# SIMULATED ANNEALING AS A SOLVER FOR THE 3-SAT PROBLEM

Lorenzo Caputi, 3240499, 15/01/2025

## ABSTRACT

The K-SAT problem is a fundamental challenge in theoretical computer science and mathematical logic, at the basis of our understanding of NP-completeness. This report investigates the performance of **Simulated Annealing as a solver of 3-SAT**, especially in relation to the asymptotic behavior of the acceptance rate and the estimation of the algorithmic threshold.

For any fixed value of the number of variables, the **acceptance rate** approaches a lower horizontal asymptote as the number of clauses increase, because of the increasing complexity of the configuration space.

The **algorithmic threshold** is found to be highly dependent on the clauses-to-variables ratio (M/N) and located at values of M/N between 3.5 and 4.2. Moreover, the collected data suggest the existence of a **limiting value** for the algorithmic threshold, represented by problem instances for which the probability of satisfiability (PSAT) is 0.5, and dependent only on the clauses-to-variables ratio (M/N) having the value 4.27.

## K-SAT AND SIMULATED ANNEALING

K-SAT is a special case of the Boolean Satisfiability problem, commonly referred to as “SAT”, one of the most general discrete optimization problems existing, and the first problem to be proven NP-complete.[1] That is, no deterministic polynomial algorithm can solve SAT, and if such an algorithm existed, by the means of reduction techniques, all non-deterministic polynomial (NP) problems could be solved by a deterministic polynomial algorithm, thus proving P=NP, one of the most important conjectures in science.

Given the exceptional relevance of the problem, numerous successful approaches have been taken in the past decades to solve SAT, such as the DPLL[2] and the WalkSAT[3] algorithms. This report will adopt a Simulated Annealing based approach.

Simulated Annealing[4] is a probabilistic technique for approximating the minimum of a function, inspired by Physics. It is an extension of the greedy approach since at every decision point, a proposed move is always accepted if the new configuration has a lower cost, but there is still some non-zero probability of accepting a move which increases the cost. This probability depends exponentially on the magnitude of the increase in cost and is reduced during the optimization process in a similar fashion as temperature decreases during the annealing process.

Especially for hard problems with a large and irregular configuration space, this added randomness is beneficial, since it provides a remedy to the high dependence of greedy approaches on the initial configuration, preventing the solution to always be the point of local minimum closest to such initial configuration.

It follows that in the Simulated Annealing approach, the choice of the optimization parameters is key, to establish the right balance between exploration and exploitation, while containing computational cost.

## OPTIMIZATION PROBLEM SET UP

As usual in the context of Simulated Annealing as a solver, one needs to define the configuration space, a cost function, and a move proposal scheme. It follows a more detailed description of each point in the context of the 3-SAT problem.

- **Configuration**: a configuration is defined as a Boolean assignment of values to variables, with a change of variable such that the value 1 is interpreted as True, and -1 as False. The choice of this representation is motivated by the easiness of flipping the sign of a variable, which accounts to multiplying it by -1.
- **Cost function**: the cost of a configuration is defined as the number of unsatisfied clauses, expressed mathematically as follows:

$$E(\hat{\mathbf{x}}) = \sum_{m=1}^{M} \left( \prod_{k=1}^{K} \frac{1 - s_{m_k} \hat{x}_{m_k}}{2} \right).$$

In the code (*KSAT.py*), a slightly different but equivalent form for the cost function is used since it was proven to be around 20% faster (*cost_efficiency.py*).

- **Move proposal**: the move proposal scheme is the trivial flipping of one variable's sign. This proposal is obviously symmetric, always proposes a different configuration than the current one and respects the properties of connectedness and aperiodicity required by the Metropolis-Hasting algorithm[5], which is at the basis of the Simulated Annealing approach.

## CHOICE OF PARAMETERS

The optimization parameters were chosen to be 1000 mcmc steps and 100 annealing steps, with a value for beta0 of 0.1 and of 10 for beta1. If not specified otherwise in the report, all experiments were conducted with a seed equal to 42 to allow reproducibility.
The main criterion considered in the choice of the parameters was to obtain sufficient exploration of the configuration space, while containing computational cost to allow for the possibility of running multiple tests and analyzing results.
It follows a more detailed description of the reasoning behind each choice:

- **Mcmc steps**: the number of move proposals for every value of the temperature. It is important to choose a high enough value for the Markov chain to provide a reliable estimate of the stationary distribution of the configurations, which will prioritize low-cost configurations more as temperature decreases. After extensive testing, the required trade-off between performance and computational cost was found at a value of 1000.
- **Annealing steps**: number of times the temperature is decreased during the annealing process. It was chosen to be 100, with the implementation of an early stopping condition which terminates the search once a configuration of cost 0 is found, so when all constraints are satisfied. This way, no computation is wasted trying to optimize an already solved problem. Moreover, harder problems will naturally require a larger number of annealing steps, so the actual value will scale with the difficulty of the problem and 100 is to be intended as an upper bound.
- **Beta0**: this is 1/T0, with T0 being the initial temperature. It was chosen to be 0.1 to allow for a sufficiently high acceptance rate at the beginning of the optimization process (around 0.9), keeping enough randomness in the acceptance scheme.
- **Beta1**: this is 1/T1, where T1 is the last temperature before the last annealing step, performed at 0 temperature, that is equivalent to running the greedy algorithm, since the probability of accepting a move which increases the cost is always 0. It was chosen to be 10 such that, considering the choices of the other parameters, the acceptance rate at the last annealing steps for sufficiently hard problems reaches a value slightly below 0.1.

In general, the parameters were chosen to be fixed and not to adapt to the size of the problem, to be able to obtain evidence on N and M as the sole determinants of the algorithmic threshold, and of the asymptotic behavior of the acceptance rate. If the goal of this report were to obtain the best possible performance, an adaptive approach setting the values of the mcmc steps based on the size and hardness of the problem, for instance by making it scale with N or with the clauses-to-variables ratio (M/N), would have been preferred.

## OPTIMIZATION AND ACCEPTANCE RATE ANALYSIS

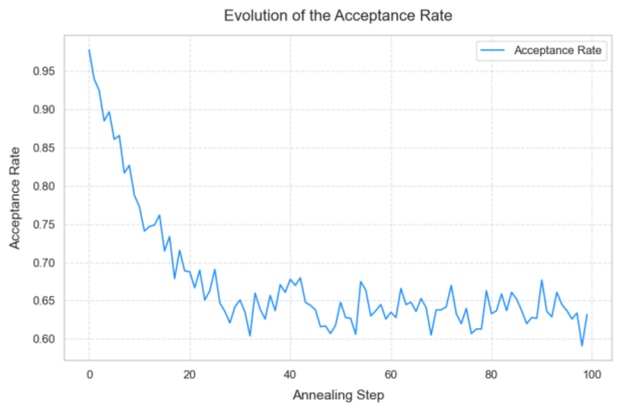

**Figure 1: Evolution of the Acceptance Rate** for N=200, M=200. Source code: *single_solver.py*

This part focuses on the quantitative analysis of the performance and of the acceptance rate during the optimization. At first, the analysis is conducted fixing the value of 200 for both N and M, then the same analysis is extended to different values for the number of clauses.

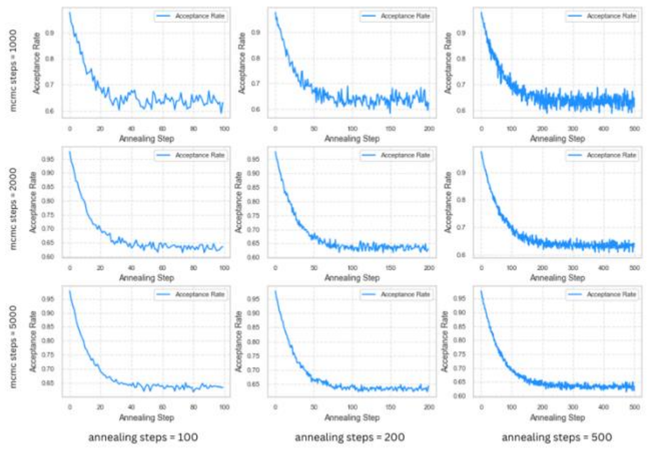

**Figure 2**: **Impact of mcmc steps and annealing steps** on the evolution of the acceptance rate, for N=200, M=200.
Source code: *multi_plot.py*

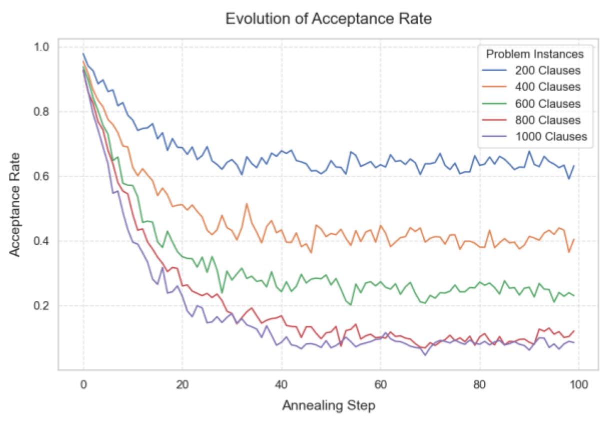

**Figure 3:** Comparing the evolution of the **acceptance rate across different values of M**, keeping N=200.
Source code: *multiple_solver.py*

Fixing N=200, M=200:
As shown in **Figure 1**, considering the values N=200, M=200, the acceptance rate starts high, above 0.9, and decays with the annealing steps, until it stabilizes at a value around 0.6. The algorithm is always able to find a solution in less than 100 annealing steps. This is due to the relatively small and well-connected configuration space found at N=200, M=200, in which the algorithm finds global minima and transitions between them relatively easily.
To show the independence of this limiting value from the depth of the exploration, the same analysis was performed changing the optimization parameters and the results are summarized in **Figure 2**.

Different values of M:
Running the same experiment with different values for M, we notice a change in performance and a different behavior of the acceptance rate, as shown in **Figure 3**. The algorithm confidently finds a solution for values of M up to 700, whether a solution is found is uncertain for values between 700 and 900, while it is never able to find a solution for values of M above 900.
As regards the acceptance rate, as M gets larger it decays faster and always keeps a lower level, heading towards a **lower horizontal asymptote**. This is again explained by the increasing complexity and irregularity of the configuration space: as the number of clauses increases, with a fixed number of variables, more constraints are imposed on the same variables, reducing the number of satisfying assignments as well as the easiness of transitioning from one to another flipping the sign of variables.

So, the problem becomes significantly harder as the relative magnitudes of M and N change towards a larger M compared to N, and this becomes evident from the asymptotic behavior of the acceptance rate.

## ALGORITHMIC THRESHOLD ANALYSIS

In this part, the focus is placed on the study of the transition phase of 3-SAT, from an easy problem to a hard one, as the relative magnitudes of M and N change, via the identification of the algorithmic threshold.
Using Simulated Annealing as a solver, the algorithmic threshold of 3-SAT for a given number of variables is the number of clauses such that the probability P(N,M) of finding a solution to a random instance is equal to 0.5.
To obtain an estimate of the algorithmic threshold, the empirical probability of finding a solution is evaluated, first keeping N fixed at a value of 200, to then assess the impact of both N and M together.

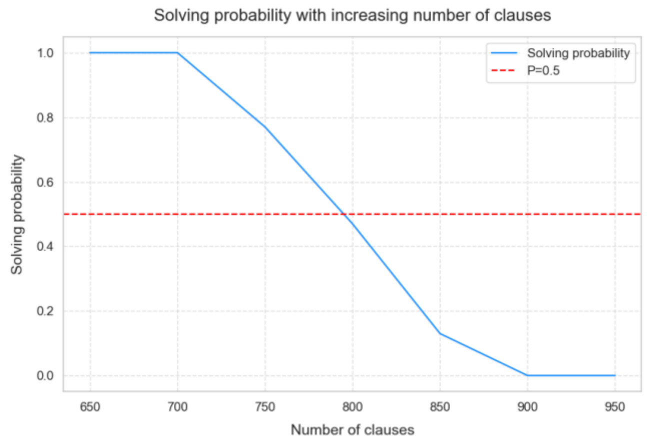

**Figure 4:** Empirical estimate of the **solving probability of a random instance** of 3-SAT, for N=200 and M between 650 and 950. Source code: *plot_probability.py*

N = 200:
To identify this probability, 30 instances of the 3-SAT problem are created, of course without setting the same seed for all, but leaving it to None, for different values of M. Then, the fraction of solved instances is used as an estimate for P(N,M). The results are summarized in **Figure 4**.
The results confirm the intuition outlined in the previous paragraph, as the algorithm confidently solves problem instances for values of M up to 700, is uncertain between 700 and 900, and cannot solve instances for M larger than 900. Adopting a linear interpolation technique, so assuming a linear behavior of the solving probability function at missing values between any two observations, an estimate of **M=795** is found for the algorithmic threshold.
As regards the shape of the solving probability function close to the algorithmic threshold, a fast decay is found, similar to that of a sigmoid function, with the characteristic S-shaped graph.

Different values of N:
To understand what happens to the system of variables and clauses as M approaches the algorithmic threshold, it is useful to repeat the same analysis for different values of N. As shown in **Figure 5** and applying the same linear interpolation technique as before, a clear pattern emerges. The algorithmic threshold is found for a value of the clauses-to-variables ratio **(M/N) in the interval (3.5,4.0)**. So, as the algorithmic threshold is approached, in the system of variables and clauses a ratio between 3.5 and 4 emerges. To provide a visual proof of this process, **Figure 6** shows a rescaling of the X axis of the plot from M to M/N and all the satisfying probability curves collapse in the range (3.5,4.0).

## LIMITING VALUE FOR THE ALGORITHMIC THRESHOLD

It is possible to note that the algorithmic threshold, expressed in terms of M/N, seems to decrease as N increases. This can be explained by the fact that, as both N and M grow keeping their ratio constant, the solutions space indeed gets larger, as more assignments of values to variables are possible. Thus, more exploration is needed to find a solution in this more difficult case. So, the reason why the algorithmic threshold expressed as M/N decreases with N is that the algorithm being used is not perfect, and it performs worse for problem instances requiring larger exploration.
In theory, if one had a perfect algorithm, the probability to find a solution to a given instance of 3-SAT would be equal to the probability that such a solution exists. More advanced techniques exploiting statistical physics[6] have led to the conclusion that the probability of the satisfiability of a random 3-SAT instance is: approximately 1 for values of M/N significantly lower than 4.27, approximately 0 for values of M/N significantly larger than 4.27, and transitions from 1 to 0 very sharply for values around 4.27. Assuming a symmetric behavior of the satisfiability probability (PSAT) around a value of M/N of 4.27, its expectation at M/N=4.27 shall be exactly 0.5. That is, the best guess for the algorithmic threshold with a perfect algorithm shall be exactly 4.27.

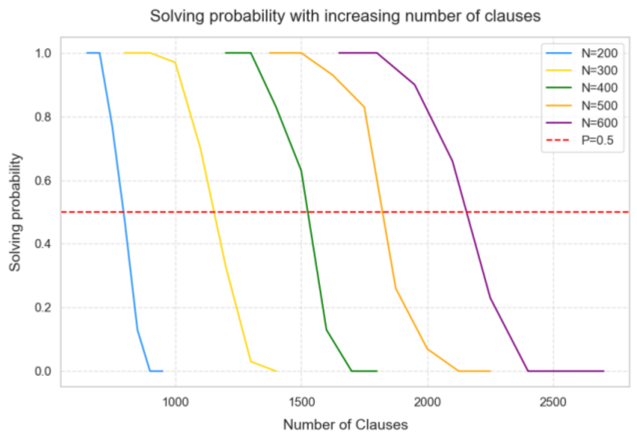

**Figure 5**: Empirical estimate of the **transition phase of the solving probability** of a random instance of 3-SAT, evaluated for N in {200, 300, 400, 500, 600}. The estimated algorithmic thresholds were: 795, 1154, 1526, 1822, 2155.
Source code: *plot_probabilities_multiple_m.py, intersections.py*

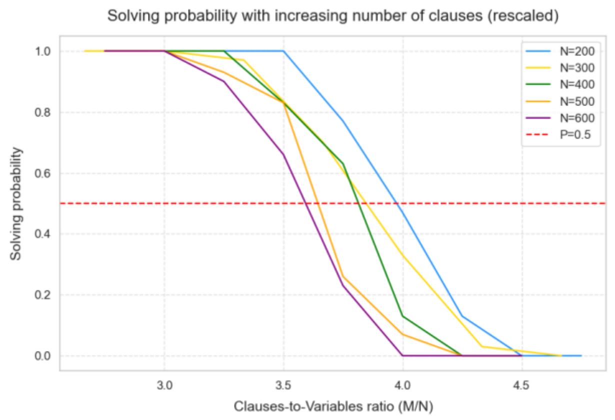

**Figure 6**: **Rescaling** of the empirical estimate of the **transition phase of the solving probability** of a random instance of 3-SAT, evaluated for N in {200, 300, 400, 500, 600}. The estimated algorithmic thresholds were: 3.97, 3.84, 3.81, 3.64, 3.59.
Source code: *plot_probabilities_multiple_m.py, intersections.py*

This motivates the idea that improving the algorithm makes the empirical estimate of the algorithmic threshold approach the theoretical value of 4.27.
To provide evidence of this intuition, the following experiment was performed: a fixed value of N=200, M=200 was chosen, and the algorithmic threshold was estimated as previously done, with different choices of the optimization parameters. The resulting solving probability curves are plotted in **Figure 7**.
The optimization parameters in the form (mcmc steps, annealing steps) were: {(1000, 100), (2000, 200), (3000, 300), (4000, 400), (5000, 100)}.
It is clear from this experiment that a better algorithm, that is, an algorithm which explores the configuration space more deeply, results in a higher estimate for the algorithmic threshold. Such estimates were: 3.97 for 1000 mcmc steps, 4.07 for 2000 mcmc steps, 4.13 for 3000 mcmc steps, 4.15 for 4000 mcmc steps, and 4.16 for 5000 mcmc steps.
All in all, this suggests the idea of the **existence of a limiting algorithmic threshold**, the one found in problem instances for which the probability of satisfiability is 0.5, which amounts to a clauses-to-variables ratio (M/N) of 4.27.

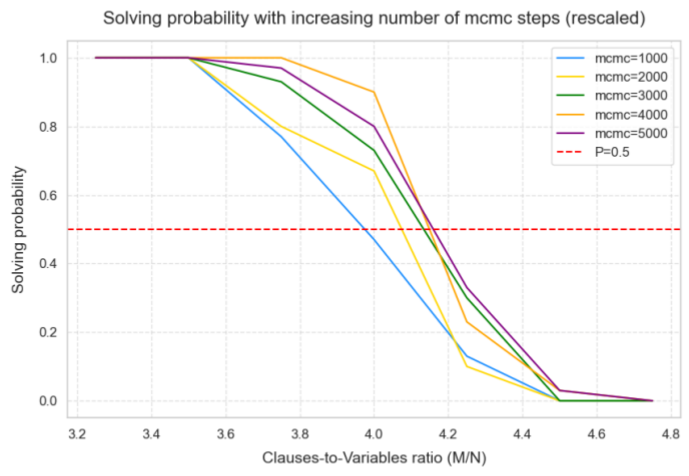

**Figure 7:** Empirical estimate of the **solving probability of a random instance** of 3-SAT, for N=200 and M=200. The **optimization parameters** in the form (mcmc steps, annealing steps) were: {(1000, 100), (2000, 200), (3000, 300), (4000, 400), (5000, 100)}. Source code: *limiting_alg_threshold.py*

## REFERENCES

[1] Cook, S. A. The complexity of theorem-proving procedures. *Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC)*, 151–158 (1971). Retrieved from: https://doi.org/10.1145/800157.805047
[2] Davis, M., Logemann, G., & Loveland, D. (1962). A machine program for theorem-proving. *Communications of the ACM*, 5(7), 394–397. Retrieved from: https://doi.org/10.1145/368273.368557
[3] Selman, B., Kautz, H. A., & Cohen, B. (1994). Noise strategies for improving local search. *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, 337–343.
Retrieved from: https://www.aaai.org/Papers/AAAI/1994/AAAI94-053.pdf
[4] Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598), 671–680. Retrieved from: https://doi.org/10.1126/science.220.4598.671
[5] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6), 1087–1092. Retrieved from: https://doi.org/10.1063/1.1699114
[6] Dubois, O., Boufkhad, Y., & Mandler, J. (2000). Typical random 3-SAT formulae and the satisfiability threshold. *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, 126–127. Retrieved from: https://arxiv.org/abs/cs/0211036