***Computer Programming Assignment 2024***

Vittorio Manfriani – 3245185 – BEMACS

***Introduction***

In this report, we investigate the use of Simulated Annealing, a stochastic optimization technique inspired by the physical process of annealing in metallurgy, to solve the k-SAT problem. This study explores the performance of simulated annealing in solving k-SAT instances of varying complexity, characterized by parameters such as the number of variables (N), the number of clauses (M), and setting the clause size (k) equal to three. Key points relate around:

1. Choosing the parameters that optimize both accuracy and computational cost.
2. Analyzing the impact of algorithmic parameters such as the number of Monte Carlo steps, annealing schedule, and cooling rate on solution quality and computational efficiency.
3. Evaluating the empirical probability of finding a satisfying solution for different problem sizes and annealing parameters.
4. Investigating the scaling behavior of the algorithm, including the relationship between the number of variables and the algorithmic threshold, where the transition from solvable to unsolvable instances occurs.

***Parameters importance***

The choice of parameters in Simulated Annealing is crucial, as they directly influence the algorithm’s ability to explore the solution space and converge to an optimal result. In this analysis, I tested various configurations of parameters. The goal was to identify the optimal parameter settings that maximize the percentage of problems solved. In the following table are showed the result I got:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ***Beta 0*** | ***Beta 1*** | ***MCMC steps*** | ***Anneal Steps*** | ***Seed*** | ***Percentage Solved*** |
| **M = 200** | 1 | 10 | 200 | 20 | 45 | 100% |
| **M = 400** | 0.7 | 10 | 400 | 20 | 45 | 100% |
| **M = 600** | 0.5 | 10 | 3000 | 20 | 45 | 100% |
| **M = 800** | 0.1 | 10 | 4000 | 20 | 45 | 83% |
| **M = 1000** | 0,01 | 10 | 20000 | 20 | 45 | 0% |

The table shows that for smaller clause the success rate is 100%, indicating that the algorithm is capable of resolving problems effectively when the clause count is manageable. However, as M increases to 800, the success rate begins to decline to 83%, and it drops sharply to 0% at M = 1000. This decline suggests that as the number of clauses grows, the problem becomes increasingly complex, and the optimization process struggles to maintain sufficient exploration and convergence.

One key metric I used to evaluate the effectiveness of my parameters is the acceptance rate. It serves as a diagnostic tool for tuning the annealing schedule and other parameters. A balanced acceptance rate allows the algorithm to explore the solution space effectively early on and refine solutions efficiently as it converges. Ideally, the acceptance rate should decrease gradually alongside the cost, reaching just a few percent by the final step, ensuring the algorithm focuses on optimization rather than aimless exploration

The plot below shows the evolution of the acceptance rate for N = 200 and M = 200 and M = 800. When M = 200, the cost reaches 0 starting at state 3, after which all subsequent states have equal cost. For comparison, I included the results for M = 800, where no states share equal cost, to provide a complete perspective.

**A graph with a line

Description automatically generated**A graph with a line

Description automatically generated

***Parameters selection for Probability Analysis***

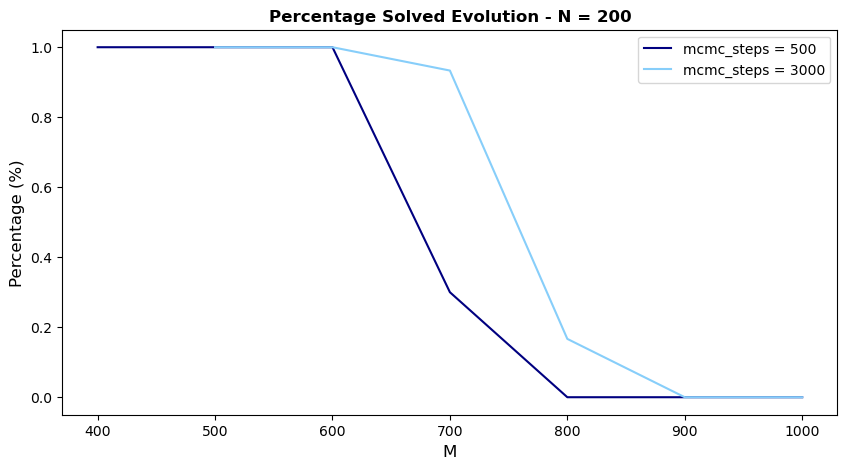
In the next section, we will run Simulated Annealing for various values of M and N . To ensure the performance comparisons are meaningful, we will use fixed parameters throughout the experiments. The selected parameter values are as follows:

* is set to 1 to ensure that, at the start of the annealing process, the acceptance rate consistently falls within an acceptable range of 30% to 80%, regardless of the specific values of N and M on which the algorithm operates.
* is set to 10.
* **MCMC steps** is set first to 500 and then to 3000 to test the results of the algorithm at different levels of accuracy. Would have been interesting to test even for even higher values but for running time issue I was not able to do so.
* **Anneal steps** are set to 20 to balance computational efficiency with the need for gradual temperature reduction, ensuring sufficient exploration of the solution space while avoiding premature convergence to suboptimal solutions.

***Ability to find solution as M grows large***

As previously stated, when increases, the Simulated Annealing algorithm does not consistently succeed in solving the 3-SAT problem or minimizing the cost to zero. Analyzing the algorithm’s performance across multiple values of M (up to 1000) while keeping N fixed at 200 reveals a significant trend: beyond a certain point, the algorithm becomes unable to always reliably find a solution. This limitation persists despite significant efficiency gains observed when increasing the number of MCMC steps from 500 to 3000.

To give a better picture of the limits of the simulated annealing, Below is the plot of how this probability changes for different values of M and N = 200 for fixed parameters of the annealing algorithm.



As the graph clearly illustrates, the Simulated Annealing algorithm struggles to consistently find a solution when M is sufficiently large and the empirical probability is decreasing in for both the mcmc\_steps values.

***Algorithmic Threshold***

The algorithmic threshold is the value of at which the empirical probability of solving a clause with Simulated Annealing equals 0.5. In the chart below, this threshold is represented by the point where the dashed line intersects the empirical probability curve at .For , as approaches , the solution space shrinks significantly due to the increasing number of constraints, resulting in a rapid decline in This sharp drop illustrates the system’s critical transition from a solvable to an unsolvable phase.

A graph of a number of data

Description automatically generated with medium confidence

The analysis was then extended to different values of , and the graph below shows how the value of algorithmic threshold changes as increases.

**A graph of a number of graphs

Description automatically generated with medium confidence**

**A graph of a mathematical equation

Description automatically generated with medium confidence**

The graphs show that the Algorithmic Threshold, , increases with  , which is intuitive because a larger number of variables    provides more flexibility to satisfy the constraints, making it easier for Simulated Annealing to find a solution. For larger  , the system can handle more clauses before the solution space collapses, meaning    must be higher to reach a probability of 0.5 of solving the problem. However, as  increases, the problem becomes harder to solve and eventually the algorithm always becomes completely unable to solve the 3-SAT problem. Also looking at the plots for different values of and demonstrate this same interplay between the solution space, constrained by the number of clauses , and the degrees of freedom introduced by the variables . Again, as approaches , again transitions sharply, with solvability dropping rapidly from likely to unlikely.

This behavior remains consistent across both levels of accuracy of the algorithm, although there are notable differences in the growth rate of , relative to . Specifically, as the number of MCMC steps increases from 500 to 3000, the growth rate of transitions to being nearly linear, following the relationship .

Conversely, when the accuracy of the algorithm decreases (e.g., with fewer MCMC steps), the algorithm struggles to maintain reliable performance as increases, causing the growth rate of to decrease and deviate from linearity. This slower growth reflects the algorithm’s inability to effectively manage the increased complexity of the problem with limited computational resources.

***Curve Collapsing***

The previous observation about the rate of growth of with respect to becomes crucial when moving to try to collapse the probability curves to one another.

First I started looking at the curves given 500 MCMC steps and I noticed that plotting vs caused all curves to collapse onto each other. Intuitively, this works because marks each curve’s own 50%‐success point. Near that point, the shape of the “difficulty profile” is fairly similar across different , so once you rescale M so that each curve’s midpoint is at x=1, they look alike, as shown in the following graph.

**A graph of a curve

Description automatically generated**

However, as previously noted, After increasing the number of MCMC steps itself is now closer to a constant times . Because of this near-linear scaling, a more natural way to make the curves “collapse” in this regime is to plotto plot vs . When we do so, the curves center around a constant —the same coefficient that ties and together that in this case seems to be something around 4.

**A graph of a falling graph

Description automatically generated**

From a broader perspective, these findings support the thesis that the difficulty of the problem can be understood through a single scaling variable that normalizes away the inherent differences in problem sizes. In the low-step regime, serves as the more nuanced scaling factor, accurately registering how many steps are required for that 50%-success threshold. In the high-step regime, where and appear linked by a near-constant ratio, scaling by itself becomes sufficient to align the curves. This scaling reveals that problem difficulty becomes a function of the ratio , highlighting the linear relationship between problem size and computational effort.