ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Simulated annealing algorithm for graph coloring

`Error 404`: E. Bugliarello, F. Fontan, E. Pastorelli

May 25, 2016

1 Algorithm Description

Starting from the algorithm provided in the project description, we have developed several implementations having different cooling procedures, such as exponentially, logarithmically and linearly dependent. We conducted our simulations by considering several 1000-node graph realizations (in the order of hundreds) and let the node degree c and the number q of coloring vary, in order to consider cases where we were sure to find a proper q-coloring (q much larger than c) and cases in which a minimum of zero might not have been reached. For fixed c and q values, we run simulations applying these different cooling functions and the returned values for the average of the energy and for their corresponding variances were very close to each other.
Guided by the experiments on these functions, we developed a custom procedure that is described in the following and implemented in `final.m` (attached to this report).
The algorithm can be split in three main phases, according to the number of iterations passed with respect to their total value.

- Due to the initial randomness of the coloring, several pairs of neighboring nodes will have the same color. For this reason, we want our algorithm not to consider possible disadvantageous configurations, but rather try in the succeeding steps with a new configuration. As a consequence, we set an initial high value of β to work with a system at low temperature. In particular, we set $\beta^1 = \frac{\texttt{maxIter}}{1000}$ and a duration of $\frac{\texttt{maxIter}}{1000}$ steps.
- After this first constant phase, we have an energy much lower than the initial one and we start taking into account the possibility of accepting configurations leading to a higher value of energy.
  This represents the trickiest part of our procedure.
  We keep trace of the number of times we reject a given configuration that would increase our energy since this might mean we are trapped in a local minimum. So, if this event happens more than $\frac{\texttt{maxIter}}{300}$ times, we reduce β to $\frac{\texttt{maxIter}}{50\cdot h}$, where h is the value of the energy at that step. This allows us giving a higher energy to a configuration representing a local minimum rather than when it is a global one.
  After this first reduction, we linearly increase this value by $\frac{20}{h}$ for the following $\frac{\texttt{maxIter}}{1000}$ steps to obtain sufficient energy to leave the (possible) minimum but not enough to neutralize the preceding moves.
  Once we have progressed for these $\frac{\texttt{maxIter}}{1000}$ steps, we reset β to $\frac{\texttt{maxIter}}{1000}$ until a new possible minimum is found.
- In the final phase, when 2% of `maxIter` iterations are left, we block β to 5 to avoid choosing a worse condition at the end of the procedure.

From now on we will refer to our algorithm as *"schedule boost"*.

[1] All the values have been derived empirically to optimize our algorithm.

2 Analysis

In this section, we exhibit the behavior of our algorithm in different scenarios.
We firstly show how the residual energy varies with time in a 1000-node Erdös-Renyi graph, as asked in the project description. The following figure is obtained choosing: `N=1000`, `q=7`, `c=5`, β=5 and `maxIter=5000`.

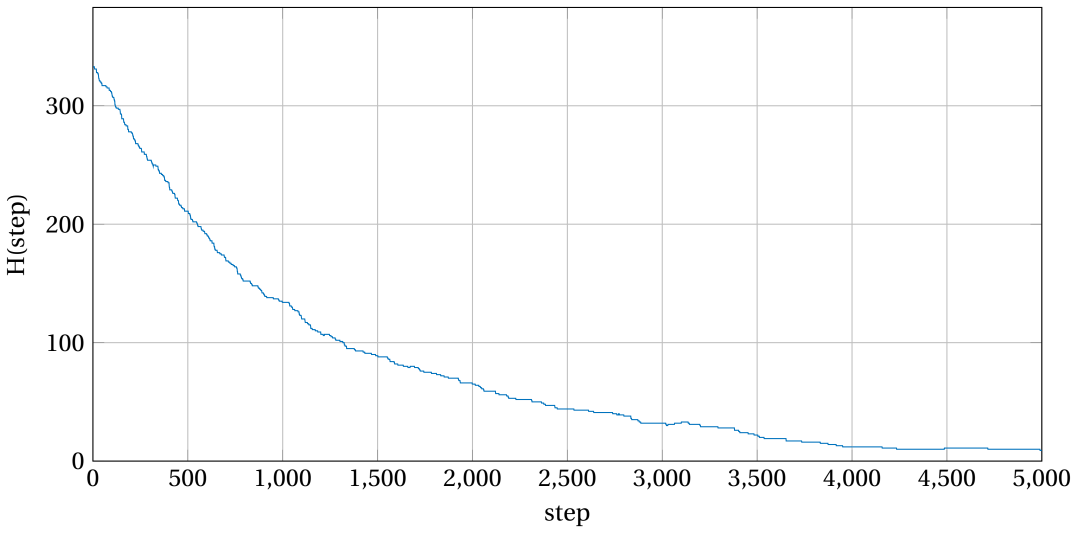

Figure 2.1: $H(x^t)$ as a function of time for $q = 5$ and $c = 5$.

The trend of this curve matches our expectations since we constructed our procedure in order to minimize this energy. Its behavior looks like a negative exponential: the energy abruptly drops at the beginning, while its decrease slows down as this value gets smaller.
These phases can be explained due to the randomness in the initial coloring which, generally, is very far from a proper coloring configuration. In the first steps, we can easily recolor a chosen vertex in such a way that the number of neighbors having the same color immediately reduces. However, as the algorithm progresses, it is less likely to pick a color for a given node that leads to a better configuration.
We now proceed by plotting the average value of residual energy as a function of c for q equals to $3, 5, 7$.

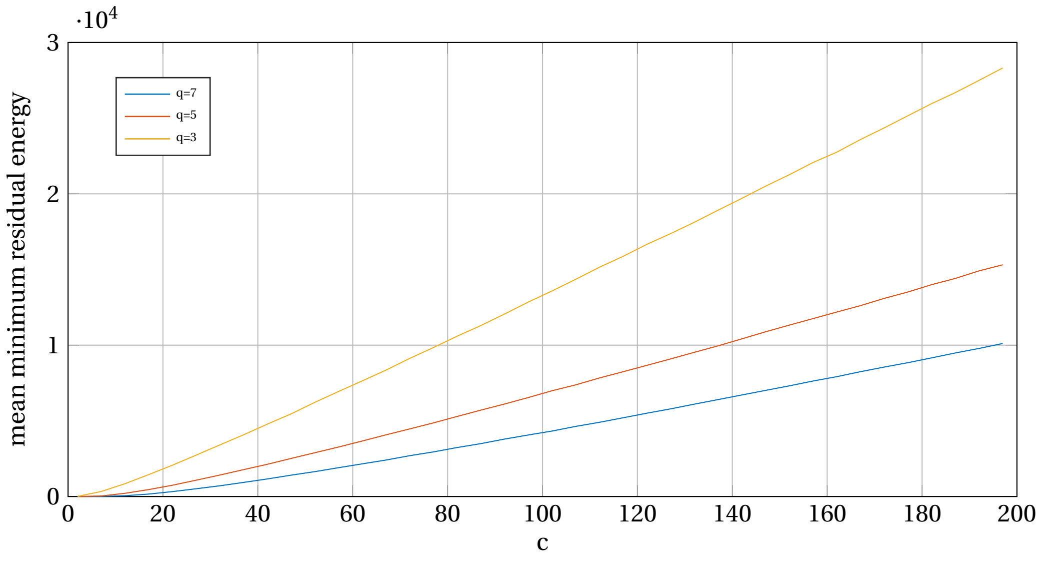

Figure 2.2: $\langle H_{min}(q, c)\rangle$ as a function of c for $q = 3, 5, 7$.

From this picture we can observe that as the parameter c increases, also the mean energy is higher. In fact, intuitively, as we increase the number of edges among nodes, it is more difficult to find a proper coloring in the graph. Moreover, for a given value of c, the energy is higher for smaller q's. We can clearly deduce this behavior because it is easier to properly color the graph when the number of colors increases: the probability of finding a color that is different from the ones of its neighbors is larger.
In particular, we have always considered values of c greater than one. By doing so, we avoided considering trivial cases. In fact, only for $c > 1$ there is a giant connected component in the graph, thus the coloring task becomes difficult to perform. On the other hand, when $c < 1$, it is possible to implement appropriate algorithms which allow to almost surely find a proper coloring of a graph. We have tried to test this trend by using our algorithm and the results are given in Figure 2.3.

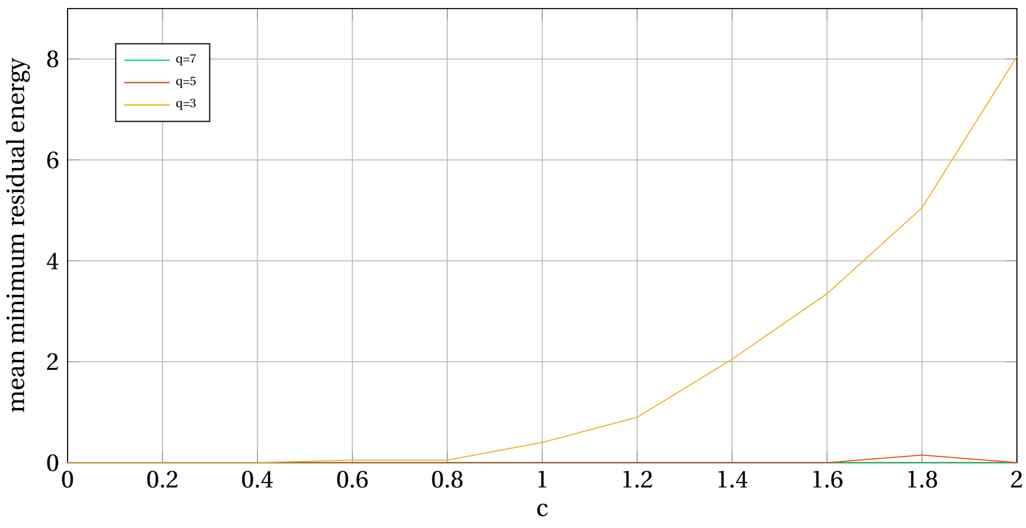

Figure 2.3: $\langle H_{min}(q,c)\rangle$ as a function of $c < 1$ for $q = 3, 5, 7$.

In this last part of the report, we illustrate the behavior of the other procedures that we have also analyzed. In particular, the following cooling algorithms have been investigated:

- $\beta = \beta - \log\left(\frac{\texttt{step}}{\texttt{maxIter}}\right) \cdot \frac{\texttt{step}}{\texttt{maxIter}}$ (schedule *log*),
- $\beta = \beta + 0.01$ (schedule *lin*),
- $\beta = \beta \cdot 0.85$ (schedule *pow*).

In the first two algorithms we update β every 10 steps, while in the third one we update it every time the current iteration number assumes a value that is a power of 2.
The following figures are obtained choosing: `N=1000`, `q=7`, `c=5`, β=5, `maxIter=10000`.

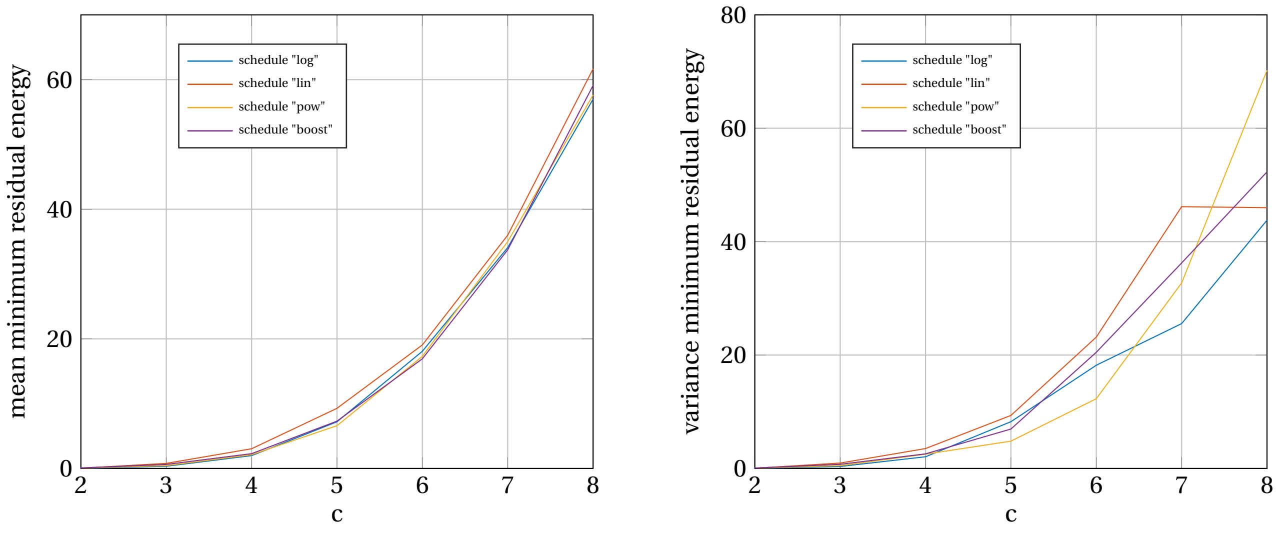

Figure 2.4: $\langle H_{min}(q,c)\rangle$ and its variance as a function of c for $q = 5$.

From the picture on the left, we notice that, among all the chosen cooling procedures, none of them is significantly better than another, although the third one seems to be preferable. Probably, the number of nodes set for the simulation is too low to reveal a real difference. However, it is not possible to establish what is the best procedure in an absolute sense; it could depend on the structure of the considered graph, q or also on the number of iterations. Furthermore, in addition to the average of the minimum residual energy, also its variance could play an important role as we can see from the plot on the right of Figure 2.4. In fact, if this quantity were too large, then it could be more advantageous to run the algorithm more times but with considering a smaller number of iterations. On the other hand, a narrower variance leads us to increase the number of iterations in order to find the minimum value for the residual energy.