· U  C ·

Evolutionary Computation

2021/2022

Project

Ernesto Costa
João Macedo

1 Introduction

We present here the practical homework, part of the students' evaluation process of this course. This work is to be done autonomously by a group up to **two** students. Each group must indicate by preference order **five** projects, sending both the **code** and the **title** of each project by email to jmacedo@dei.uc.pt, no later than **19 of March**. The assignment of the projects to each group will be communicated no later than **26 of March**. The deadline to upload the project is, is **14 of May** via **UCStudent**. Besides a **document** you also have to provide all the **code** used, together with a short **user manual** that enables the complete replication the experiments and verify the results.

The quality of your work will be judged as a function of the value of the technical work, the written description, and the **public defence**. All sources used to perform the work (including the code) must be clearly identified. The document may be written in Portuguese or in English, using a word processor of your choice[1]. The written report is limited to **10** pages long, but in special, justified, cases (e.g., the need of presenting many images and/or tables), that number may be increased accordingly. The document should be well structured, including a general introduction, a description of the problem, the experimental setup, the analysis of the results, and a conclusion. **The final mark will be given to not to the group but to each student individually**.

There are two types of problems in the following list. A first group, includes those that try to do a statistical comparison of different algorithms or the same algorithm with different values for the parameters. Here, the choice of the benchmark problem is not critical but must be justified. In these cases we value the technical correction of the design of the experiment and the critique of the obtained results. A second group, has the main goal of finding a high performing algorithm for a specific kind of problem. In that case, we will value the novelty of the algorithm proposed and the critique of existent approaches. That said, the production of new code is not a goal in itself, and that's enough to mention the source of the used code.

2 Experimentation

To do the work the student may consult any source he/she wants. Nevertheless, plagiarism will be not allowed and, if detected, it will imply failing

[1] We strongly suggest the use of LaTeX.

the whole course. While doing the work and when submitting it, you should pay particular attention to the following aspects (whose relative importance depends on the type of the work done):

- description of the general architecture of the algorithm used;
- description of the representation, variation operators and fitness function used;
- description of the experiment, including a table with the parameters used;
- description of the measures used for the statistical work: quality of the final result, efficacy, efficiency, diversity, or any other most appropriate;
- the minimum number of runs is 30;
- the comparisons must be fair, i.e., you should give to all variants the same resources (number of evaluations);
- in certain projects you may choose the benchmarks problems. We provide some examples below, but you are free to choose different ones, indicating a short justification for your choices.

Do not forget, besides what was just said, that it is fundamental: (1) to do a correct statistical analysis, including the reasons for choosing a particular method; (2) to do an informed discussion about the results obtained; (3) to put in evidence the advantages of the chosen alternative.

3 Problems

In the following we present a brief description of each project. Feel free to ask for complimentary explanations. Once again: do not forget to provide the code and the title of the projects that you prefer.

3.1 TP1- Variable Variation Operators

Brief Description

Variation operators, like mutation and crossover, have been object of discussion for a long time. Usually, once we choose their probabilities of being used they are kept fixed over the entire run. Nevertheless, many researchers say that crossover is fundamental in the beginning of the evolutionary process, while mutation is most important near the end of the run. Let's try to clarify this issue, using (at least) two benchmark problems of different classes. Some examples of those problems are provided at the end of this text.

Goals

Do some preliminary tests that help you to define reasonable values for the parameters of your Evolutionary Algorithm. Define three versions of your algorithm. One, with fixed, common values, for the probabilities of applying the variation operators during the entire run. A second one, with fixed values for the probabilities of applying the variation operators during a particular number of generations: a high value for the probability of crossover in the first 70% of generations and zero thereafter, while mutation is only applied after 70% of generations have occurred. A third one, with variable number of the probabilities for both crossover and mutation, with the former decreasing over time and the latter increasing over time.

For each version of the algorithm, and each problem, do **thirty runs**. Store quality measures, like the performance at the end of the run and the moment the best result was obtained. Do a statistical analysis of those measures and draw your conclusions.

3.2 TP2- Parent's Selection

Brief Description

Convergence premature is a well known problem of evolutionary algorithms, with some researchers advocating the need to keep the diversity of the popu-

lation at high levels to avoid it. In class we discussed some ways of selecting the parents claiming that some favor diversity and thus positively influence the quality of the attained result. It is time to analise this issue scientifically, using proper benchmark problems.

Goals

Do some preliminary tests that help you to define reasonable values for the parameters of your Evolutionary Algorithm. Define three versions of your algorithm. One, based on roulette wheel, another using tournament selection, and a third one with stochastic uniform sampling.

For each version of the algorithm, and each problem, do **thirty runs**. Store quality measures, like the performance at the end of the run and the diversity over the run. Justify the way you measured the diversity. Do a statistical analysis of those measures and draw your conclusions.

3.3 TP3- Survivors' Selection

Brief Description

Convergence premature is a well known problem of evolutionary algorithms, with some researchers advocating the need to keep the diversity of the population at high levels to avoid it. In class we discussed some ways of selecting the survivals. It is time to analise scientifically if the choice of the survivors mechanism used is important for the final result using proper benchmark problems.

Goals

Do some preliminary tests that help you to define reasonable values for the parameters of your Evolutionary Algorithm. Define three versions of your algorithm. One, based on generational selection, another using steady state selection, and a third one with generational with elitism selection. Do not forget that the number of individuals involved in the two latter options may be relevant.

For each version of the algorithm, and each problem, do **thirty runs**. Store quality measures, like the performance at the end of the run and the diversity over the run. Justify the way you measured the diversity. Do a statistical analysis of those measures and draw your conclusions.

3.4 TP4- Recombination Operators

Brief Description

Crossover is a variation operator that many claim it is essential to promote exploration of the search space contributing to the final quality of the result. There are many types of crossover operators depending on the type of problem and representation used. You are going to study in what way the **recombination operators** promote a balanced search of the space of candidate solutions, avoid premature convergence and, as a consequence, foster the quality of the results obtained. Use benchmark problems, justifying why those problems are helpful in clarifying this issue.

Goals

Do some preliminary tests that help you to define reasonable values for the parameters of your Evolutionary Algorithm. Define three versions of your algorithm. One, with one-point crossover, another with two-points crossover and a third one relying on uniform crossover.

For each version of the algorithm, and each benchmark problem, do **thirty runs**. Store quality measures, like the performance at the end of the run and the moment the best result was obtained. Do a statistical analysis of those measures and draw your conclusions.

3.5 TP5- Mutation Operators in Genetic Programming

Brief Description

Some researchers state that the **mutation operators** influence the performance of genetic programming algorithms. You are going to clarify this issue. For that you will use appropriated benchmark problems.

Goals

Do some preliminary tests that help you to define reasonable values for the parameters of your Evolutionary Algorithm. Define two versions of your algorithm: one, with sub-tree mutation, and another with node mutation. For each version of the algorithm, and each benchmark problem, do **thirty runs**. Store quality measures, like the performance at the end of the run and the moment the best result was obtained. Do a statistical analysis of those measures and draw your conclusions.

3.6 TP6- Self-Adaptation in Evolutionary Strategies

Brief Description

It seems that changing the parameters of an evolutionary algorithm during a run improves the results obtained. For example, the size of the population or the probabilities of the variation operators. We may define ourselves how this happens during the run. But it is a better idea to let evolution decide the best values to use at each moment. This is what we are going to study in the context of the so-called **evolutive strategies**.

Goals

Your work will involve the comparison of two algorithms. One, with a representation that includes the values of the standard deviations of a normal distribution that we use to mutate each gene; another, with a normal representation. The benchmark problems to use are those describing functions whose optimum we want to find.

For each version of the algorithm, and each benchmark problem, do **thirty runs**. Store quality measures, like the performance at the end of the run and the moment the best result was obtained. Do a statistical analysis of those measures and draw your conclusions.

3.7 TP7- *Bloat* in Genetic Programming

Brief Description

One of the most problematic problems when we use genetic programming is the uncontrolled growth of the individuals without a corresponding improvement of their quality, a problem called **bloat**. Over the years there have been many proposals of solutions for this problem. We are going to study a few of them, that are related with the admissible tree depth.

Goals

Do some preliminary tests that help you to define reasonable values for the parameters of your GP algorithm. Use three versions of the basic algorithm: one, without limit for the maximum depth, another with a fixed maximum depth limit, and a third one with a limited depth that may be surpassed if the quality of the offspring is better than the one of the parents.

For each version of the algorithm, and each benchmark problem, do **thirty runs**. Store quality measures, like the performance at the end of

the run. Store also the average depth of the individuals over generations. Do a statistical analysis of those measures and draw your conclusions.

3.8 TP8- Initial Population

Brief Description

When we discussed optimization methods based in one individual only, we stress the problem of choosing the initial point for the search. In the case of population-based methods this seems not to be a problem, for we start from several distinct points of the search space that we choose randomly using an uniform distribution. But we may question if this is the best way to proceed. Let's study the problem.

Goals

The work will be done in the context of function optimization problems. Define two versions of a standard evolutionary algorithm. One, that initializes the population as described above; a second one, that initializes the population based on the idea of dividing the search space in hyper volumes, and choosing the same number of individuals for each hyper volume.

For each version of the algorithm, and each benchmark problem, do **thirty runs**. Store quality measures, like the performance at the end of the run and the moment the best result was obtained. Do a statistical analysis of those measures and draw your conclusions.

3.9 TP9- Unfeasible Individuals

Brief Description

Many optimization problems involve constraints. For a particular problem the choice we made for the representation and variation operators, may produce unfeasible individuals, i.e., individuals that violate some or all the constraints. There are two major ways to deal with this problem. First, we penalize the individuals so their fitness will be low, increasing the probability of their disappearance from the population, Second, we may repair the individuals until they become feasible. Let's see which one is better.

Goals

Choose benchmarks problems with restrictions (e.g., the 0/1 Knapsack Problem). Implement two versions of the standard EA, one with each of the

mentioned methods. For each version of the algorithm, and each benchmark problem, do **thirty runs**. Store quality measures, like the performance at the end of the run. Do a statistical analysis of those measures and draw your conclusions.

3.10 TP10- Mutation Operators for Permutations

Brief Description

Some problems, e.g., N-queens and TSP, whose solution is based on a permutation of integers. Swap mutation is the most common, and simple, operator used for this representation. But there are some more, like the inversion mutation operator and the shift mutation operator. Does the choice of the mutation operator influence the quality of the final result? That's what we are going to study.

Goals

Implement three versions of a standard evolutionary algorithm, one for each of the three mutation operators mentioned above. For each version of the algorithm, and each benchmark problem, do **thirty runs**. Store quality measures, like the performance at the end of the run. Do a statistical analysis of those measures and draw your conclusions.

3.11 TP11- Mutation Operators for Reals

Brief Description

The most common mutation operator used with reals is the one that changes a gene using a gaussian distribution. We may proceed differently, like choosing a new value value within the range of possible values based on a uniform distribution, or using a non-uniform distribution as defined below (the gene x_k will be mutated into x'_k):

$$x'_k = \begin{cases} x_k + \Delta(t, S_k - x_k) & \text{se} \quad \mathit{choice}(0,1) = 0 \\ x_k - \Delta(t, x_k - I_k) & \text{se} \quad \mathit{choice}(0,1) = 1 \end{cases} \tag{1}$$

where S_k is the upper limit and I_k in the lower limit for x_k, and $\Delta(t, y)$ a function that gives, non uniformly, a result in the interval $[0, y]$ and t is a generation counter. $\Delta(t, y)$ is defined by:

$$\Delta(t, y) = y \times r \times (1 - \frac{t}{T})^b$$

where r is a random number between 0 and 1, T the maximum number of generations, and b a system parameter that controls the degree of non uniformity. Let's see in what way these approaches may lead to different results.

Goals

Implement two versions of a standard evolutionary algorithm, one for each of the two mutation operators mentioned above. For each version of the algorithm, and each benchmark problem, do **thirty runs**. Store quality measures, like the performance at the end of the run. Do a statistical analysis of those measures and draw your conclusions.

3.12 TP12- Crossover Operators for reals (I)

Brief Description

Over time different crossover operators were proposed with their proponents claiming their superiority. Let's study this issue. One, is a standard one point crossover, while another of the operators is called arithmetical crossover operator: suppose you want to cross two vectors of reals, x_1 and x_2 to generate two offsprings, defined as follows:

$$\begin{cases} x'_1 = a \times x_1 + (1 - a) \times x_2 \\ x'_2 = a \times x_2 + (1 - a) \times x_1 \end{cases} \tag{2}$$

where a a real number in the interval [0..1]. The operation is done component wise.

Goals

Implement two versions of a standard evolutionary algorithm, one for each of the two crossover operators mentioned above. For each version of the algorithm, and each benchmark problem, do **thirty runs**. Store quality measures, like the performance at the end of the run. Do a statistical analysis of those measures and draw your conclusions.

3.13 TP13- Crossover Operators for Reals (II)

Over time different crossover operators were proposed with their proponents claiming their superiority. Let's study this issue. Besides the standard one-point crossover, another operators was proposed called **heuristic crossover**, that produce **one** offspring from the parents x_1 and x_2 according to the following rule:

$$x_f = r \times (x_2 - x_1) + x_2$$

where r is a real number in the interval [0..1]. This operator introduces a bias for the offspring will be closer of the best of the parents. It may happen that the offspring will be unfeasible, in which case the procedure must be repeated with a different value for r. If after fixed number of trials the resulting offspring is still unfeasible the process ends without generating an offspring.

Goals

Implement two versions of a standard evolutionary algorithm, one for each of the two crossover operators mentioned above. For each version of the algorithm, and each benchmark problem, do **thirty runs**. Store quality measures, like the performance at the end of the run. Do a statistical analysis of those measures and draw your conclusions.

3.14 TP14- Crossover Operators for Permutations

Brief Description

There are several crossover operators for permutations each one claiming its superiority. Let's study this issue for the case of the PMX and of the OX operators.

Goals

Implement two versions of a standard evolutionary algorithm, one for each of the two crossover operators mentioned above. For each version of the algorithm, and each benchmark problem, do **thirty runs**. Store quality measures, like the performance at the end of the run. Do a statistical analysis of those measures and draw your conclusions.

3.15 TP15- Dynamic Environments

Brief Description

Real world problems are quite often dynamic, which implies that the best solution, the optimum, changes over time. There are several methods to adapt evolutionary algorithms to this situation. We are going to study two of these methods in the context of the 0/1 Knapsack Problem.

Goals

There are two interesting approaches to harness dynamics environments. One is based on the concept of **hypermutation**: once a change in the environment is detected the mutation probability is greatly increased for a certain number of generations before returning to the initial, pre-defined, value. Another one is based on the concept of explicit **memory**: for each situation the best individual is kept in a memory, and every time the environment changes, we extract from memory the individuals most appropriated for the new situation.

Implement a version of these two methods and a third one where you proceed without doing nothing special. For each version of the three versions, do **thirty runs**. Store quality measures, do a statistical analysis of those measures, and draw your conclusions. Be cautious when you define the possible ways of changing the environment and, in particularly, with the way you define your quality measure. In this case it is more important to assess the capacity of each of the algorithms to recover after the change.

3.16 TP16- Immigrants Insertion

Brief Description

Convergence premature is a well known problem of evolutionary algorithms, with some researchers advocating the need to keep the diversity of the population at high levels to avoid it. We will study if the inclusion of **new individuals** during the run influencing the diversity improve the performance of the algorithm.

Goals

Do some preliminary tests that help you to define reasonable values for the parameters of your Evolutionary Algorithm. Implement three variants of

your algorithm: one, a standard EA without any modification; another, involving the introduction of random individuals at each generation that replace the worst ones; a third one, where mutations of the best individuaL are introduced at each generation replacing also the worst ones. Notice that you have to define the percentage of immigrants that you will use. For each version of the algorithm, and each benchmark problem, do **thirty runs**. Store quality measures, like the performance at the end of the run. Do a statistical analysis of those measures and draw your conclusions.

3.17 TP17- Multiple Populations

Brief Description

Some researchers advocate the use of more than one population with evolutionary algorithms. Here we are going to use a slightly different approach. For a particular problem we use two different algorithms, but, from times to times, solutions are exchanged between the two populations. We want to compare this approach with another one where, also from time to time, we introduce new random individuals in the population that replace the worst ones.

Goals

Implement the two versions and test their performance against a benchmark problem. It is important to analyze the relevance of the frequency of the migration process and the number of individuals exchanged, for the first algorithm, while, for the second one, the choice of the frequency and the number of random immigrants, is and important parameter. Define appropriated measures of quality, and do a proper statistical analysis.

3.18 TP18- Assignment of Homeworks

Brief Description

The difficulty of assigning the projects to students, when they may choose up to three projects, was referred during class. The professor want to maximize the degree of satisfaction of all students, when their choices are provided sorted. We want to study this problem using an evolutionary algorithm.

Goals

Define the architecture for at least two evolutionary algorithms (or variants of the same algorithm) to solve this problem and implement them. You may use whatever code was already provided or code done by others, indicating the author. Define clearly, and justify, your options for the representation, the variation operators and the fitness function. Admit that the number of projects is greater or equal to the number of students. Remember that the gap between the number of projects and the number of students is an important aspect, making the best solution easier or harder to find.

For each version of the algorithm(s), and each benchmark problem, do **thirty runs**. Store quality measures for each version, like the performance at the end of the run. Do a statistical analysis of those measures and draw your conclusions.

3.19 TP19- Discovery of Physical Laws

Brief Description

Kepler's Laws describe the movement of the planets around the Sun. The Third Law says that the square of the orbital period of a planet is proportional to the cube of the semi-major axis of its orbit:

$$P^2 = kD^3$$

We want to prove that an evolutionary algorithm can be used to (re)discover this law (or another one of the same type ...).

Goals

The following table (see 1) show the values that relate the period (measured in years) with thew distance (measured in $10^{10}m$).

Choose the best alternative of an EA for this problem, including the value of the constant k. You do not need to invent an algorithm yourself, for you may use whatever solution was already proposed. Do a study involving at least two different algorithms (or variants of a particular one). Store quality measures for each version, like the performance at the end of the run. Do a statistical analysis of those measures and draw your conclusions.

Planet	Distance	Period
Mercury	5.79	0.241
Venus	10.8	0.615
Earth	15.0	1.0
Mars	22.8	1.88
Jupiter	77.8	11.9
Saturn	143	29.5
Uranus	297	84
Neptune	450	165

Table 1: Planets: distances and periods

3.20 TP20- Sudoku: an evolutionary solution

Brief Description

The Sodoku is a well-known problem. It is possible to find EAs that solve the problem? That's the task of this project.

Goals

Search for evolutionary algorithms that solve this problem. Implement at least two variants, and analyze statistically their performance. You may use others proposals (and code), but we will valorize positively your own solution.

3.21 TP21- Heart Diseases

Brief Description

There are many indicators that can be used to say if someone has, or has not, a cardiac problem. We are going to try to discover a binary classifier, using an appropriate evolutionary algorithm.

Goals

In https://archive.ics.uci.edu/ml/datasets/Heart+Disease you will find a concrete dataset, involving 14 indicators, with a correct classification, to be used to train, validate and test the classifier, together with complementary information. You must devise the architecture of your solution, and do a proper analysis of the performance of the binary classifier obtained.

3.22 TP22- Who's best: ACO or PSO

Brief Description

ACO and PSO are two algorithms discussed in class, which can be applied to the TSP problem. We know from the No Free Lunch Theorem that there os not an algorithm that is the best for all kind of problems and instances. Nevertheless, for concrete problems and instances one algorithm may be better than all the others.

Goals

Our goal is to see which algorithm, ACO and PSO, is better to solve the TSP problem. Choose three instances of the problem (one small, another medium and a third one of big size). For each instance and each algorithm, do **thirty runs**. Store quality measures for each version, like the performance at the end of the run. Do a statistical analysis of those measures and draw your conclusions. **Note** that you must do a fair choice of the parameters used by each algorithm.

3.23 TP23- Transposition, a bio-inspired operator

Brief Description

In nature there are many different mechanisms for exchanging genetic material between two chromosomes. One such mechanism is called transposition. One version of transposition consists in the identification in a chromosome of a sequence of DNA that is delimited by two identical flanking sequences. Then, we look up the flanking sequence in a second chromosome and if such a sequence exist the transposon of the first chromosome replaces the DNA of the second chromosome after the flanking sequence. The sequence that is replaced in the second chromosome is copied to the place where the transposon was in the first chromosome.

The figure 1 show a possible initial situation.

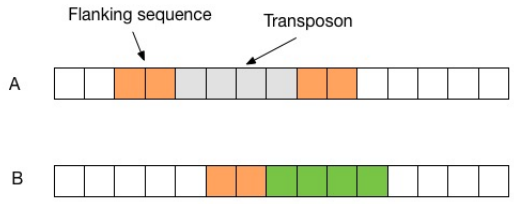

Figure 1: Initial situation

After the modification the two chromosomes will be in the situation presented at figure 2.

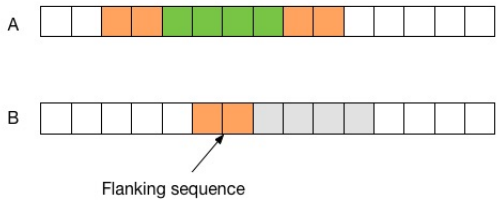

Figure 2: Final situation

Goals

We want to study how the use of such a mechanism improves the performance of a evolutionary. Choose an appropriated benchmark problem and two versions of an evolutionary algorithm: one with a standard uniform crossover and a second one with the transposition operator. Do **thirty runs** for each configuration. Store quality measures for each version, like the performance at the end of the run. Do a statistical analysis of those measures and draw your conclusions. **Note** that the size of the flanking sequence is an important parameter and you must choose a good value for that parameter.

4 Benchmark Problems

We present here some standard **benchmark problems**, commonly used to test stochastic global search heuristics. They can be used with the different evolutionary algorithms that will be under study in your work, but can also be applied to other type of heuristics like PSO, ACO and the like.

Using benchmark problems simplify the issue of assessing the performance of the algorithms and allow the comparison with others work. The examples below are only a small fraction of the examples mostly used. If you decide to use a problem not mentioned here ask first for permission. If the project you choose do not force you to use a particular example, or kind of problem, you may choose the ones you thing they are the most appropriated for the problem at hand.

Do not forget that the dimension of the problem is relevant. For example, it is harder to find the minimum of a 30 dimension function when compared with, say, a 2 dimension function. As a consequence, you must choose an appropriate instance of the benchmark problem, so the problem will be difficult to solve and the statistics will be important to determine eventual differences among the candidate algorithms.

4.1 *Trap*

There are functions that were conceived to make the life harder for the evolutionary algorithms. They have the property of being **deceptive**. A classical example is the function known as **trap**. A candidate solution is a binary vector, and its fitness is given by the number of zeros present in the string, unless there is none in which case the fitness is maximal and equal to the length of the vector plus one. More formally (3):

$$f(\langle x_1, \ldots, x_n \rangle) = (n - \sum_{i=1}^{n} x_i) + (n+1) \prod_{i=1}^{n} x_i \qquad x_i \in \{0, 1\} \qquad (3)$$

4.2 SAT: Satisfiability of a boolean function

The SAT problem ask for an interpretation of a boolean formula (i.e., an assignment of true values to boolean variables), that result in the formula evaluating to true. The formula is given, without loss of generality, in the conjuntive notmal form. An example with four boolean variables, follows:

$$(x_1 \vee x_3) \wedge (x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4)$$

There may exist more than one solution. In general, it is a hard problem and the set of solutions may be a singleton when the number of variables is high. For example, with 100 variables the size of the search space is equal to 2^{100}.

4.3 João Brandão's Numbers

already mentioned in class. We have a set comprising the first n integers and we want to find a set of maximal length (size), subject to the constraint that no number in the solution set may contain three elements such that one is the average of the other two.

4.4 Travelling Salesman Problem

Another classical problem already discussed. A salesman must visit one and only one set of cities, starting in one city and returning to that same city) minimizing a certain criterium (e.g., the distance). Technically, we want to find an Hamiltonian tour in a directed graph of minimal cost. For the problem become harder the number of cities must be hight.

For the **dynamic variant** of the TSP, we may consider that the matrix of the distances between pairs of cities may change over generations. The more the number of the distances that change in a particular moment, the harder the problem.

4.5 0/1 Knapsack Problem

Yet another well known problem. Given a knapsack and a set of n items x_i, each with a certain weight w_i and value v_i, which ones to choose to maximize the value without surpassing the capacity C of the knapsack. Formally:

$$max(\sum_{i=1}^{n} v_i \times x_i)$$

subject to:

$$\sum_{i=1}^{n} w_i \times x_i \leq C$$

with

$$x_i \in \{0, 1\}$$

For the **dynamic variant** the knapsack capacity changes over the generations. As you know the choice of the characteristics of the problem determine its hardness. There are three main forms of defining them:

Uncorrelated

The weights w_i and the values v_i are obtained from a uniform distribution in the interval $[1..v]$:

w_i = uniform([1..v])
v_i= uniform([1..v])

Weakly correlated

w_i = uniform([1..v])
$v_i = w_i$ + uniform([-r..r])

Note: negative values are not allowed.

Strongly Correlated

w_i = uniform([1..v])
$v_i = w_i$ + r

The more correlated the data, the harder the problem.

4.5.1 Parameters

You may use the following values:

- $v = 10$
- $r = 5$
- Number of items $n \in \{100, 250, 500\}$

The knapsack capacity can be calculated in two different ways.
$C = 2 \times v$
$C = 0.5 \times \sum_{i=1}^{n} w_i$

4.6 Sum of a subset of integers

This is a particular case of the Knapsack problem. Given a set of n distinct integers $\{a_1, a_2, \ldots, a_n\}$, not necessarly consecutives, find a subset whose sum is equal to a given particular value X. Suppose that you want to minimize the size of the set. For example, for $\{5, 8, 4, 11, 6, 12\}$ and $X = 20$ we have two solutions $\{8, 12\}$ and $\{4, 5, 11\}$, the first one being the best.

4.7 Golomb Rulers

we may describe the Golomb Rulers Problems as follows. A Golomb Ruler has marks, not necessarily evenly spaced, that may be used to measure integers distances. In the Golomb Rulers all distance between two consecutive marks are unique, and we say that the ruler os optimal (an OGR) if its size is the minimal for a certain number of marks. There may be more than one OGR for a certain numbers of marks, We say that the OGR is **perfect** if we can measure all lengths from 1 up to its length. Figure 3 shows a Perfect Golomb Ruler for four marks and length six.

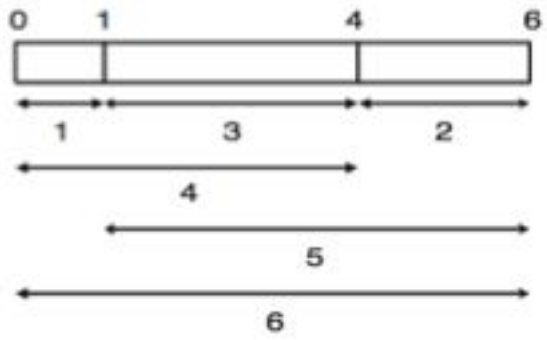

Figure 3: Perfect Golomb Ruler

The Golomb Rulers may be represented in different ways. In abstract they are just a vector $< a_1, a_2, \ldots, a_n >$, where each a_i represents a mark. In the case of the 3 we have $< 0, 1, 4, 6 >$. By convention a_1 is all ways put in position 0, while a_n represent the length of the ruler.

4.8 Symbolic Regression

A classical benchmark problem for genetic programming. We want to approximate a polinomial of a certain degree, using a training set composed of input-output pairs. Example of simple polinomials are:

$$x^4 + x^3 + x^2 + x \tag{4}$$
$$x^2 + x + 1 \tag{5}$$

In the context of genetic programming the condition for the experiments are given in the table 2.

Characteristics	Description
Goal	Best approximation for $x^4 + x^3 + x^2 + x$ with $x \in [-1, 1]$
Function set	$+, -, *$ and protected division
Terminal set	variable x (no constants)
Fitness	Squared Error
CTest set	21 cases in the interval $[-1, 1]$

Table 2: Symbolic regression: : parameters

4.9 Five-Even Bit Parity Detector

Another example for genetic programming. We want to evolve a solution for detecting if the number of ones in a binary sequence of size five is even or not.

Characteristics	Description
Goal	Correct classification for the 2^5 cases
Function set	$and, or, nand, nor$
Terminal set	variables x_1, x_2, x_3, x_4, x_5 (no constants)
Fitness	32 - number of correct classifications
Test set	subset of the 2^5 possible cases

Table 3: Parity detector: parameters

4.10 Artificial Ant

We want to evolve the **strategy/controller** of an artificial ant so it can follows a discontinuous trail with 89 food pellets. The food is in a toroidal grid of size 32×32. In the beginning the ant is in the upper left cell of the grid facing east, and it has up to 600 actions to attain its goal, i.e., eat all 89 food pellets. Figure 4 illustrate the situation.

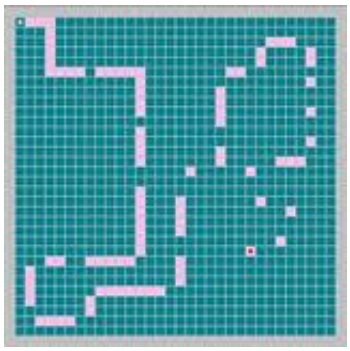

Figure 4: **Santa FE Ant Trail**

Characteristics	Description
Goal	Eat all 89 food pellets limited to a maximal number of 600 actions
Function set	$\{if\_food\_ahead, progn2, progn3\}$
Terminal set	$\{left, right, move\}$ (no variables)
Fitness	Number of food pellets not eatean

Table 4: Santa Fe Ant Trail: parameters

4.11 Functions

A sample set of possible functions whose goal is to find their optimal value. You may choose another one in accordance with the professor. Remember that these functions are only difficult if the number of dimensions is high (greater than 10).

Sphere (De Jong's F1 function)

One on the De Jong's functions set. It is the**F1** function: continuous, convex, unimodal. The minimum is at the origin and the maxima are at the extreme

values of the domain.

$$Min(F_1) = F_1(0, \ldots, 0) = 0$$

In figure 5 you can see the 2D version.

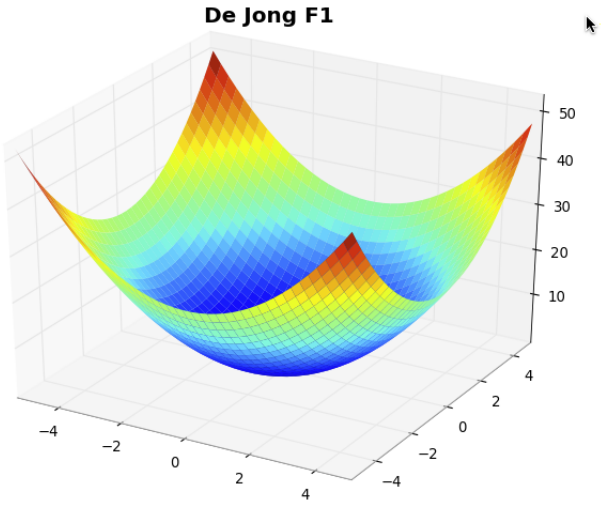

Figure 5: **De Jong F1**

Defined by the equation 6.

$$f(\langle x_1, \ldots, x_n \rangle) = \sum_{i=1}^{n} x_i^2 \qquad x_i \in [-5.12, 5.12] \tag{6}$$

4.11.1 *Rosenbrock*

De Jong F2 function: continuous, non convex and unimodal. In figure 6 the 2D version.

$$Min(F_2) = F_2(1, \ldots, 1) = 0$$

Defined by the equation 7.

$$f(\langle x_1, \ldots, x_n \rangle) = \sum_{i=1}^{n-1} (1-x_i)^2 + 100 \times (x_{i+1} - x_i^2)^2 \qquad x_i \in [-2.048, 2.048] \tag{7}$$

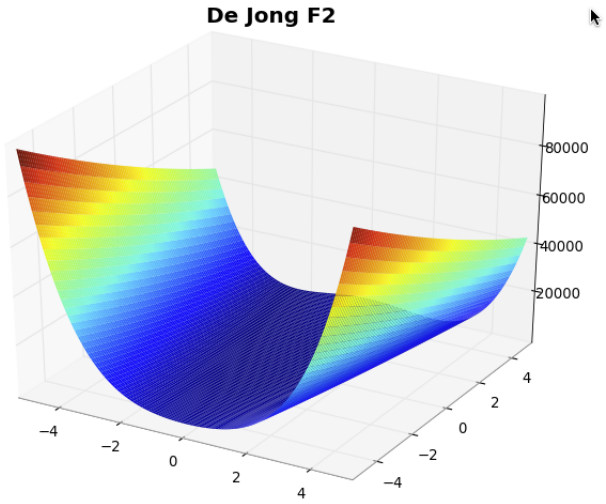

Figure 6: **De Jong F2**

4.11.2 *Step*

De Jong F3 function:non continuous, non convex and unimodal. In figure figura 7 the 2D version.

$$Min(F_3) = F_3(-5.12, \ldots, -5.12) = 0$$

Defined by the equation 8.

$$f(\langle x_1, \ldots, x_n \rangle) = 6n + \sum_{i=1}^{n} \lfloor x_i \rfloor \qquad x_i \in [-5.12, 5.12] \tag{8}$$

4.11.3 *Quartic*

De Jong F4 function: continuous, convex and unimodal. In figure 8 the 2D version.

$$Min(F_4) = F_4(0, \ldots, 0) = 0$$

Defined by the equation 9.

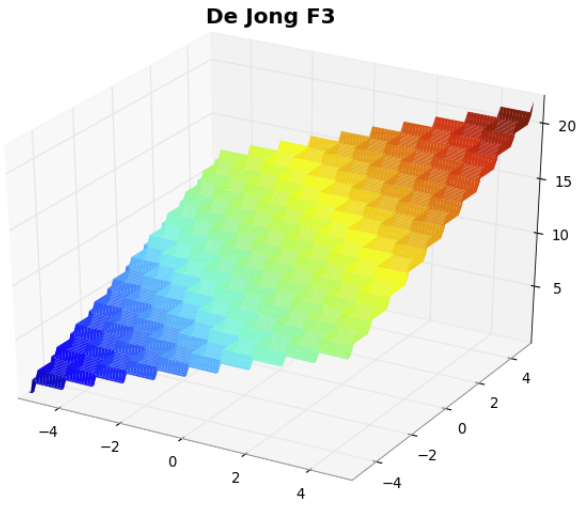

Figure 7: **De Jong F3**

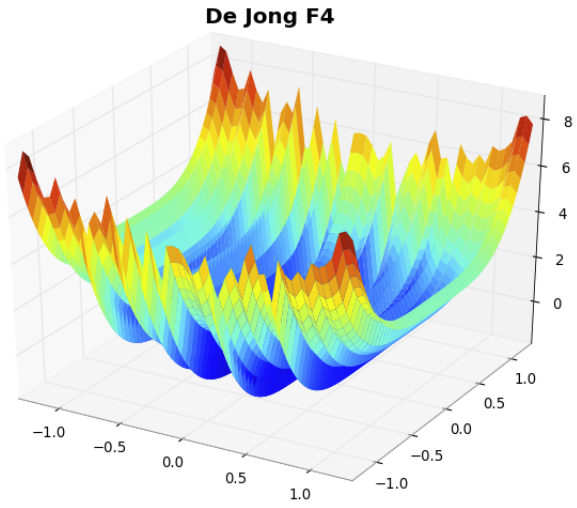

Figure 8: **De Jong F4**

$$f(\langle x_1, \ldots, x_n \rangle) = (\sum_{i=1}^{n} i \times x_i^4) + \mathcal{N}(0,1) \qquad x_i \in [-1.28, 1.28] \qquad (9)$$

with $\mathcal{N}(0,1)$ a gaussian noise of average 0 and standard deviation 1.

4.11.4 *Rastrigin*

A multimodal function with many local minima. In figure 9 a 2D version.

$$Min(Rastrigin) = Rastrigin(0, \ldots, 0) = 0$$

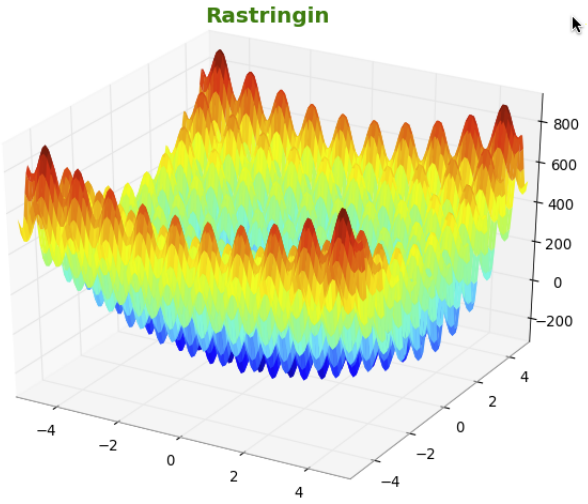

Figure 9: **Rastrigin**

Defined by the equation 10.

$$f(\langle x_1, \ldots, x_n \rangle) = A \times n + (\sum_{i=1}^{n} x_i^2 - A \times cos(2\pi x_i) \qquad x_i \in [-5.12, 5.12] \quad (10)$$

with A usually equal to 10.

4.11.5 *Schwefel*

Another multimodal function with many local minima. In figure 10 a 2D version.

$$Min(Schwefel) = Schwefel(420.9687, \ldots, 420.9687) = n \times 418.9829$$

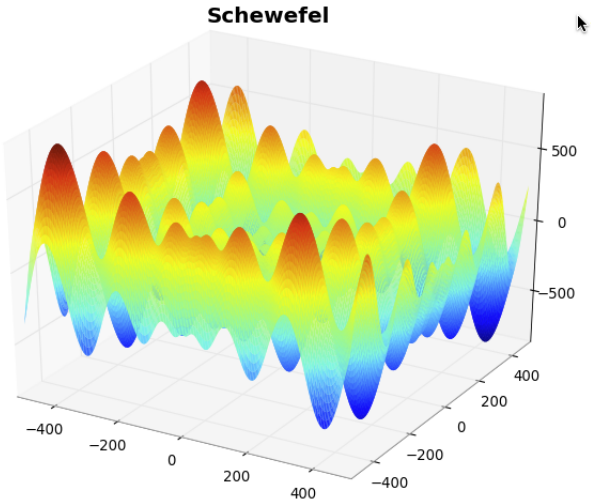

Figure 10: **Schwefel**

Defined by the equation 11.

$$f(\langle x_1, \ldots, x_n \rangle) = \sum_{i=1}^{n} -x_i \times sin(\sqrt{|x_i|}) \qquad x_i \in [-500, 500] \qquad (11)$$

4.11.6 *Griewangk*

Yet another multimodal function, similar to the Rastrigin function. In figure 11 the 2D version.

$$Min(Griewangk) = Griewangk(0, \ldots, 0) = 0$$

Defined by the equation 12.

$$f(\langle x_1, \ldots, x_n \rangle) = 1 + \frac{1}{4000}(\sum_{i=1}^{n} x_i^2) + \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{(i)}}) \qquad x_i \in [-600, 600] \quad (12)$$

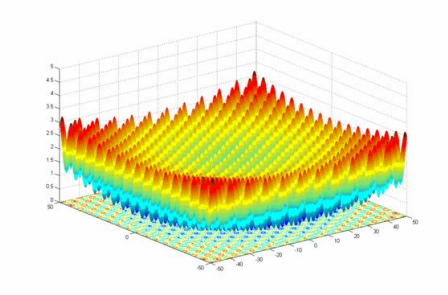

Figure 11: **Griewangk**

5 Conclusion

A few short comments. First, the control of the progression of your work will be done during the classes (T and PL). Moreover, you can discuss directly with me eventual problems by presenting yourself at my office at defined hours. Second, the projects reflect for the most part your actual knowledge. The rest will be object of lecturing soon after Easter. Third, we try to balance the difficulty of all the works, but we are aware that this is not an easy task and it is somehow a subjective matter. Forth, we try to ask a work load compatible with the value of the work for the final mark.

Methodological issues, like the statistical background, were elucidated during the previous lectures. You may use the statistical tool you feel at easy with, including the Python code that I provided. Finally, even if this is a work that asks you to do simulations and analyze the results, i.e., it has a practical flavor, there is however a theory behind the work, and you are advised to consult the necessary literature, e.g., [1, 2, 3].

Good luck!

References

[1] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.

[2] Sean Luke. *Essentials of Metaheuristics*. Lulu Press, 2011.

[3] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 1996.