Mini-project : Deploying a 5G Network in a country

Justin Deschenaux, Federico Betti, Guillaume Follonier
COM-516 Markov chains and algorithmic applications, EPFL Lausanne, Switzerland

***Abstract*—Over the course of the semester, we have been introduced to the concept of Markov Chains with the objective of applying the newly acquired knowledge to different optimization problems that may arise in computer and communication science. More specifically, we studied how the Metropolis-Hasting (MH) algorithm can allow us to sample from a distribution that is computationally infeasible to simulate exactly, as the Gibbs distribution, with its intricate log-partition function. This report presents our work on a mini-project involving Markov chain simulation, to solve a problem of optimal location of telecom antennae.**

I. Introduction

We tried various approaches to solve the problem of allocating limited resources optimally. First, we considered the simulated annealing procedure. As we saw in class, we can use it to find extremas of a function. To that end, we considered a Gibbs distribution biased in favor of cities configurations that yields the highest reward. In order to sample from such complicated function, we used a version of the Metropolis-Hasting (MH) algorithm. Hence, we first had to define a chain with the sought after stationary behaviour. Another reason of concern is the mixing time, which depends on the spectral properties of the graph representing the random walk. For some specific instances, one could derive a bound on the total variation distance using the decomposition of the transition matrix. However, this is often not possible in practice. An alternative option that yields *exactly* distributed samples is the coupling from the past algorithm, proposed by James Propps and David Wilson in 1996. In short, it leverages a special property of certain chains and cost functions. However, we strongly believe that it is not possible to apply it in our case. Finally, we will present a method based on MCMC principles that allowed us to find an approximate solution to the problem in reasonable time.

II. Models and methods

A. Simulated Annealing

As it is the case in general for MH algorithm, we are working on the state space of a Markov chain and explore it step by step. In order to encode the current position, we used a similar representation as in the case of the Ising model. Let $\mathcal{C} = \{1, 2, ..., N\}$ denote the ordered set of cities. The state space is defined as $\mathcal{S} = 2^N$. Each $\sigma \in \mathcal{S}$ is a N-dimensional binary vector representing a subset $c \subseteq C$. Hence,

$$\sigma_i = \begin{cases} 1 & \text{if city } i \in c \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The need for a clever algorithm is justified here as the number of states is exponential (2^N). Note that although it is not reflected in the encoding of the selected cities, each of them has a an associated position in space, which determines the cost incurred when compared with the location of other towns, as we will see shortly when we talk about the cost.

Given such a state space, the natural choice of a base chain is the random walk on the N dimensional cube. That is,

$$\psi_{\sigma,\sigma'} = \begin{cases} \frac{1}{N} & \text{if } d(\sigma, \sigma') = 1 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where $d(\sigma, \sigma')$ is the Hamming distance between states σ and σ'. Therefore, from a given state we can move to one of its closest neighbour with uniform probability. Such a choice is appropriate, as it is isomorphic (up to vertices labelling) to the base chain of an Ising model we studied in class. Observe that it is straightforward to verify that $\psi_{\sigma\sigma'} > 0 \iff \psi_{\sigma'\sigma} > 0$ by symmetry of the Hamming distance, i.e. $d(\sigma, \sigma') = d(\sigma', \sigma)$. Hence, detailed balance holds. Moreover, starting from any state σ, we can reach any other position τ by flipping each coordinate of σ one by one, for all $i \in C$ where $\sigma_i \neq \tau_i$. Although the number of states can be huge, it is finite. As we know, irreducibility coupled with a finite space implied positive-recurrence. However the chain has a period of 2. It is nonetheless not an issue, as we know from our study of the Ising model.

Thirdly, one has to define the distribution we want on the chain (which in turns yields the acceptance probabilities). Those terms are defined using the function to minimize $g(\sigma) := -f(\sigma)$ where

$$f(\sigma) = \sum_{i=1}^{N} v_i - \lambda N \max_{\substack{i,j \\ \sigma_i = \sigma_i = 1}} \pi \frac{d(x_i, x_j)^2}{4} \tag{3}$$

Let $\Delta g(\sigma \to \sigma') := g(\sigma') - g(\sigma)$ represent the cost change when moving from state σ to state σ'. Let $\pi_\sigma := \frac{1}{Z} e^{-\beta g(\sigma)}$ where Z is the log-partition function. We can recognize the same distribution used in the Ising model. Similarly, the acceptance probabilities are $a_{\sigma\sigma'} = \min\left(1, \frac{\pi_{\sigma'}}{\pi_\sigma}\right) = \min\left(1, e^{-\beta \Delta g(\sigma \to \sigma')}\right)\ \forall \sigma, \sigma' \in \mathcal{C} : \sigma \neq$

σ'. Observe that moves $\sigma \to \sigma'$ that decrease the cost ($\Delta g(\sigma \to \sigma') < 0$) are certainly accepted. Indeed, in this case $e^{-\beta \Delta g(\sigma \to \sigma')} > 1$.

B. *Tolerance threshold algorithm*

The simulated annealing procedure raises different concerns (when to decrease β, how many simulated steps before we get close to the expected Gibbs distribution). Consequently, we also developed a simpler approach to manage the transitions of the random walk.

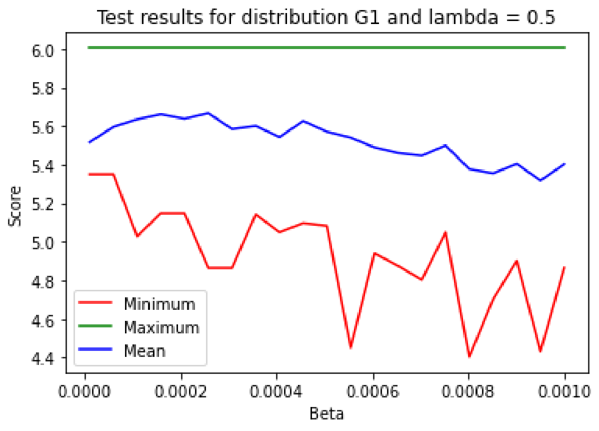
We accept a move if and only if the new score is greater than the old score minus the tolerance, that is if and only if $s \geq s' - tol$, tol being a parameter in function of time. This shrewdness permits to avoid really bad moves that decrease the value of the function in an important and almost irrecoverable way. The idea of introducing such a parameter should be a good compromise between the will of not remaining stuck in local maximums and the need of converging to the overall best in a reasonable time. The idea is perhaps to decrease linearly in time the tolerance parameter starting from an appropriate initial value. After accepting also some reasonably bad moves initially, the algorithm proceeds to accept only ”decreasingly bad moves” until the point in which $tol = 0$ and only increasing moves are accepted, and then it converges towards the nearest local maximum.

III. Results

In this section we discuss briefly some results obtained with a classical MH algorithm on the Gibbs distribution and simulated annealing, then for the reasons explained in the previous section we discuss more in detail the obtained results with the use of the tolerance threshold, which turned out to be the fastest and most efficient method.

A. *Simulated annealing*

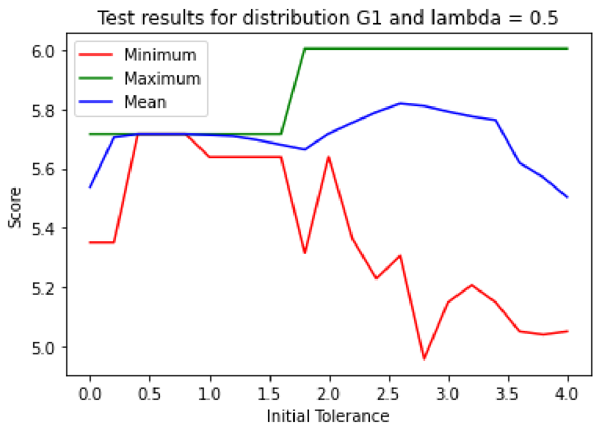
The following plot shows the results obtained by tuning first the value of β small:



The values obtained are small in the average case (which is the one of interest), meanwhile in the minimum case there is more fluctuation with scores arriving as low as 4.4. We will see that for an appropriate choice of the tolerance the obtained results are much better and obtained in a much faster time.

B. *Initial tolerance*

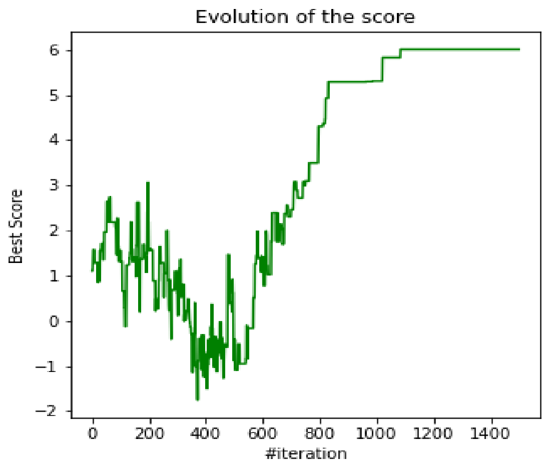
We will start from justifying the choice of the initial tolerance referred to in the previous section. After running the algorithm with different values of tol in [0,4] for the distribution G1 and for fixed $\lambda = 0.5$ the following was observed:



Hence it seems reasonable to think that the best choice of tolerance is around $tol_{init} = 2.5$, as for this value the mean curve attains its maximum and such a choice permits convergence to the absolute best on the green curve. Adding to this the fact that such a result was observed for a central value of $\lambda = 0.5$, we believe it was the best choice. Consider the red curve of the minimal result obtained by one of the chains.

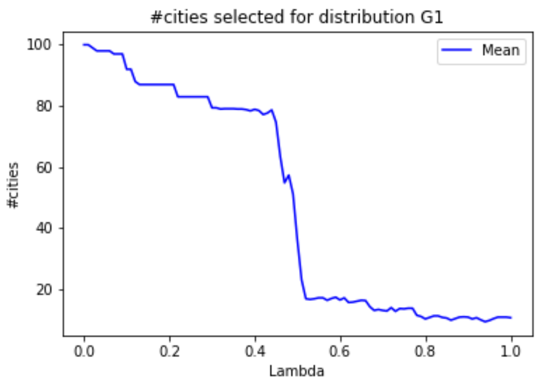
Contrarily to what one might intuitively think, since one wants to lose as much dependence on the initial condition as possible, the choice of a medium-high initial tolerance is not necessarily unfavorable. This in fact allows the state space to be explored uniformly initially. On the other hand, if one chooses a too high tol_{init} (e.g. 4), the risk is that the chain gets stuck in an area with low scoring states from which it is no longer able to exit as the tolerance is then decreased in the next steps. Consequently, as in the spirit of simulated annealing, the idea is to find a good balance between having an almost uniform distribution initially and the need to converge to the best possible result. From the graphical evidence and reasons aforementioned, $tol_{init} = 2.5$ appears to be the one that fulfills such requirements.

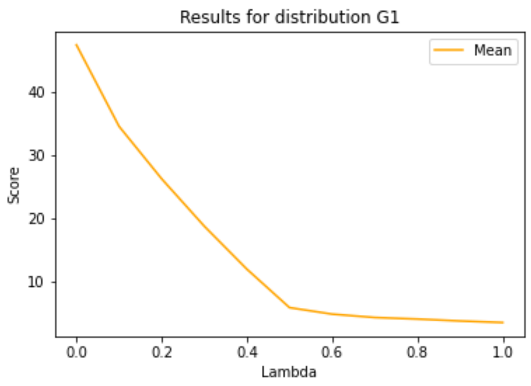
To give an intuition of how the algorithm works with the choice of bringing in this tol parameter, see the following plot again for G1 and $\lambda = 0.5$ which describes the evolution of the chain; note in particular that from approximately 800 iterations on, the algorithm proceeds to accept only increasing moves:



C. Results from test on distribution G1

What stands out immediately is that, as λ is increasing, both the expected number of selected cities and the expected maximum score are decreasing.
This intuitive evidence coming from the fact that λ measured the importance attributed to the deployment cost is confirmed also in the following plots:

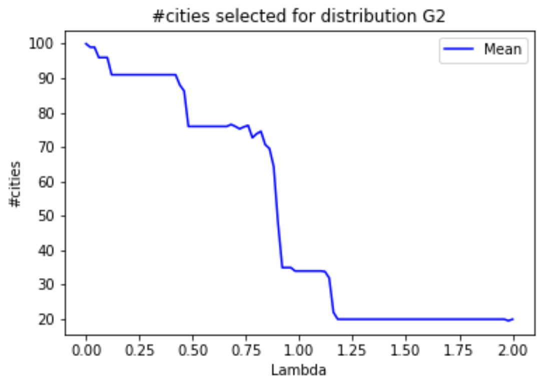


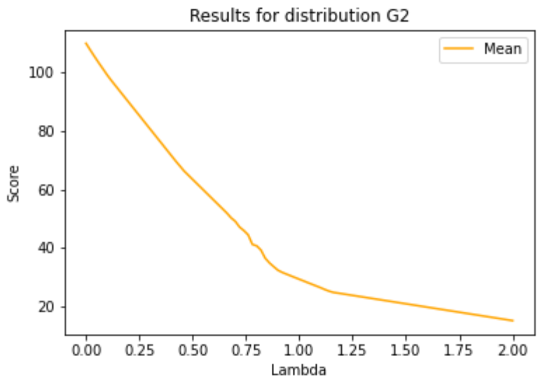


Note that the function $\lambda \mapsto \mathbf{E_{G1}}(|\mathbf{S}^{\star}(\lambda)|)$ seems to have a critical value $\lambda_{cr} \approx 0.5$. Perhaps a small perturbation of λ in a neighborhood of such a value λ_{cr} corresponds to a large perturbation in the expected number of selected cities. Moreover, also $\lambda \mapsto \mathbf{E_{G1}}(\mathbf{f}(\lambda, \mathbf{S}^{\star}(\lambda)))$ seems to have a decently smooth behaviour with the exception of a cusp exactly in a neighbourhood of λ_{cr}.

D. Results from test on distribution G2

As in the previous subsection for G1, as λ is increasing, both the number of selected cities and the maximum score are decreasing; in this case the behaviour of the mentioned quantities is the following:





where in this case, being $\lambda \in$ [0,2], one can even see the flattening of the blue curve for values of $\lambda \in$ [1,2]. Note that as in the results for $G1$, again there are some critical values of λ, but for $\lambda \approx 0.5$ and for $\lambda \approx 1.2$ this happens albeit in a less obvious and stunning way than before: for those values, the changes in $\mathbf{E_{G2}}(|\mathbf{S}^{\star}(\lambda)|)$ are more contained this time and this results coherently in the function $\mathbf{E_{G2}}(\mathbf{f}(\lambda, \mathbf{S}^{\star}(\lambda)))$ being able to maintain a certain degree on smoothness in a neighbourhood of those values of λ. Nevertheless, once again it seems that the central value of $\lambda \approx 1$ is the one that creates the most problems, as for that value $\mathbf{E_{G2}}(|\mathbf{S}^{\star}(\lambda)|)$ registers an abrupt change of values attained and $\mathbf{E_{G2}}(\mathbf{f}(\lambda, \mathbf{S}^{\star}(\lambda)))$ changes behaviour and starts decreasing faster, although this time being able to maintain some smoothness.

E. *Comments and conclusions*

First note that all the graphs are representing continuous functions everywhere on the domain of λ, i.e. there is continuous dependence on such a parameter. As discussed just above, the functions $\lambda \mapsto \mathbf{E}_{\mathbf{G_i}}(|\mathbf{S}^\star(\lambda)|)$ and $\lambda \mapsto \mathbf{E}_{\mathbf{Gi}}(\mathbf{f}(\lambda, \mathbf{S}^\star(\lambda)))$ for i = 1,2 have a smooth behaviour in general, except around values λ_{cr} for which the first registers a big "jump" and the latter seems to have the tendency to lose smoothness, although it has already been noted earlier that this is slightly more evident for i = 1. Nevertheless in both cases it is reasonable to think that this two phenomena are related to each other, i.e. as an abrupt change of behaviour in $\mathbf{E}_{\mathbf{G_i}}(|\mathbf{S}^\star(\lambda)|)$ leads to $\mathbf{E}_{\mathbf{G_i}}(\mathbf{f}(\lambda, \mathbf{S}^\star(\lambda)))$ losing its smoothness for the corresponding value of λ_{cr}. This translates mathematically in saying that the function $\mathbf{E}_{\mathbf{G_i}}(|\mathbf{S}^\star(\lambda)|) \mapsto \mathbf{E}_{\mathbf{G_i}}(\mathbf{f}(\lambda, \mathbf{S}^\star(\lambda)))$ has an average rate of change which is potentially unbounded for i = 1,2 in a neighbourhood of λ_{cr}.

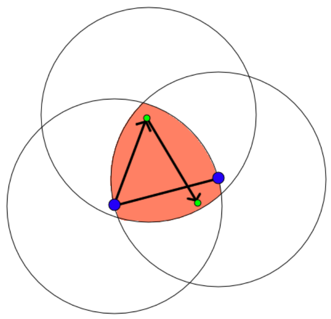
IV. Discussion and summary

A. *Strengths and weaknesses of the proposed algorithm*

Our implementation works well for relatively small instances, (20 minutes for running 5 chains over 2000 cities). We tried to optimize our algorithm by using the parallelism provided by numpy, we also cached a matrix containing the pairwise distances between all the cities. Those two things drastically reduced the computation time, but those two optimization are not enough when we are dealing with much larger data sets. As a result, we could use this implementation for countries like Switzerland that have few localities (as there are 2202 towns and villages in Switzerland). To increase the speed of the algorithm when there are a lot of cities, we could aggregate the cities that are in the same area into a larger agglomeration whose population is the sum of the smaller population, this would give an approximate solution.
Another limitation of our approach is that the tolerance parameter has to be manually changed when the scale of the problem changes, as it was chosen perhaps ad-hoc for the distributions on which the first test were made. Given the uncertainty of the input data sets the choice of tol cannot be made more precise than this, as we have seen in the $Results$ section that a perturbation of the order of $10^{-1} \div 10^0$ on the initial tolerance turns out to make a big difference between converging to a very good results or not.
Nevertheless, the algorithm has the clear merit of being able to converge to a medium to high quality approximation of the absolute best score in a fast time and for reasonably large instances, and it is able to overcome the initial issues presented by applying the simulated annealing heuristic.

B. *Adjustments in the case of larger data sets*

Regarding the upcoming competition, we had to find a faster algorithm. Indeed, our solution computes the distance between all pairs of points, which is intractable with a large number of cities. To that end, we leveraged the geometry of the problem and the fact that the random walk is iterative. Indeed, given we are at a state σ and that we know which are maximally distant points (which we call support points), one can find the maximal distance of the next state in linear time. The algorithm to determine theses points for the proposed next state is as follows:


In blue, the support points, in green, the new ones when removing one blue point. Green points are found by computing the maximally distant points twice from the remaining blue point. The orange area is where selected cities can be.

1) Suppose that the proposed move is to add a new city v_i. In order to determine whether the support points need to be updated, it is sufficient to compute the distance between v_i and the currently selected points. If there is a point at a distance larger than the current diameter, then we found the new points that determine the diameter and otherwise, we keep the ones we had previously.
2) Suppose now that the proposed move is to remove the city v_i from the selection. In case v_i is not a support point, the maximal distance of the possible next state is the same as the current one. However, if we want to remove one of the points on the diameter, we must be cautious. Let v_j be the remaining support point, that is, the point on the diameter that we do not propose to remove. We begin by computing the maximal distance among the remaining points and v_j. Call v_k the city we obtain. v_k will be our first support point for the proposed state. We finally compute the maximal distance between v_k and the same remaining points. The furthest city is the second support point.

It is possible to make the above algorithm completely rigorous using a geometry argument. Note that with this approach, we compute the distance between a point and the current selection a fixed number of time. Hence, the update of the cost term takes linear time at each iteration. For more detailed implementation details, please refer to the attached script.