Project of Computer Programming: Investigating 3-SAT Properties and Optimization

Tommaso Pascotto

15-01-2024

Performance of Simulated Annealing algorithm

First, we must choose proper values for $MCMCsteps$, $annealsteps$, β_0, and β_1. These parameters affect the program's ability to complete the task successfully; however, different parameters will result in different computational costs.

For $N = 200$ and $M = 200$, it is hard to find good parameters for an optimal implementation of the simulated annealing algorithm, since even with $\beta = \infty$, the acceptance rate rarely falls under 50% and acceptance rates fluctuate oddly. This is due to the nature of the problem: as we will see later during our analysis, problems with a ratio $M/N \leq 3$ are relatively "easy" to solve and have a high acceptance rate. With the ratio $M/N = 1$, solving the problem with simulated annealing is like "killing a mosquito using a cannonball" - the goal of solving the problem is satisfied but more computational cost than necessary is spent in the process.

Table 1: Initial Parameters for Analysis

Parameter	Value
MCMC steps	200
Annealing steps	20
β_0	1
β_1	10
Seed	123

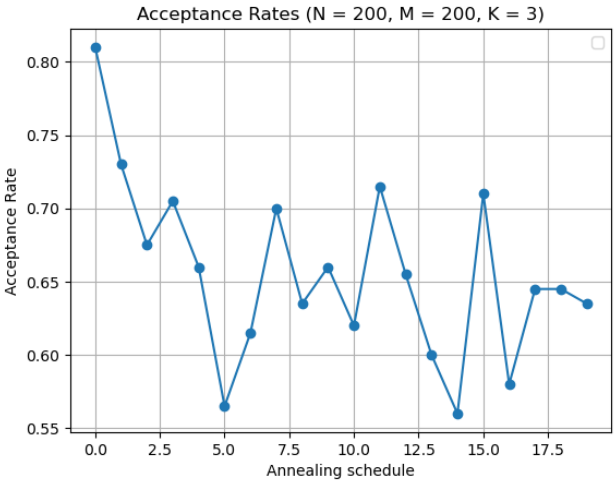
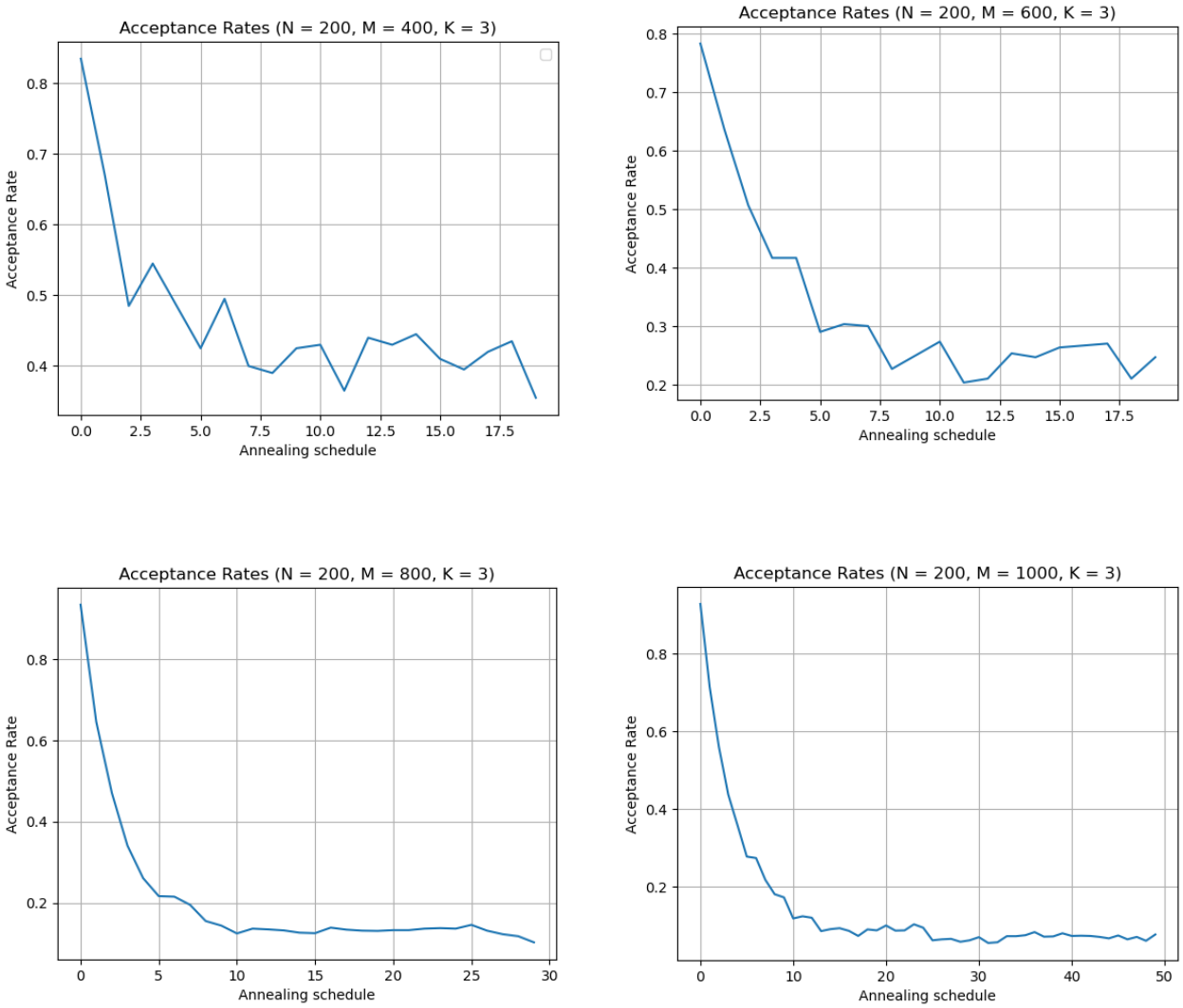


Table 2: Parameters for Different M Values

M	MCMC Steps	Annealing Steps	β_0	β_1
400	200	20	0.5	10
600	300	20	0.5	10
800	1600	30	0.1	15
1000	1600	50	0.1	15



For $N = 200$, the algorithm performs well in solving problems with $M \leq 600$, while for more complex issues ($M > 600$), finding a low cost depends mostly on the parameters chosen, and for $M = 1000$ the probability is close to 0.

Also, as we see in the graphs, the acceptance rate evolution decreases more uniformly and to a few digits of percentage as the problem increases in complexity.

When approaching more complex problems, we need to remember that different parameters of the algorithm help solve different types of problems. When working with $M > 700$, the best approach is to use a slow cooling strategy:

- Low starting temperature

- High final temperature
- Many annealing steps
- Many steps per temperature

In this way, we ensure a gradual temperature-decreasing exploration of the problem at each temperature, and it is more likely to find a global minimum. This is the best approach for complex landscapes with many local minima, but obviously, a short running time is exchanged for a good-quality solution.

General 3-SAT Properties

To understand the change in the solving probability as M increases we use the following approach: for each value of M, we run several trials to study the empirical solution probability $P(M, N)$. More precisely, we keep $n = 30$ as the number of trials we run for each value of M. The problem increases in complexity as M increases, and therefore the parameters are chosen properly at each value of M. This adaptive approach is necessary to save computational cost, as it would be very inefficient to adopt uniform parameters as the complexity increases: a larger M requires more intense research for an answer, while low M might be less critical and, thus, not require such attention. Mostly, throughout the execution, the following parameters are adapted: $annealsteps$, $MCMCsteps$, and the inverse temperature range from β_0 to β_1. These parameters are gradually increased along with the rise of M approaching the threshold of the algorithm. This would allow a wide exploration of solution space. As M increases the complexity of the problem, these parameters are also increased gradually. This would allow an exhaustive search to reach $cost = 0$. For example, considering $M \leq 700$, the solution space has fewer constraints, and the solutions are easily found, therefore we opt for an aggressive cooling schedule with fewer steps. When M approaches bigger values, higher $mcmcsteps$ are needed to enable more in-depth sampling at each temperature. This enables the algorithm to test a wider range of solutions.

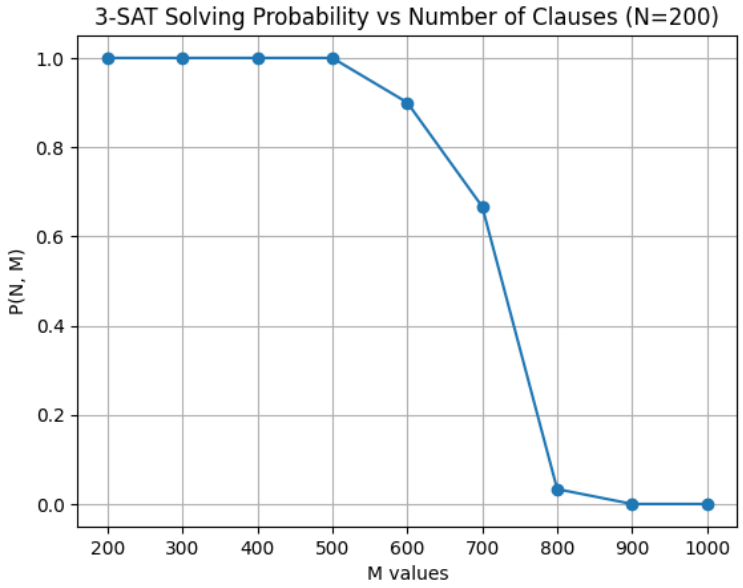


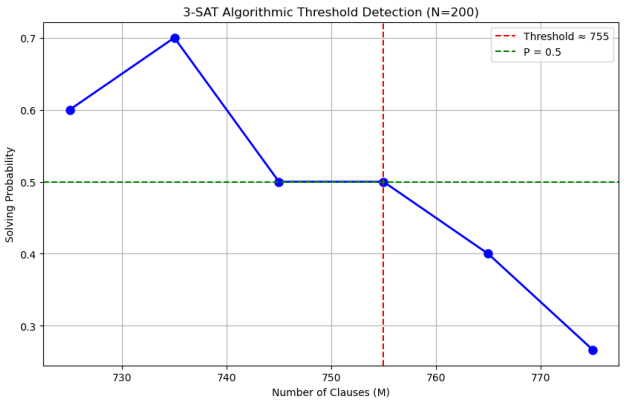
Table 3: Parameters for Solving Probability Analysis

M	MCMC Steps	Annealing Steps	β_0	β_1	Seed
200	200	20	1	10	random
300	200	20	1	10	random
400	400	20	1	10	random
500	600	20	0.1	10	random
600	800	20	0.1	15	random
700	1000	20	0.1	20	random
800	1100	20	0.1	25	random
900	1400	20	0.1	30	random
1000	1700	20	0.1	30	random

Because of the large sample size required and the associated computational cost, an exact threshold is hard to determine. Initially, a binary search approach was attempted, but this couldn't be pursued as it would have required too long processing time to yield any meaningful results. Thus, an upper limit between $M = 725$ and $M = 775$ was first determined and, in this range, 6 M values were evenly space. This method allows us to focus on a more narrow range to identify better the threshold. The threshold was identified as the point crosses the $p = 0.5$ curve without any further recovery. Given that the range is quite narrow in this case the only parameter to change is *mcmcsteps*.

Table 4: Parameters for Threshold Detection

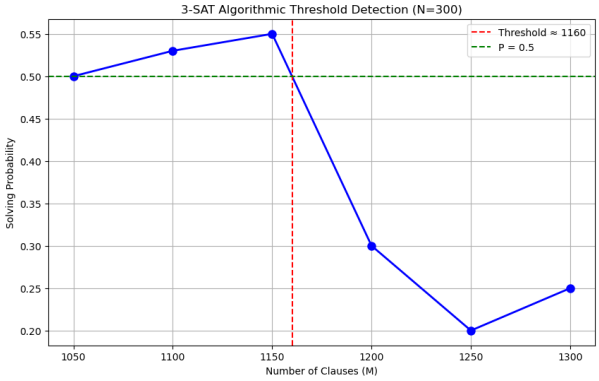
Parameter	Value
MCMC steps	$M * 2$
Annealing steps	20
β_0	0.1
β_1	25
Seed	Random for each iteration

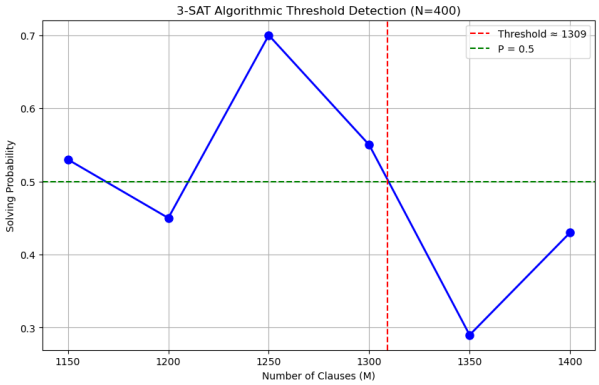


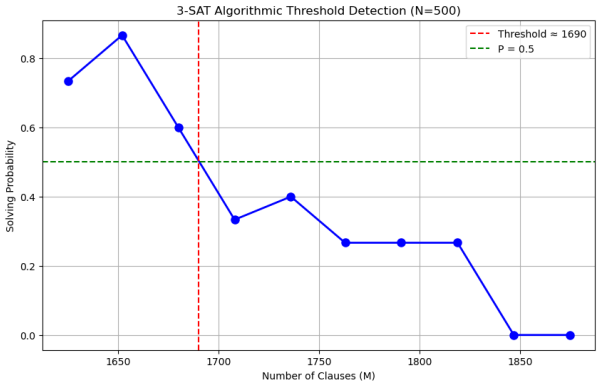
Looking at the evolution in solving probability, we can see a dramatic change in behavior as we approach what we call the algorithmic threshold, which represents a critical point in the relationship between the number of clauses (M) and variables (N). The shape of the probability curve near the threshold resembles what in mathematics is called a step function, where the probability drops sharply from almost 1 to nearly 0 around $M \approx 700 : 800$ for $N = 200$. When we have fewer clauses the algorithm can easily "move" between them to find a valid solution, that is: find a variable assignment that simultaneously satisfies multiple clauses,

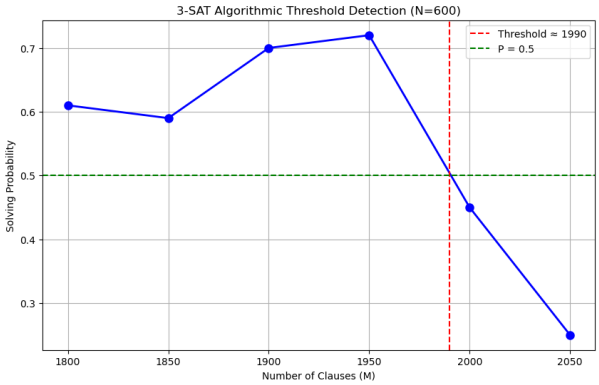
this is why the probability in this case is close to 1. However, when the number of clauses increases, the algorithm can rarely find a solution for a clause which is also a solution for other clauses and that is why there is a clear boundary between "easy" and "hard" instances.

-

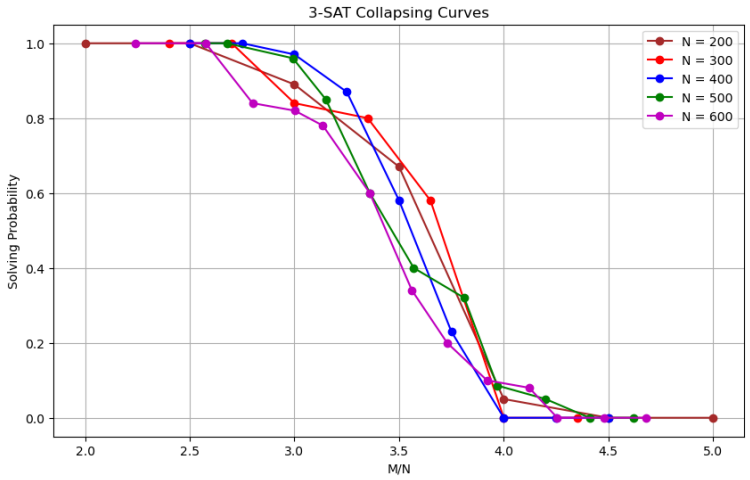








Looking at the graphs above which show the threshold for different values of N, we can see that the boundary increases as N increases. That is because the algorithm can handle more clauses M while maintaining solvability. This is because additional variables give the algorithm more "room" to find valid solutions, balancing the constraints added by new clauses. The boundary at which the algorithm's performance undergoes a significant change is, therefore, determined also by the number of N. However, it is important to note that as N grows larger the search space increases exponentially requiring more computational resources or iterations to find a satisfying solution.



Specifically, as we see in the collapsing curve, we can see that the solving probability approaches the threshold as $M/N \approx 3.5$. For any given number of variables N, we can estimate that the threshold M will be around $3.5 \times N$. This means we don't have to test every possible value of M - we can focus our search around this estimate, which saves a lot of computing time and effort.