CS 8803
Logic in Computer Science

Project 1 – Final Report

Name: Saranya Kavileswarapu
Email: skaviles3@gatech.edu
GTId: 903869435
Link to Github: https://github.com/saranya-05/DPLL-SAT-Solver

DPLL Heuristics

1. **Random-choice:** This heuristic is used in SAT-solving to select unassigned literals randomly from a CNF formula. It collects unassigned literals from each clause, which represent variables that have not yet been assigned truth values. The function checks for an empty list, indicating all literals are assigned, and returns None if so. Essentially, this heuristic introduces randomness into SAT-solving by randomly choosing unassigned literals, aiding exploration of different branches in the search tree. While it can enhance the efficiency of finding satisfying assignments, it's a stochastic heuristic, relying on chance rather than specific problem patterns or heuristics.
2. **2-Clause:** This heuristic focuses on identifying literals in a CNF formula that appear in exactly two clauses. It begins by counting the occurrences of each literal in the remaining clauses, storing this information in a dictionary. Then, it filters the list of literals to retain only those that meet the criterion of appearing twice. If no such literals are found, the heuristic defaults to choosing a random literal from the first clause, or None if the CNF formula is empty. However, if there are literals that match the criterion, it selects one of them randomly. The purpose of this heuristic is to leverage the presence of literals in precisely two clauses to potentially improve the efficiency of SAT-solving by exploring promising paths in the search for a satisfying assignment.
3. **Max-literal-weight:** This heuristic calculates a weight for each literal in a CNF formula based on its appearance in clauses. It creates a defaultdict to store these weights, where each literal's weight is initially set to zero. It then iterates through the formula's clauses and literals, updating the weights according to a specific formula: each literal's weight is increased by 2 raised to the power of the negative length of the clause it belongs to. Finally, it returns the literal with the highest weight as the selected literal. This heuristic aims to prioritize literals that appear in shorter clauses, as they are often considered more critical in the SAT-solving process, with the goal of potentially improving the efficiency of finding a satisfying assignment.

Results

100 experiments were performed for each heuristic as a function of L/N where N=[100,150,200], L/N=[3,3.2,3.4,3.6,3.8,4,4.2,4.4,4.6,4.8,5,5.2,5.4,5.6,5.8,6]. For some larger values of N and ratios the code terminated timed out as it was taking a lot of time to compute. The results are presented below:

Random Choice				
N	L/N	Compute time (s)	No. of DPLL	Probability
100	3	0.0177	129	1
100	3.2	0.1219	129	1
100	3.4	0.3497	129	1
100	3.6	1.0129	129	1
100	3.8	2.6069	129	1
100	4	9.4719	129	0.96
100	4.2	19.3018	129	0.73
100	4.4	24.404	129	0.46
100	4.6	22.9287	129	0.22
100	4.8	19.7711	129	0.05
100	5	15.8595	129	0
100	5.2	11.2696	129	0
100	5.4	9.2136	129	0
100	5.6	7.3807	129	0
100	5.8	6.2339	129	0
100	6	5.2079	129	0
150	3	0.1999	129	1
150	3.2	0.9282	129	1
150	3.4	4.1764	129	1
150	3.6	23.8184	129	1
150	3.8	135.2142	129	1
150	4	#N/A	129	
150	4.2	#N/A	129	
150	4.4	#N/A	129	
150	4.6	#N/A	129	
150	4.8	#N/A	129	
150	5	#N/A	129	
150	5.2	#N/A	129	
150	5.4	#N/A	129	
150	5.6	#N/A	129	
150	5.8	#N/A	129	
150	6	#N/A	129	
200	3	1.073	63	1
200	3.2	23.0003	63	1
200	3.4	34.2755	63	1
200	3.6	486.9011	63	1
200	3.8	#N/A	63	
200	4	#N/A	63	
200	4.2	#N/A	63	
200	4.4	#N/A	63	
200	4.6	#N/A	63	
200	4.8	#N/A	63	
200	5	#N/A	63	
200	5.2	#N/A	63	
200	5.4	#N/A	63	
200	5.6	#N/A	63	
200	5.8	#N/A	63	
200	6	#N/A	63	

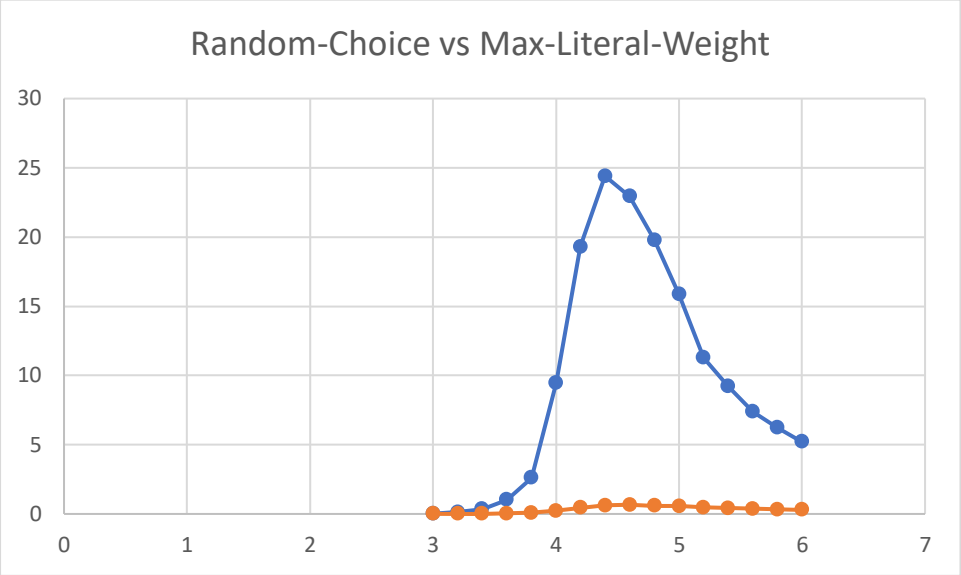
2-clause				
N	L/N	Compute time (s)	No. of DPLL	Probability
100	3	8.178	6	1
100	3.2	39.6343	6	1
100	3.4	126.0792	6	1
100	3.6	#N/A	6	
100	3.8	#N/A	6	
100	4	#N/A	6	
100	4.2	#N/A	6	
100	4.4	#N/A	6	
100	4.6	#N/A	6	
100	4.8	#N/A	6	
100	5	#N/A	6	
100	5.2	#N/A	6	
100	5.4	#N/A	6	
100	5.6	#N/A	6	
100	5.8	#N/A	6	
100	6	#N/A	6	
150	3	#N/A	57	
150	3.2	#N/A	57	
150	3.4	#N/A	57	
150	3.6	#N/A	57	
150	3.8	#N/A	57	
150	4	#N/A	57	
150	4.2	#N/A	57	
150	4.4	#N/A	57	
150	4.6	#N/A	57	
150	4.8	#N/A	57	
150	5	#N/A	57	
150	5.2	#N/A	57	
150	5.4	#N/A	57	
150	5.6	#N/A	57	
150	5.8	#N/A	57	
150	6	#N/A	57	
200	3	#N/A	57	
200	3.2	#N/A	57	
200	3.4	#N/A	57	
200	3.6	#N/A	57	
200	3.8	#N/A	57	
200	4	#N/A	57	
200	4.2	#N/A	57	
200	4.4	#N/A	57	
200	4.6	#N/A	57	
200	4.8	#N/A	57	
200	5	#N/A	57	
200	5.2	#N/A	57	
200	5.4	#N/A	57	
200	5.6	#N/A	57	
200	5.8	#N/A	57	
200	6	#N/A	57	

Max-literal-weight				
N	L/N	Compute time (s)	No. of DPLL	Probability
100	3	0.0082	64	1
100	3.2	0.0106	64	1
100	3.4	0.0174	64	1
100	3.6	0.0398	64	1
100	3.8	0.0806	64	1
100	4	0.2152	64	0.96
100	4.2	0.4521	64	0.78
100	4.4	0.6035	64	0.43
100	4.6	0.6644	64	0.21
100	4.8	0.5942	64	0.04
100	5	0.5732	64	0
100	5.2	0.4763	64	0
100	5.4	0.4206	64	0
100	5.6	0.3732	64	0
100	5.8	0.3268	64	0
100	6	0.3065	64	0
150	3	0.0194	64	1
150	3.2	0.0321	64	1
150	3.4	0.0554	64	1
150	3.6	0.2	64	1
150	3.8	1.0874	64	1
150	4	2.6263	64	1
150	4.2	8.5616	64	0.7
150	4.4	11.0698	64	0.37
150	4.6	10.8332	64	0.08
150	4.8	8.6896	64	0.02
150	5	6.1676	64	0
150	5.2	5.1724	64	0
150	5.4	4.1598	64	0
150	5.6	3.6539	64	0
150	5.8	3.1928	64	0
150	6	2.6052	64	0
200	3	0.0308	64	1
200	3.2	0.0725	64	1
200	3.4	0.1212	64	1
200	3.6	0.8432	64	1
200	3.8	3.7729	64	1
200	4	23.9345	64	1
200	4.2	129.6009	64	0.75
200	4.4	160.3261	64	0.29
200	4.6	135.0932	64	0.04
200	4.8	86.8682	64	0
200	5	63.8232	64	0
200	5.2	46.4274	64	0
200	5.4	37.6855	64	0
200	5.6	26.8314	64	0
200	5.8	21.7385	64	0
200	6	17.5758	64	0

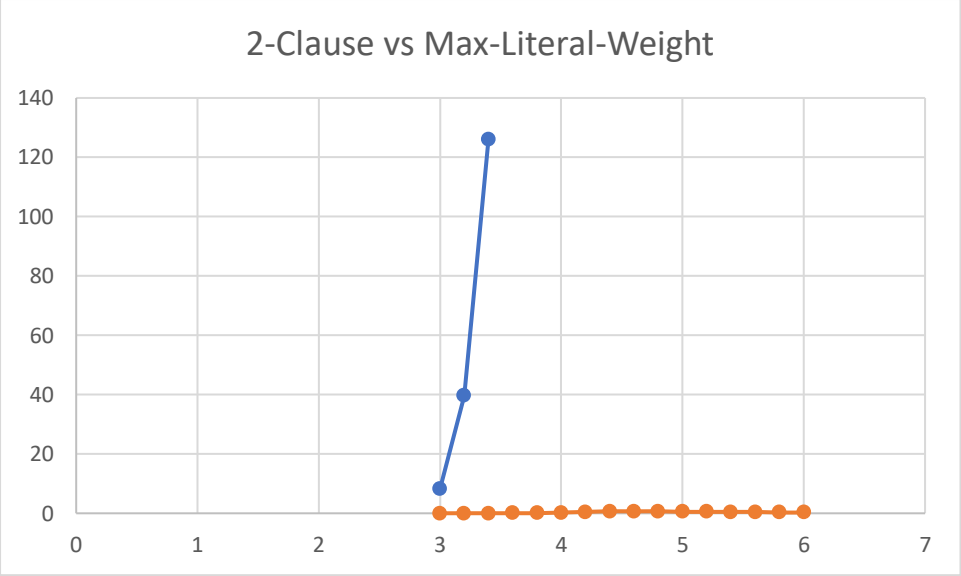
The results demonstrate a noticeable decrease in the probability of satisfiability as the L/N ratio increases. This phenomenon is indicative of larger datasets being processed by the DPLL satisfiability algorithm, which, as data size grows, tends to yield lower probabilities of satisfiability. This behaviour aligns with the concept of a transition phase in random SAT instances, wherein a shift occurs from a regime where most instances are satisfiable to one where most instances are unsatisfiable due to increased problem constraints. This

observation holds significance in gaining insights into the intricacies of solving SAT and other NP-hard problems.
The below plots have the x-axis as L/N ratio and y-axis as computation time.

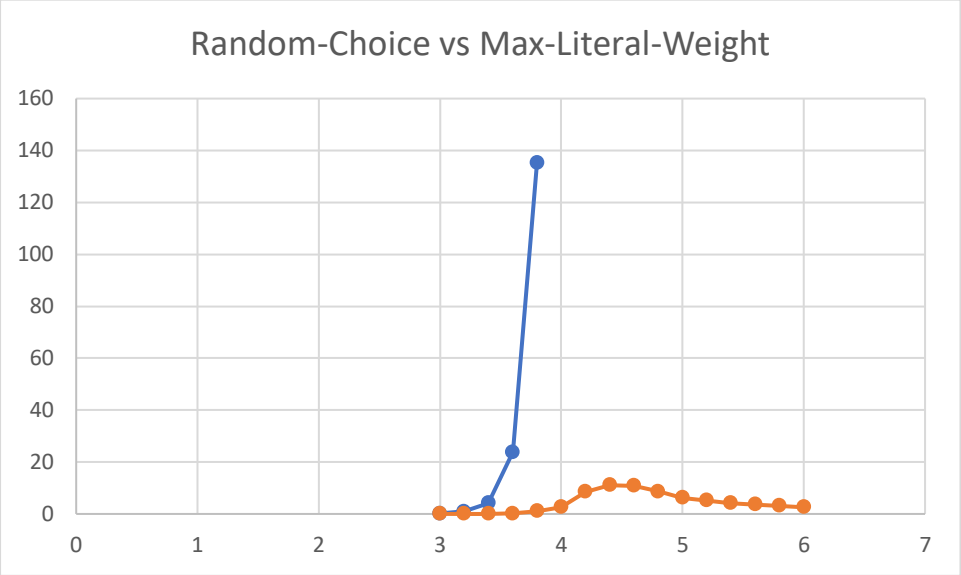
Plot of compute time of Random-choice and Max-literal-weight as a function of L/N where N=100. The blue line depicts the performance of random-choice and orange line shows the max-literal-weight.



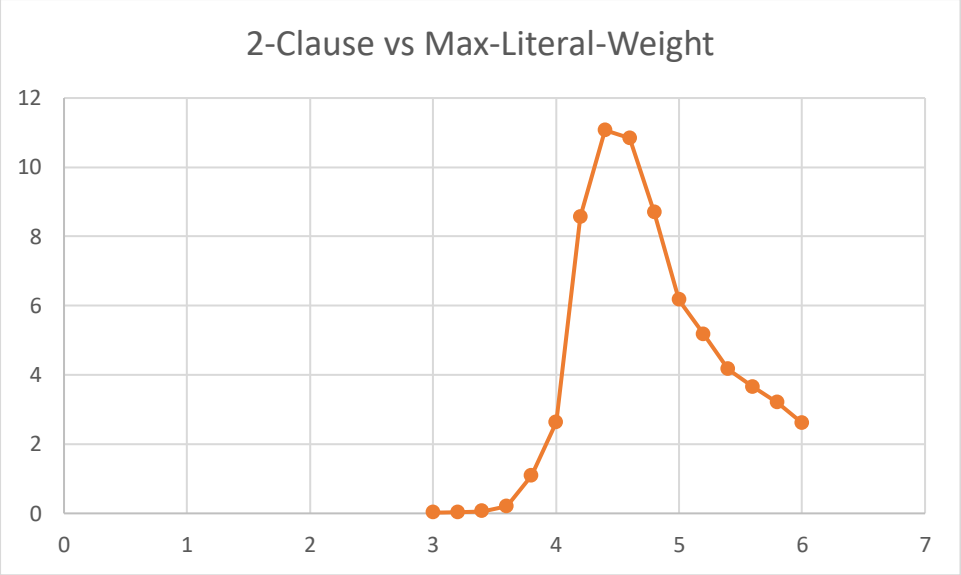
Plot of compute time of 2-Clause and Max-literal-weight as a function of L/N where N=100. The blue line depicts the performance of 2-clause and orange line shows the max-literal-weight. After certain ratio, the 2-clause heuristic timed out as the compute time was very high hence the graph shows null value, but in practice it is a larger value.



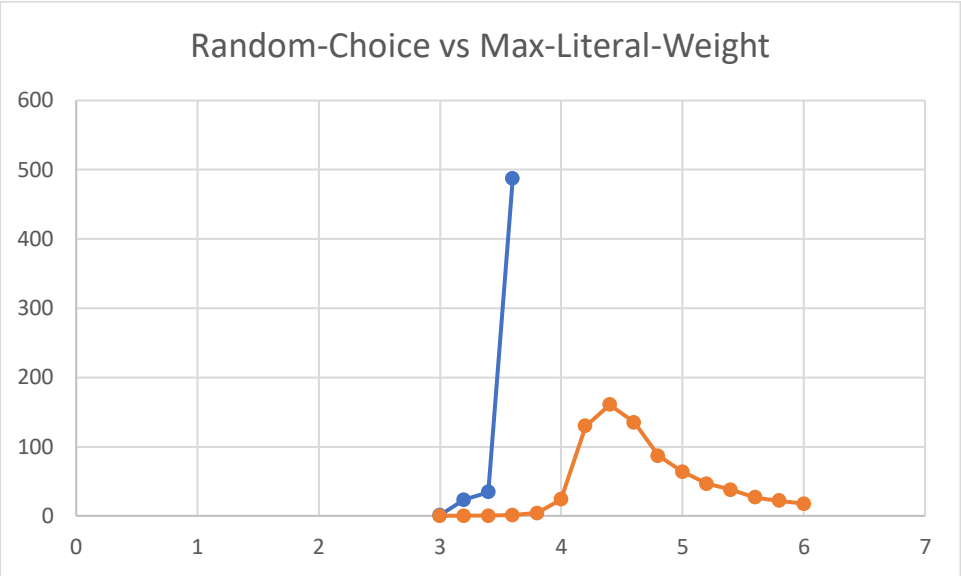
Plot of compute time of Random-choice and Max-literal-weight as a function of L/N where N=150. The blue line depicts the performance of random-choice and orange line shows the max-literal-weight. After certain ratio, the random-choice heuristic timed out as the compute time was very high hence the graph shows null value, but in practice it is a larger value.



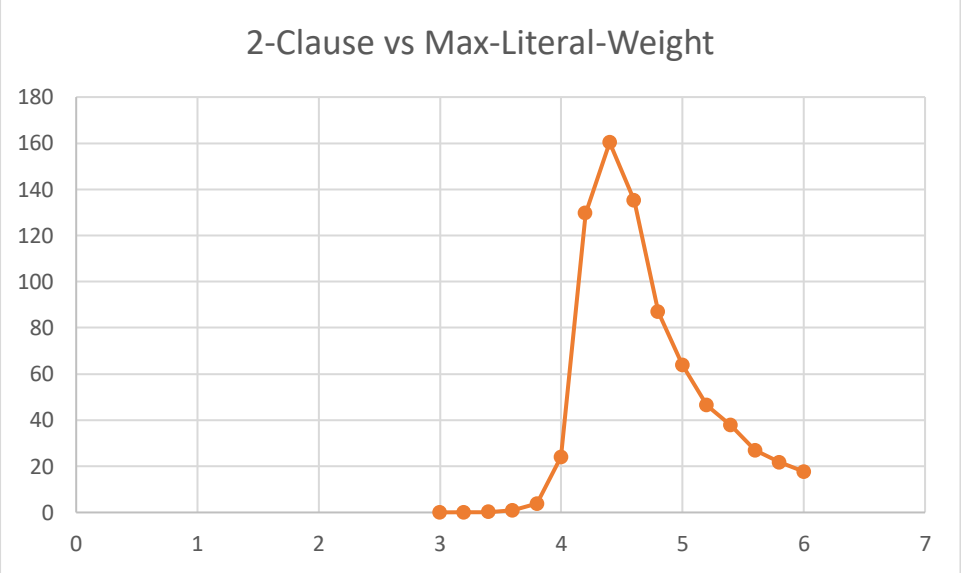
Plot of compute time of 2-Clause and Max-literal-weight as a function of L/N where N=150. The blue line depicts the performance of 2-Clause and orange line shows the max-literal-weight. After certain ratio, the random-choice heuristic timed out as the compute time was very high hence the graph shows null value, but in practice it is a larger value.



Plot of compute time of Random-choice and Max-literal-weight as a function of L/N where N=200. The blue line depicts the performance of random-choice and orange line shows the max-literal-weight. After certain ratio, the random-choice heuristic timed out as the compute time was very high hence the graph shows null value, but in practice it is a larger value.



Plot of compute time of 2-Clause and Max-literal-weight as a function of L/N where N=200. The blue line depicts the performance of 2-Clause and orange line shows the max-literal-weight. After certain ratio, the random-choice heuristic timed out as the compute time was very high hence the graph shows null value, but in practice it is a larger value.



In summary, the "Max-Literal-Weight" heuristic offers a potentially advantageous approach for solving SAT problems when compared to the random-choice and 2-clause heuristics. It operates by assigning weights to literals based on the lengths of the clauses they inhabit, favouring shorter clauses. This prioritization can be beneficial since shorter clauses typically hold more constraints and significantly impact satisfiability. Moreover, the heuristic concentrates on critical variables by elevating the weights of those in shorter clauses, facilitating more effective exploration of pivotal paths in the search space. However, its effectiveness depends on the specific characteristics of the SAT problem, and alternative heuristics may perform better in some scenarios. The choice of heuristic often involves trade-offs and should be tailored to the problem's unique attributes.

The implementation of the mentioned heuristics reveals a noteworthy trend: as the dataset size increases, computational time experiences an exponential rise, while the probability of satisfiability declines across all the heuristics employed. This observation underscores an important phenomenon in computational complexity. As the scale of the problem grows, the computational demands also escalate significantly. The decreasing probability of satisfiability with larger datasets suggests a transition from scenarios where most instances are satisfiable to ones where most are unsatisfiable. This phenomenon holds significance in the realm of understanding problem complexity, particularly for NP-hard problems like SAT.

Problem Complexity: I've gained insights into the complexity of solving NP-hard problems like SAT (Satisfiability). SAT is a computationally challenging problem, and its difficulty increases as the problem size grows. This complexity is a fundamental aspect of computer science and algorithmic analysis.

Heuristic Selection: I've explored different heuristics used in SAT-solving algorithms, including random-choice, 2-clause, and a custom heuristic that considers clause length. I've learned that the choice of heuristic can significantly impact the efficiency and effectiveness of the solver.

Trade-offs: I've recognized that there are trade-offs when selecting heuristics. While one heuristic may perform well on certain instances, it might not be suitable for others. This highlights the importance of adaptability and the need to tailor heuristics to specific problem characteristics.

Data Scaling: I've observed how computational time and satisfiability probability change as the dataset size increases. This scaling behavior is a crucial concept in computer science, as it reflects the challenges posed by larger datasets and the limitations of solving complex problems within reasonable timeframes.

Overall, this project has provided me with a deeper understanding of algorithmic problem-solving, heuristic selection, and the impact of problem size on computational performance. These skills are essential for tackling real-world computational challenges across various domains.