TP1 - NK Landscape

Project structure

The project is developed in **python 3**, using only core libraries for implementing the metaheuristic and using some mathematics and plot library to analyse the results (*matplotlib*, *scipy* and *numpy*).

- **NKLandscape.py** : contain the meta-heuristics algorithm and the data structures needed.
- **tp1.py** : executes the simple test code, as requested in the section **2.2** of the TP.
- **analyse.py** : executes the analysis over the 2 algorithms and plots the results.

NK-Landscape

This model depends on the values of N and K, where the first is controlling the size of the solution space S, and K control the roughness of the landscape. The formal definition is:

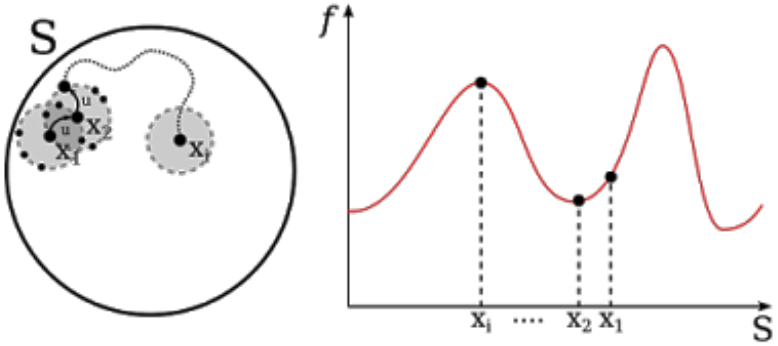
$$S = \{0,1\}^N \quad x = (x_1, x_2, .., x_N)$$
$$F(x) = \sum_{i=1}^{N-K} f_K(x_i, x_{i+1}, \ldots, x_{i+K})$$

Where $f_K(\ldots)$ is the local fitness and $x_{i+1}, .., x_{i+K}$ are the neighbors of x_i.
A simpler case is when $K = 0$ (K0 NK-Landscape), as in the **MaxOne** problem.

In the TP1 the configuration used is $N = 21$ and three different $K\,(0, 1, 2)$ and f_K.

Metaheuristic

A metaheuristic is an algorithm that proceeds to explore the solution space to search an acceptable solution that maximize (or minimize) the fitness function.



The problem is that S, the solution space, is huge (in our configuration the size of S is 2^{21}) and often it will take too much time to explore it all.
This means that we have to define how the algorithm choose the next solution between the neighbors (exploration), and when it stops the searching for the optimal solution.
A metaheuristic also need a starting solution, that in general is a random $x \epsilon S$.

Two metaheuristic algorithms will be studied to optimize the NK-Landscape: the deterministic hill-climbing and the probabilistic hill-climbing.

Deterministic Hill-Climbing

This metaheuristic is very simple and the exploration operator simply chooses the neighbor with highest fitness **if** it's higher than the fitness of the current solution (so this is also the stop condition).
This algorithm, as the name suggest, is deterministic and with the same input will always give the same result.

$$x_{i+1} \in V(x_i) \ : \ F(x_{i+1}) \geq F(y) \wedge F(x_{i+1}) > F(x_i) \quad \forall y \in V(x_i)$$
$$\text{where } V(x) \text{ return the neighbors of the solution } x$$

Probabilistic Hill-Climbing

To improve the performance of the previous algorithm in rough landscape it's possible to implement a probabilistic variant of it.
Instead of choosing always the neighbor with highest fitness, the algorithm will chose

randomly the next solution to explore.
The probability of each neighbor to be selected is **not uniformly distributed** but proportional to its fitness.
On top of that, a **aspiration** process is also applied, that impose to select a neighbor if it's fitness is higher than the highest fitness ever found.
The **stop condition**, this time, is a maximum number of iterations. The algorithm has to keep record of the maximum fitness and the best solution found.

$$x_{i+1} \in V(x_i) \ : \ F(x_{i+1}) \geq F(y) \ \ \forall y \in V(x_i) \ \wedge \ F(x_{i+1}) > F(x_{best})$$
$$\vee$$
$$x_{i+1} \text{ extracted from } V(x_i) \text{ with } \vec{P}(V(x_i)) = \left(\frac{F(x_{v1})}{\sum_{y \in V(x_i)} F(y)}, \dots, \frac{F(x_{vN})}{\sum_{y \in V(x_i)} F(y)} \right)$$
$$\text{where } V(x) \text{ return the neighbors of the solution } x$$
$$\text{and } x_{best} \text{ is the best found solution}$$

Implementation

To represent a solution I implemented a class `BitArray`, that gives me the possibility to integrate many functionalities, in particular the `neighbors()` method that returns the list of all neighbors of a solution, or the overloading of the operator `-`, to compute the Hamming distance between two solutions.

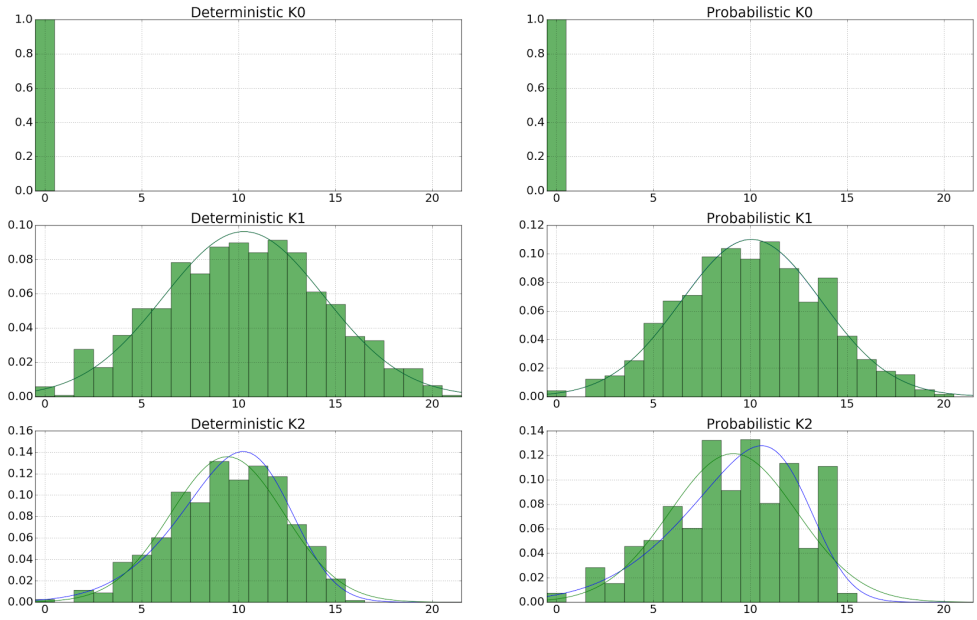
The fitness function is an instance of the class `NKFitness`, that manages, using a dictionary passed in the constructor parameters, any kind of K and local fitness.

Finally, the most important part of the work, the two metaheuristics, `det_high_climbing` and `prob_high_climbing`, are both implemented as recursive functions, both need a start solution and fitness value (it's own fitness) and a counter (in the opposite way, the deterministic one increases it to count the number of recursions done, the probabilistic decreases it to stop the recursion when it reaches 0).
The probabilistic function has the need to keep in memory the maximum fitness found and the respective solution.
In both cases (only for the aspiration variants of the probabilistic one) if 2 or more solutions are found with the same maximum fitness the first solution is always chosen.

Analysis of the results

More K is big more the landscape is rough, so intuitively I was expecting that the *stability* of the solutions would have worsened with the increasing of K.
Trough the Hamming distance histogram it's possible to see that the stability doesn't evolve in the pictured way.
Between $K0$ (where the optimum solution is always found, as there are no local max.) and $K1$ the stability worsened a lot, and the distribution of the solutions hamming distance is a normal distribution.
But instead between $K1$ and $K2$ the situation reverse and the maximum Hamming distance doesn't exceed 15/16 (compared to the previous 21, the maximum distance possible). Also the distribution change and become a skew normal distribution (as possible to see in the plots fit).
This depends on the fact that the local fitness K2 configuration in the example present less diversity than the K1 local fitness, as most of the configuration results in a local fitness of 0 or 1.

Hamming Distance histogram



The performances of the two algorithm are very similar for lower K (in particular for $K = 0$ the two perform in the same way).
But for higher values of K the probabilistic algorithm explores a bigger portion of S and (statistically) finds better solutions (with highest fitness) then the deterministic one, that falls easily for local maximum.
The similarity of the solutions at lower K is due the implementation of the aspiration process.

Fitness histogram