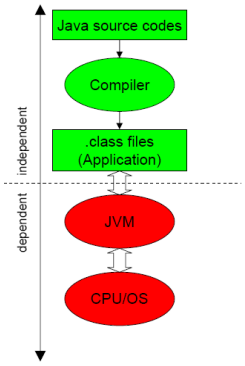


Figure 1: compilation in Java

- **Automatic memory management**:

  This aspect let programmers not to be worried about memory management. When the programmer has to allocate memory manually, several "unpleasant" situations can appear: if he forgets to deallocate memory previously allocated or writes code that fails trying to make the task, a memory leak occurs and the program can consume an arbitrarily large amount of memory; if the program attempts to deallocate the region of memory more than once, the result is undefined and the program may become unstable and may crash. Automatic memory management try to avoid all these kind of problems.

  In Java, automatic memory management is realized by **automatic garbage collector**. The programmer determines when objects are created, and the Java runtime is responsible for managing the object's lifecycle. When there are no references to an object, this one is eligible for release by the Java garbage collector and it may be freed automatically by the garbage collector at any time.

  The use of garbage collection in a language can also affect programming paradigms. If, for example, the developer assumes that the cost of allocate memory or free space of memory is low, he may choose to construct objects instead of pre-initializing, holding and reusing them.

  Other characteristic about garbage collector in Java is that it is virtually invisible to the developer: developers may have no notion of when garbage collection will take place. This can be an advantage or a disadvantage, depending on the intention of the application.

  To end this aspect, it has to be said that Java does not support pointer arithmetic because the garbage collector may relocate referenced objects. If arbitrary manipulation of pointers is allowed, the safety and security of the automatic memory management wouldn't be guaranteed.

3.1.3 Advantages

Some of the advantages extracted from [WP5].that the use of Java brings to its users are:

- Java is **simple**
  - Java has replaced the complexity of multiple inheritance in languages like C++ with a simple structure called interface, and also has eliminated the use of pointers.
  - Java uses automatic memory allocation and garbage collection.
  - The number of language constructs in Java is small for such a powerful language. The clean syntax makes Java programs easy to write and read.
- Java is **Distributed**: this let programmers construct distributed programs in an easy way: writing network programs in Java is like sending and receiving data to and from a file.
- Java is **Portable**
  - The programs implemented in Java can run on any platform without having to be recompiled.

- There are no platform-specific features on the Java language specification.

- Java is **Architecture Neutral**: this is a consequence of platform independent.

- **Security**: The compiler, interpreter, and Java-compatible browsers contain several levels of security measures to reduce the risk of security compromise, loss of data and program integrity, and damage to system users.

- **Reliability**:

  - Pointers and automatic type conversion, features that are detrimental to program reliability, are avoided in Java, independently the platform used to implement the application.

  - The Java compiler provides several levels of additional checks to identify type mismatches and other inconsistencies, independently the platform used to implement the application.

  - The Java runtime system duplicates many of the checks and performs additional checks to verify that the executable bytecodes form a valid Java program.

- **Multimedia**: Images, Sounds and Animation: JAVA provides extensive multimedia facilities that will enable a programmer to start developing powerful multimedia applications immediately.

- **Networking**: JavaBeans make networking easy to write reusable components that can be strung together with a minimum of additional coding added to the original codes.

- Java is **Robust.** This is managed because Java has several characteristics that makes this:

  - Early checking for possible errors realized during the implementation of the application.

  - Java does not support pointers, which eliminates the possibility of overwriting memory.

  - Java has a runtime exception-handling feature to provide programming support for robustness, and can catch and answer to an exceptional situation so that the program can continue its normal execution and terminate successfully when a runtime error occurs.

- Java is **Multithreaded.** This is managed because Java has several characteristics that makes this:

  - In Java, multithreaded programming has been integrated into Java, while in other languages, operating system-specific procedures have to be called in order to enable multithreading.

  - Multithreading is especially useful in the development of GUI and network programming.

- Java is **Dynamic**: A programmer can add to the class new methods and properties without affecting the clients of the class. Also, Java is able to load classes as needed at runtime.

3.1.4 Disadvantages

- The main disadvantage is **speed**: An interpreter must first translate the Java binary code into the equivalent microprocessor instruction.

4 Developed aplication

In this chapter it is going to be exposed the different elements that works together to manage the objectives expressed in Chapter 1, based on the theorical aspects developed in chapter 2. This chapter will be divided in 2 sections: Global view, where a global view about the work of the system is made; and Developed classes, where all the classes of the system are explained. On the other hand, this second section has 2 parts: General classes, where the classes that are present in all the developed problems are specified; and Specific classes for the different problems, where the rest of the classes are presented.

4.1 Global view

This section is written to let the user have a global view about the working of the application. It is going to be exposed the kind of instances that, in the most of the time, are going to be working in the system while the system is developing the algorithm (when all the input data has been introduced) for every one of the solved problems by the application:

- **"Greatest binary number":** there will be instances of classes Application, Main Frame, FrameGAB, controlGAB and Control.
- **"Most ones followed by one cero"**: there will be instances of classes Application, Main Frame, FrameGAB, controlGAOneZero and Control.
- **"TSP"**: there will be instances of classes Application, Main Frame, FrameTSP, controlGATSP and Control.
- **"Greatest sin in a defined interval"**: there will be instances of classes Application, Main Frame, FrameGAInterval, controlGAInterval and Control.
- **"Greatest real number"**: there will be instances of classes Application, Main Frame, FrameGAReal, controlGAReal and Control.
- **"Greatest binary real number"**: there will be instances of classes Application, Main Frame, FrameGABReal, controlGABReal and Control.

4.2 Developed classes

4.2.1 General classes

4.2.1.1 Aplication

- **Overview**

This class is the responsible of beginning all the operation of the algorithm genetic, independently of the problem that the user has selected to be resolved.

- **Attributes**

  - **boolean packFrame**: it is an attribute, initialized to false, that is used, depending on his value, in the method of creation of this class.

- **Methods**

  - **Application()**: It is the constructor of this class. In this method, the main frame of the application is created.

4.2.1.2 Elemento

- **Overview**

This class is the responsible of representing every member of the actual population, independently of the step of the process that is being developed.

- **Attributes**

  - **double[] valor**: In this attribute, it is saved the value of the element. In the problem called greatest real number, it is simply a real number. In the rest of the cases, it is an array of integer values.

  - **double fitness**: This attribute represents the fitness of the element in the actual problem.

  - **double fitnessProbability**: This other attribute represents the probability that has the element to be selected in the reproduction process.

  - **double expectedCount**: This represents the average number of elements expected like this one in the future population.

- **Methods**

  - **public Elemento(int length)**: this is one of the creators of this class.

  - **public Elemento(double v[],int length)**: This is the other creators of the class.

  - **double getFitness()** : this method returns the value of the attribute “fitness”.

  - **void setFitnessProbability(double poblationFitness)**: this method is used to initialize the attribute “fitnessProbability”.

  - **double getFitnessProbability()**:this method returns the value of the attribute “fitnessProbability”.

  - **void setExpectedCount(int n)**: this method is used to set a value in the attribute expectedCount.

  - **double getExpectedCount()**: this method is used to obtain the value of the attribute expectedCount of the object.

- **double[] getValor()**: this method is used to obtain the set of values of the attribute valor of the object.

- **void setValor(double[] aux,int pos, int posFinal)** : copy the vector specified in the parameter “aux” from the position specified in the parameter "pos" until position specified in the parameter "posFinal" in the same positions of the attribute valor.

- **void setFitness(double Fitness)**: this method is used to initialize the attribute “fitness”

4.2.1.3 GA

- **Overview**

This interface represents the operation that are going to have the classes in charge of developed the main operations of the genetic algorithms in the different problems offered in the developed application.

- **Attributes**

This element doesn’t have attributes because it is an interface.

- **Methods**

- **void ForceMutation(List childrenPopulation)**: this method is called to force a mutation in the population specified in the parameter “childrenPopulation”.

- **void setLength(int lengthP)**: This method is used to set the value of the attribute length of the classes that implements this interface.

- **void setNumPopulation(int numPopulationP)**: This method is used to set the value of the attribute numPopulation of the classes that implements this interface.

- **void setNumberBits(int numberBitsP)**: This method is used to set the value of the attribute numberBits of the classes that implements this interface.

- **void setMutationProbability(float mutationProbabilityP)**: This method is used to set the value of the attribute length of the classes that implements this interface.

- **int getNumberBits()**: returns the number of bits of the object of one of the class that implements this interface.

- **void reproduction(List population,List childrenPopulation)**: calling this method in one object of one of the classes that implements this interface is developed the process of reproduction. This method has 2 parameters: “population”, that specifies the parent population and “childrenPopulation” that represents the population that is going to be obtained after the process of the reproduction of the population specifies in parameter “population”.

- **void DividePoblation(float porcentajeRemain, List population,List parents,List offsprings,List indParents,List indOffsprings)**: Calling this method, the object that develops this method divide the population specified in the parameter "population" between 2 populations specified in the parameters "parents" and "offsprings". This

division is made having in mind the value of the parameter “porcentajeRemain”. This percentage indicates the percentage of elements that is going to form the population “parents” and the percentage of elements that is going to form the population “population”. This method also store the positions of the elements, referred to the population "population", of both populations (“parents” and “offsprings”) in the parameters "indParents" and "indOffSptring"

- **void crossoverBreeding(List childrenPopulation,int numberPoints)**: thanks to this method, the object called develops the process of crossover breeding over the population specified in the parameter childrenPopulation. This crossover process can be developed in 3 different ways, with one point of crossover, with 2 points of crossover or with 3 points of crossover. This last data is specified in the parameter called “numberPoints”.

- **void randomMutation(List childrenPopulation,int typeMutation)**: In this method is developed the process of mutation. This process of mutation is made over the population indicated in the parameter “childrenPopulation”. There can be 3 different type of mutations indicated in the parameter “typeMutation”.

- **List createInitialPopulation()**:Calling this method, the object that develops this method creates the initial population of the algorithm. The elements of this initial population are created randomly and are different between them.

4.2.1.4 MainFrame

- **Overview**

This class is the one that represents the main frame of the application, I mean, the one when you can select the problem to be solved.

- **Attributes**

This class only has attributes that represents elements of the user interface of the main window.

- **Methods**

  - **public MainFrame()** : it is the creator of the class.

  - **void jbInit()**: in this method, all the elements of the object are initialized.

  - **void processWindowEvent(WindowEvent e)**: This method defines the process realized when an event relationates with the window appears.

  - **Methods relationates with the action to realized when a particular button is pressed.**

4.2.1.5 Control

- **Overview**

This class is the responsible of making some operations that are not special of genetic algorithms, like checking that the user has introduced all the necessary parameters, compares the fitness of the elements of the population and stores the best one, calculate if the algorithm has to end, etc.

- **Attributes**

  - **int length**: length of the elements that forms part of the different populations.
  - **int ChosenEnd**: used to know if there has to be a new iteration of the algorithm or not.
  - **int numIterWithoutChange**: actual number of iterations without change produced.
  - **int TotalNumIterWithoutChange**: stores the maximum number of iterations without change in the best element that the algorithm has to execute.
  - **int tipoProblema**: this attribute indicates the problem selected by the user.
  - **int numberIterations**: maximum number of iterations that the algorithm can execute.
  - **int numberMutations**: it is the total number of mutations that is going to be forced.
  - **Int end**: this attribute indicates if the algorithm has to continue or not.
  - **int numPopulation**: shows the number of elements of the different populations.
  - **int mutationsForced**: the number of mutations forced until the actual moment.
  - **int typeMutation**: type of process of mutation selected by the user.
  - **String fileName:** name of the input file (necessary in “TSP”).
  - **float porcentajeRemain**: percentage of crossover introduced by the user.
  - **int numberPoints**: indicates the number of points used in the crossover process.
  - **float lowerLimit**: in the "Greatest sin in a defined interval" indicates the lower limit of that interval.
  - **float upperLimit**: in the "Greatest sin in a defined interval" indicates the upper limit.
  - **int realLength**: in the problem called "Greatest binary real number" indicates the length of the real part.

- **Methods**

  - **public Control()**: it is the creator of the class.
  - **void setLength(int lengthP)**: sets a new value in the attribute “length”.

- **void setNumPopulation(int numPopulationP)**: stores a new value in the attribute "numPopulation". This new value is specified in the parameter of the method.

- **void setNumberPoints(int numberPointsP)**: this method is referenced to set a new value in the attribute "numberPoints". This new value is specified in the parameter of the method.

- **void setTypeMutation(int typeMutationP)**: this method is used to set a new value in the attribute "typeMutation". This new value is specified in the parameter of the method.

- **void setCrossOver(float crossOverP)**: this method is called to stablish a new value in the attribute "porcentajeRemain". This new value is specified in the parameter of the method.

- **void setRealLength(int lengthP)**: this method is used to store a new value in the attribute "realLength". This new value is specified in the parameter of the method.

- **void setLowerLimit(float lowerLimitP)**: this method is called to set a new value in the attribute "lowerLimit". This new value is specified in the parameter of the method.

- **void setUpperLimit(float upperLimitP)**: this method is referenced to establish a new value in the attribute "upperLimit". This new value is specified in the parameter of the method.

- **int getLength()**: returns the value of the attribute "length".

- **int getNumPopulation()**: is used to return the value of the attribute "numPopulation".

- **int getNumberPoints()**: returns the value of the attribute "numberPoints".

- **int getTypeMutation()**: returns the value of the attribute "typeMutation".

- **float getCrossOver()**: is used to return the value of the attribute "porcentajeRemain".

- **int getRealLength()**: returns the value of the attribute "realLength".

- **float getLowerLimit()**: returns the value of the attribute "lowerLimit".

- **float getUpperLimit()**: is used to return the value of the attribute "upperLimit".

- **void setTipoProblema(int tipoProblemaP)**: sets a new value in the attribute called "tipoProblema".

- **int getTipoProblema()**: returns the value of the attribute "tipoProblema".

- **void setNumberIterations(int numberIterationsP)**: sets the value of attribute called "numberIterations".

- **int getNumberIterations()**: returns the value of the attribute "numberIterations".

- **int getNumIterWithoutChange()**: this method returns the value of attribute called "numIterWithoutChange".

- **void setNumIterWithoutChange(int numIterWithoutChangeP)**: establishes the value of the attribute “numIterWithoutChange”.

- **void setMutationsForced(int mutationsForcedP)**: sets a new value of attribute called “mutationsForced”.

- **int getMutationsForced()**: obtains the value of the attribute “mutationsForced”.

- **int initialCkecking(String name1,String name2,String name3)**: checks if the user has completed the input fields that appear in the instance of “FrameGAB” that calls this method. Returns 0 if everything is correct; other number, if something is not correct.

- **int initialCkeckingReal(String name1,String name2)**: the same than the previous method for the problem “Greatest real number”.

- **int initialCkeckingBReal(String name1,String name2,String name3,String name4)**: the same than the previous method for the problem "Greatest binary real number".

- **int initialCkeckingTSP(String name1,String name2, String name3)**: the same than the previous method for the problem "TSP".

- **int initialCkeckingInterval(String name1,String name2,String name3,String name4,String name5)**: the same than the previous method for the problem "Greatest sin in a defined interval".

- **void reset()**: when a thread is stopped, let the things ready for the next possible thread.

- **int initialization(int lengthP, int numberElementsP, int criteriumIterationP, int numberPointsP, int typeMutationP, float crossOverP, int tipoProblemaP)**: with this method, the attributes of the object of this class are initialized when the actual problem is “Greatest binary number" or “Most ones followed by one cero".

- **int initializationGABReal(int lengthP,int realPartLenghtP,int numberElementsP,int iterationCriteriumP,int numberPointsP,int typeMutationP,float crossOverP)** : the same than the previous method for the problem "Greatest binary real number".

- **int initializationGABInterval(int lengthP,int numberElementsP,float lowerLimitP,float upperLimitP,int iterationCriteriumP,int numberPointsP,int typeMutationP,float crossOverP)** : the same than the previous method for the problem "Greatest sin in a defined interval".

- **int initializationGATSP(int iterationCriteriumP, int numberPointsP, int typeMutationP, float crossOverP, int numberElementsP)**: the same than the previous method for the problem “TSP”.

- **int initializationGAReal(int numberElementsP, int iterationCriteriumP, int typeMutationP, float crossOverP)**: the same than the previous method for the problem “Greatest real number”.

- **void copy(List childrenPopulation, List parents, List offsprings, List posParents, List posOffsprings)**: copy the elements of the populations specified in “parents” and “offsprings” in the population specified in “childrenPopulation” in positions “posParents” and “posOffsprings”.

- **void calculateBestEver(List childrenPopulation, Elemento bestEverSolution)**: calculate the best ever solution until the actual moment. The parameter "childrenPopulation" contains the elements whose fitness is going to be compared with the fitness of the element specified in the parameter "bestEverSolution".

- **int testEnd(int actualNumberIteration, GA ga, List childrenPopulation)**: returns one if the ending condition of the genetic algorithm is satisfied. The parameters "ga" and "childrenPopulation" are necessary if a forced mutation has to be produced.

- **int testEnd2(int actualNumberIteration, GAReal ga, List childrenPopulation)**: The same function than the previous method "testEnd", but for GAReal instances (the only one that doesn't implement interface "GA").

- **Different methods necessary to solve the "TSP"**:

  - int numberLines(File archive): calculates the number of lines of the file "archive".

  - void initializeCoordinates(double[][] coordinates,File archive) : initializes the parameter "coordinates" using as input file the file specified in parameter "archive".

  - void Fill(String line,int row,double[][] coordinates) : Given a line, specified in the parameter "line", of the input file initializes the row number "row" of the parameter "coordinates".

  - void calculateDistances(double[][] coordinates, double[][] distances, int length): calculates the distances between the different cities specified in parameter "coordinates". The results are stored in the parameter "distances".

4.2.1.6 FrameData

- **Overview**

This class is the responsible of creating the frame that let the user introduce the extra information needed when the criterion iteration selected is 1, I mean, the number of iterations of the algorithm.

- **Attributes**

  - **int numIterations:** this attribute contains the number of iterations of the algorithm.

  - **FrameGAB framegab, FrameGABReal framegabreal, FrameGAInterval framegainterval, FrameGAReal framegareal**, **FrameTSP frametsp**: indicate the possible "father", the possible creator of the object.

  - **int fatherType**: this indicates the kind of father of the object. It is useful to know which of the "frame attributes" has to be referenced.

  - **Attributes that represents the user interface:** JPanel jPanel1, JLabel jLabel1, JTextField jTextField1, JButton jButton1.

- **Methods**

  - **public FrameData()**: it is the default creator of the class.

  - **int getNumberIterations()**: returns the value of the attribute “numIterations”.

  - **void jbInit()**: is the method that initializes the elements of the class.

  - **String getDataTextField1()**: returns the text that is in the attribute jTextField1.

  - **void jButton1_actionPerformed(ActionEvent e)**: This method defines the operations to be realized when the Button1 (the one with the string “OK”) is called. One of the aspects that this method does is to validate if the user hasn’t introduced all the data when this button is pressed.

  - **void setFather(FrameGAB f)**: initializes “framegab” and “fatherType” too.

  - **void setFather(FrameGABReal f)**: initializes “framegabreal” and “fatherType” too.

  - **void setFather(FrameGAInterval f)** initializes “framegainterval” and “fatherType” too.

  - **void setFather(FrameGAReal f)**: initializes “framegareal” and “fatherType” too.

  - **void setFather(FrameTSP f)**: initializes “frametsp” and “fatherType” too.

4.2.1.7 FrameData2

- **Overview**

This class is the responsible of creating the frame that let the user introduce the extra information needed when the criterion iteration selected is 2, I mean, the number of iterations without change that can be produced until ending the algorithm.

- **Attributes**

  - **int numIterationsWithoutChange:** this attribute contains the number of iterations without change that can be produced.

  - **Attributes framegab**, **framegabreal**, **framegainterval**, **framegareal**, **frametsp**, **fatherType and attributes that represents the user** interface of frameData..

- **Methods**

  - **public FrameData2()**: it is the default creator of the class.

  - **int getNumberIterationsWithoutChange()**: returns the value of the attribute “numIterationsWithoutChange”.

  - **void jbInit()**: is the method that initializes the elements of the class.

- **Methods getDataTextField1(), jButton1_actionPerformed(ActionEvent e) and methods setFather of FrameData.**

4.2.1.8 FrameData3

- **Overview**

This class is the responsible of creating the frame that let the user introduce the extra information needed when the criterion iteration selected is 3, I mean, the number of iterations without change that can be produced until forced a mutation and the total number of mutations that can be produced until ending the algorithm.

- **Attributes**

  - **int numIterationsWithoutChange:** this attribute contains the number of iterations without change that can be produced until forced a mutation.

  - **int mutationsForced:** this attribute stores the total number of mutations that can be produced.

  - **Attributes framegab**, **framegabreal**, **framegainterval**, **framegareal**, **frametsp**, **fatherType of frameData and several attributes that represents the user interface.**

- **Methods**

  - **public FrameData3()**: it is the default creator of the class.

  - **int getNumberIterationsWithoutChange()**: returns the value of the attribute "numIterationsWithoutChange".

  - **int getMutationsForced()**: returns the value of the attribute "mutationsForced".

  - **Methods getDataTextField1(), jButton1_actionPerformed(ActionEvent e) and methods setFather of FrameData.**

  - **String getDataTextField2()** : returns the text that is in the attribute jTextField2

  - **void jbInit()**: is the method that initializes the elements of the class.

  - **int initialChecking()**: this method returns 0 when all the necessary extra data have been introduced and other number in otherwise.

4.2.2 Specific classes for the different problems

4.2.2.1 "Greatest binary number"

- **GAB**

  - **Overview**

This class is one of the classes that implement the interface GA specified in the section 4.3 of this memory. This class is going to be the responsible of doing the main operations relative to the genetic algorithm in the problem of finding the greatest binary number.

  - **Constants**

    - float crossoverProbability **:** represents the value of the probability of crossover, 0.7 (extracted of one of the sources of the bibliography specified in this memory).

  - **Attributes**

    - **int numPopulation**: attribute saves the value of the number of elements.
    - **int length**: represents the length of the elements.
    - **int numberBits**: represents the total number of bits of the population.
    - **float mutationProbability**: represents the mutationProbability of the algorithm.

  - **Methods**

    - **public GAB()**: this is the creator of this class.
    - **Implementations of the methods of interface GA**.
    - **void cross(Elemento crossElement, Elemento crossElement2, int posBegin, int posEnd)**: this is a method that is not part of interface GA. This method is called to cross the bits of the 2 elements specified in the parameters "crossElement" and "crossElement2" from the initial position specified in the parameter "posBegin" until the position specified in parameter "posEnd".
    - **void calculateFitness(Elemento e)**: this is a method that is not part of interface GA. This method is called to calculate the fitness of the element specified in the parameter "e". This value is saves in the attribute fitness of that element specified in parameter "e".
    - **void calculateStadistics(List population)**: This is a method, like the previous one, that doesn't appear in the interface GA. It is called to calculate the new value of the attributes "fitness", "fitnessProbability" and "expectedCount" of all the elements specified in the parameter "population".

- **FrameGAB**

  - **Overview**

This class is the one that represents the main frame of the problems called "Greatest binary number" and “Most ones followed by one cero". Other important general aspect of this class is that extends class JFrame.

  - **Attributes**

    - **int tipoProblema**: saves the type of problem that we are solving.

    - **boolean extraDataIntroduced**: used to control if all the input data has been introduced.
    - **controlGAB cGAB**: represents the instance of the class controlGAB that is going to be part in the actual process.
    - **FrameData fd**: represents the possible instance created by the actual object of the class “FrameData”. This happens when the user press the button “extraData” when the iteration criterion selected is 1.

    - **FrameData2 fd2**: represents the possible instance created by the actual object of the class “FrameData2”. This happens when the user press the button “extraData” when the iteration criterion selected is 2.

    - **FrameData3 fd3**: represents the possible instance created by the actual object of the class “FrameData3”. This happens when the user press the button “extraData” when the iteration criterion selected is 3.

    - **Control c**: symbolizes the instance of class Control that is will take part in the actual process.

    - **Attributes that represents the user interface**: JPanel contentPane, JPanel JpanelParameters, JPanel JpanelAction, JPanel JpanelResults, JLabel jLabel1, JTextField JTextField1, JLabel jLabel2, JTextField JTextField2, JLabel jLabel3, JComboBox JComboBox3, JLabel jLabel4, JLabel jLabel5, JLabel jLabel6, JComboBox JComboBox4, JComboBox JComboBox5, JTextField JTextField6, JTextField JTextField7, JLabel jLabel7, JTextField JTextField8, JLabel jLabel8, JButton jButtonStart, JButton jButtonStop, JButton jButton1, JScrollPane jScrollPane1, JTextArea jTextArea1, JTextArea jTextArea2, JPanel jPanel1.

  - **Methods**

    - **public FrameGAB(String name)** : it is the creator of the class.

    - **void jbInit(String name)** : it is the method that initializes the elements of the class.

    - **void showErrors(int error)** :. If any of the numerical numbers introduce by the user is incorrect, the system shows a window saying where is the mistake.

    - **void showStringErrors(int error)** : this method will show a window saying which parameter hasn’t been introduced if this situation appears.

- **int initialCkecking()** : in this method is specified the behaviour that the object of this class makes to check if the to validate if the user has introduced all the input parameters.

- **void jButtonStart_actionPerformed(ActionEvent e)** : In this method the operations to realize when the user presses the jButtonStart are defined.

- **void jButtonStop_actionPerformed(ActionEvent e)** : In this method the operations to realize when the user presses the jButtonStop are defined.

- **void jButton1_actionPerformed(ActionEvent e)** : This method defines the operations to be realized when the Button1(the one with the string "moreData") is called. Depending the value of the parameter "iteration criterion", the actions will be different:

  - ✓ If iteration criterion is 1, it will appear a window (instance of FrameData) asking the number of iterations.

  - ✓ If iteration criterion is 2, it will appear a window(instance of FrameData2) asking the number of iterations without change

  - ✓ If iteration criterion is 3, it will appear a window(instance of FrameData3) asking 2 parameters:

    - The number of iterations without change until force a mutation.

    - The number of forced mutations that the user wants to force.

  - ✓ In the rest of the cases, it will not appear any window.

- **void setActualIteration(int actualIteration)** : this method is called to set a new value in JTextField7(the text field that shows the actual iteration).

- **void setExtraDataIntroduced(boolean extraDataIntroducedP)** : this method is called to set a new value in attribute "extraDataIntroduced".

- **void setBestElement(double[] bestElement)** : this method is called to set a new value in JTextField8(the text field that shows the best element after the actual iteration).

- **void setTextjTextArea2(String text)** : it is used to set text in the attribute jTextArea2.

- **void setTextjTextArea1(String text)** : it is used to set text in the attribute jTextArea1.

- **void appendTextjTextArea1(String text)** : it is used to add text in the attribute jTextArea1.

- **void resetTextFields()** : it is used to reset the text fields that represent the actual iteration and the best element. It is called when the user press the button start.

- **ControlGAB**

  - **Overview**

This class is the responsible of developing(administering) all the genetic algorithm in the problems called "Greatest binary number" and “Most ones followed by one cero".

  - **Attributes**

    - **int length**: length of the elements that forms part of the different populations.

    - **int numberElements**: number of elements that are going to be in the different populations.

    - **int iterationCriterium**: this attribute saves the iteration criterion selected by the user.

    - **int numberPoints**: the number of points in the crossover process selected by the user.

    - **int typeMutation**: type of mutation selected by the user.

    - **int tipoProblema**: this attribute represents the problem that is being developed.

    - **double crossOver** percentage of elements that are not going to be in process of crossover.

    - **Control c**: instance of the class Control that is going to be part in the actual process.

    - **FrameGAB frame** : place where the results obtained have to be written.

    - **int end**: this attribute symbolizes if the algorithm has to finish or not.

  - **Methods**

    - **public controlGAB()**: it is the default creator of the class.

    - **public controlGAB (int lengthP , int numberElementsP , int iterationCriteriumP , int numberPointsP , int typeMutationP , double crossOverP , int tipoProblemaP , Control c , FrameGAB frame)**: it is other creator of the class.

    - **void end()**: this method calls to the method reset of an object of the class Control to reset the actual Iteration.

    - **void run()**: it is the most important method of this class. Calling this method, the called object makes the necessary calls to the different elements (object of the class GAB, FrameGAB, Control) to administer all the process, I mean, to realize the genetic algorithm to solve the actual problem.

4.2.2.2 "TSP"

- **GATSP**

  - **Overview**

This class is one of the classes that implement the interface GA specified in the section 4.3 of this memory. This class is going to be the responsible of doing the main operations relative to the genetic algorithm in the TSP (Travelling Salesman Problem).

  - **Constants**

The same than in GAB.

  - **Attributes**

The same than in GAB.

  - **Methods**

    - **public GATSP():** this is the creator of this class.

    - **Implementations of the methods of interface GA**..

    - **boolean esta(double vec[] ,List bitList)**: This method check if the vector specified in the parameter “vec” is a member of the list specified in the parameter “bitList”.

    - **void calculateFitness(Elemento e,double[][] distances)**: This method is called to calculate the fitness of the element specified in the parameter “e”.

    - **void calculateStadistics(List population,double[][] distances)**: It is called to calculate the new value of the attributes “fitness”, “fitnessProbability” and “expectedCount” of all the elements specified in the parameter “population.

    - **void cross(Elemento crossElement, Elemento crossElement2,int posBegin,and int posEnd)**: This method is called to cross the bits of the 2 elements specified in the parameters “crossElement” and “crossElement2” from the initial position specified in the parameter “posBegin” until the position specified in parameter “posEnd”.

- **FrameTSP**

  - **Overview**

This class is the one that represents the main frame of the problem called "TSP". Other important general aspect of this class is that extends class JDialog.

  - **Attributes**

    - The same attributes than in frameGAB except attribute cGAB.

    - **controlGATSP cGATSP** : instance of the class controlGATSP that is going to be part in the actual process.

- **Attributes that represents the user interface**: JPanel jPanel1, JPanel jPanel2, JPanel jPanel3, JPanel jPanel4, JLabel jLabel1, JTextField jTextField1, JButton jButton1, JLabel jLabel2, JComboBox jComboBox1, JLabel jLabel3, JComboBox jComboBox2, JLabel jLabel4, JComboBox jComboBox3, JLabel jLabel5, JTextField jTextField2, JButton jButton2, JButton jButton3, JLabel jLabel6, JTextField jTextField3, JLabel jLabel7, JTextField jTextField4, JFileChooser jFileChooser1, BorderLayout borderLayout1, JButton jButton4, JTextArea jTextArea1, JLabel jLabel8, JTextField jTextField5, JLabel jLabel9, JTextField jTextField6.

o **Methods**

- **public FrameTSP()** : it is the creator of the class.

- **Methods void jbInit ()** , **void showErrors(int error) , int initialCkecking ( ) , void setExtraDataIntroduced (boolean extraDataIntroducedP) , void setActualIteration (int actualIteration) , void setBestElement (double [ ] bestElement)** , **void setTextjTextArea1 (String text) and void resetTextFields()** of **FrameGAB.**

- **void jButton1_actionPerformed(ActionEvent e)** : This method defines the operations to be realized when the Button1(the one with the string "File") is called.

- **void jButton2_actionPerformed(ActionEvent e)** : In this method the operations to realize when the user presses the jButton2 are defined.

- **void jButton3_actionPerformed(ActionEvent e)** : In this method the operations to realize when the user presses the jButton3 are defined.

- **void jButton4_actionPerformed(ActionEvent e)** : The same function of **void jButton1_actionPerformed(ActionEvent e) of FrameGAB.**

- **Graphics getGraphicsJPanel3()** : it is used to obtain the graphics of the jPanel3 , I mean, returns "jPanel3.getGraphics()".

- **int getWidthJPanel3()** : it is used to obtain the width of the jPanel3.

- **int getHeightJPanel3()** :it is used to obtain the height of the jPanel3.

- **void setDistanceBestElement(double distance)**: it is used to set a value in the attribute jTextField6. This value is specified in the parameter "distance".

➢ **ControlGATSP**

o **Overview**

This class is the responsible of developing (administering) the entire genetic algorithm in the problem called "TSP".

o **Attributes**

- **Attributes iterationCriterium, numberPoints, typeMutation, tipoProblema, c, end and numberElements and crossOver** of ControlGAB.

- **String fileName** : it saves the complete path that describes the position of the input file, file with the positions of the different cities.

- **FrameTSP frame** : represents the place where the results obtained by the object of this class have to be written.

o **Methods**

The same than in controlGAB: default creator, a creator with parameters to initialize attributes of the new object and methods void end() and void run().

4.2.2.3 "Greatest sin in a defined interval"

➢ **GAInterval**

o **Overview**

This class is other class that implements the interface GA specified in the section 4.3 of this memory. This class is going to be the responsible of doing the main operations relative to the genetic algorithm in the problem of finding the greatest value of the function "sin" in a defined interval (for defining the interval, user will have to specify the lower and upper limits of that interval).

o **Constants**

The same than in GAB.

o **Attributes**

The same than in GAB.

o **Methods**

- **Public GAInterval()**: this is the creator of this class.

- **Implementations of the methods of interface GA**.

- **double getRealNumber(Elemento elemento,float lowerLimit, float upperLimit)**: This method is used to calculate the real number value represented as a binary number in the attribute "valor" of the element "elemento".

- **void calculateFitness(Elemento e,float lowerLimit, float upperLimit)**: The same than the method of the same name(although different parameters) in GAB.

- **void calculateStadistics(List population,float lowerLimit, float upperLimit)**: The same than the method of the same name(although different parameters) in GAB.

- **void cross(Elemento crossElement, Elemento crossElement2,int posBegin,int posEnd)**: The same than the method of the same name(although different parameters) in GAB.

- **FrameGAInterval**

  - **Overview**

This class is the one that represents the main frame of the problem called. "Greatest sin in a defined interval. Other important general aspect of this class is that extends class JDialog.

  - **Attributes**

    - **The same attributes than in frameGAB except attribute cGAB.**

    - **controlGAInterval cGAI** : This attribute represents the instance of the class controlGAI that is going to be part in the actual process.

    - **Attributes that represents the user interface**: JPanel contentPane, JPanel JpanelParameters, JPanel JpanelAction, JPanel JpanelResults, JLabel jLabel1, JTextField jTextField1, JLabel jLabel2, JTextField jTextField2, JLabel jLabel3, JTextField jTextField3, JLabel jLabel4, JLabel jLabel5, JLabel jLabel6, JTextField jTextField4, JComboBox JComboBox5, JComboBox JComboBox6, JComboBox JComboBox7, JButton jButtonStart, JButton jButtonStop, JLabel jLabel7, JTextField jTextField8, JLabel jLabel8, JTextField jTextField9, JLabel jLabel9, JTextField jTextField10, JLabel jLabel10, JButton jButton1, JPanel jPanel1, JPanel jPanel2, JTextArea jTextArea1, JLabel jLabel11, JTextField jTextField5.

  - **Methods**

    - **public FrameGAInterval()** : it is the creator of the class.

    - **Methods void jbInit ( )** , **void showErrors (int error) , void showStringErrors (int error) , int initialCkecking() ,void setExtraDataIntroduced(boolean extraDataIntroducedP), void setActualIteration(int actualIteration), void setBestElement(double[] bestElement)** , **void setTextjTextArea1(String text) and void resetTextFields()** of **FrameGAB.**

    - **void jButtonStart_actionPerformed(ActionEvent e)** : In this method the operations to realize when the user presses the jButtonStart are defined.

    - **void jButtonStop_actionPerformed(ActionEvent e)** : In this method the operations to realize when the user presses the jButtonStop are defined.

    - **void jButton1_actionPerformed(ActionEvent e)** : The same function of void jButton1_actionPerformed(ActionEvent e) of FrameGAB.

    - **Graphics getGraphicsJPanel1()** : it is used to obtain the graphics of the jPanel1 , I mean, returns “jPanel1.getGraphics()”.

    - **Graphics getGraphicsJPanel2()** : it is used to obtain the graphics of the jPanel2.

    - **int getWidthJPanel1()** : it is used to obtain the width of the jPanel1.

    - **int getHeightJPanel1()** : it is used to obtain the height of the jPanel1.

- **int getWidthJPanel2()**: this method is used to obtain the width of the jPanel2.

- **int getHeightJPanel2()**: it is used to get the height of the jPanel2.

- **void setRealValueBestElement(double bestElement)**: this method is used to put the real value of the best element in the jTextField10.

- **void setSinValueBestElement(double bestElement)**: it is used to put value of the sin of the best element in the jTextField5. This value is specified in the parameter "bestElement".

➢ **controlGAInterval**

o **Overview**

This class is the responsible of developing(administering) the entire genetic algorithm in the problem called "Greatest sin in a defined interval".

o **Attributes**

- **Attributes iterationCriterium, numberPoints, typeMutation, tipoProblema, c, end, numberElements, length and crossOver** of ControlGAB.

- **float lowerLimit**: this attribute contains the lower limit of the interval.

- **float upperLimit**: this attribute stores the upper limit of the interval.

- **FrameGAInterval frame** : the place where the results obtained are written.

o **Methods**

The same than in controlGAB: default creator, a creator with parameters to initialize attributes of the new object and methods void end() and void run().

4.2.2.4 "Most ones followed by one cero"

➢ **GAOneZero**

o **Overview**

This class is one of the classes that implement the interface GA specified in the section 4.3 of this memory. This class is going to be the responsible of doing the main operations relative to the genetic algorithm in the problem of finding the binary number that has more ones followed by cero, I mean, more couples "10".

o **Constants**

The same than in GAB.

- **Attributes**

The same than in GAB.

- **Methods**

  - **public GAOneZero()**: this is the creator of this class.

  - The rest of methods are the same than in GAB.

4.2.2.5 "Greatest real number"

- **GAReal**

  - **Overview**

This class doesn´t implements the interface GA specified in the section 4.3 of this memory(it has a different crossoverBreeding), but developes the most part of the operations specified in that interface This class is going to be the responsible of doing the main operations relative to the genetic algorithm in the problem of finding the greatest real number between 0 and 100.

  - **Constants**

    - float crossoverProbability : The same than in GAB.

    - int MAX: This constant represents the value of the biggest real number of our problem. In our aplicaction, this value is 100.

  - **Attributes**

The same than in GAB.

  - **Methods**

    - public GAReal(): this is the creator of this class.

    - The same than in GAB, except that it doesn´t have method "void cross(Elemento crossElement, Elemento crossElement2,int posBegin,int posEnd)".

- **FrameGAReal**

  - **Overview**

This class is the one that represents the main frame of the problem called "Greatest real number". Other important general aspect of this class is that extends class JFrame.

  - **Attributes**

    - The **same attributes than in frameGAB** except attribute cGAB.

    - **controlGAReal cGAReal** : instance of the class controlGAReal that will be in the process.

- **Attributes that represents the user interface**: JPanel contentPane, JPanel JpanelParameters, JPanel JpanelAction, JLabel jLabel3, JTextField jTextField3, JLabel jLabel4, JLabel jLabel6, JComboBox JComboBox4, JComboBox JComboBox6, JButton jButtonStart, JButton jButtonStop, JTextField jTextField7, JLabel jLabel7, GridLayout gridLayout1, JButton jButton1, JPanel JpanelResults, JLabel jLabel1, JLabel jLabel2, JTextField jTextField1, JTextField jTextField2, JScrollPane jScrollPane1, JTextArea jTextArea1, JTextArea jTextArea2, JPanel jPanel1 = new JPanel().

o **Methods**

- **public FrameGAReal()** : it is the creator of the class.

- **Methods** void jbInit() , void showErrors(int error), void showStringErrors(int error), int initialCkecking(),void setExtraDataIntroduced(boolean extraDataIntroducedP), void setActualIteration(int actualIteration), void setBestElement(double[] bestElement) , void setTextjTextArea1(String text), void appendTextjTextArea1(String text), void setTextjTextArea2(String text) and void resetTextFields() **of FrameGAB**.

- **void jButtonStart_actionPerformed(ActionEvent e)** : In this method the operations to realize when the user press the jButtonStart are defined. The most important one is create the instance of the class controlGAReal and call the method start of that object when everything is ready.

- **void jButtonStop_actionPerformed(ActionEvent e)** : In this method the operations to realize when the user press the jButtonStop are defined. The most important ones are call the method "end" of that object of the class "controlGAReal"and stop the execution of the instance of the same object.

- **void jButton1_actionPerformed(ActionEvent e)** : The same function of void jButton1_actionPerformed(ActionEvent e) of FrameGAB.

➢ **controlGAReal**

o **Overview**

This class is the responsible of developing(administering) all the genetic algorithm in the problem called "Greatest real number".

o **Attributes**

- **Attributes** iterationCriterium, typeMutation, tipoProblema, c, end, numberElements, length and crossOver **of ControlGAB**.

- **FrameGAReal frame** : represents the place where the results obtained by the object of this class have to be written.

o **Methods**

The same than in controlGAB: default creator, a creator with parameters to initialize attributes of the new object and methods void end() and void run().

4.2.2.6 "Greatest binary real number"

- **GABReal**

  - **Overview**

This class is the last class that implements the interface GA specified in the section 4.3 of this memory. This class is going to be the responsible of doing the main operations relative to the genetic algorithm in the problem of finding the greatest real binary number.

  - **Constants**

The same than in GAB.

  - **Attributes**

    - The same than in GAB.
    - **int realLength**: length of the binary number that represents the real part.

  - **Methods**

    - **public GABReal()**: this is the creator of this class.
    - Implementations of the methods of interface GA.
    - **void setRealLength(int lengthP)**:. sets a new value in the attribute realLength.
    - **void cross(Elemento crossElement, Elemento crossElement2,int posBegin,int posEnd)**: The same than the method of the same name in GAB.
    - **void calculateFitness(Elemento e)**: The same than the method of the same name in GAB.
    - **void calculateStadistics(List population)**: The same than the method of the same name in GAB.

- **FrameGABReal**

  - **Overview**

This class is the one that represents the main frame of the problem called "Greatest binary real number". Other important general aspect of this class is that extends class JFrame.

  - **Attributes**

    - The same attributes than in frameGAB except attribute cGAB.
    - **controlGABReal cGABReal** : This attribute represents the instance of the class controlGABReal that is going to be part in the actual process.

- **Attributes that represents the user interface**: JPanel contentPane, JPanel JpanelParameters, JPanel JpanelAction, JPanel JpanelResults, JLabel jLabel1, JTextField jTextField1, JLabel jLabel2, JTextField jTextField2, JLabel jLabel3, JTextField jTextField3, JLabel jLabel4, JLabel jLabel5, JLabel jLabel6, JComboBox JComboBox4, JComboBox JComboBox5, JComboBox JComboBox6, JButton jButtonStart, JButton jButtonStop, JTextField jTextField7, JLabel jLabel7, JTextField jTextField8, JLabel jLabel8, JTextField jTextField9, JLabel jLabel9, JButton jButton1, JScrollPane jScrollPane1, JTextArea jTextArea1,JTextArea jTextArea2,JPanel jPanel1.

o **Methods**

- **public FrameGABReal()** : it is the creator of the class.

- **Methods** void jbInit() , void showErrors(int error), void showStringErrors(int error), int initialCkecking(),void setExtraDataIntroduced(boolean extraDataIntroducedP), void setActualIteration(int actualIteration), void setBestElement(double[] bestElement) , void setTextjTextArea1(String text) , void setTextjTextArea2(String text) , void appendTextjTextArea1(String text) and void resetTextFields() **of FrameGAB**.

- **void jButtonStart_actionPerformed(ActionEvent e)** : In this method the operations to realize when the user press the jButtonStart are defined.

- **void jButtonStop_actionPerformed(ActionEvent e)** : In this method the operations to realize when the user press the jButtonStop are defined.

- **void jButton1_actionPerformed(ActionEvent e)** : The same function of void jButton1_actionPerformed(ActionEvent e) of FrameGAB.

➢ **controlGABReal**

o **Overview**

This class is the responsible of developing(administering) all the genetic algorithm in the problem called "Greatest binary real number".

o **Attributes**

- **Attributes** iterationCriterium, typeMutation, tipoProblema, numberPoints, c, end, numberElements, length and crossOver **of ControlGAB**.

- **int lengthRealPart**: symbolizes the length of the real part of the number.

- **FrameGABReal frame** : place where the results obtained have to be written.

o **Methods**

The same than in controlGAB: default creator, a creator with parameters to initialize attributes of the new object and methods void end() and void run().

5 Experiments

In this chapter is going to be analyzed the different results obtained because of the running of the developed software system over the 6 different problems proposed. This chapter will have 2 different sections: an introduction, where it is explained basically the methodology that has been developed to obtain the different analyses of the obtained results; and one section, titled "Problems solved", where it is going to be explained each one of the problems used to apply the algorithm and a following analysis of the results obtained in each problem.

5.1 Introduction

In this sections is exposed the experimental work made with the developed genetic algorithm. The main idea of this experimental work is to see the performance of the algorithm and the different considered alternatives in the problems explained in the next section.

- Objetives of the experiments:
  - See how good are the solutions obtained.
  - Make a comparison between the different options.
  - See which parameters are the most important in the different exposed problems.
- Criteria that shows how good are the realized experiments:
  - The best value of the 2 executions that has the same values in the input parameters.
  - The mean of both executions with the same values in the input parameters.
  - If the best elements was developed as consequence of applying the algorithm or was in the initial population.
- Methodology of the experiments: to realize the experiments have been used the next values for the different input parameters :
  - The used lengths of the elements are five, ten and twenty five.
  - The used number of elements in the different population used is five, ten and twenty five.
  - The only used lower limit used is 1.5.
  - The used upper limits are 4.64 and 7.8
  - The only length of the real part used has been 2.
  - There has been used the iteration criteria number 4, 5 and 6.
  - There has been used one, two and three points of cross (more information in section 2.2.7).
  - There has been used both kind of mutations explained in section 2.2.7.

- The used crossover percentages are 10, 50 and 90 (more information in section 2.2.7).

Finally it has to be said that not all the combinations are possible in all the problems. For example, the parameters referred to the lower and upper limits of an interval only can be used in the problem called "Greatest sin in a defined interval" (section 5.2.3). To make this clearer, the next table, Table 2, show us the executions made in the differents considered problems:

Considered problem	Number of executions
"Greatest binary number"	36
"TSP"	30
"Greatest sin in a defined interval"	42
“Most ones followed by one cero"	36
“Greatest real number”	20
"Greatest binary real number"	36

Table 2: number of executions of the considered problems

We can see in Table 2 that it hasn’t been made the same number of executions for the different problem, because as it can be seen next, not all the problems have the same input parameters.

5.2 Problems solved

5.2.1 "Greatest binary number"

Introduction

Given a length, the objective of the algorithm is trying to find the greatest binary number that has that length. It is clear that there can be executions of the algorithm that the best result obtained by the algorithm can’t be different of the “real best result”.

In this problem, every member of the iteration represents a binary number. For example: If the length of elements is 4, the binary number “1111” represents the decimal number “15”.

In this problem, the user will have to introduce always the next parameters:

- Length of the elements.
- Number of elements of the different populations.
- The iteration criterion.
- The number of points in the crossover process.
- The type of mutation.
- % of crossover.

As it was said in the method “void jButton1_actionPerformed(ActionEvent e)” of the class FrameGAB, the user will have to introduce more data if the iteration criterion selected is less than 4:

- If iteration criterion is 1, it will appear a window (instance of FrameData) asking the number of iterations.

- If iteration criterion is 2, it will appear a window (instance of FrameData2) asking the number of iterations without change.

- If iteration criterion is 3(instance of FrameData3), it will appear a window asking 2 parameters:

  - The number of iterations without change until force a mutation

  - The number of forced mutations that the user wants to force.

Analysis of obtained results

From the results obtained following the methodology exposed previously, we can obtain several conclusions about the different parameters that appear in this problem:

- Length of the elements: when the length is very small, obtain the greatest number is easier than if the length is quite big. This is very normal, because when the length is smaller, the number of possible numbers where the algorithm has to look for the best is smaller.

- Number of Elements in the populations: when the number of elements is very big, the results obtained are better than when the number of elements in the different populations is smaller. This is logic, because if the different populations have more elements, the algorithm will be exploring more possible solutions and probably in more different zones of the solutions apace.

- Iteration criterion used: between the iteration criterion number 4 and the iteration criterion number 5 there hasn't been a lot of differences in the best element obtained. In one of the executions that used as iteration criterion the number fourth, the best element has been found (11111), but this has been because this element was generated in the initial population. The one that obtains bet results is the iteration criterion number 6. Using this iteration criterion, the best element was found, even with a population whose best element (11001) was quite far from the best possible element (11111). It is true too that with this iteration criterion, the algorithm has a much more time cost than with the other ones. Using the iteration criterion 4, the number of iterations were 5 in both executions, using the criterion number 5 were 6 in both executions, but using the criterion number 6, the number of executions were 35 and 32.

  In conclusion, the user has to decide what is more important for him to select one of these 3 iteration criteria: the time or the fitness of the best element obtained.

- Number of points used in the crossover: To compare this, it has been made experiments with 2 different lengths:

  - When the elements that forms the different populations have length 5, the best elements obtained using 1 or 2 points of cross were the best element of the different initial population. However, when the number of points used in the cross is 3, the best result was obtained in both cases and, what is more, in both cases this best element doesn't appear in the initial population. Then, it can be said that for this case, the best option is used 3 points of cross.

  - When the elements of the different populations have length 25, it is not obtained the best possible result (25 ones) using neither 1, 2 nor 3 points of cross. The results obtained

using one or two points of cross are more or less the same. However, like when the length was 5, the best results are obtained with 3 points of cross.

- Type of mutation used: To compare this, it has been made experiments with 2 different lengths:
  - When the elements that form the different populations have length 5, the best elements obtained using both kinds of mutations were the best element of the different initial population. Then not any special conclusion can be obtained of these tests.
  - When the elements of the different populations have length 25, it is not obtained the best possible result (25 ones) using the first or second kind of mutation. It has to be said too that, analyzing the results, the second kind of mutation obtain better results.
- Crossover percentage: To compare this, it has been made experiments with 2 different lengths:
  - When the number of elements that forms the different populations is 5, the best results are obtained using 10 as crossover percentage and the worst results are obtained using 90 as crossover percentage.
  - When the number of elements of the different populations is 25, the three options obtain the best possible solution. This best solution was in the initial populations in the 6 executions (2 per option of crossover percentage).

5.1.1 "TSP"

The objective of the algorithm is try to find the best solution of the travelling Salesman problem: given a number of cities and the positions of the cities, find the least-cost round-trip route that visits each city exactly once and then returns to the starting city.

In this problem, every member of the populations symbolizes a possible solution of this problem.

In this problem, the user will have to introduce always the next parameters:

- Input file where it will appear all the positions of the cities. .
- The iteration criterion.
- The number of points in the crossover process.
- The type of mutation.
- % of crossover.

As it was said in the method "void jButton4_actionPerformed(ActionEvent e)" of the class FrameTSP, the user will have to introduce more data if the iteration criterion selected is less than 4:

- If iteration criterion is 1, it will appear a window (instance of FrameData) asking the number of iterations.

- If iteration criterion is 2, it will appear a window(instance of FrameData2) asking the number of iterations without change

- If iteration criterion is 3(instance of FrameData3), it will appear a window asking 2 parameters:

  - The number of iterations without change until force a mutation

  - The number of forced mutations that the user wants to force.

Analysis of obtained results

From the results obtained following the methodology exposed previously, we can obtain several conclusions about the different parameters that appear in this problem:

- Number of Elements in the populations: when the number of elements is very big, the results obtained are better than when the number of elements in the different populations is smaller. This is logic, because if the different populations have more elements, the algorithm will be exploring more possible solutions and probably in more different zones of the solutions apace.

- Iteration criterion used: In the experiments realized to compare the three possible options, the best result is obtained using the iteration criterion number 5 (distance of: 45.49494940178282). Making a comparison between the mean of the 2 executions of the three criteria, the best one if the iteration criterion number 6 and the worst one if the criterion number 4.

  Like in the rest of problems, the number of iterations using the iteration criterion number 6 is much bigger than using the other two ones.

- Number of points used in the crossover: In the experiments realized, the best results are obtained using three points in the crossover process, and the worst results are obtained using only one point of cross.

- Type of mutation used: To compare this, it has been made experiments with 2 different lengths:

  - When the number of the elements that forms the different populations is 5, the type of mutation number two obtains better results.

  - When the number of the elements that forms the different populations are 25, , the type of mutation number one obtains better results, although the difference is quite small.

- Crossover percentage: To compare this, it has been made experiments with 2 different lengths:

  - When the number of elements that forms the different populations is 5, the best results are obtained using 50 as crossover percentage, but there is not a lot of difference with the results obtained with 10 as percentage, and the worst results are obtained using 90 as crossover percentage.

  - When the number of elements of the different populations is 25, the best result is obtained in one of the executions having as percentage 90, although looking the means of

the distances of the best solutions obtained, the best option is to using 10 and the worst one is using 90 as percentage.

5.2.2 "Greatest sin in a defined interval"

Given an interval, the objective of the algorithm is trying to find the real number whose sin is the biggest one of all the numbers that belongs to the interval. It is clear that there can be executions of the algorithm that the best result obtained by the algorithm can't be different of the "real best result".

In this problem, every member of the different populations represents a real number that belongs to the interval.

In this problem, the user will have to introduce always the next parameters:

- Length of the elements.
- Number of elements of the different populations.
- The lower limit of the interval.
- The upper limit of the interval.
- The iteration criterion.
- The number of points in the crossover process.
- The type of mutation.
- % of crossover.

As it was said in the method "void jButton1_actionPerformed(ActionEvent e)" of the class FrameGAInterval, the user will have to introduce more data if the iteration criterion selected is less than 4:

- If iteration criterion is 1, it will appear a window (instance of FrameData) asking the number of iterations.
- If iteration criterion is 2, it will appear a window(instance of FrameData2) asking the number of iterations without change
- If iteration criterion is 3(instance of FrameData3), it will appear a window asking 2 parameters:
  - The number of iterations without change until force a mutation.
  - The number of forced mutations that the user wants to force.

Analysis of obtained results

About this problem, it has to be said that the lower limit of the interval chosen to make the experiments is 1.5 and the upper limits has been 4.64.

From the results obtained following the methodology exposed previously, we can obtain several conclusions about the different parameters that appear in this problem:

- Length of the elements: when the length is very small, obtain the greatest number is easier than if the length is quite big. This is very normal, because when the length is smaller, the number of possible numbers where the algorithm has to look for the best is smaller. That is the reason that the best value of the sin has been reached when the length is 5. In the other tested cases, where length is 10 or 25, the results of the sin are very good too.

- Number of Elements in the populations: when the number of elements is very big, the results obtained are better than when the number of elements in the different populations is smaller. This is logic, because if the different populations have more elements, the algorithm will be exploring more possible solutions and probably in more different zones of the solutions apace. In this problem this fact happens: the mean of the results obtained when the number of elements is 5 is the worst, and the one obtained when the number of elements is 25 is the best, although it has to be said that in all the cases the sin values of the best elements are very good.

- Iteration criterion used: In this problem, the difference between the results obtained one or other criterion is very small. Then, taking in mind the cost in iterations of the iteration criterion number six, probably this one is the worst of the three for this problem.

- Number of points used in the crossover: To compare this, it has been made experiments with 2 different lengths:

  - When the elements that forms the different populations have length 5, the best option seems to be using 2 points of cross, and the worst one seems to be using 3 points of cross.

  - When the elements of the different populations have length 25, the best option seems to be using 2 points of cross, that obtains results very similar to use one point of cross, and the worst one seems to be using 3 points of cross.

- Type of mutation used: To compare this, it has been made experiments with 2 different lengths:

  - When the elements that form the different populations have length 5, it seems to be better to use the first kind of mutation.

  - When the elements of the different populations have length 25, again it seems to be better to use the first kind of mutation.

- Crossover percentage: To compare this, it has been made experiments with 2 different lengths:

  - When the number of elements that forms the different populations is 5, the best results are obtained using 10 as crossover percentage and there is no big difference between using 90 or 50 as crossover percentage.

  - When the number of elements of the different populations is 25, the three options obtain excellent solutions. It seems not to be one that is clearly better than the other ones.

- Interval defined: To compare this, it has been other upper limit (7.8), and like this we have two intervals: the interval used in the other experiments [1.5, 4.64] and the new interval [1.5, 7.8]: it has been compared 6 executions of both intervals, and it has been better used the first interval. In all the previous executions the value of number elements were 5, the iteration

criterion selected were the number 4, number of points used were one, the type of mutation was the first kind and the percentage of crossover used were 10. The lengths used were 5, 10 and 25.

5.2.3 "Most ones followed by one cero"

Given a length, the purpose of the algorithm is trying to find the binary number that has more ones followed by cero, I mean, more couples "10". It is clear that there can be executions of the algorithm that the best result obtained by the algorithm can't be different of the "real best result".

In this problem, every member of the iteration represents a binary number. For example:

- If the length of elements is 4, the binary number "1111" has 0 couples "10".
- If the length of elements is 4, the binary number "1010" has 2 couples "10".

In this problem, the user will have to introduce always the next parameters:

- Length of the elements.
- Number of elements of the different populations.
- The iteration criterion.
- The number of points in the crossover process.
- The type of mutation.
- % of crossover.

As it was said in the method "void jButton1_actionPerformed(ActionEvent e)" of the class FrameGAB, the user will have to introduce more data if the iteration criterion selected is less than 4:

- If iteration criterion is 1, it will appear a window (instance of FrameData) asking the number of iterations.
- If iteration criterion is 2, it will appear a window (instance of FrameData2) asking the number of iterations without change.
- If iteration criterion is 3(instance of FrameData3), it will appear a window asking 2 parameters:
  - The number of iterations without change until force a mutation.
  - The number of forced mutations that the user wants to force.

Analysis of obtained results

From the results obtained following the methodology exposed previously, we can obtain several conclusions about the different parameters that appear in this problem:

- Length of the elements: when the length is very small, obtain the best possible element is easier than if the length is quite big. This is very normal, because when the length is smaller,

the number of possible elements (where the algorithm has to look for the best) is smaller. It can be seen too that when the length is bigger, the fitness of the normal best element obtained is further from the best possible fitness.

- Number of Elements in the populations: normally when the number of elements is very big, the results obtained are better than when the number of elements in the different populations is smaller. But in this experiment, this hasn't happened: in all the executions an element with the maximum possible fitness has been obtained except in one of the executions where the length was 5 and the number of elements was 10. This has been produced because in the reproduction process, there was only one element with the best possible fitness and it wasn't selected.

- Iteration criterion used: In this problem, when the length of the elements is and odd number, there is more than one solution with the best possible fitness. This is the reason that produces that in our experiments, like we have used a value of 5 for the parameter length, in both executions of the 3 kinds of criteria, an element with the best possible fitness has been obtained as the best element.
  To end the study of this parameter in this problem, as in the rest of the problems developed, the number of iterations using the iteration criterion number 6 is much bigger than using the other two ones.

- Number of points used in the crossover: To compare this, it has been made experiments with 2 different lengths:

  - When the elements that forms the different populations have length 5, an element with the best possible fitness are obtained in both executions using 1, 2 or 3 points of cross.

  - When the elements of the different populations have length 25, it is not obtained the best possible result (12 pairs "10") using neither 1, 2 nor 3 points of cross. The results obtained using one, two or three points are very similar: the fitnesses of the best elements obtained with 1 cross point are both 7, the fitnesses of the best elements obtained with 2 cross points are 7 and 9, and the fitnesses of the best elements obtained with 3 cross points are 7 and 8.

- Type of mutation used: To compare this, it has been made experiments with 2 different lengths:

  - When the elements that form the different populations have length 5, the best elements obtained using both kinds of mutations were the best element of the different initial population. Then not any special conclusion can be obtained of these tests.

  - When the elements of the different populations have length 25, it is not obtained the best possible result (12 pairs "10") using the first or second kind of mutation. It has to be said too that, analyzing the results, the second kind of mutation obtain better results the first one.

- Crossover percentage: To compare this, it has been made experiments with 2 different lengths:

  - When the number of elements that forms the different populations is 5, the best results are obtained using 10 or 90 as crossover percentage and the worst results are obtained using 50 as crossover percentage.

- When the number of elements of the different populations is 25, the three options obtain the best possible solution. This best solution was in the initial populations in the 6 executions (2 per option of crossover percentage).

5.2.4 "Greatest real number"

The objective of the algorithm is to find the greatest real number between 0 and 100. It is clear that there can be executions of the algorithm that the best result obtained by the algorithm can't be different of the "real best result".

In this problem, every member of the iteration represents a real number. It is obvious that the length of the elements in this problem is going to be 1.

In this problem, the user will have to introduce always the next parameters:

- Number of elements of the different populations.
- The iteration criterion.
- The type of mutation.
- % of crossover.

As it was said in the method "void jButton1_actionPerformed(ActionEvent e)" of the class FrameGAReal, the user will have to introduce more data if the iteration criterion selected is less than 4:

- If iteration criterion is 1, it will appear a window (instance of FrameData) asking the number of iterations.
- If iteration criterion is 2, it will appear a window(instance of FrameData2) asking the number of iterations without change
- If iteration criterion is 3(instance of FrameData3), it will appear a window asking 2 parameters:
  - The number of iterations without change until force a mutation.
  - The number of forced mutations that the user wants to force.

Analysis of obtained results

From the results obtained following the methodology exposed previously, we can obtain several conclusions about the different parameters that appear in this problem:

- Number of Elements in the populations: normally when the number of elements is very big, the results obtained are better than when the number of elements in the different populations is smaller. This is what happens in the executions of this problem.
- Iteration criterion used: Usually the best results are obtained by the criterion number six, but here the best results were obtained by the criterion number four. This has happened because the initial populations created in the executions of the other two criteria were much worse than the ones created in the criterion number four.

- Type of mutation used: In the experiments realized, the second kind of mutation obtains better results that the first kind of mutation. But this doesn't have a lot of importance, because in both executions of both kinds of mutations, the best element obtained was in the initial population created by the algorithm.

- Crossover percentage: To compare this, it has been made experiments with 2 different lengths:

  - When the number of elements that forms the different populations is 5, the best results are obtained using 50 as crossover percentage and the worst results are obtained using 90 as crossover percentage.

  - When the number of elements of the different populations is 25, the best results are obtained using 10 as crossover percentage and the worst results are obtained using 90 as crossover percentage.

5.2.5 Greatest binary real number"

Given the length of the binary elements and the length of the real part of the elements, the objective of the algorithm is trying to find the greatest binary real number. It is clear that there can be executions of the algorithm that the best result obtained by the algorithm can't be different of the "real best result".

In this problem, every member of the iteration is composed by several integer numbers, like the normal binary number. But his value is real .For example:

If length is 4 and the length of the real part 2, the binary number "1111" represents the real number 3.75(2+1+0.5+0.25).

In this problem, the user will have to introduce always the next parameters:

- length(number of bits) of the elements.

- The length of the real part of the elements.

- number of elements of the different populations.

- the iteration criterion.

- the number of points in the crossover process.

- the type of mutation.

- % of crossover.

As it was said in the method "void jButton1_actionPerformed(ActionEvent e)" of the class FrameGABReal, the user will have to introduce more data if the iteration criterion selected is less than 4:

- If iteration criterion is 1, it will appear a window (instance of FrameData) asking the number of iterations.

- If iteration criterion is 2, it will appear a window(instance of FrameData2) asking the number of iterations without change

- If iteration criterion is 3(instance of FrameData3), it will appear a window asking 2 parameters:

  - The number of iterations without change until force a mutation

  - The number of forced mutations that the user wants to force.

Analysis of obtained results

The length of the real part o the elements of this problem have been set to 2. This element is not important to study the different possibilities of this problem.

From the results obtained following the methodology exposed previously, we can obtain several conclusions about the different parameters that appear in this problem:

- Length of the elements: Generally speaking, when the length is very small, obtain the greatest number is easier than if the length is quite big. But this doesn't happen in the experiments realized in our case. This is because the initial populations that appear in our experiments have been better in the 2 executions of the biggest length.

- Number of Elements in the populations: when the number of elements is very big, the results obtained are better than when the number of elements in the different populations is smaller. This is logic, because if the different populations have more elements, the algorithm will be exploring more possible solutions and probably in more different zones of the solutions apace.

- Iteration criterion used: between the iteration criterion number 4 and the iteration criterion number 5 there haven't been a lot of differences in the best elements obtained, although it is a bit better the iteration criterion number 5.

  The one that obtains the best results is the iteration criterion number 6. Using this iteration criterion, the best element was found, even with a population whose best element (11010) was quite far from the best possible element (11111). It is true too that with this iteration criterion, the algorithm has a much more time cost than with the other ones, as we have said in the other problems too.

  Finally, as it was said in the "greatest binary number", the user has to decide what is more important for him to select one of these 3 iteration criteria: the time or the fitness of the best element obtained.

- Number of points used in the crossover: To compare this, it has been made experiments with 2 different lengths:

  - When the elements that forms the different populations have length 5, the best elements obtained using 1 or 2 points of cross were the best element of the different initial population. However, when the number of points used in the cross is 3, the best result was obtained in one of the cases and is the only one that obtains a better solution than the best element in its initial population. Then, it can be said that for this case, the best option is used 3 points of cross.

- When the elements of the different populations have length 25, it is not obtained the best possible result (25 ones) using neither 1, 2 nor 3 points of cross. However the mean of the results obtained with 2 points of cross is the best and the worst is the mean of 3 points.

- Type of mutation used: To compare this, it has been made experiments with 2 different lengths:

  - When the elements that forms the different populations have length 5, the best elements obtained using both kinds of mutations were elements of the different initial population. The best one seems to be the second type.

  - When the elements of the different populations have length 25, it is not obtained the best possible result (25 ones) using the first or second kind of mutation. It has to be said too that, analyzing the results, the first kind of mutation obtain better results.

- Crossover percentage: To compare this, it has been made experiments with 2 different lengths:

  - When the number of elements that forms the different populations is 5, the best results are obtained using 50 as crossover percentage and the worst results are obtained using 90 as crossover percentage(but this results are very similar that percentage=10).

  - When the number of elements of the different populations is 25, two of the three options obtain the best possible solution(10% and 50%) and 10% in both executions. In this case, it can be said that the best option is 10%, although the three options options very similar results.

6 Conclusions

In this chapter it is going to be exposed some conclusions that have been extracted from the elaboration of the previous chapters.

From the development, introduction and application of the genetic algorithm the next conclusions have been extracted:

- As it was explained in the section 2.2.7, there has been taken several ways to develop the different processes that form a genetic algorithm: some of them have been taken from the bibliography (for example the different kinds of crossover) and other ones, for example the different kinds of mutations, have been generated making interpretations of the specified bibliography.

- Conclutions relatives to Java:

  - The main advantage of Java over other object oriented languages as for example C++ is the automatic memory management that makes the implementation much easier to realize.

  - The main advantage of Java over languages as C is that let the programmer use object oriented paradigm that let the user get all the advantages that offers the object oriented programming.

  - Java has other advantages as:

    - Offers the possibility of implementing user interfaces, with a very rich set of possibilities of implementation.

    - Offer a very easy way to use programming elements as

      - ✓ Structures data: list, arrays,….

      - ✓ Threads.

    - Gives to the implementer the possibility of developing a code that is valid to generate executables in different operative systems: UNIX, MAC OS X, Windows, etc.

- Referred to the application of the algorithm to the six different problems (chapter 5), the next conclusions can be extracted referred to the different possible parameters:

  - Respect to the length of the elements: when the length is very small, obtain the greatest number is easier than if the length is quite big.

  - Respect to the number of elements, normally when the number of elements is very big, the results obtained are better than when the number of elements in the different populations are smaller. As it was said in chapter number 5, this fact is logic, because if the different populations have more elements, the algorithm will be exploring more possible solutions and probably in more different zones of the solutions apace.

  - Respect to the iteration criteria, the one that generally obtain the best results are the number 6. The problem of this criterion is that its time cost is much bigger than the other 2 compared criteria. Where the other two criteria obtain quite good results, it is not necessary to use the criterion number 6.

- Respect to the points of cross in the crossover process, there is no one that is clearly the best. In some of the developed problems, the best option are the third one (for example in “TSP” or in "Greatest binary number"), in others the best options are the first and second (for example in "Greatest sin in a defined interval").

- Respect to the kinds of mutations, sometimes the first type of mutation obtains better results than the first one and in other times viceversa. It can’t be said that one of the options is clearly better than the other ones.

- Respect to the crossover percentage, it can be said that the worst results are usually obtained using as crossover percentage the value “90” and the best one is “10”, although in several times using “10” and “50” as crossover percentage the results obtained are very similar. This is very logic, because crossover process is very important in genetic algorithm, and when you put a very high percentage, there are less elements that participate in the crossover process.

- It is very important the influence of the initial population: If the initial population doesn’t have any member of a good zone of the solution space, it is going to be more difficult that the genetic algorithm find a good solution.

- Some future works taking the developed application as initial point might be the next:

  - Using the interface GA new classes that resolve new kind of problems can be created.

  - New kinds of kind of mutations can be implemented.

  - New kinds of crossover processes can be added, possibly with more points of cross.

  - During, for example, the processes of mutation, can be added a possibility for every bit to mutate, I mean, not consider like in the actual application that all the bits has the same probability of mutation.

  - A probability of selection can be added to the selection during the crossover.

  - In future works can be added 2 parameters more to be configured by the user: the mutation probability and the crossover probability. Even, it can be offered the possibility to define them from every member of the population.

Bibliography

- ***Books and articles***

[1] Aliev R.A., Aliev R.R., *Soft Computing and its applications*, World Scientific Co. Pte. Ltd., 2001, pages 251-283.

[2] Juez Martel P., Díez Vegas F. J.,*Probabilidad y Estadística en Medicina: Aplicaciones en la Práctica Clínica y en la Gestión Sanitaria*, Ed. Díaz de Santos, Madrid 1996.

[3] Kecman V., *Learning and Soft Computing: Support Vector Machines, Neural Networks, and Fuzzy Logic Models*, The MIT Press., 2001.

[4] Mitchell Melanie, *An Introduction to Genetic Algorithms*, MIT Press, 1999.

[5] Mitra S., Achazya T., *Data Mining Multimedia, Soft Computing and Bioinformatics*, Ed. John Wiley and Soks, pp.67-69, 2003.

[6] Munakata Toshinori, *Fundamentals of the New Artificial Intelligence*, Springer, 1998, pages 65-101.

[7] Pal S. K., Mitra S., *Neuro-Fuzzy Pattern Recognition: Methods in Soft Computing*, Wiley Series on Intelligent Systems, 1999.

[8] Trippi R. R., *Chaos & Nonlinear Dynamics in the Financial Markets: Theory, Evidence, and Applications*, McGraw-Hill/Irwin, 1995.

- ***Web pages***

[WP1] Auton lab Web page article, *Bayes Nets for representing and reasoning about uncertainty*, http:// www. Autonlab. Org/ tutorials/ bayesnet09.pdf**,** February 2008.

[WP2] Adam Marczyk web page article, *Genetic Algorithms and Evolutionary Computation*, http://www.talkorigins.org/faqs/genalg/genalg.html**,** February 2008.

[WP3] Díez Vegas F. J. Web page, http://www.ia.uned.es/~fjdiez/, February 2008.

[WP4] Franco Busettin Web page article, *Genetic algorithms overview*, www.geocities.com/francorbusetti/gaweb.pdf , February 2008.

[WP5] Pradnya Choudhari web page article, *Java Advantages & Disadvantages*, http://arizonacommunity.com/articles/java_32001.shtml, March 2008.

[WP6] Wikipedia Web page, *Artificial neural network*, http:// en. wikipedia. Org/ wiki/ Neural_computing **,** February 2008.

[WP7] Wikipedia Web page, *Evolutionary computation*, http:// en. wikipedia. Org/ wiki/ Neural_computing **,** February 2008.

[WP8] Wikipedia Web page, *Java*, http://en.wikipedia.org/wiki/Java_(programming_language), March 2008.

Appendix A: user‘s guide

- Generation of the executable file

To generate the .exe file, it has been used the application “Borland Jbuilder 2005”. It has been generated only the executable for Windows, but using that program executables for MAC OS X or UNIX can be generated from the implemented source code.

- Main window:

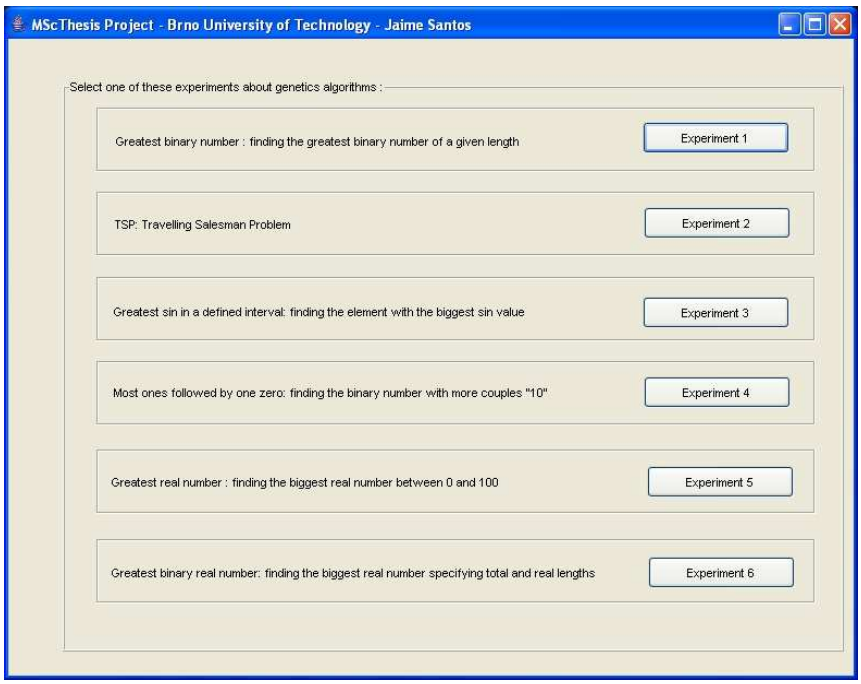


Figure 2: Main Window of the application

It can be seen in this Figure 2 the aspect that has the main windows of the application. The user will have to press the button that represents the experiment that he has to execute. This buttons will be:

- If the problem that the user wants to develop is the problem called "Greatest binary number", the user has to press the button “Experiment 1”.Due to this action, it will appear a window with the title "Greatest binary number".

- If the problem that the user wants to develop is the problem called "TSP", the user has to press the button “Experiment 2”. Due to this action, it will appear a window with the title "TSP".

- If the problem that the user wants to develop is the problem called "Greatest sin in a defined interval", the user has to press the button “Experiment 3”. Due to this action, it will appear a window with the title "Greatest sin in a defined interval".

- If the problem that the user wants to develop is the problem called “Most ones followed by one cero", the user has to press the button “Experiment 4”. Due to this action, it will appear a window with the title "Most ones followed by one zero".

- If the problem that the user wants to develop is the problem called “Greatest real number”, the user has to press the button “Experiment 5”. Due to this action, it will appear a window with the title "Greatest real number".

- If the problem that the user wants to develop is the problem called "Greatest binary real number", the user has to press the button “Experiment 6”. Due to this action, it will appear a window with the title "Greatest binary real number".

➢ Greatest binary number window:

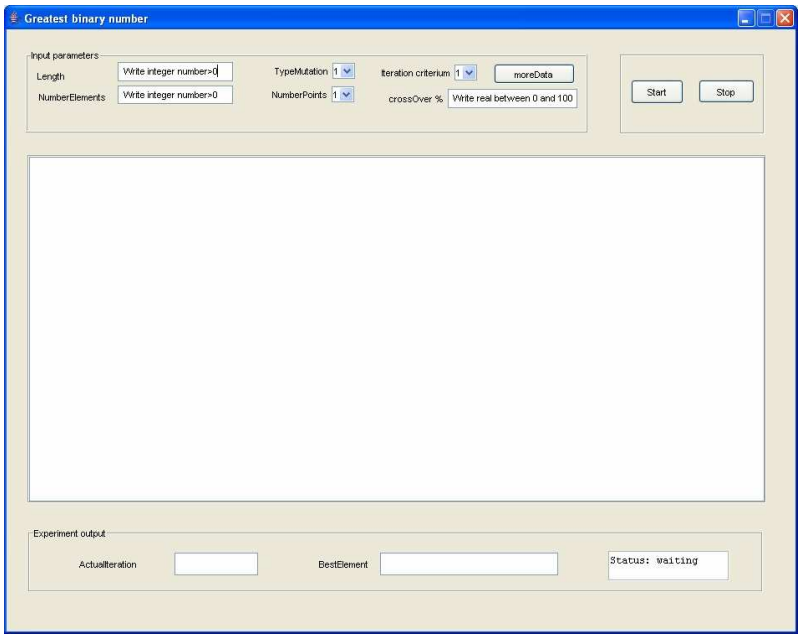


Figure 3: main window of the problem "Greatest binary number"

As it can be seen in the Figure 3, the window is divided in 4 main parts or panels:

- A panel called “Input parameters” that contains the input parameters that the user has to define before using the aplication. These parameters are:

  - Length: In this field the user has to set the length of the binary number.

  - numberElements: In this field the user has to specify number of elements of the initial population.

  - TypeMutation: In this field the user has to particularize the kind of mutation that the user wants to be selected. There are only two options.

  - NumberPoints: In this field the user has to set the number of points that he wants to be used in the crossover process.

  - Iteration criterium: In this field the user has to particularize the iteration criterion that he wants to be used.

  - crossover %: In this field the user has to specify the percentage of crossover of the algorithm.

- A panel that has 2 buttons: Start and Stop. Pressing one time the Start button the application begins and pressing the Stop button the algorithm is stopped.

- A panel with white background, where it will appear the results obtained in the different iterations of the algorithm.

- A panel called “Experiment output”, where the user can see the best element obtained after the iteration “actual iteration”. It can be seen too the status of the algorithm:

  - “waiting”: the user hasn’t pressed the button start or not all the needed data have been introduced.

  - “processing”: the algorithm is running.

  - “ended by user”: the algorithm was running until the user pressed the button “Stop”.

  - “Ended”: the algorithm has ended its execution alone (without an user intervention).

It has to be said that to introduce the value of “Length”, ”numberElements” or “crossover %”, the user will have to delete the content of the field and put the new value that he desires.

Finally, if the user wants to use the iteration criterion number one, two or three, the user will have to specify the value and after this specify the extra data needed to execute the algorithm. To specify this extra data, the user has to click the button called “moreData”. Due to this action, it will happen the next:

- If Iteration criterion is 1, it will appear a window titled “Introduce extra data of iteration criterion number one”.

- If Iteration criterion is 2, it will appear a window titled “Introduce extra data of iteration criterion number two”.

- If Iteration criterion is 3, it will appear a window titled “Introduce extra data of iteration criterion three”.

After introducing all the parameters, the user has to select with the mouse the button “Start”. If the user wants to stop the execution, the user has to press the button called “Stop”.

To end this section, it has to be said two things:

- If Iteration criterion is 1, 2 or 3 and the user hasn’t specified the extra data and clicks the button start, the application is not going to do anything at all.

- If the user introduce a numerical value out of the range of the parameter or don´t introduce anything in one or more of the parameters that appear in this window (window titled “Greatest binary number”), the application will warn the user about this.

Example:

If the user wants the length of elements of the algorithm to be 4, the number of elements in the initial population to be 4, the kind of mutation to be the first one, the number of points in the crossover process to be 1, the iteration criterion to be the fourth one and the percentage of the crossover to be 15 %, the input parameters will show the next aspect (see Figure 4):

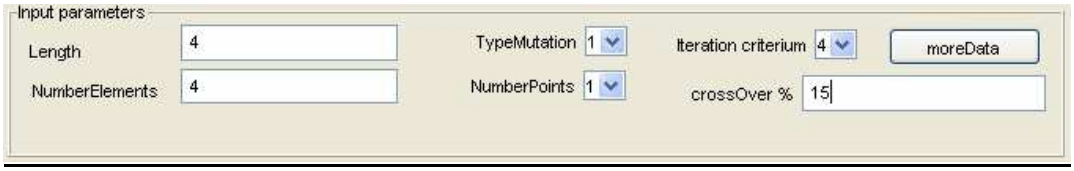


Figure 4: input parameters panel of the example of the "Greatest binary number" problem section

After this, if the user clicks the “Start” button, it will appear something like:

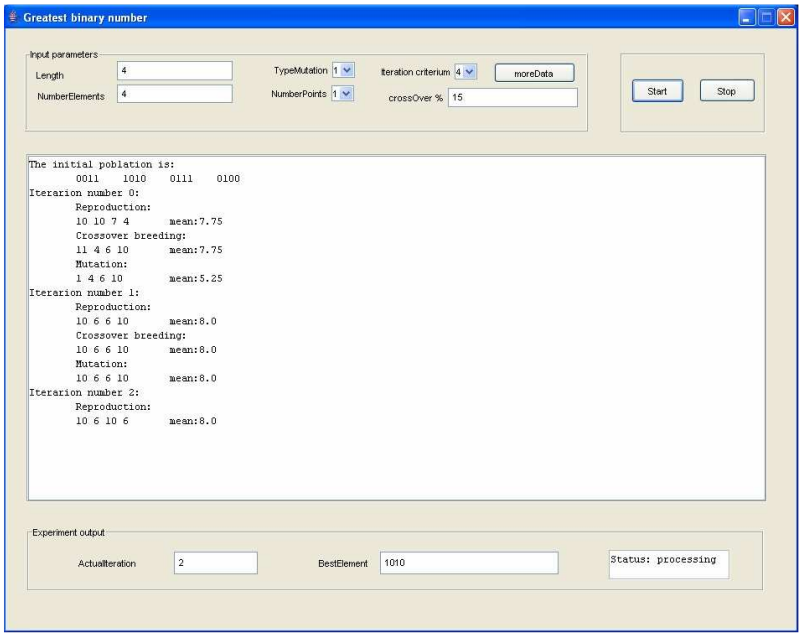


Figure 5: Example of “Greatest binary number" window with status “processing”

In the previous figure, Figure 5, it can be seen that the status of the algorithm is “processing” (bottom right corner).

When the execution has ended, the application will show something similar to:

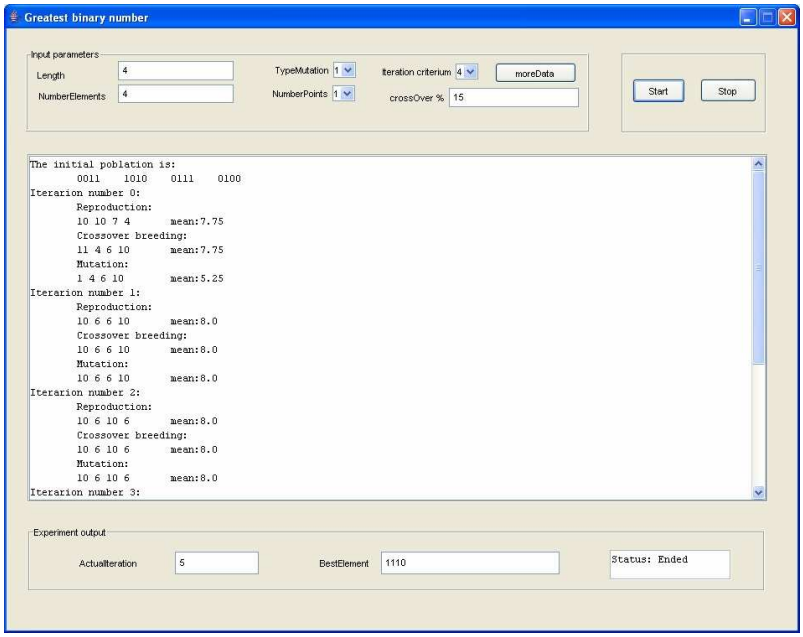


Figure 6: Example of “Greatest binary number" window with status “Ended”

In the previous picture, Figure 6, it can be seen that the status of the algorithm is “Ended” (bottom right corner).

- "TSP" window:

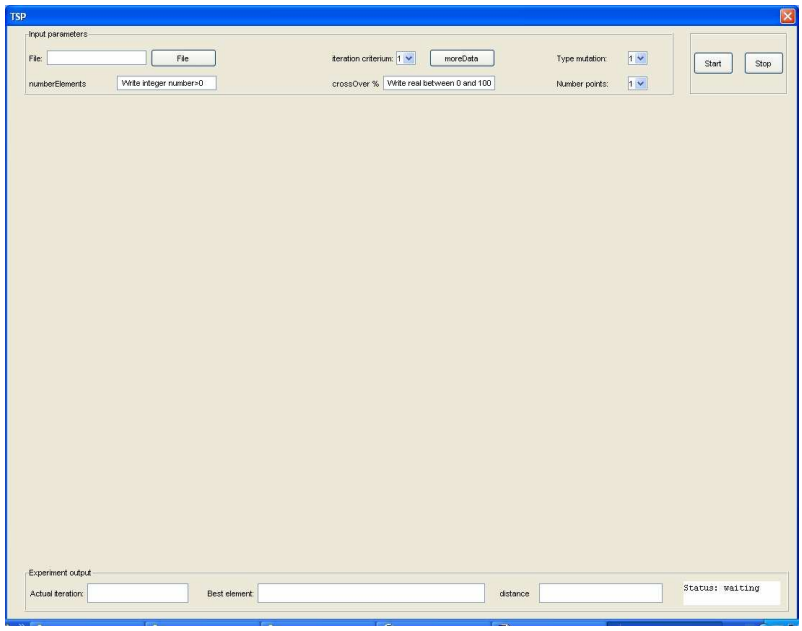


Figure 7: main window of the problem "TSP"

As it can be seen in the Figure 7, the window is divided in 4 main parts or panels:

- A panel called “Input parameters” that contains the input parameters that the user has to define before using the aplication. These parameters are:

  - File: In this field the user has to set the file that is going to be the input file, I mean, the file that contains the positions of the different cities. To select this file, the user has to press the button “File” and then it will appear a window where it can be chosen the input file.

- The rest of elements have the same meaning that the elements with the same name in Greatest binary number window.

o A panel with 2 buttons, Start and Stop, as in Greatest binary number window.

o A panel called “Experiment output”, as in Greatest binary number window.

o An “invisible” panel (because doesn’t have borders and has the same background colour than the content pane) between “Input parameters” and “Experiment output”, where it will appear

- points that represents the different cities.
- red lines between the points that represents the different ways of the path of the best solution found by the algorithm.
- Numbers in black colour that represents the names of the cities.
- Numbers in red that represents the order that the algorithm has followed to paint the different ways of the path that represents the best element.

Now it is going to be shown one example of the execution with this window when the algorithm has ended its execution. The file used is in C:\Documents and Settings\Jaime\Mis documentos\burma14.tsp (contains the coordinates x and y of 14 different cities), the number of elements chosen is 50, the iteration criterion selected is 4, the percentage of crossover chosen is 10, the type of mutation selected is 1 and the number of points in the crossover process selected is 3. The result obtained is:

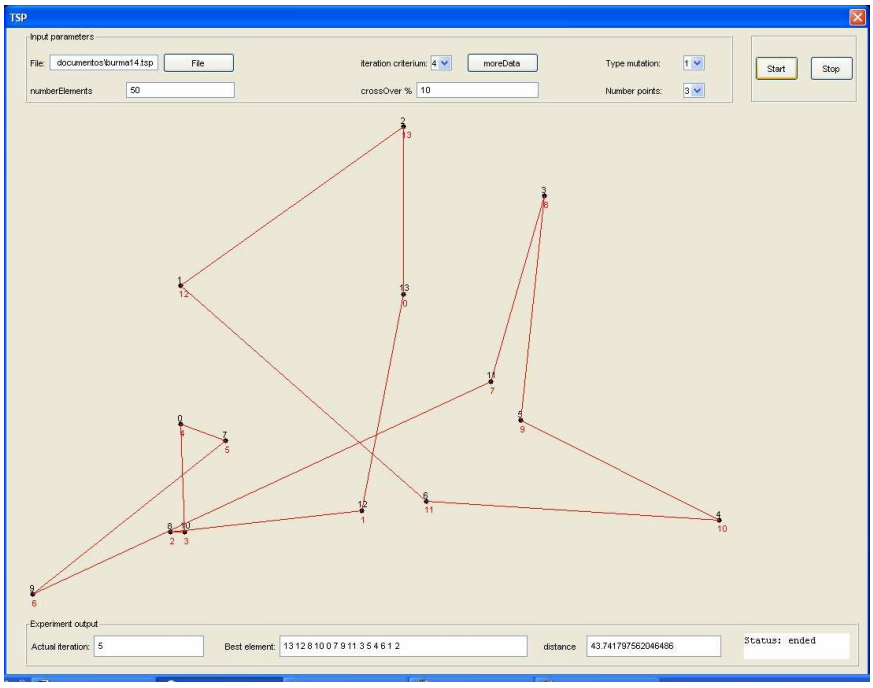


Figure 8: Example of “TSP" window with status “ended”

As it can be seen in Figure 8, the best element found is going from city 13 to city 12, then to city 8, then to city 10, after this go to city 0, after this go to city 7, after this go to city 9, after this go to city 11, after this go to city 3, after this go to city 5, after this go to city 4, after this go to city 6, after

this go to city 1 and finally go to city 2. The total distance covered is 43.741797562046486.

- Greatest sin in a defined interval window:

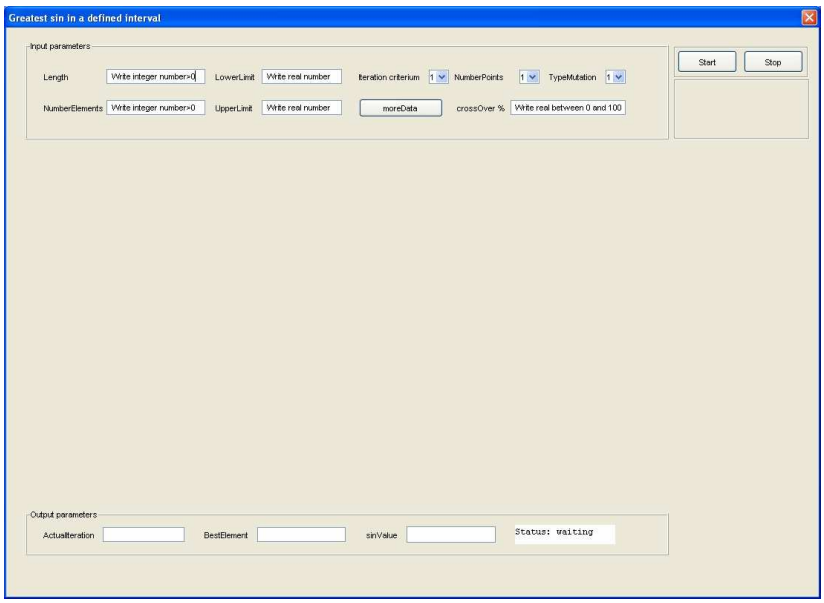


Figure 9: main window of the problem "Greatest sin in a defined interval"

As it can be seen in Figure 9, this window is very similar to "TSP" window, except in three things:

- A new panel just under the panel of the buttons "Start" and "Stop" where it will appear the legend of the graphic that is going to appear in the middle of the window.
- This window has 2 fields that haven't appeared before referred to the interval where it is going to look for the maximum "sin value": "lowerLimit", where the user has to specify the lower limit of the interval and "upperLimit" where the user has to indicate the upper limit of the interval.
- In the panel called "Output experiment" it is shown a new data, the value of the sin of the best element.

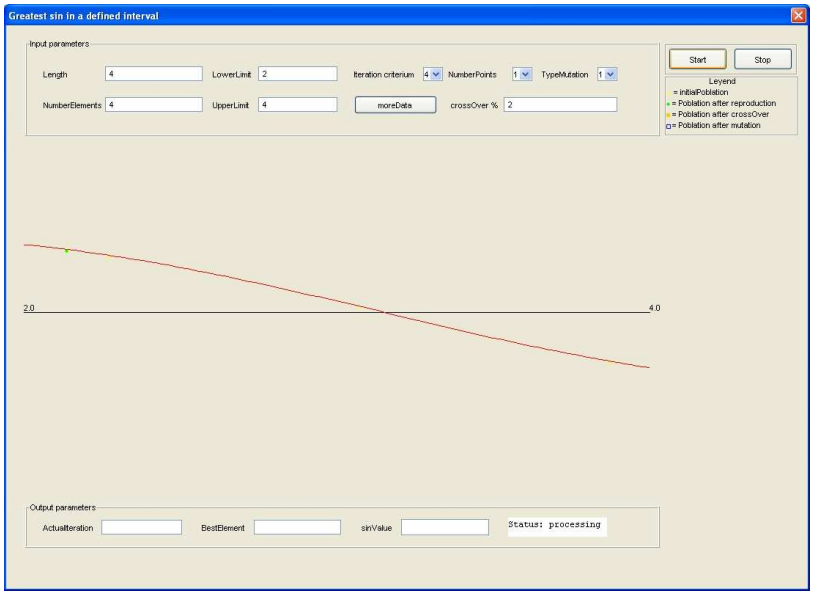


Figure 10: Example of "Greatest sin in a defined interval "window with status "processing"

In the previous picture, Figure 10, it can be seen that the status of the algorithm is "processing" (bottom right corner). It can be seen 4 yellow points, that seeing the legend, represents the 4 elements of the initial population. After the reproduction process, we see that we have 4 elements that represent the same element (element drawn in green).

The next picture represents the end of other execution with the same input data:

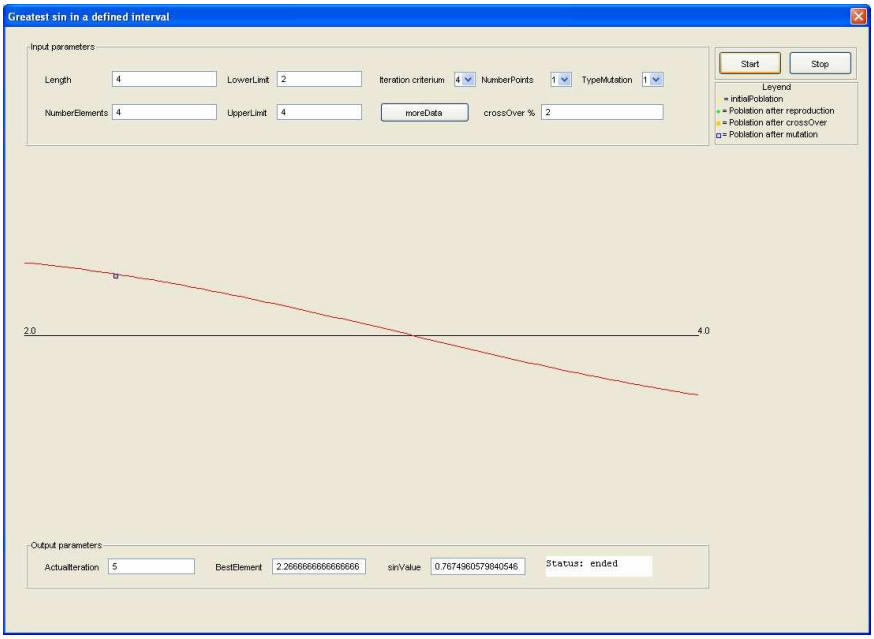


Figure 11: Example of "Greatest sin in a defined interval "window with status "ended"

In the previous picture, Figure 11, it can be seen that the status of the algorithm is "Ended" (bottom right corner) and that the final population, the population after the process of mutation is formed by 4 elements that represents the same element: element represented by the upper left corner of a blue square in the graphic.

- Most ones followed by one zero window:

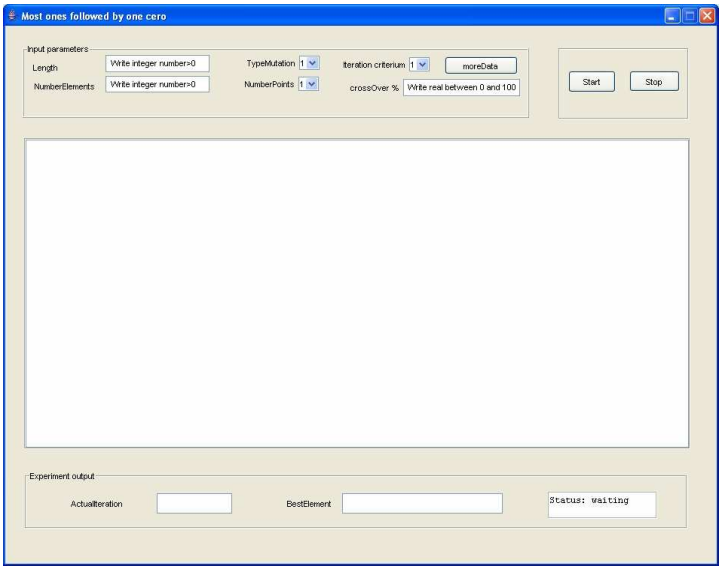


Figure 12: main window of the problem " Most ones followed by one zero"

This window, Figure 12, has the same elements that the Greatest binary number window.

- Greatest real number window:

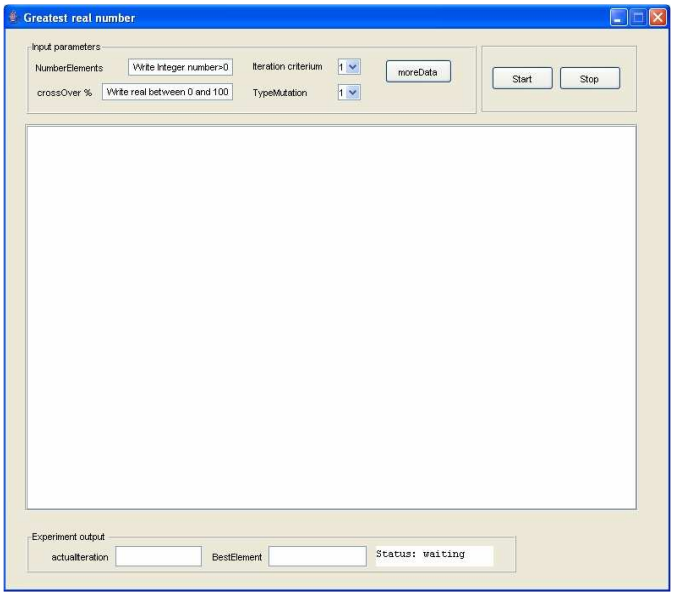


Figure 13: main window of the problem "Greatest real number"

There is nothing special to say about this window (Figure 13), except that the elements of this window are the same elements that the elements with the same name of the window titled "Greatest binary number".

- Greatest binary real number window:

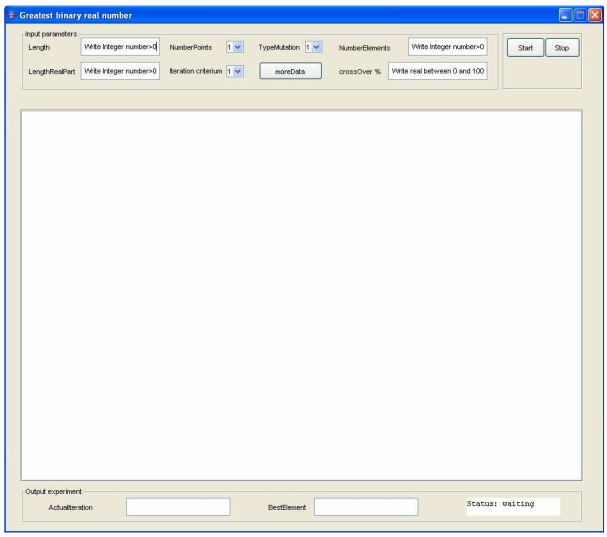


Figure 14: main window of the problem "Greatest binary real number window"

This window (Figure 14) has only one more element than the Greatest binary number window: the option "LenghRealPart", where the user has to specify the number of bits of the binary number that forms the real part of it.

Appendix B: CD content

The CD contains two folders:

- "Documentation": is the folder that contains the documents relatives to the documentation. It is formed by the next files:
  - "MScThesis.doc": is the file that contains the memory that has been printed.
  - "MScThesis.pdf": is the "pdf" version of the file "Memory.doc".

- "src": is the folder that contains the needed source code of the application and the executable file of the application.

- "Experiments": is the folder that contains the Microsoft Word documents that stores the information obtained in the different experiments realized over the different problems specified in section 5.2 of the memory.