CS3105 Practical 2 Report

170011474

April 2020

Contents

1 Overview

This practical requires students to implement WalkSAT algorithm as well as conducting analysis and other alternatives would be preferred. `Dimacs` format input file is accepted and the algorithm can determine proper solution for the SAT formula.

2 Design and Implementation

2.1 WalkSAT Search

This WalkSAT implementation is based on Practical 2 Specification PDF and also from Fukunaga in 2014.

The WalkSAT selects a random unsatisfied clause and iterates over each literal of it. Afterwards, choosing the literal that brings the largest number of satisfied clauses after it is flipped. There is also possible that it can choose random walk rather than the above strategy to choose a literal, which depends on the possibility `p`.

An random assignment is allocated as a start point, stored in a `HashMap<Integer, Integer>` where the key is a literal (x or $\neg x$) and the value is 1(true) or 0(false). Each flip would modify the assignment map and clauses containing the corresponding literal would modify its assignment as well.

2.2 Novelty Search

Novelty search also has multiple versions of implementation, here I adopted the paper (Wen and Grochow, 2016) to find the most different-looking phenotype P , defined as $[s_1, s_2, s_3, ...]$ where s_i is the state (1 for true, 0 for false) for clause i. All used phenotype would be stored in `phenotypeLib` for calculating `novelty score` for the new phenotype. To choose a variable to flip, we calculate 2 merits to determine:

1. `novelty score` is product of hamming distance of P_{new} from each phenotype in the `phenotypeLib`. The hamming distance is defined as the number of positions where the chars differ in strings.

2. `fitness score` is the number of satisfied clauses.

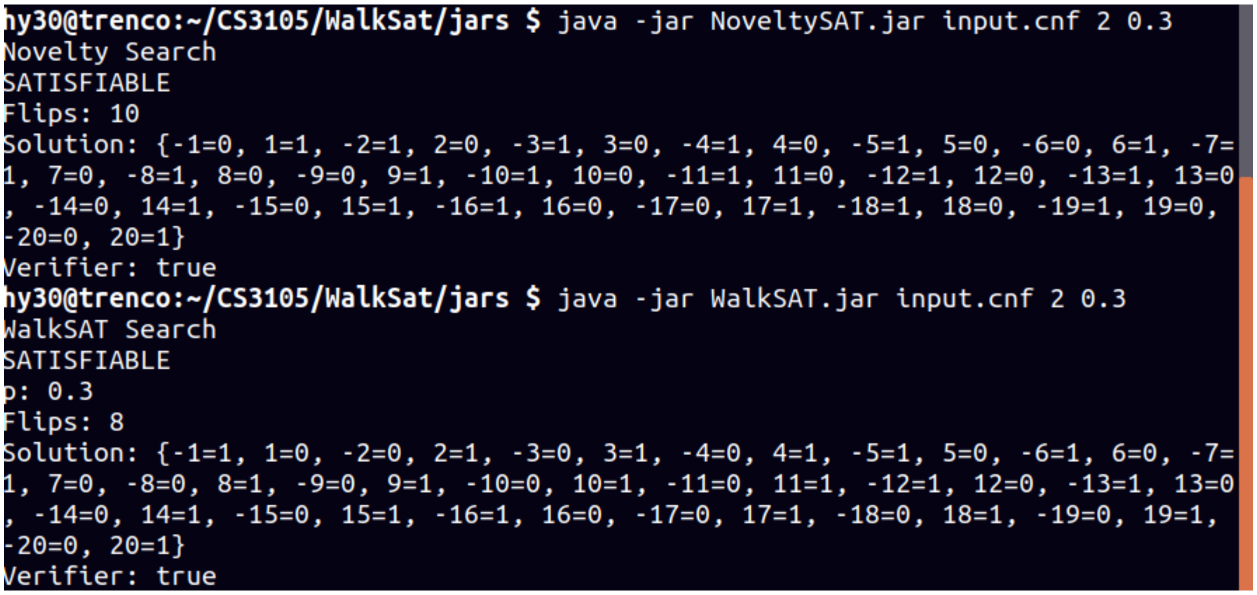
If there is a var leading to both maximum novelty and fitness, then that var is chosen. If only one merit is matched then we choose the var with the largest fitness score. Sometimes we may have multiple qualified vars and then we choose a random one from this range.

This version of Novelty Search requires more running time than others, since other versions record `age` of a pheonotype as `novelty score` without calculating hamming distance. **This is the most complicated implementation in Novelty Search. The reason to choose it is to investigate the "worse case" in Novelty Search.**

2.3 Correctness Testing and I/O Format

To verify the solution returned by the algorithm, a `verifier()` is used in `Helper.java`. This checker takes all clauses and iterate over each. If there is a literal is true according to the solution assignment map, then the current clause is satisfied. If all clauses are satisfied, then the solution is correct and `Verifier` turns `true`. 100 instances are tried including different sizes and all `Verifier` shown true.

Running WalkSAT and Novelty Search on lab machine, please see Figure 1:



```
hy30@trenco:~/CS3105/WalkSat/jars $ java -jar NoveltySAT.jar input.cnf 2 0.3
Novelty Search
SATISFIABLE
Flips: 10
Solution: {-1=0, 1=1, -2=1, 2=0, -3=1, 3=0, -4=1, 4=0, -5=1, 5=0, -6=0, 6=1, -7=
1, 7=0, -8=1, 8=0, -9=0, 9=1, -10=1, 10=0, -11=1, 11=0, -12=1, 12=0, -13=1, 13=0
, -14=0, 14=1, -15=0, 15=1, -16=1, 16=0, -17=0, 17=1, -18=1, 18=0, -19=1, 19=0,
-20=0, 20=1}
Verifier: true
hy30@trenco:~/CS3105/WalkSat/jars $ java -jar WalkSAT.jar input.cnf 2 0.3
WalkSAT Search
SATISFIABLE
p: 0.3
Flips: 8
Solution: {-1=1, 1=0, -2=0, 2=1, -3=0, 3=1, -4=0, 4=1, -5=1, 5=0, -6=1, 6=0, -7=
1, 7=0, -8=0, 8=1, -9=0, 9=1, -10=0, 10=1, -11=0, 11=1, -12=1, 12=0, -13=1, 13=0
, -14=0, 14=1, -15=0, 15=1, -16=1, 16=0, -17=0, 17=1, -18=0, 18=1, -19=0, 19=1,
-20=0, 20=1}
Verifier: true
```

Figure 1: Running WalkSAT on Lab Machine

Please use `java -jar [jar name] [input file] [max cpu time] [p]` to run the Jar. When it cannot find solution, then the output should be (**more in README**):

```
WalkSAT Search
Unable to find solution
```

When it can find a solution, then the output format should be:

```
WalkSAT Search
SATISFIABLE
p: 0.4
Flips: 38
Solution: {-1=0, 1=1, -2=1, 2=0, -3=1, 3=0, -4=0, 4=1, -5=1, 5=0, -6=0, 6=1,
    -7=1, 7=0, -8=1, 8=0, -9=1, 9=0, -10=0, 10=1, -11=1, 11=0, -12=1, 12=0,
    -13=0, 13=1, -14=0, 14=1, -15=0, 15=1, -16=1, 16=0, -17=0, 17=1, -18=1, 18=0,
    -19=1, 19=0, -20=0, 20=1}
Verifier: true
```

3 Analysis and Evaluation

3.1 Scalability

The dataset is from https://studres.cs.st-andrews.ac.uk/CS3105/Practicals/CS3105. Maximum problems it provided have 250 variables with 1065 clauses. **Due to Covid-19 and current situations, there is no other dataset can be found so only those problems are used to test.** Actually, they are supposed to solve more complicated instances with more variables.

	Largest Problem can solve	Smallest Problem can't solve
WalkSAT	250 vars, 1065 clauses	Not in this dataset
Novelty Search	250 vars, 400 clauses	125 vars, 400 clauses

3.2 Flip Frequency of WalkSAT

For general algorithm implemented, *when number of variables and number of clauses increase, the flip frequency (flip times per second) will decrease.* Please use `plot.py` to generate 3D graph. Please see figure 2.

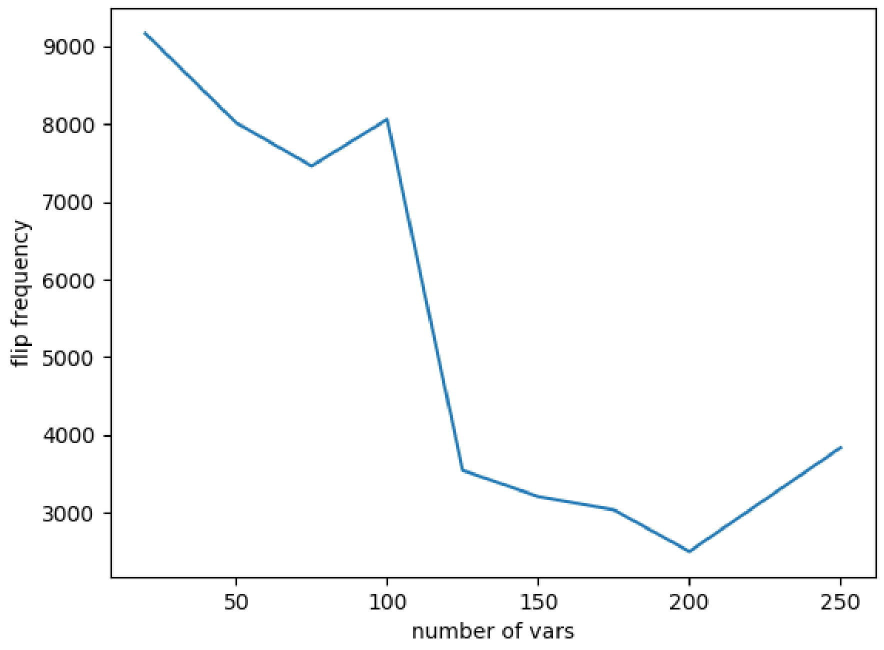
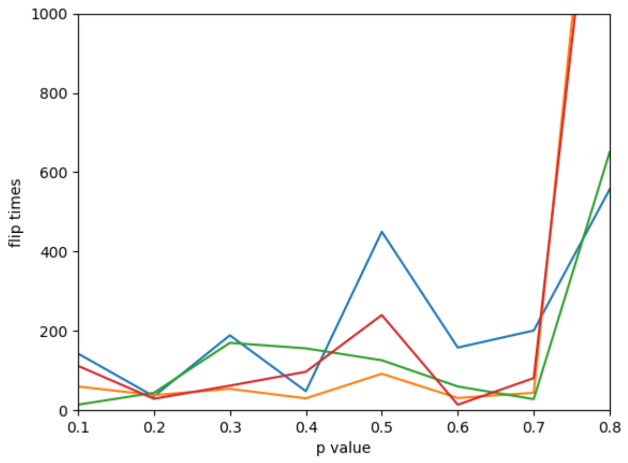


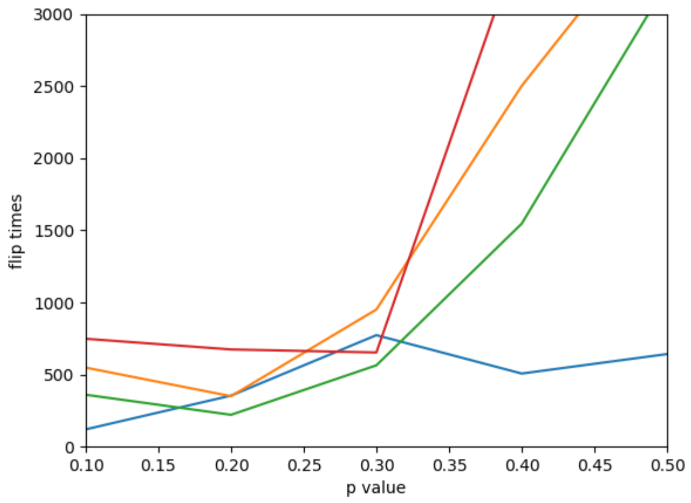
Figure 2: relation between flip frequency and problem size

3.3 P Value

The same random assignment map is used to control error, so only p value is the independent variable and each time algorithm starts to search from an identical assignment map with



(a) m = 20



(b) m = 50

Figure 3: p-value for WalkSAT and Novelty Search

different p values. Please refer to Figure 3.

3.3.1 WalkSAT Search

Select a range where p values vary: 0.1, 0.2, ..., 0.8. Then store the resulted flip times each solution requires into `p-eval.csv`. Use `plot.py` to plot and record the p values which leads to minimum flip times. It was found that when p is between `0.3` and `0.5`, the searching algorithm needs least flip times to find a solution.

3.3.2 Novelty Search

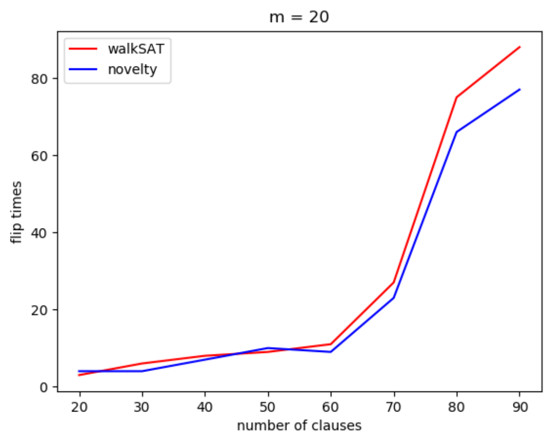
Similar to above, the best p value for Novelty Search in the current case is from 0.1 to 0.3 which can lead to less flip times. P values above 0.3 make flip times for a particular problem instance dramatically increase. To use a typical example, we choose a problem with 50 variables to test rather than 20. The reason why 50 is more typical than 20 is illustrated in Section 4.4.

3.4 WalkSAT vs. Novelty Search: Running Time and Search Steps

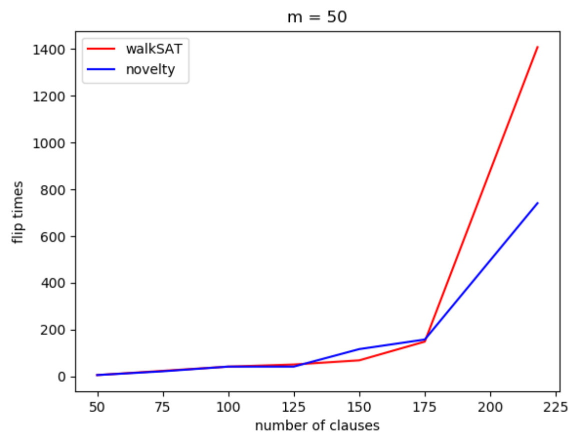
Evaluation is conducted in `Eval.java`. To compare different algorithms, one same case is used. It runs 20 times and for each iteration a random assignment is initialised for both WalkSAT and Novelty Search, so they start from the same point.

With the same instance to be solved, the running time of Novelty Search could be generally longer that running time of WalkSAT, since Novelty Search requires Novelty and fitness calculation. While calculating the novelty score, obtaining phenotype is troublesome.

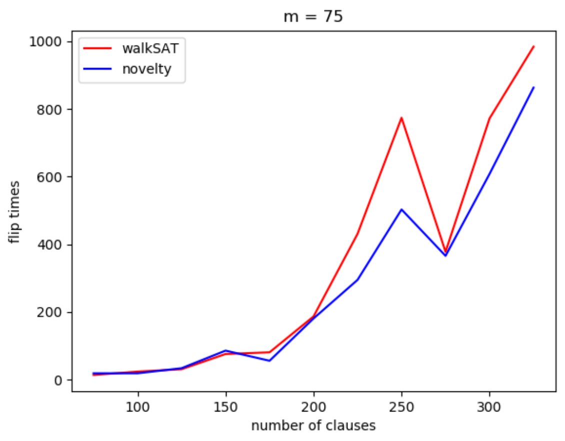
However, with the same solvable case, the average flips required by Novelty Search is less than that by WalkSearch. The reason is that Novelty Algorithm can explore more search space. The following figures show that the novelty search requires less search steps (flip times) than WalkSat does, particularly when the problem size goes up:



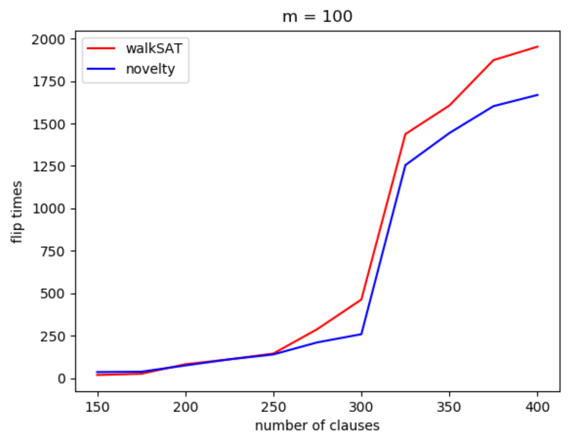
(a) m = 20



(b) m = 50



(c) m = 75

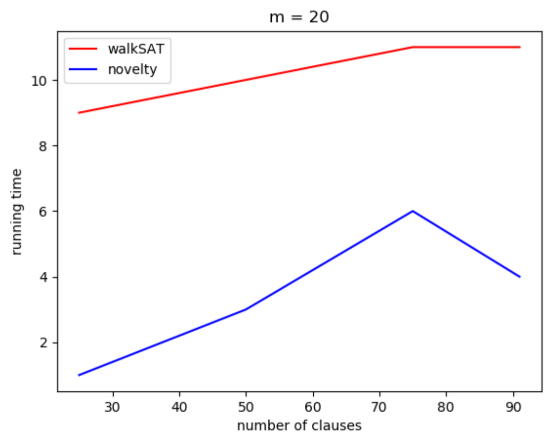


(d) m = 100

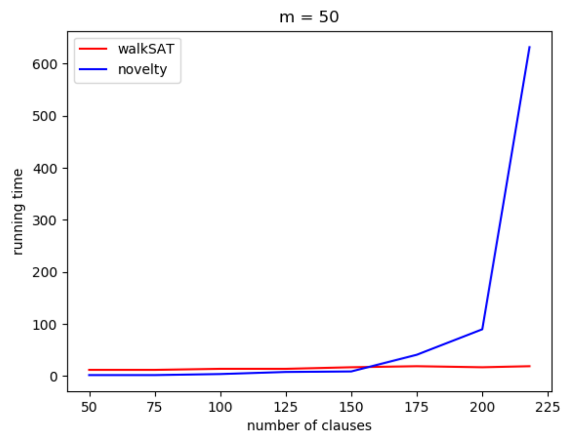
Figure 4: m is the number of variables in a problem

From above, we can see Novelty Search theoretically improve the capability of solving SAT problems as long as better CPU is provided. The capability and scalability for those search algorithms are listed in the table in Section 4.1 Scalability.

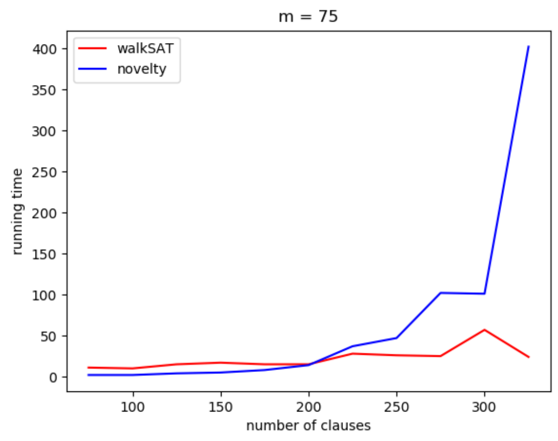
The running time for each algorithm is also represented in Figure 5. We can see that when the size of problems is small, the Novelty Search takes less time to find a solution. However, when the size rises, the Novelty Search running time dramatically increases. The reason is that it has to look at the phenotype and calculate hamming distance for each step. When the size of the problem goes up, the impact shows. So a typical example used to represent Novelty Search is a problem with more than 20 variables, since at that time the novelty search can fulfill its advantages.



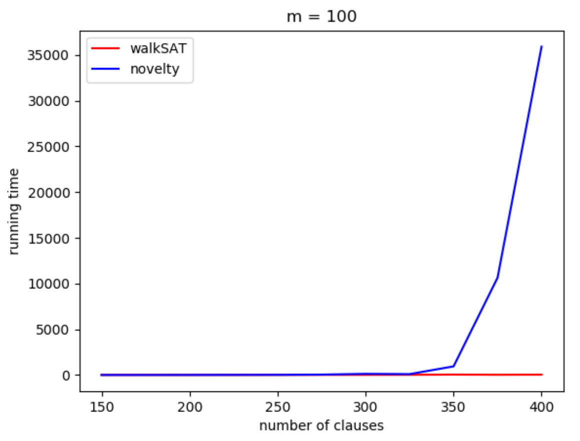
(a) m = 20



(b) m = 50



(c) m = 75



(d) m = 100

Figure 5: m is the number of variables in a problem. The unit of time is milliseconds.

Additionally, since the current version of Novelty Search is more complicated than others, we can conclude that for most other versions, Novelty Search can outperform WalkSAT Search dramatically, with less running time as well as less search steps (flip times) required.

3.5 Comparison with Other Literature

From Wen and Grochow in 2016, the paper shows the *there is no significant difference in scalability of the 2 search algorithms. Besides, Novelty Search requires less search steps.* This conclusion is similar to author's conclusion. However, they directly linked this success to running time and made a conclusion that Novelty Search also takes less time to complete its searching, which is not convincing. They failed to provide concrete implementation in their paper, except some basic concepts of Novelty Search.

For the best possible p values in WalkSAT, in 2004, Kautz and Selman states that *most random 3-SAT problems can be solved in* $p = 0.5$ *most quickly. If the instance is structured, then a lower p value is preferred.* In this case, the p value analysis in current report suits this conclusion, with an optimal range $[0.3, 0.5]$ presented. Besides, it also presents that the optimal p value is close to the minimal μ/σ, where μ is the average number of unsatisfied clauses in the running time, and σ is the standard deviation of μ.

4 Problems Encountered

The problems encountered are how to improve running time in those algorithm implementation. Limited paper resources about Novelty Search in SAT solving and unstable VPN in China also bring troubles.

5 Check Table

Please see Figure 6

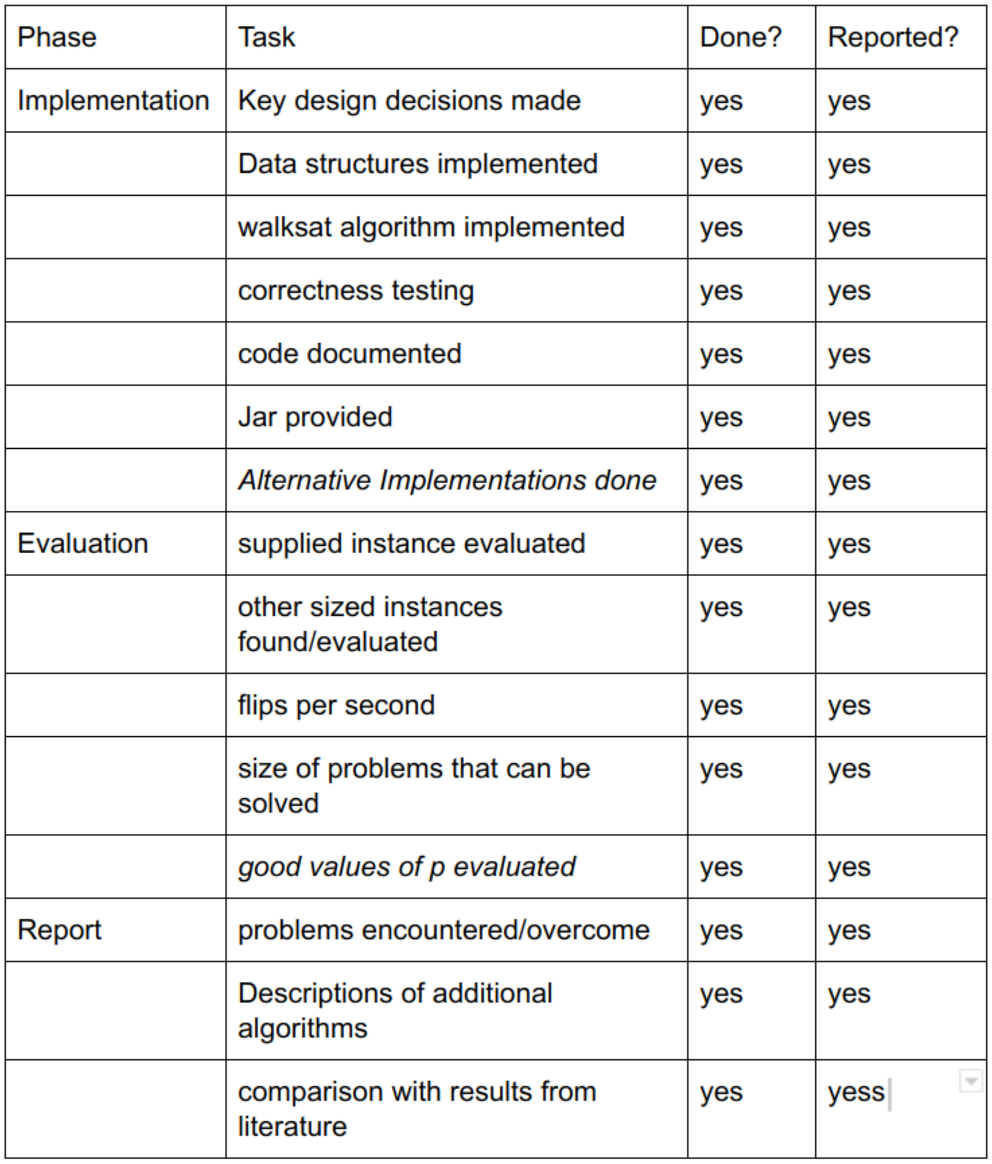
References

[1] Fukunaga, A., 2002. *Automated Discovery Of Composite SAT Variable-Selection Heuristics.* [online] Aaai.org. Available at: https://www.aaai.org/Papers/AAAI/2002/AAAI02-096.pdf [Accessed 28 April 2020].

[2] Fukunaga, A., 2004. *Efficient Implementations Of SAT Local Search.* [online] Satisfiability.org. Available at: http://www.satisfiability.org/SAT04/programme/106.pdf [Accessed 2 May 2020].

[3] Kautz, H. and Selman, B., 2004. Walksat In The 2004 SAT Competition. [online] Available at: https://www.researchgate.net/profile/Bart_Selman/publication/250012712 [Accessed 28 April 2020].

[4] Wen, R. and Grochow, J., 2016. *Novelty Search For SAT*. [online] Roujiawen.com. Available at: https://www.roujiawen.com/portfolio/pdfs/novelty_paper.pdf [Accessed 11 April 2020].



Phase	Task	Done?	Reported?
Implementation	Key design decisions made	yes	yes
	Data structures implemented	yes	yes
	walksat algorithm implemented	yes	yes
	correctness testing	yes	yes
	code documented	yes	yes
	Jar provided	yes	yes
	Alternative Implementations done	yes	yes
Evaluation	supplied instance evaluated	yes	yes
	other sized instances found/evaluated	yes	yes
	flips per second	yes	yes
	size of problems that can be solved	yes	yes
	good values of p evaluated	yes	yes
Report	problems encountered/overcome	yes	yes
	Descriptions of additional algorithms	yes	yes
	comparison with results from literature	yes	yess

Figure 6: Check Table