Metaheuristic Optimization – Assignment 2


```
Author: Melwyn D Souza
Student Number: R00209495
Email: melwyn.dsouza@mycit.ie
Course: MSc Artificial Intelligence
Module: Metaheuristic Optimization
Assignment: 2
Date: 02/01/2022
```


Abstract

For this assignment, three CNF instance files are selected from uf20-91 set (click here), the set contains 1000 3SAT instances, 20 variables, 91 clauses in each instance file and all are satisfiable, the goal of this project is to run local search algorithms namely GSAT, GWSAT, GWALKSAT and GSAT+TABU LIST, find the solutions which satisfy the formulae F, tune the parameters to find the best configuration and finally evaluate and analyse the results with the help of Run Time Distribution plots

1 Introduction

A Boolean propositional logic formula F is formed using variables, logical operators (AND, OR, NOT) and parenthesis. In computer science, a SAT problem is problem of finding if there is any interpretation which will satisfy the formula or in other words, the formula F returns TRUE or 1 for variable assignments. This variable assignment which will satisfy SAT formula F will hence be the solution

F = (V, C) where V indicates variables and C indicates the CNF clauses

There are different ways to find the solution for SAT problems, in this project we are focusing of 3-SAT instances, the local search algorithms are used to find the solutions or to satisfy the SAT instances. Local search is an optimization technique to solve computationally hard problems in various areas of AI. There are different types of local search algorithms such as Hill Climbing, Simulated Annealing, 2OPT etc, for SAT problems we are mainly focusing of four algorithms as mentioned earlier

2 Research

2.1 Local Search Algorithms for SAT

The local search is used to solve computationally hard search problems, the idea is to move around in the search space created by the problem instance in order to find a solution, steps include initializing search at some point in the search space, checking if the current point is the solution, if not, move from the current search position to a neighbouring position, the decision is based on the local neighbourhood information only, hence the name local search since at every step, the algorithm only checks its local neighbouring search space and not global [1] the figure 1 below shows the general local search methodology for SAT problems

```
procedure StochasticLocalSearch for SAT
    input CNF formula Φ, maxTries, maxSteps
    output satisfying assignment of Φ or "no solution found"
    for i := 1 to maxTries do
        s := initAssign(Φ);
        for j := 1 to maxSteps do
            if s satisfies Φ then return s;
            else
                x := chooseVariable(Φ, s);
                s := s with truth value of x flipped;
            end if
        end for
    end for
    return "no solution found";
end StochasticLocalSearch for SAT
```

Figure 1: Local search procedure for SAT problems [1]

If S is the solution for F at T and S1 is the solution at T1,

Net gain = UNSAT Clauses because of S at T - UNSAT Clauses because of S1 at T1

Negative Gain = Clauses UNSAT at T1 which were not UNSAT at T

Positive Gain = Clauses SAT at T1 which were not SAT at T

2.2 GSAT

A greedy local search-based method, hence the name Greedy-SAT or in short 'GSAT'. It starts with assigning values to variables randomly and creating an initial solution. This initial solution is checked if it satisfies all clauses in the formula, else each variable is flipped (0 to1 and vice versa) and the variable with the highest net gain is then selected as the best variable. This new solution with highest net gain if satisfies the formula will be our interpretation and the search is complete, else the latter step is repeats for given number of iterations (number of flips) until a solution is found. If the algorithm reaches maximum flips, the search is terminated without a solution [2].

```
Input: a set of clauses α, MAX-FLIPS, and MAX-TRIES
Output: a satisfying truth assignment of α, if found
begin
   for i := 1 to MAX-TRIES
       T := a randomly generated truth assignment
       for j := 1 to MAX-FLIPS
           if T satisfies α then return T
           p := a propositional variable such that a change
                in its truth assignment gives the largest
                increase in the total number of clauses
                of α that are satisfied by T
           T := T with the truth assignment of p reversed
       end for
   end for
   return "no satisfying assignment found"
end
```

Figure 2: GSAT general procedure [2]

Figure 2 above shows the methodology of GSAT, as mentioned above all the variables are flipped one after another and the net gain is calculated, the variable which results in the highest net gain in the neighboring search space is selected as the best variable. This step is repeated until a solution for the

problem is found or if the maximum iterations are exhausted, if there are several variables with the same net gain then one variable is randomly chosen as the best variable

This is not the most efficient local search technique since at each iteration all the variables are flipped, this increases the total steps of the local search algorithm and becomes computationally expensive when there is large number of variables

GSAT can get stuck in the local minima or local maxima depending on the type of problem. To avoid this problem we use random restarts, this removes the position in search space and reinitiate it at a different point in the search space, which could escape the local maxima or local minima

2.3 GWSAT

In this method, along with the greedy local search, another local search step called random walk is introduced. Firstly, one of the unsatisfied clauses are selected at random, then one of the variables in the selected UNSAT clause is selected as the best variable to flip. The idea of GWSAT is to decide whether to perform GSAT or random walk step at each iteration depending on the walk probability 'wp' [3].

2.4 GSAT+TABU

This technique helps GSAT not to get stuck in the local optima, the idea here is to forbid the reversing the move for several iterations which also is the length of a tabu list. So, if the tabu list can hold 3 variables, the latest variable which is flipped cannot be flipped for at least 3 iterations. The steps include storing of 'tl' number of flipped variables in the tabu list sequentially, the tabu list is more like a FIFO queue of length 'tl', when selecting the best variable, check if its not in the tabu list, if it is, randomly choose the next best variable which doesn't exist in the tabu list. The issues with tabu lists is, if the list is too small, the search can still get stuck in the local optima and need restarts, if the list length is too long, the search space might not have enough best variables to flip which will make it difficult to reach a solution [4].

2.5 GWALKSAT

The similarity between GWSAT and GWALKSAT is uncanny but there are major differences between two methods. GSAT has a random walk step with probability wp which is also present in GWALKSAT, but the difference here is that in GWALKSAT, first a random unsat clause is selected, the variables are then flipped and if there exist a variable with 0 negative gain, it is directly selected as the best variable, if more than one variables have 0 negative gain, one of them is randomly selected, if no variables have 0 negative gain, then there is a random walk step with wp probability and GSAT. The latter steps are same as GWSAT. The initial selection of the best variable with 0 negative gain gives an upper hand [1].

3 Methodology

- Randomly select 3 out of 1000 instances from uf20-91 set
- Read the instance file and extract information such as number or variables and clauses
- Store the variables and clauses in memory
- Randomly assign values (0s or 1s) to the variables, store the SAT and UNSAT clauses and memory
- If the length of UNSAT list is 0, it means there are no UNSAT clauses, hence solution is found
- If not, depending on the local search scheme, select the best variable to flip
- If the solution gives same gain for 'n' iterations, then restart at a different point in search space (reinitialize the variable values randomly)
- Else, repeat the above steps until maximum flips are exhausted

4 Evaluation and Analysis

The three files are randomly selected from 1000 instance files using python library 'random' with random.seed(209495), and the 3 files used for all experiments are

1. uf20-0877.cnf
2. uf20-0498.cnf
3. uf20-0471.cnf

As discussed in the research part, I have used 4 local search schemes for this assignment and results are listed in the tables below for each instance file. The columns in the tables indicate the steps taken to find the solution by each algorithm, the bottom row is the average steps for 10 runs, the efficiency of the model depends on the steps taken by the model to find solution that satisfies the SAT instance

The default parameters used for this experiment are,

1. algorithms = gsat, gwsat,gtabu,gwalksat
2. nRuns = 10
3. nBaditers = 20
4. nRestarts = 50
5. wp = 0.1
6. tl = 5

RUN	GSAT	GWSAT	GTABU	GWALKSAT
1	1680	72	200	510
2	940	597	2360	555
3	2500	540	2860	816
4	800	144	260	564
5	1580	2358	2320	132
6	320	72	540	1185
7	220	1194	220	282
8	20000	75	340	1104
9	20000	441	740	393
10	980	1113	1060	588
Average steps	**4902**	**660.6**	**1090**	**612.9**

Table 1: File inst/uf20-0877.cnf default parameter results

As can be seen from table 1, I have conducted gsat, gwsat, gwalksat and gtabu schemes on "uf20-0877.cnf", GSAT is not an efficient local search method as we have discussed earlier, this is because of the number of steps it must undergo in each iteration, it can be seen from the results above, the average steps for GSAT are way high compared to other schemes.

The 8th and 9th step are marked red since the GSAT could not find solution to the instance even after exhausting all the iterations, this is a major concern when using GSAT. To tackle the issue of getting stuck in local optima (which leads to not finding the global optimal solution, maxIters = 1000, variables = 20, total steps = maxIters*variables for GSAT) we use other schemes.

We can see that GWSAT, GWALKSAT and GTABU have found solutions for every run and from the evaluation and careful analysis of the results, I conclude that GWALKSAT is performing the best out of all 4 schemes, followed by GWSAT.

GWALKSAT and GWSAT are very similar in action, and since GWALKSAT has an advantage of selecting variables with 0 negative gain before switching between random walk and GSAT, it is ought to perform better than all other schemes and the same is proven from the above results.

GTABU on the other hand maintains a list of latest variables flipped, this gives it an advantage over GSAT but it still has to flip all variables at every iteration, it is performing better than GSAT since the TABU list won't let the search point to be stuck in local optima

From table 1, we could say that GWALKSAT < GWSAT < GTABU < GSAT when average steps are taken into consideration

RUN	GSAT	GWSAT	GTABU	GWALKSAT
1	400	60	240	36
2	540	60	300	69
3	220	45	260	63
4	340	117	860	21
5	1780	105	420	93
6	400	366	320	90
7	200	48	1040	36
8	1040	66	640	258
9	120	42	120	18
10	140	21	140	57
Average steps	**518**	**93**	**434**	**74.1**

Table 2: File inst/uf20-0498.cnf default parameter results

Table 2 indicates the results when default parameter local search schemes are applied on uf20-0498.cnf instance file, compared to the previous instance, it takes less steps to find solution which satisfied the current instance file

Comparing all local search scheme results, my conclusion would be similar to the previous conclusion

From the average steps, we can agree that GWALKSAT < GWSAT < GTABU < GSAT, which means GWALKSAT is the best performing model and GSAT is the worst model

RUN	GSAT	GWSAT	GTABU	GWALKSAT
1	2180	63	1620	39
2	1440	21	180	843
3	2040	195	420	957
4	2120	123	440	93
5	400	24	200	42
6	1060	549	1700	345
7	240	75	520	21
8	720	60	180	246
9	280	333	280	495
10	500	297	680	237
Average steps	**1098**	**174**	**622**	**322.1**

Table 3: File inst/uf20-0471.cnf default parameter results

The results (steps for each run and average steps(10 runs) is from the SAT instance file uf20-0471.cnf, the same behaviour as other two files is noticeable from table 3, surprisingly GWSAT is performing better compared to GWSAT

In conclusion, GSAT is very inefficient on its own, when techniques like TABU list, random walk with walk probability are applied along with GSAT, it shows great improvements

5 Parameter Tuning

I have conducted a few experiments to tune the parameters to give out best configuration for each local search scheme, the goal is to reduce the total number of steps involved in finding the solution. The tables below show the results of different tests run on 3 instance files, the rows indicating the instance files and the columns indicate the average of 10 runs on each configuration.

5.1 nBadIters

Table 4 (row 1, column 1) which is marked yellow indicates the average of 10 runs over file uf20-0877.cnf using GSAT with nBaditers = 0, likewise in the same row, column 2 indicates average of 10 runs with nBaditers = 5

Configuration:

1. algorithms = gsat
2. nRuns = 10
3. nBadIters = 0, 5, 10, 15, 20
4. nRestarts = 50
5. wp = 0.1
6. tl = 5

nBadIters	0	5	10	15	20
uf20-0877.cnf	4234	9338	10310	5482	4898
uf20-0498.cnf	2708	600	424	612	518
uf20-0471.cnf	1958	830	856	768	1098

Table 4: Default parameters except nBaditers (GSAT)

As we can see when we use 0 for bad iterations, the GSAT model is performing poorly on all files, whereas when we increase it further, the average steps have reduced. From this experiment, I have concluded that nBadIters in range (10-20) is a good value for improving performance of GSAT, selecting default value of nBaditers as 20 for RTD

5.2 nRestarts

Every cell in the table below indicates the average of 10 runs over given parameters, I have tested my assignment 2 with restarts ranging from 0-5, I'm aware of default value = 50 but since I have noticed on average restarts have a value of around 5

We can notice that for nRestarts = 0, the average steps are very low, this is because the GSAT gets stuck in local optima and never finds a solution, the average is calculated on only good solutions, hence the small values are not the best values, anyways with increase in restart numbers, most nRuns find a solution compared to nRestarts = 0, which indicates that GSAT will get a fresh restart when it gets stuck at the same net gain/local optima, since the total number of good solutions increase with increase in nRestart values, I have selected the default value of 50 for further experiments and RTD

Configuration:

1. algorithms = gsat
2. nRuns = 10
3. nBadIters = 20
4. nRestarts = 0, 1,2,3,4
5. wp = 0.1
6. tl = 5

nRestarts	0	1	2	3	4
uf20-0877.cnf	2264	2264	4414	6478	6594
uf20-0498.cnf	280	280	424	424	567
uf20-0471.cnf	348	348	520	736	836

Table 5: Default parameters except nRestarts

5.3 TABU Length

Table 6 (row 1, column 1) indicates the average of 10 runs over file uf20-0877.cnf using GTABUSAT with tl = 1, likewise in the same row, column 2 indicates average of 10 runs with tl = 3

Configuration:

1. algorithms = gtabu
2. nRuns = 10
3. nBadIters = 20
4. nRestarts = 50
5. wp = 0.1
6. tl = 1,3,5,7,9

As can be seen from the table below, taking the average of all 3 files, tabu length of 5 is helping the local search scheme perform well, hence the default value of tl=5 will be used for RTD

Very low tabu lengths will still have higher chances of getting stuck in a local optimum since the same variables will flip infinite times, also high values of tabu lengths can degrade the performance of the scheme since it won't let the local search to select the best variable at each iteration since most of the best and good variables might be still in tabu list, this can be noticed in the table below, for small 'tl' values, average steps are high, likewise for large values of 'tl' the average steps are high. So, selecting a proper length is very important

Tabu length	1	3	5	7	9
uf20-0877.cnf	1298	2446	1090	2440	3802
uf20-0498.cnf	408	338	434	842	1446
uf20-0471.cnf	1194	700	622	602	872

Table 6: Default parameters except tabu length

5.4 Random walk probability

This step is used only by GWSAT and GWALKSAT since they have a random walk step, it checks for the random walk probability 'wp' to decide between random walk step and other heuristics

Table 7 and table 8 are the results of varying random walk probability for GWSAT and GWALKSAT, we can notice the wp value of 0.1 has a great impact on the performance of both schemes

We have to be careful while selecting a value for wp, because if it is very high, our search point which could have been the best in the search space will be lost since the next seach point will be initiated randomly, this will lead to the loss of best variables and in turn reduces the performance of the models

It can be noted that the average steps decrease with increasing wp, this might seem like performance enhancement but in reality most of the nRuns end up with no solutions, hence the average steps are very low compared to other columns

Configuration:

1. algorithms = gwsat, gwalksat
2. nRuns = 10
3. nBadIters = 20
4. nRestarts = 50
5. wp = 0.0, 0.1, 0.2, 0.3, 0.4
6. tl = 5

wp	0.0	0.1	0.2	0.3	0.4
uf20-0877.cnf	368	74	118	110	62
uf20-0498.cnf	157	174	356	249	285
uf20-0471.cnf	490	331	380	489	480

Table 7: Default parameters except random walk probability (GWALKSAT)

GWSAT (varying noise wp) (0.0, 0.5,0.1)

wp	0.0	0.1	0.2	0.3	0.4
uf20-0877.cnf	1146	660	987	540	361
uf20-0498.cnf	2256	612	1180	905	1160
uf20-0471.cnf	99	98	72	76	78

Table 8: Default parameters except random walk probability (GWSAT)

6 Run Time Distribution

I have used inst/uf20-0498.cnf instance file for this part of the project, local search schemes GSAT and GWSAT are compared. Each scheme is run 100 times on the instance file to produce 100 solutions, few runs will take more time depending on the number of steps. Since run time alone cannot be used for comparison, we use number of steps taken to reach a solution point. The number of steps taken by GSAT will be very high since it checks the gains of each variable to select the best on every iteration, where as the GWSAT scheme checks gains of variables in one randomly selected clause, so the steps are dependent on the number of variables in the caluse, If it's a 3 SAT instance, we'll deal with 3 variables for GWSAT.

The runtime is measured using pythons time library, timer starts at the start of the variable search and ends when the algorithm finds best variable, the total time is the time taken for every iteration until a solution is found

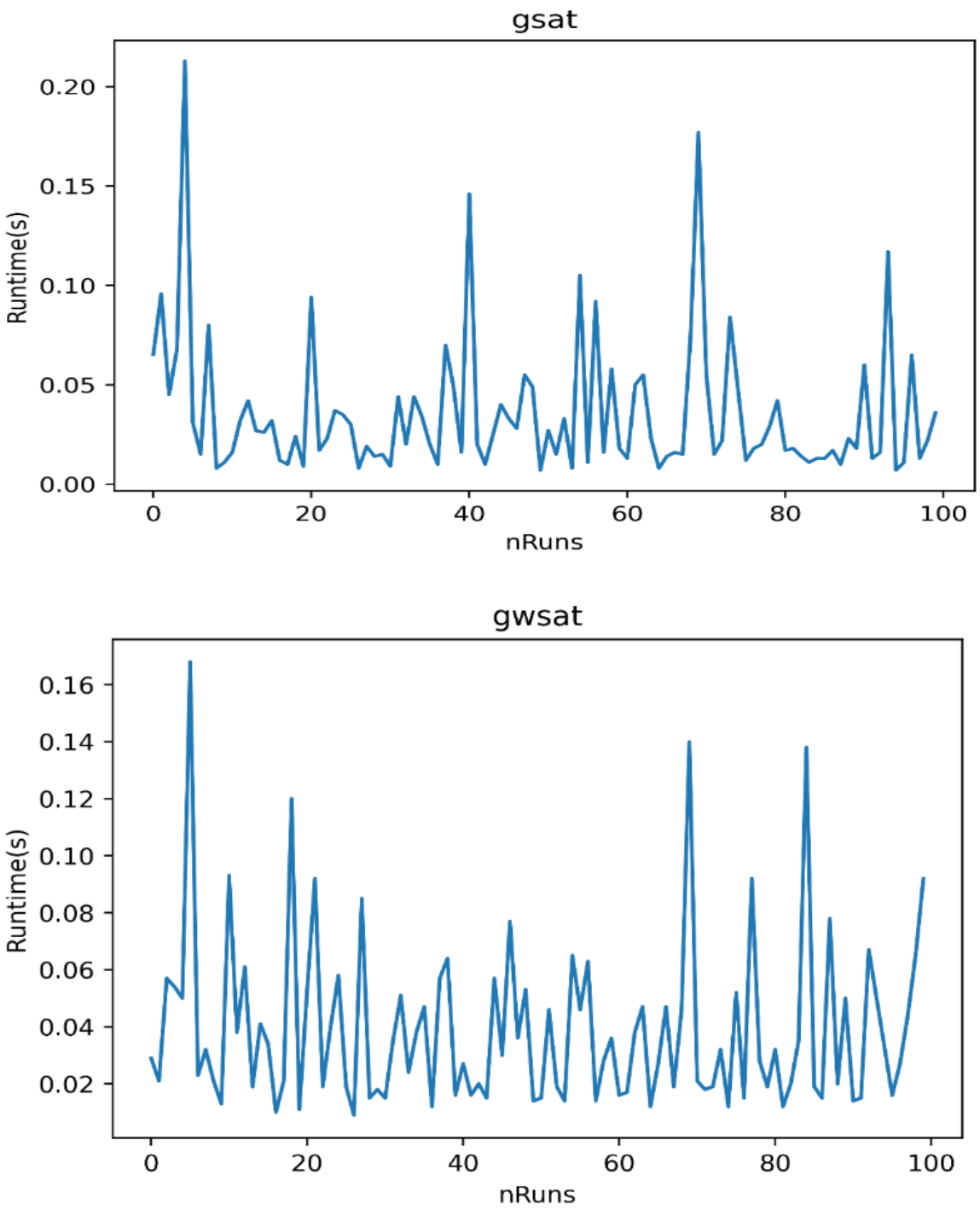

Figure 3: Runtime in seconds vs number of runs for GSAT (top) and GWSAT (bottom)

The figure 3 indicates the run time for each iteration, we can notice that the maximum run time for GSAT is around 0.20 and for GWSAT is around 0.16, GWSAT is performing well compared to GSAT as it deals with less steps for each iteration

Few iterations have very low runtime in GSAT, this is because it selects the best variable out of all variables and the algorithm will get lucky sometimes, whereas GWSAT only selects variable with 0 negative gain, else it does random walk or GSAT!

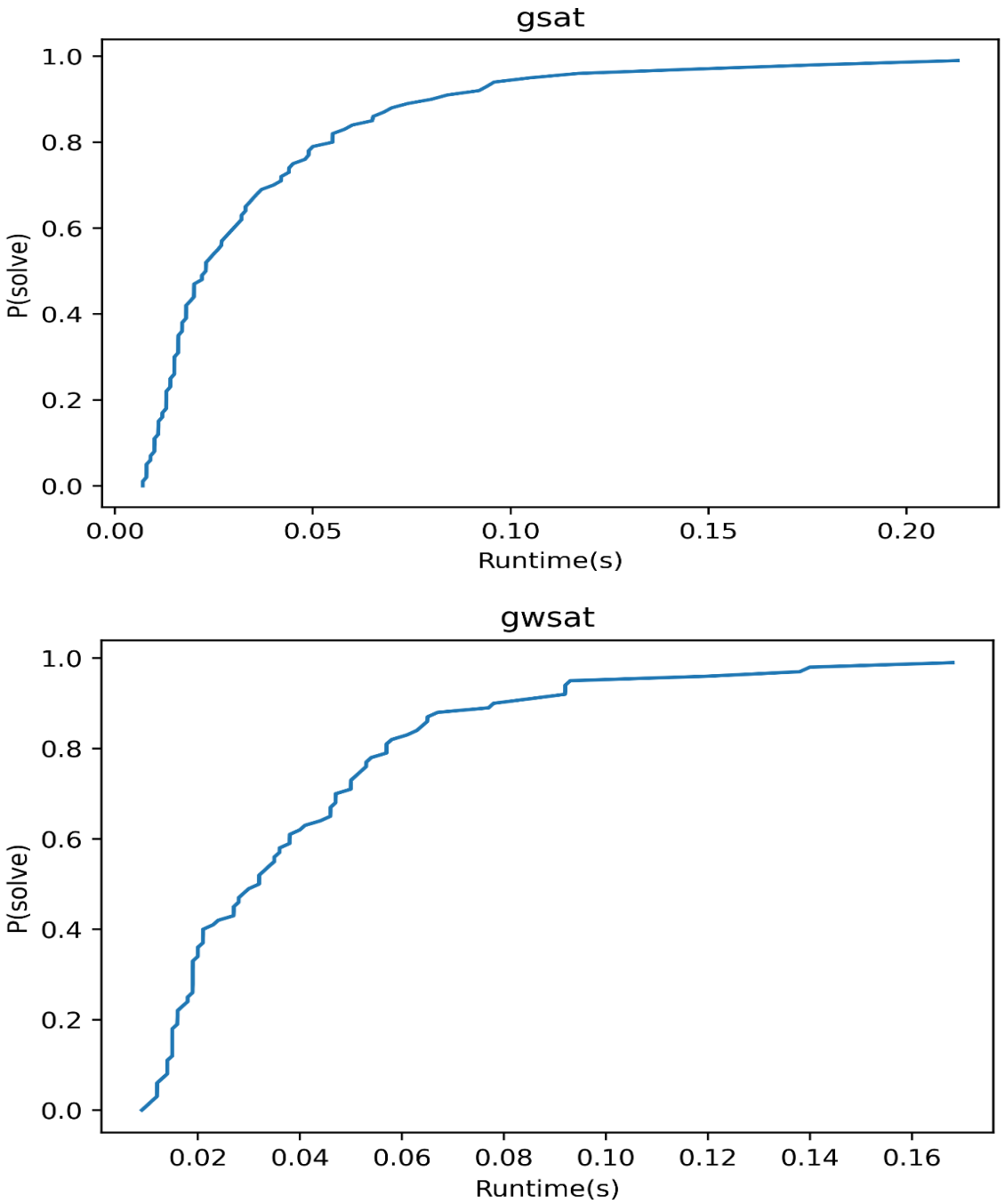

Figure 4: Runtime in seconds vs Probability of solving p(solve) for GSAT (top) and GWSAT (bottom)

The plots above indicate the probability of finding a solution in the given runtime, for example, at the runtime 0.06 seconds, GSAT will have around 0.7 P(solve) which means the probability of finding solutions within 0.06 seconds is 70% for GSAT, whereas for GWSAT it is around 80%

Probability is 100% or 1 at 0.15-0.16 seconds for GWSAT, and for GSAT its 0.17-0.20 seconds

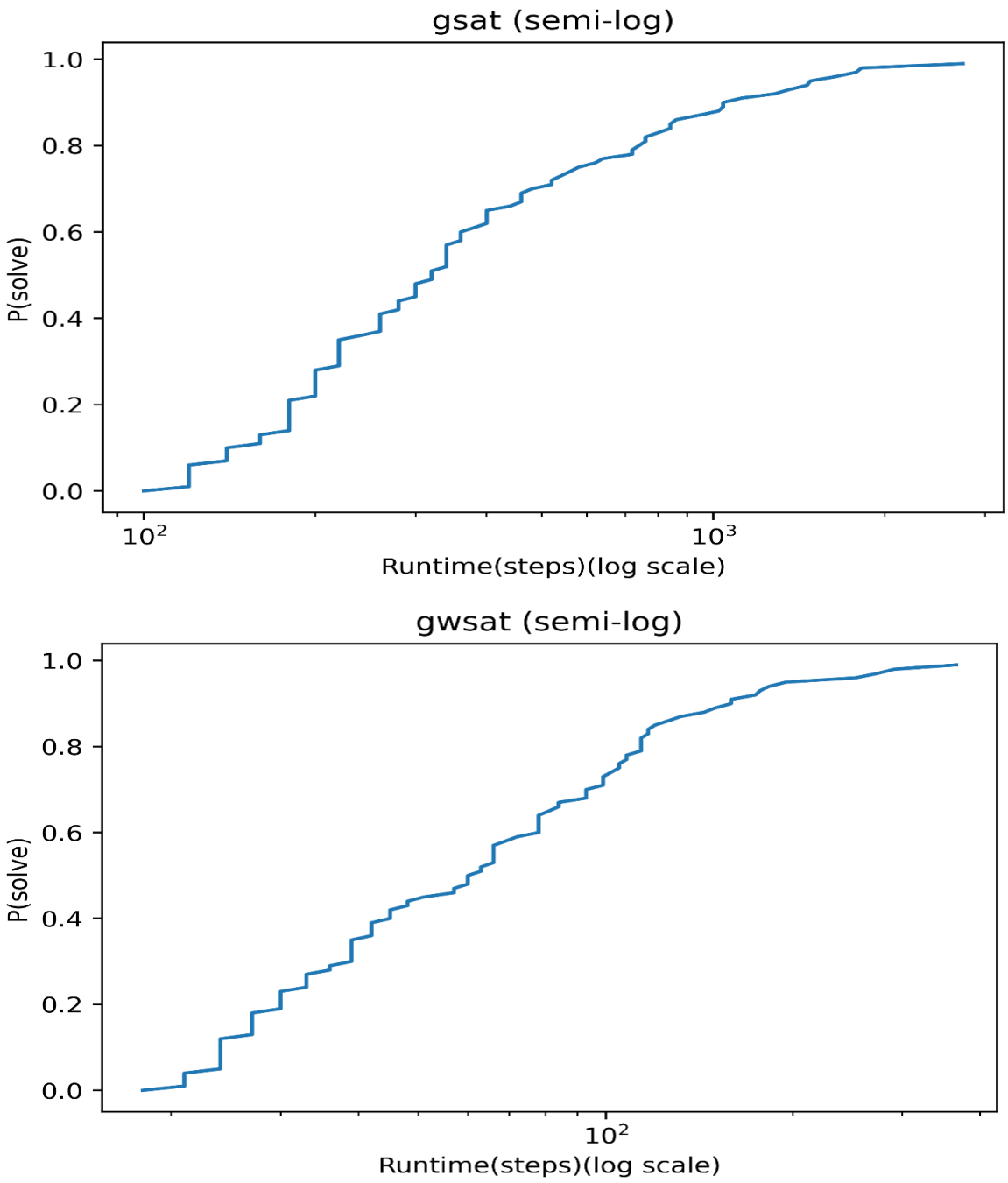

Figure 5: Semi-log RTD of GSAT and GWSAT

From the semi log plots above, we can notice that the total number of steps taken by GSAT and GWSAT has a huge difference for P(solve) = 1, this is the main advantage of GWSAT over GSAT that its computationally inexpensive when compared to GSAT

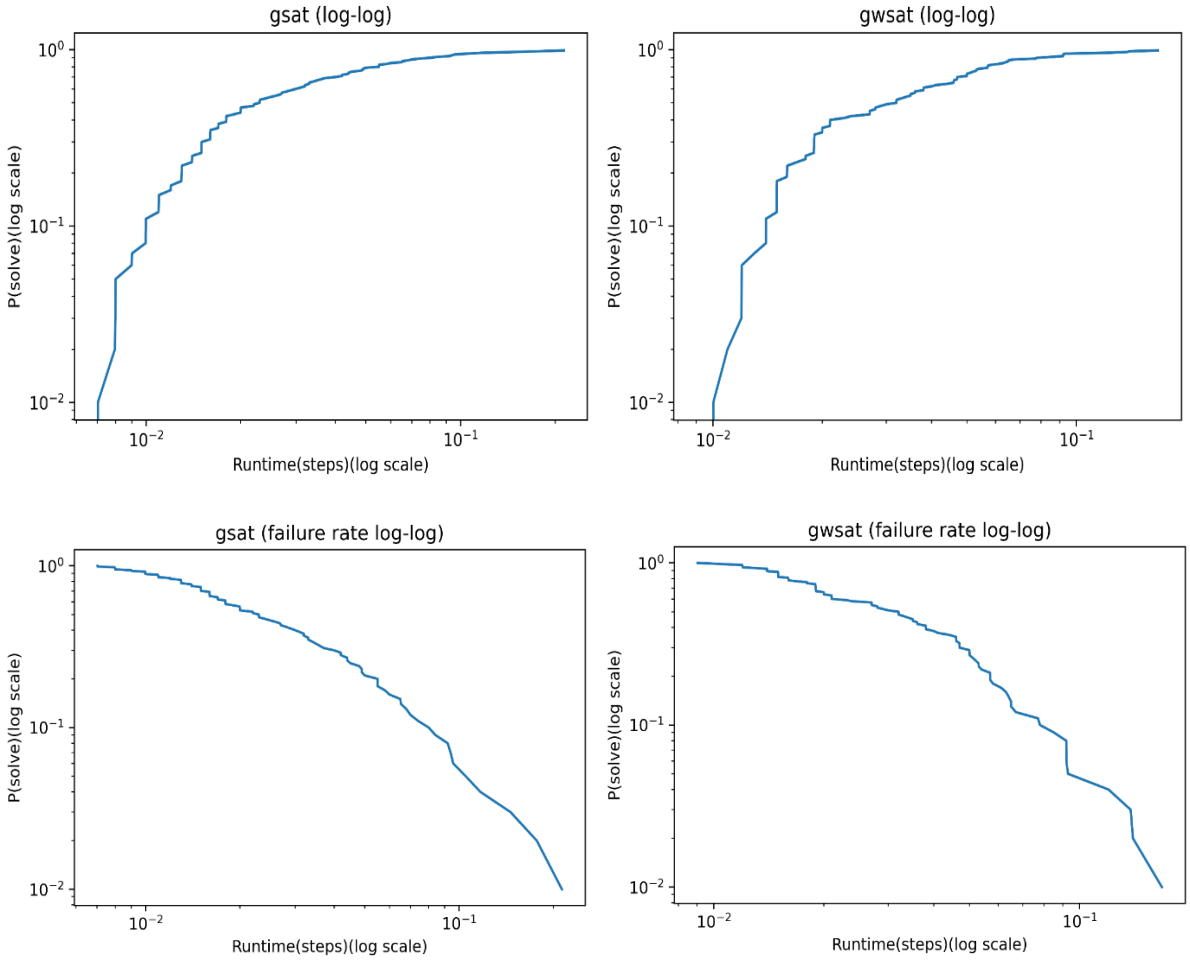

Figure 6: Various graph representations of RTD (GSAT and GWSAT)

I have plotted few more plots which are different types of representing RTD, the Log-Log plots and failure rate decay function are used to analyse the behaviour of any models for very short or long runs

7 Conclusion

The Local Search schemes play a vital role in finding correct assignments for the variables to satisfy a propositional formula. All local search methods are very well designed but there are some gaps which can be found in future to reduce the number of steps involved hence reduce the total run time making it computationally inexpensive, also future work for me would be to implement various researched techniques like 'Novelty', 'Adaptive Novelty' etc which have been proved to give excellent results. As an outcome of completing this assignment, I have also improved my scripting skills, dealt with various python libraries, and most importantly have learned the metaheuristic optimization techniques for solving SAT problems.

References

[1] Hoos, Holger & Stützle, Thomas. (2000). Local Search Algorithms for SAT: An Empirical Evaluation. Journal of Automated Reasoning. 24. 421-481.

[2] Selman, Bart & Levesque, Hector & Mitchell, David. (1992). A New Method for Solving Hard Satisfiability Problems. Proceedings Tenth National Conference on Artificial Intelligence.

[3] Selman, Bart, Henry A. Kautz, and Bram Cohen. "Noise strategies for improving local search." AAAI. Vol. 94. 1994.

[4] Glover, Fred. "Tabu search—part I." *ORSA Journal on computing* 1.3 (1989): 190-206.