Université de Genève

Metaheuristics for optimization

TP 1: NK-landscape models

Author: Joao Filipe Costa da Quinta

E-mail: Joao.Costa@etu.unige.ch

October 3, 2021

1 Introduction

In this TP we will try to solve the problem of finding the maximum of a given function. Our function represents the landscape of a mountain top, to which we can add more ruggedness by changing the parameters. By having higher ruggedness we should witness higher difficulty in finding the global maximum, as we might tricked by local maximums.

2 Problem

2.1 Representation

To represent our mountain top we will use a simple N-bit string, which makes $\{0,1\}^N$ our search space. Let x be a random point in our search space, to check if is a maximum we will use a fitness function $F(x)$, this is the function we would like to maximize.

$$F(x) = \sum_{i=1}^{N-K} f_K(x_i, ..., x_{i+K})$$

The difficulty will entirely depend on the value K, in this TP we will set $K = \{0, 1, 2\}$, varying K will make $F(x)$ easier or harder to maximize. Depending on K, f_K will have to consider more or less input possibilities, $\{0,1\}^{K+1}$ to be precise.

$K = 0$ This is the easiest difficulty, this means that $f_K()$ gets only 1 parameter x_i at a time, which means that there are $\{0,1\}^1$ input possibilities. Therefore there is only one maximum $F(x) = \sum_{i=1}^{N} f_0(x_i)$.

$K = 1$ $f_K()$ gets 2 parameters x_i at a time, therefore there are $\{0,1\}^2$ input possibilities. Which means our mountain top will have multiple mountains or multiple local maximums. $F(x) = \sum_{i=1}^{N-K} f_1(x_i, x_{i+1})$.

$K = 2$ $f_K()$ gets 3 parameters x_i at a time, so there are $\{0,1\}^3$ input possibilities. It will make it even harder to find the global maximum. $F(x) = \sum_{i=1}^{N-K} f_2(x_i, x_{i+1}, x_{i+2})$.

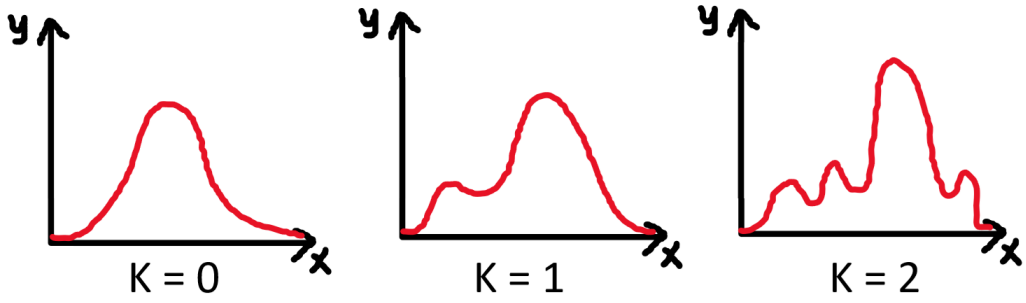

Figure 1: Simple 2-D representation on how a higher K increases the ruggedness of the function F()

3 Algorithms

To solve this problem we will use the mountain climbing algorithm, we will use the deterministic version as well as the probabilistic, we will see which one does better with higher K. The goal is to find an $x \in \{0,1\}^N$ that maximises $F(x)$.
Let's also define what is the neighbourhood of a point $x \in \{0,1\}^N$ as the algorithms will use it. The neighbourhood of a point $x \in \{0,1\}^N$ or $V(x)$ is defined by all the points $x' \in \{0,1\}^N$ at a hamming distance of 1 from x. The hamming distance between two strings of the same length, is the sum of different individual values, so 2 strings are of hamming distance 1, if only one bit between the two is different.

3.1 Deterministic Hill-Climbing

Lets see the a step by step analysis of the algorithm:

(1) generate random $x \in \{0,1\}^N$

(2) compute the fitness of x given by $F(x)$, let's call it fit_x

(3) generate its neighbourhood or $V(x)$

(4) compute the fitness of all $x' \in V(x)$

(5) find the maximum fitness computed in step (4), and let's call it $fit\_x_{neighbor}$

(6) compare fit_x and $fit\_x_{neighbor}$

(6.a) $fit\_x < fit\_x_{neighbor} \longrightarrow$ we set x = neighbor with higher fitness and go back to step (2)

(6.b) $fit\_x \geq fit\_x_{neighbor} \longrightarrow$ we return x as a maximum and finish the computation

In other words, at each step of the algorithm we look for the highest fit neighbor, and set it as the new x, if we don't find a higher fit neighbor, it means we are at a mountain top, so we stop the algorithm. We have to be ware that it might be a local maximum, and not at a global maximum.

3.2 Probabilistic Hill-Climbing - with aspiration

Lets see the a step by step analysis of the algorithm (some steps are the same as the deterministic version):

(1) generate random $x \in \{0,1\}^N$

(2) compute the fitness of x given by $F(x)$, let's call it fit_x

(3) generate its neighbourhood or $V(x)$

(4) compute the fitness of all $x' \in V(x)$

(5) find the maximum fitness computed in step (4), and let's call it $fit\_x_{neighbor}$

(6) compare $fit\_x_{neighbor}$ and the best fitness ever seen during the computation x_fit_best

(6.a) $x\_fit\_best < fit\_x_{neighbor} \longrightarrow$ we set x = neighbor with higher fitness and go back to step (2)

(7) compute $total\_fitness$ the sum of all the fitness in the neighborhood, $total\_fitness = \sum_{x \in V(x)} F(x)$

(8) for every $x' \in V(x)$ compute $F(x')/total\_fitness$ from step (7), every neighbor has a normalised probability that is proportional to their own fitness

(9) use a probabilistic method from TP0 (roulette method), to chose randomly the next neighbor x depending on the probabilities computed in step (8), go back to step (2) with new x

(stop) we stop after a given number of 'rounds', and return the best ever x (we have to keep track of them)

If we skip steps (5) and (6) we will have the Probabilistic Hill-Climbing algorithm without aspiration.

3.3 Comment on the algorithms

What changes between the 2 is the choosing of the next x, as the deterministic version always chooses the best neighbor possible, and the probabilistic version chooses randomly between the neighbors. Which means the probabilistic version might not get stuck in a local maximum, where the deterministic version will naively think it has done it's job perfectly while finding the local maximum.

4 Results

To try both algorithm's performances, we will create a random x string of $N = 21$ bits, making $\{0,1\}^{21}$ the search space, and use the algorithm with $K = \{0,1,2\}$. Since the result highly the depends on the first random x, we will run the same algorithm 50 times. To check how different the results were, we will compute the hamming distance between them.

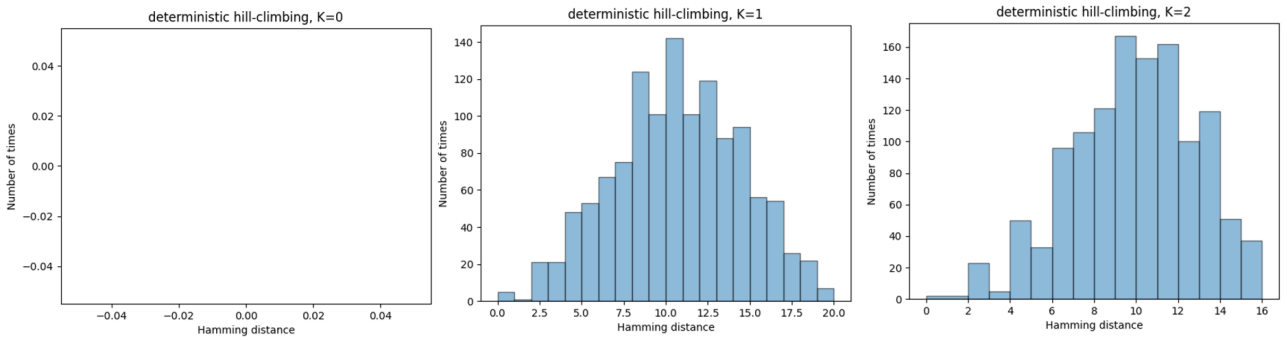

Figure 2: Histogram representing the Hamming distance for deterministic hill-climbing $K = \{0,1,2\}$

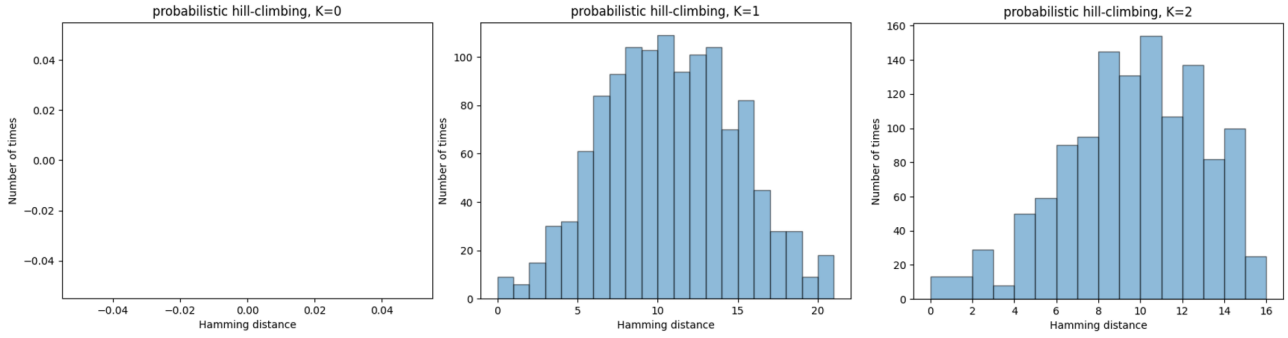

Figure 3: Histogram representing the Hamming distance for probabilistic hill-climbing with aspiration $K = \{0,1,2\}$

Looking at figure 2 and 3, we can analyse how good both algorithms are, when $K = 0$ both algorithms were able to find the global maximum every time. This is to be expected, the deterministic version only stops at a maximum, since there is only one maximum, it is easily found. The probabilistic variant with aspiration does have a deterministic side, as we make sure to explore a higher fitted x if we've never seen one that is as good.

Looking at the other histograms in both figures, we can't see much difference between the two methods, as both of the graphs for a given K value show a similar gaussian. We can see that the bins are a bit more even in the probabilistic method, but it doesn't tell us that it is definitely better than the deterministic version.

To try and have a definitive idea of which method is best, we can plot the fitness of each returned x by both methods with $K = \{1,2\}$

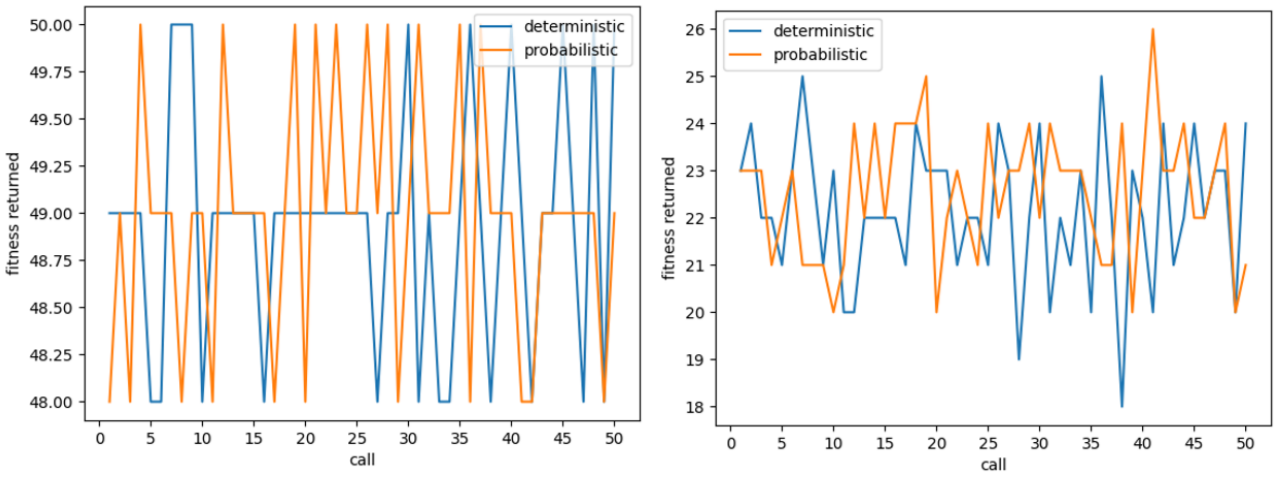

Figure 4: Graphs representing the fitness of the x returned by both algorithms, $K = \{0, 1, 2\}$

For K=1 we can see that both methods returned the same results, the same amount of real maximums, and the same amount of local maximums. Where it gets interesting, is with K=2, where we see that the deterministic could never achieve the global maximum.