Metaheuristic and Optimization - Assignment 2 Report

GWSAT:

GWSAT is a combination of GSAT and WalkSAT. Whether to select GSAT or WalkSAT for variable selection depends on the walk probability (wp). If the generated random number is less than the walk probability (wp) then variable to flip is selected using WalkSAT, otherwise GSAT is used.
The code for GWSAT has been implemented and can be found with the submitted file (Doshi_R00183334_GWSAT.py). Initially, GWSAT was evaluated on the basic configuration specified in the assignment document. This algorithm was evaluated on 3 different instances; ie uf20-01.cnf, uf20-02.cnf and uf50-01.cnf. The first 2 configuration files have 91 clauses and 20 variables each and the latter one have 218 clauses and 50 variables.

Note: *For each execution we are changing the random seed to bring more randomness in our solutions.*

instance name	Executions	Iterations	Restarts	wp	Valid Solutions	CPU runtime (seconds)
uf20-01.cnf	30	1000	10	0.4	30	0.76
uf20-02.cnf	30	1000	10	0.4	30	0.47
uf50-01.cnf	30	1000	10	0.4	19	141.42

The above table showcase the results of baseline evaluation as specified in the assignment. As we can observe that for all the instances we ran 30 executions and for each execution step we have 10 restarts and for each restart we have 1000 iterations. This means that in each restart maximum of 1000 variables can flipped. The column "total successful" indicates that how many executions provided valid solutions. So for uf20-01 and uf20-02, we got valid solution in all the execution steps and for uf50-01 we got only 19 valid solutions. But the runtime for all the instances are different. As we can see instance name **"uf20-02.cnf"** gave 30 valid solutions in just 0.47 seconds. On the other hand, uf20-21.cnf took around 0.76 seconds to generate 30 valid solutions. As the instance "uf50-01.cnf" is larger than the other 2 gave the same number of valid solutions in 141.42 seconds.

As we have evaluated our algorithm using baseline configurations, let’s try by varying these parameters.

1. uf20-01.cnf

Executions	Iterations	Restarts	wp	Valid Solutions	CPU runtime
30	100	10	0.1	28	2.53
30	100	10	0.6	30	0.56
50	100	10	0.1	46	3.93
50	100	10	0.6	50	0.97

The above table shows the evaluation of uf20-01.cnf instance file. It can be observed that there are 2 readings for 30 executions and 2 readings for 50 executions. The value of wp is set to 0.1 and 0.6 for both different number of executions. We can see that for lesser value of wp we get lesser number of valid solutions and the CPU runtime is greater

2. uf20-02.cnf

Executions	Iterations	Restarts	wp	Valid Solutions	CPU runtime
30	100	10	0.1	30	1.23
30	100	10	0.6	30	0.28
50	100	10	0.1	50	2.01
50	100	10	0.6	50	0.46

The above table shows the evaluation of uf20-02.cnf file. We can see that all the config gave valid solutions in all the executions but we can observe that the CPU runtime of greater wp is half the CPU time of low value of wp.

3. uf50-01.cnf

Executions	Iterations	Restarts	wp	Valid Solutions	CPU runtime
30	1000	10	0.1	10	328.97
30	1000	10	0.6	28	88.7
50	1000	10	0.1	10	511.86
50	1000	10	0.6	45	150.29

The above table shows the evaluation for uf50-01.cnf file. We can observe that for greater values of wp we get valid solutions for all the executions and also the CPU runtime is very less. On the other hand, for lesser value of wp, we can see that the CPU time is very large and we don't get valid solutions for all the executions.

WalkSAT\SKC with Tabu Search

WalkSAT\SKC starts with randomly selecting a clause from the list of unsatisfied clauses. One the we get the list of clauses, negative gain is calculated for each variable in that clause. Then, if any variable is having negative gain of 0 then that variable is selected directly to flip in the main configuration. Otherwise, variable selected for flipping is based on the value of wp. If the random number generated is less than wp than randomly a variable is selected from that clause, otherwise, the variable with minimal negative gain is selected. After selecting that variable we check whether that variable was previously selected within **tl** steps before. **'tl'** is known as tabu list. If that variable is in the tabu list then we don't flip that variable.
The code for WalkSAT\SKC has been implemented and can be found with the submitted file (Doshi_R00183334_WalkSAT.py). Initially, WalkSAT\SKC was evaluated on the basic configuration specified in the assignment document. This algorithm was evaluated on 3 different instances; ie uf20-01.cnf, uf20-02.cnf and uf50-01.cnf. The first 2 configuration files have 91 clauses and 20 variables each and the latter one have 218 clauses and 50 variables.

instance name	Executions	Iterations	Restarts	wp	Tabu Tenure	Valid Solutions	CPU runtime
uf20-01.cnf	30	1000	10	0.4	5	30	1.84
uf20-02.cnf	30	1000	10	0.4	5	28	2.93
uf50-01.cnf	30	1000	10	0.4	5	30	34.15

The above table shows the results for the baseline configuration. As we can see that we got valid solutions in all the executions for instance uf20-01 and uf50-01.cnf but for uf20-02.cnf we got only 28 valid solutions, respectively. The CPU runtime is also comparatively lesser for instance uf20-01.cnf, as compared to other instances. The largest instance gave 27 valid solutions in 24.15 seconds. Also, for each execution we are changing the random seed to bring more randomness in our solutions.

As we have evaluated our algorithm using baseline configurations, let’s try by varying these parameters.

1. uf20-01.cnf

Executions	Iterations	Restarts	wp	Tabu Tenure	Valid Solutions	CPU runtime
30	1000	10	0.1	5	27	6.60
30	1000	10	0.4	5	30	1.68
30	1000	10	0.1	10	27	6.00
30	1000	10	0.4	10	29	2.50

The above table is evaluation of uf20-01.cnf instance for WalkSAT\SKC. In the above evaluation, we have tried to vary tabu tenure and wp. As we can see that, varying the tabu tenure does not actually increase or decrease the number of valid solutions but the value of wp does affect the CPU runtime as well as valid solutions.

2. uf20-02.cnf

Executions	Iterations	Restarts	wp	Tabu Tenure	Valid Solutions	CPU runtime
30	1000	10	0.1	5	26	7.06
30	1000	10	0.4	5	28	2.85
30	1000	10	0.1	10	22	10.92
30	1000	10	0.4	10	30	2.42

The above table shows that length of tabu list does not affect the CPU runtime or number of valid solutions.

2. uf50-01.cnf

Executions	Iterations	Restarts	wp	Tabu Tenure	Valid Solutions	CPU runtime
30	1000	10	0.1	5	14	92.10
30	1000	10	0.4	5	27	42.58

30	1000	10	0.1	10	8	111.97
30	1000	10	0.4	10	18	78.60

The above table also proves what we have stated in examples before.

Run-Time Distribution Evaluation

Run-time distribution allows us to study the behaviour of local search algorithms on a single problem instance as well as on whole subclass of SAT. The measurement of RTDs does not incur any additional significant overhead and it is easy to get basic descriptive statistics of run-time like mean, median and standard deviation. The next part will compare empirical RTD of GWSAT and WalkSAT for following configurations:

executions=500
restarts(per execution)=100
iterations(per restart)=1000
wp=0.4
tabu list(for WalkSAT)=5

1. uf20-21.cnf

1) GWSAT:

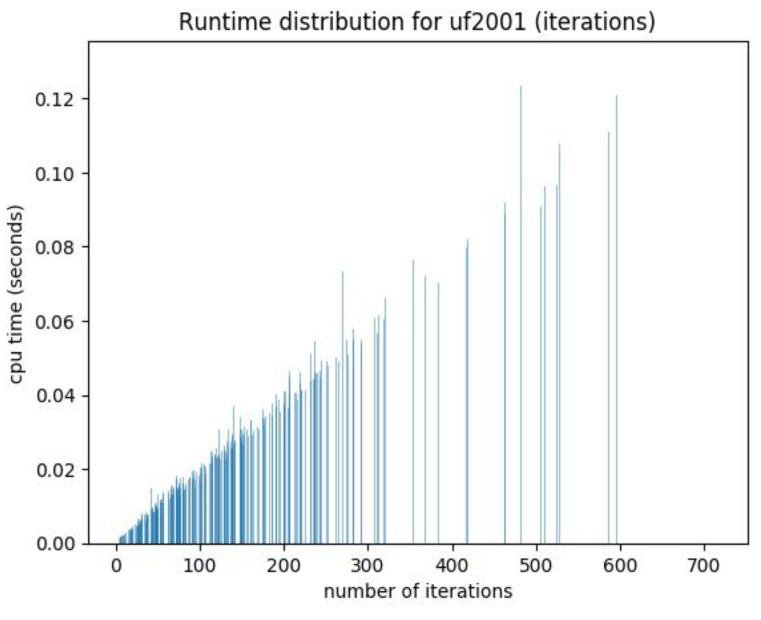

Figure 1

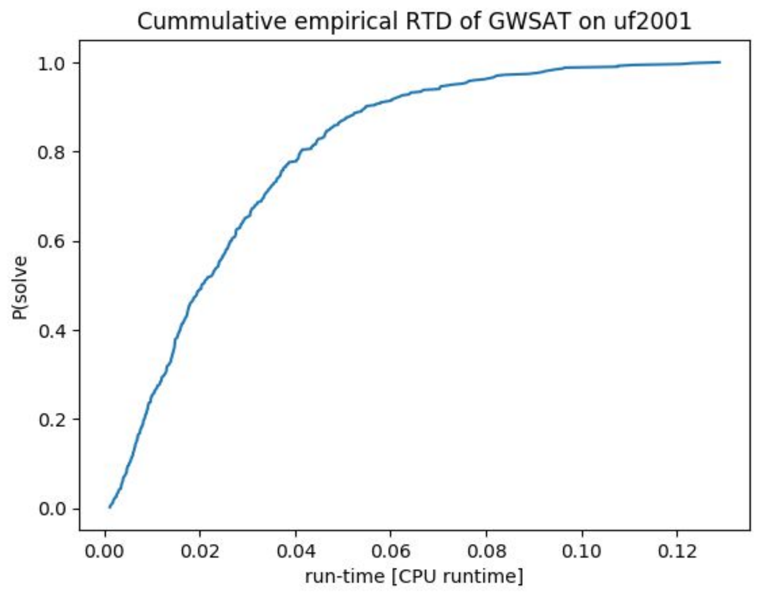

Figure 2

Figure 1 shows the runtime distribution of number of iterations as compared to the cpu runtime. We can see that majority of the valid solutions are before 400 iteration count. Figure 2 shows that after a certain time the CPU runtime gets constant.

2) WalkSAT:

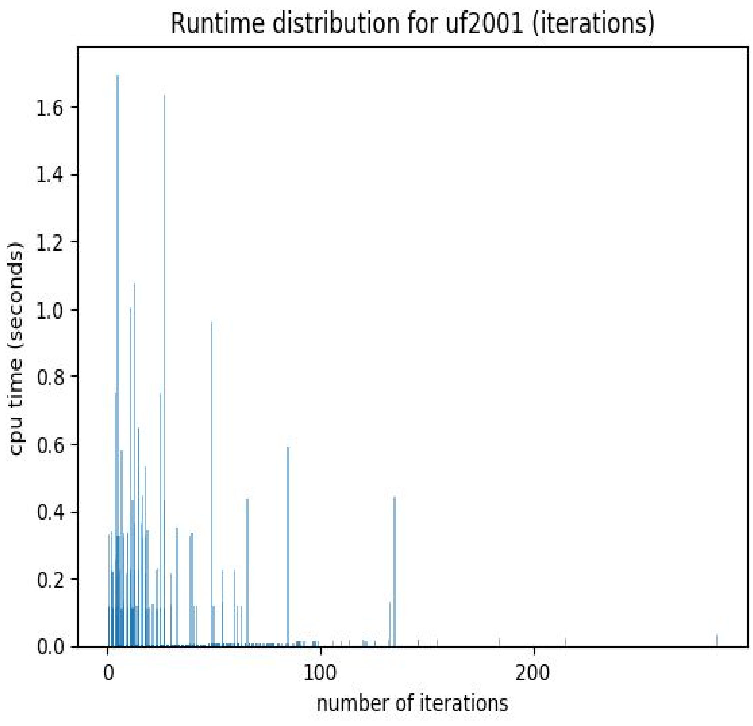

Figure 3

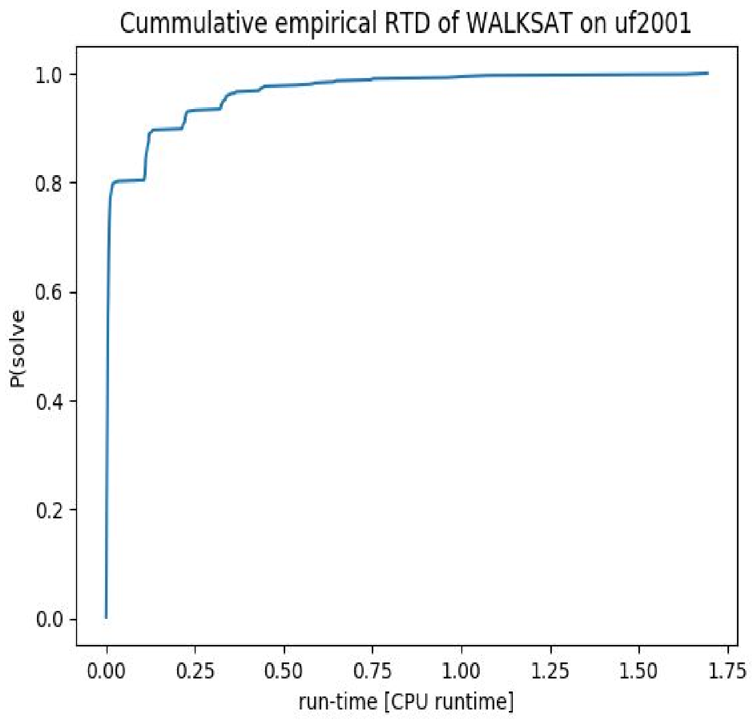

Figure 4

From Figure 3, we can identify that we get a majority of the solution before 100 iterations so we can set 150 iterations as maximum tries. In Figure 4, the valid solutions are very quick, which means that the algorithm is performing very fast and after a particular time the time gets constant.

<u>Comparing GWSAT and WalkSAT for uf20-01.cnf instance:</u>

- When comparing the runtime results of both the SAT algorithms for uf20-01.cnf instance, we can say that,
  - WalkSAT is getting valid solutions in very less number of iterations
  - the maximum CPU runtime for some iterations goes around 1.75 seconds
  - there is a chance of getting lesser number of valid solutions.

- On the other hand,
    - GWSAT is getting valid solutions in many different iteration number ranging from 0 till 650
    - maximum CPU runtime is comparatively very low than that of WalkSAT.
    - More chances of getting valid solutions

2. uf20-02.cnf

1) GWSAT

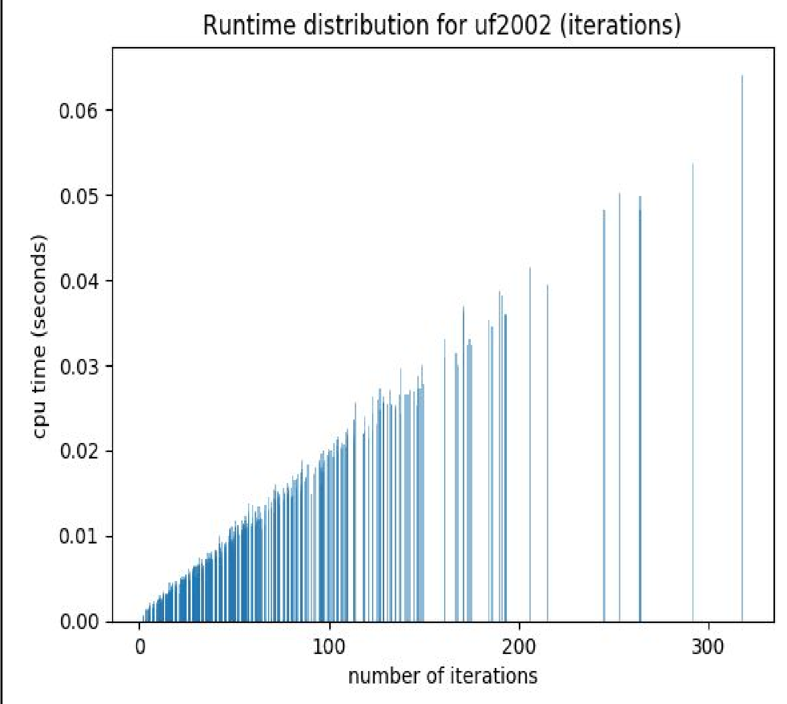

Figure 5

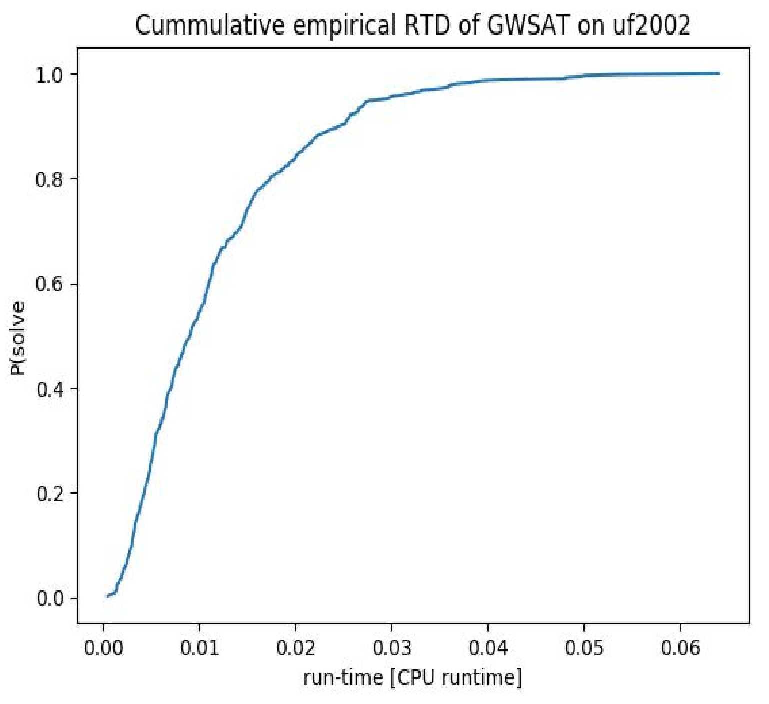

Figure 6

From Figure 5, we can say that there are a wide range of possible iterations that can take place to get a valid solution. The maximum iteration count comes around 350 so we can run this SAT algorithm for a maximum of 400 iterations per restart. In Figure 6, we see that after 0.02 seconds, the runtime reaches its threshold and gets constant.

2) WalkSAT

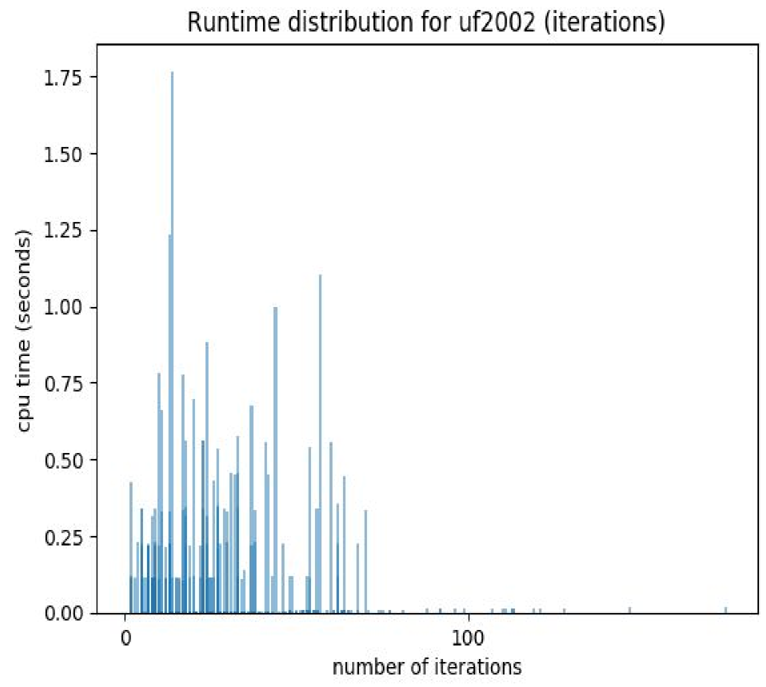

Figure 7

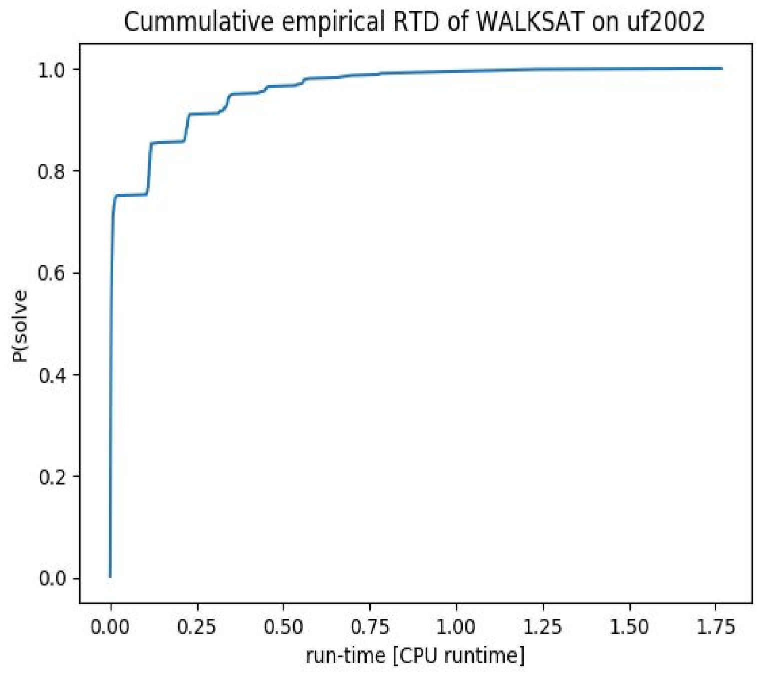

Figure 8

From Figure 7, we can see that the maximum number of valid solutions comes under 100 iterations so we can 100 "**maximum tries"** parameter. Figure 8 indicates that majority of the valid solutions take very less cpu runtime.

Comparing GWSAT and WalkSAT for uf20-02.cnf instance:

- When comparing the runtime results of both the SAT algorithms for uf20-02.cnf instance, we can say that,
  - WalkSAT is getting valid solutions in very less number of iterations (0-100 iterations)
  - the maximum CPU runtime for some iterations is somewhat more than 1.75 seconds
  - Possibility of getting lesser number of valid solutions.
- On the other hand,
  - GWSAT is getting valid solutions in many different iteration number ranging from 0 till 350
  - but the maximum CPU runtime is comparatively very low than that of WalkSAT.
  - Possibility of getting more valid solutions than WalkSAT.