

Sat Solvers

Problem: In this problem we are trying to implement a solution to a given sat problem.

The files used in the problem and for the demonstration can be found in the zip file included with the pdf and contain two files with the name:

- ☑ uf20-01.cnf
- ☑ uf20-02.cnf

Disclaimer - the results produced can vary in the execution time for someone else, at the time of making this pdf the results were obtained on my machine with system specs:

Model Name:	MacBook Pro
Model Identifier:	MacBookPro15,2
Processor Name:	Quad-Core Intel Core i5
Processor Speed:	2.4 GHz
Number of Processors:	1
Total Number of Cores:	4
L2 Cache (per Core):	256 KB
L3 Cache:	6 MB
Hyper-Threading Technology:	Enabled
Memory:	8 GB
Boot ROM Version:	1037.60.58.0.0 (iBridge: 17.16.12551.0.0,0)

Let's begin this problem with an introduction to what actually is a SAT problem and why it is important and why do we need to really build a sat solver

Introduction -

What is a SAT problem ?

A sat problem is an abbreviation for the satisfiability problem, we are here given a boolean Expression with a set of variables and we are asked to find a solution that actually sets the final Result of the entire expression to be True.

The SAT problem can be used to convert many complex problems into simple NP complete as first originally proposed by Cook-Levin theorem.

Problems look like,

$$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge \neg x_1$$

Where we are given a set of AND operation on clauses where each clause consist of certain variables which I will call them literals , that are within an OR operation.

The problem is called NP complete because given a solution it can be verified within a deterministic polynomial time.

The challenge proposed over here was that even if just one of the clause in the expression of all the clauses is set to False then the entire Expression becomes False. Moreover literals can be used in more than one expression, Hence setting a literal True can set that same literal False if it is used in negation in some other clause.

Hence we are required to find the unique or list of unique combination of these boolean literals that actually make the assignment of the entire expression to be true.

In the proposed solutions we actually try to solve and explore the many possibilities of a set of unique solutions that actually make up the expression to be true, However trying to optimise the tradeoff with time by using these deterministic approaches such as GWSAT and walkout.

There are many ways to solve the sat problem, but in this report we will only focus on the two namely:

- ◆ GWSAT
- ◆ Walksat with Tabu

And try to explain the run time analysis,

Algorithms Explanations -

What is the GWSAT solution ?

In the GWSAT approach we are following some guidelines to expiration why we are following these guide lines will be provided later

Step 1: We initialise the problem by reading all the clauses and literals present in the expression

Step 2: We will initialise a random solution to all the literals present in the problem with a 50% chance that the literal can be positive or negative.

Step 3: We begin the problem by generating a random number between 0 and 1 if this random number is below the chosen walk probability set in the step 1. We will go to step 3 (a) else we will move to step 3 (b)

Step 3 (a) : we will now pick a random literal in the set of all the literals that are currently set to make the problem unsatisfied. (we read all the clauses in step 1)

Step 3 (b) : we will compute the Net gain for each of the variables present in the expression and we will flip that particular variable that computes to the maximum net gain of the entire sat expression after we flip it.

Step 4 : we will terminate the program if we are out of the maximum number of flips allowed and then we might have to begin with the random assignment or a new restart heading back to the step 2 (why ? Discussed in the explanation section)

Step 5: If the sat problem is solved by calling sat checker we will begin the next execution otherwise we will move to another restart and back to Step 2

Explanations on GWSAT:

- ☑ Net gain is defined as the positive gain minus the negative gain
- ☑ Positive gain is defined as the total clauses that become satisfied upon flipping that particular variable that were previously unsatisfied
- ☑ Negative gain is defined as the total number of clauses that become Unsatisfied that were previously satisfied upon flipping the particular variable
- ☑ We calculate the random number to decide if want to change all the literals or specifically want to pick the literals involved in unsat clauses
- ☑ A lot of deciding whether we want to flip the variable from all clauses or only unsat clauses depends upon the walk probability provided at the initialisation

- ☑ In order to ensure that we are not stuck at local minima we will restart with a random assignment if the set number of iteration or flips are not enough to reach to a point that the assignment of the variables satisfies the final expression

What is the Walksat (with Tabu) solution ?

Step 1: We initialise the problem by reading all the clauses and literals present in the expression

Step 2: We will initialise a random solution to all the literals present in the problem with a 50% chance that the literal can be positive or negative. We will also initialise a tabu list which is QUE data structure and has the maximum occupancy equal to the tab space taken from the program. A new variable can only be added to this due if the variable is not present (why ? Discussed in the explanation)

Step 3: We begin the problem by selecting a random UNSAT clause and then computing the Negative gain for each of the variables present in the clause. If we get any variable(s) that actually have their gain equal to 0 we will flip that variable . If multiple we will break the ties by randomly selecting any one variable that has 0 gain. If there no variable with gain 0 we will move to step 3 a We will add the variable to our tabu list if the variable is currently not present in the tabu list.

Step 4 : we will now generate a random number between 0 and 1 if the number generated is less than the set walk probability we will go to step 4 (a) otherwise we will go to step 4 (b)

Step 4 (a) : we will select any variable from the currently choosen unsat clause and flip it.

Step 4 (b): we will flip the variable with the minimum negative gain and we will flip it.

Step 5: If the sat problem is solved by calling sat checker we will begin the next execution otherwise we will move to another restart moving back to Step 2

Explanations on Walksat (with Tabu):

- ☑ Tabu list controls the track of the variables that have already been flipped making sure that a new variable is flipped after certain number of iterations, hence can actually waste a lot of computing power.
- ☑ Tabu list has parameter tabu space that controls the memory of the tabu list meaning the high value for the variable will actually not permit entry of variables that are not new to the list
- ☑ Negative gain is defined as the total number of clauses that become Unsatisfied that were previously satisfied upon flipping the particular variable.
- ☑ We calculate the random number to decide if want to pick a random variable from the clause or specifically want to pick the variable with minimum negative gain involved in unsat clauses
- ☑ A lot of deciding wether we want to flip the variable at random or at deterministically depends upon the walk probability provided at the initialisation.
- ☑ In order to ensure that we are not stuck at local minima we will restart with a random assignment if the set number of iterations or flips are not enough to reach to a point that the assignment of the variables satisfies the final expression

Interpreting the results (standard configuration)-

(below are the discussed results on running the algorithm over different instances)

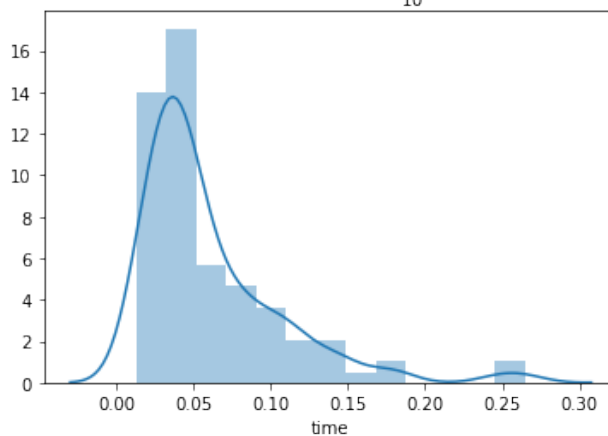
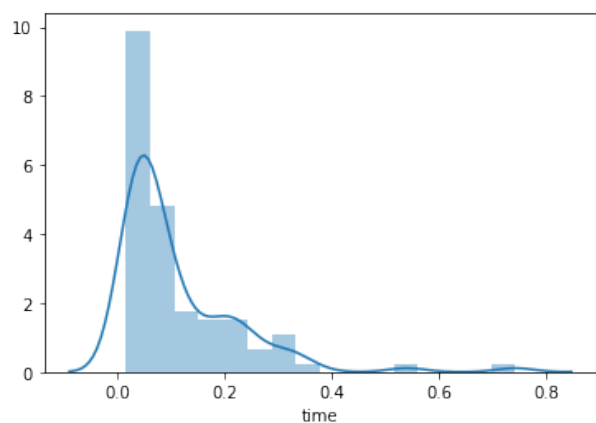
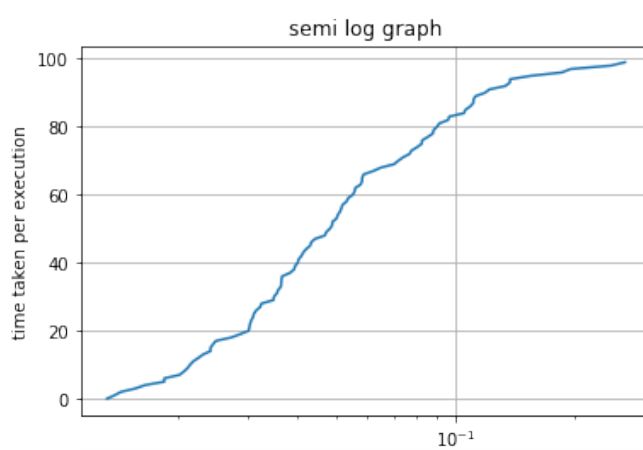
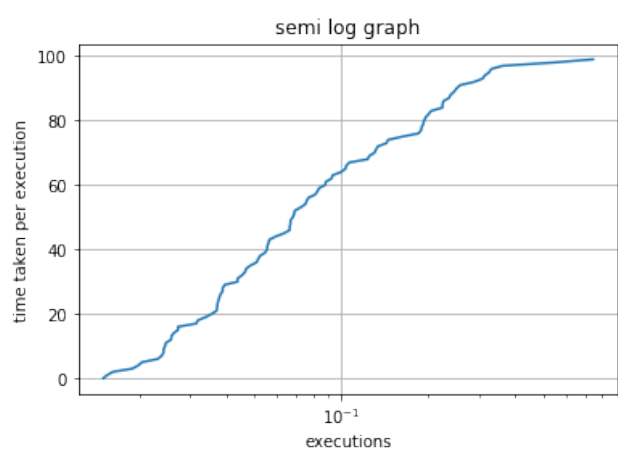
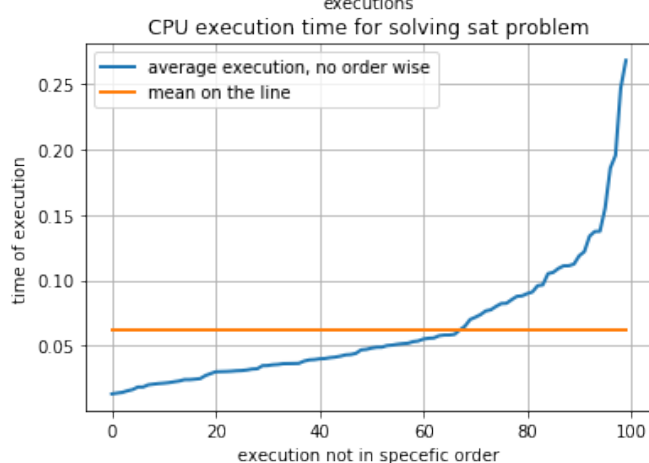
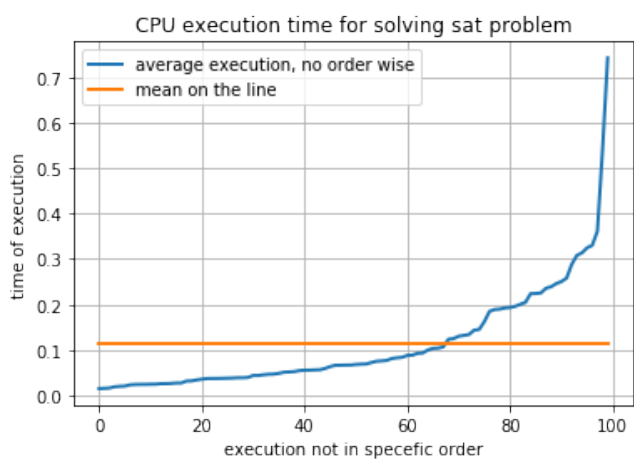
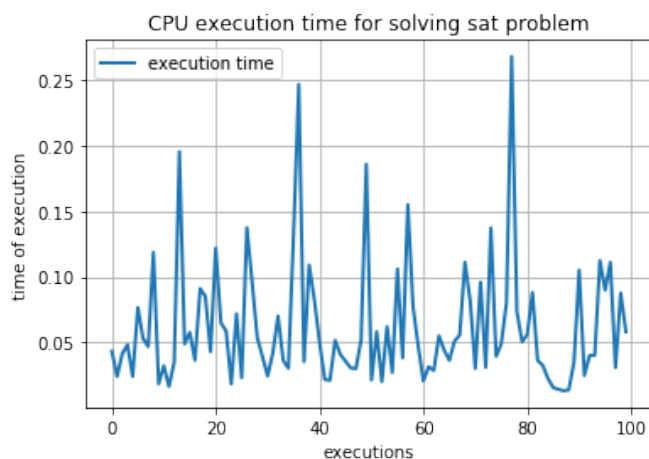
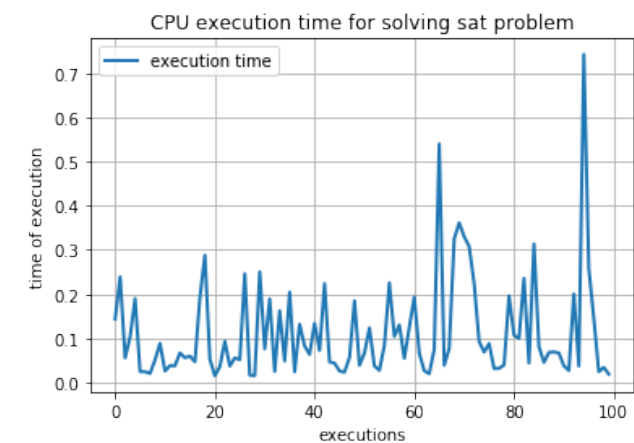
I encourage you to kindly run the algorithm for the set configuration get the similar results

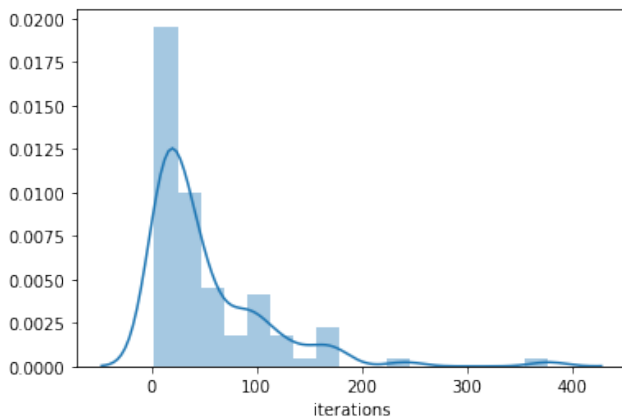
In this section the results generate will be based on the standard configuration, as provided by the assignment specification. On the left are outputs obtained on the 01 cnf file and on the right are outputs from the 02 file.

GWSAT

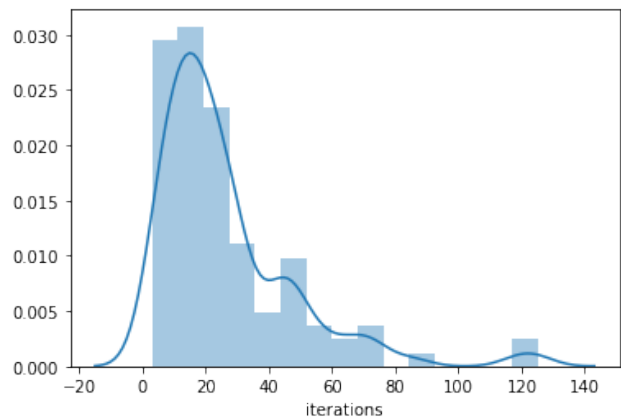
(cnf -01)

(cnf-02)





*****this is the final verdict*****
 this is the length of soln list: 100
 this is the length of unique soln list: 8
 executions with successful solutions: 100



*****this is the final verdict*****
 this is the length of soln list: 100
 this is the length of unique soln list: 26
 executions with successful solutions: 100

On evaluating an algorithm we have made five different plots to analyse the working.

- * time taken vs the execution number actually demonstrates that which execution seed took the most time to calculate the output
- * Cpu execution time plot arranges and tries to find the mean time taken per execution
- * A semi log plot explains it better as the function is monotonically increasing and can be used to evaluate the performances
- * Time taken and iterations follow a skewed gamma distribution

Conclusion-

The 01 cnf file almost 30 iterations were sufficient to calculate for the most of the solutions meaning we got the solution within 30 iterations for most of the executions

The mean time taken for 01 file is roughly 0.1 sec and most occurring time (not average but from the gamma distribution) was found to be 0.12 seconds.

For 01 file the peak or the most number of executions were solved in roughly 25-30 iterations.

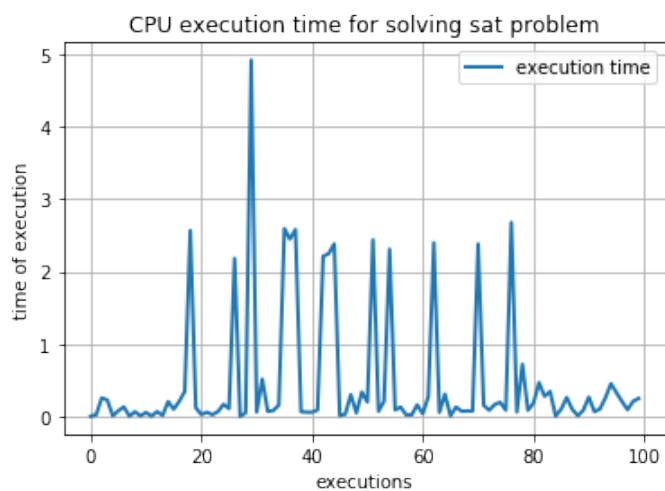
The 02 cnf file almost 25 iterations were sufficient to calculate for the most of the solutions meaning we got the solution within 25 iterations for most of the executions

The mean time taken for 02 file is roughly 0.06 sec and most occurring time (not average but from the gamma distribution) was found to be 0.05 seconds.

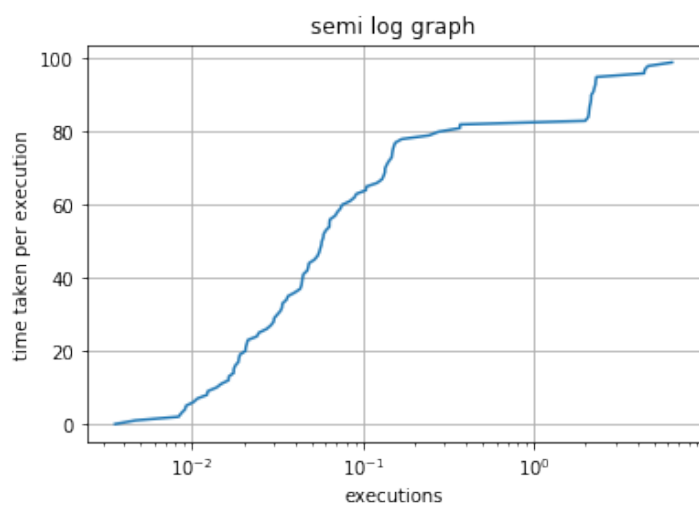
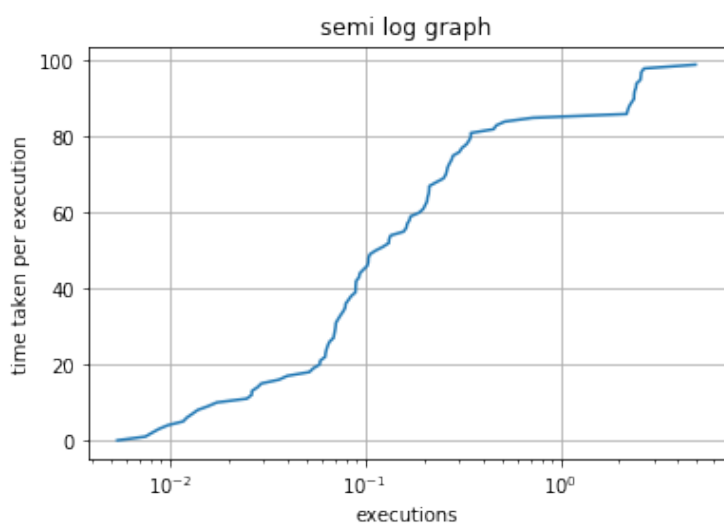
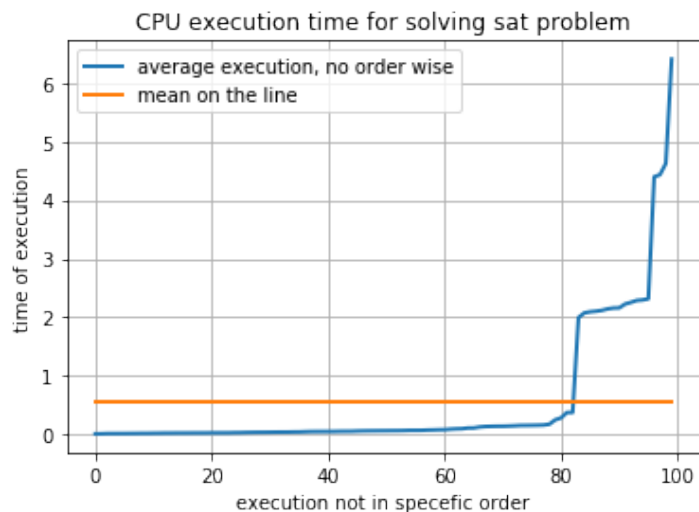
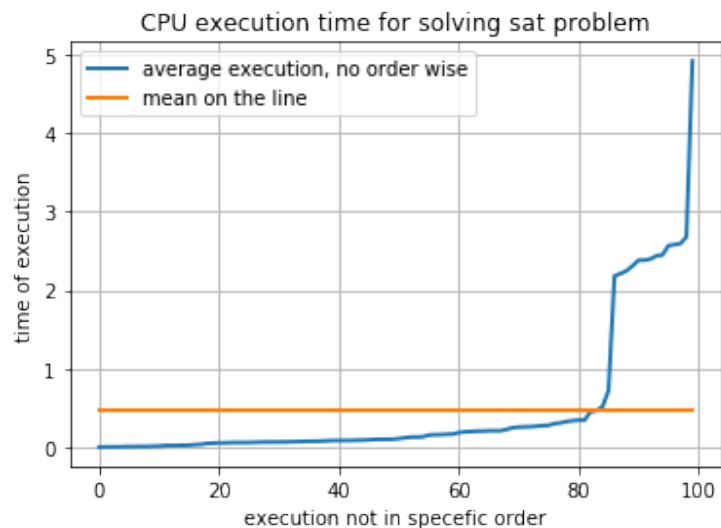
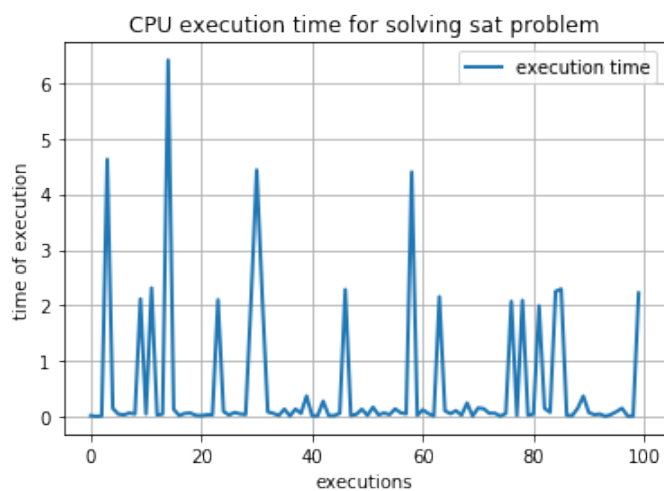
For 02 file the peak or the most number of executions were solved in roughly 20 iterations.

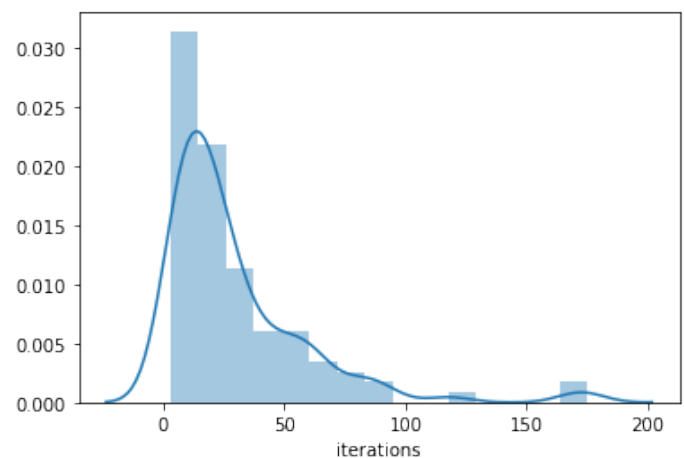
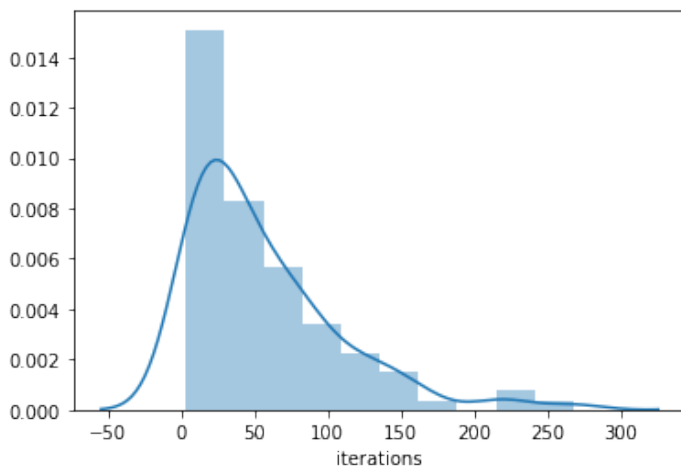
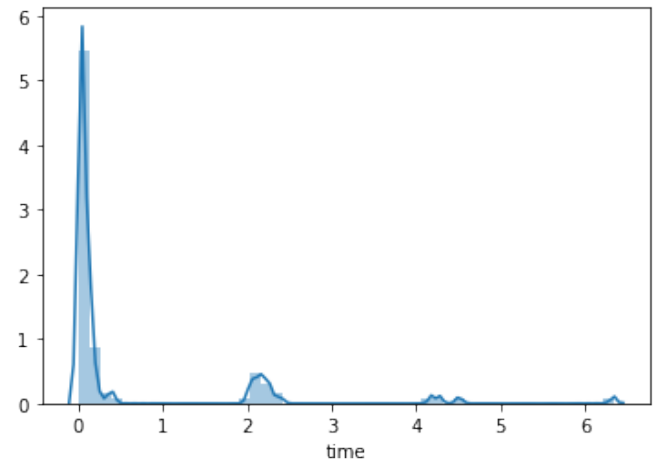
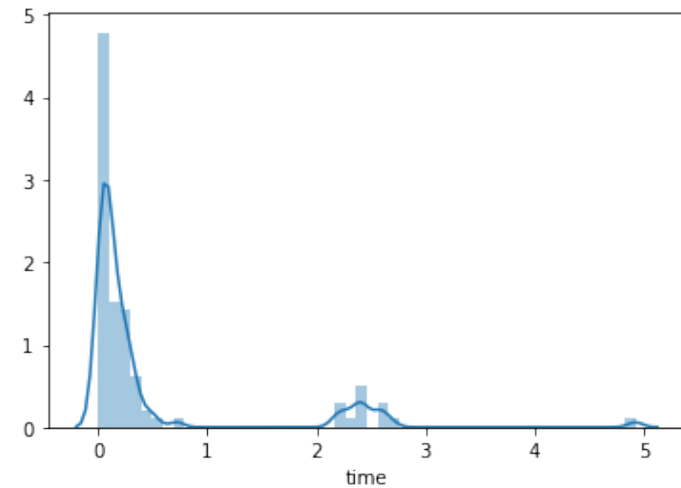
Walksat (with tabu)

(cnf-01)



(cnf-02)





*****this is the final verdict*****
 this is the length of soln list: 100
 this is the length of unique soln list: 8
 executions with successful solutions: 100

*****this is the final verdict*****
 this is the length of soln list: 100
 this is the length of unique soln list: 26
 executions with successful solutions: 100

Conclusion-

The 01 cnf file almost 30 iterations were sufficient to calculate for the most of the solutions meaning we got the solution within 30 iterations for most of the executions

The mean time taken for 01 file is roughly 0.5 sec and most occurring time (not average but from the gamma distribution) was found to be 0.2 seconds.

For 01 file the peak or the most number of executions were solved in roughly 25-30 iterations.

The 02 cnf file almost 25 iterations were sufficient to calculate for the most of the solutions meaning we got the solution within 25 iterations for most of the executions

The mean time taken for 02 file is roughly 0.5 sec and most occurring time (not average but from the gamma distribution) was found to be 0.2 seconds.

For 02 file the peak or the most number of executions were solved in roughly 25 iterations.

Comparing GWSAT with walkSAT-

In both walk sat and GWSAT equal number of solutions were explored (unique) however in the Walksat approach

There were usually less time taken in walk approach (as the update occurs only if the element is present not in the tabu list que) this can also cause a waste of computation power if originally the solution was not found and hence can increase the computation time to average 0.5 seconds meaning this approach takes more time than GWSAT approach.

What happens if we vary the standard parameters ?

In this part we will see the impact of the following parameters on the GWSAT and walk sat problem

- ☒ Iterations
- ☒ Restarts
- ☒ Walk probability
- ☒ Tabu space (only for walk sat)

Iterations-Total number of iterations does not increase the performance any better however decreasing the set number of iterations can actually interfere in finding the solution meaning we will find less number of solutions.

Lets try to visualise with 10 (very low) number of iterations as this can will save time and see the impact on both approaches.

For GWSAT-

01 cnf-

```
this is the length of soln list: 68
this is the length of unique soln list: 8
executions with successful solutions: 68
```

02 cnf-

```
this is the length of soln list: 86
this is the length of unique soln list: 27
executions with successful solutions: 86
```

Walk sat with tabu-

01 cnf-

```
this is the length of soln list: 60
this is the length of unique soln list: 8
executions with successful solutions: 60
```

02 cnf-

```
this is the length of soln list: 87
this is the length of unique soln list: 27
executions with successful solutions: 87
```


Clearly we see not all the executions were solved with lowering the iteration count moreover both approaches equally explored different unique solutions possible for SAT problem.

Restarts- Random assignments allowed to get out of the local minima, if we are out of the iterations. We will compare both the approaches and see which one of them actually gets stuck in the local minima yielding in no solution in set time.

Set the restart count to just 1(very low)

For GWSAT-

01 cnf-

```
this is the length of soln list: 100
this is the length of unique soln list: 8
executions with successful solutions: 100
```

02 cnf-

```
this is the length of soln list: 100
this is the length of unique soln list: 26
executions with successful solutions: 100
```

Walk sat with tabu-

01 cnf-

```
this is the length of soln list: 86
this is the length of unique soln list: 8
executions with successful solutions: 86
```

02 cnf-

```
this is the length of soln list: 83
this is the length of unique soln list: 25
executions with successful solutions: 83
```

This will also yield in a time reduction to solve the problem however there is no or very little difference in the execution time, so can be ignored (graph not added because of no significant improvement found but can be seen in the code)

WalkSat has more tendency to run of restarts as the assignment of the variable depends heavily upon the tabu list and can yield to no results if there are not many restarts allowed.

Walk probability - this allows problem to be solved in the random manner the more the probability the less is the solution dependent on the deterministic min gain rather than deterministically flipping the maximum gain literal.

In GWSAT

01 cnf- decreasing the walk prob- 100 solutions found
increasing the walk prob- 100 solutions found

02 cnf- decreasing the walk prob- 100 solutions found
increasing the walk prob- 100 solutions found

In walksat(with tabu)

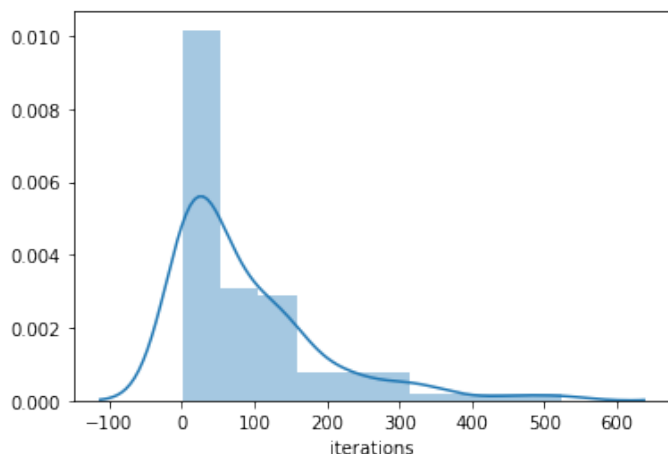
01 cnf- decreasing the walk prob- 100 solutions found

increasing the walk prob- 100 solutions found

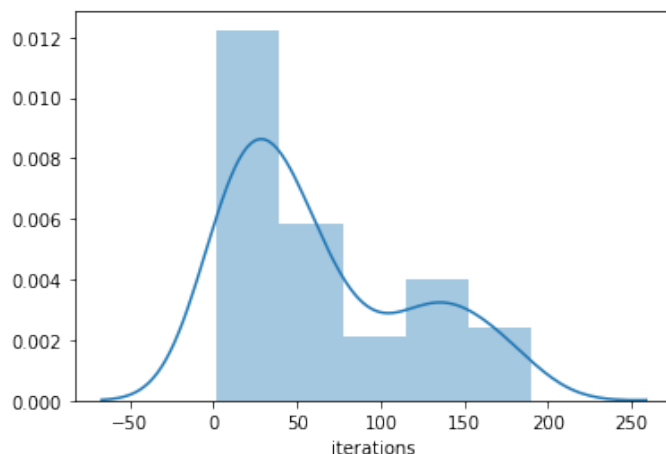
02 cnf- decreasing the walk prob- 100 solutions found
increasing the walk prob- 100 solutions found

Walk sat (with tabu for 01 cnf)-

Walk prob is 0.1(89.72 mean iterations)

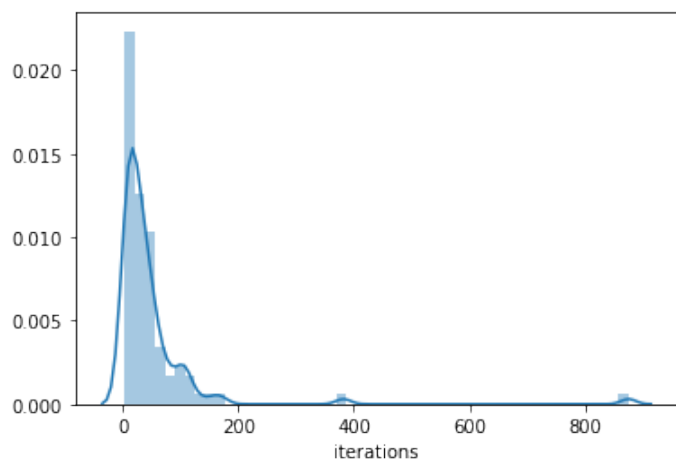


Walk prob is 0.9(64.72 mean iterations)

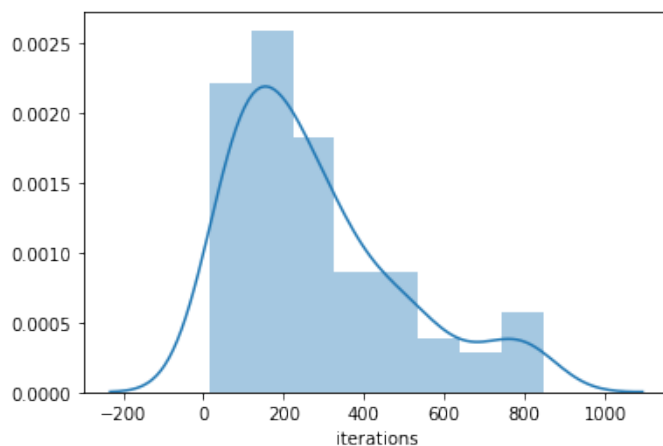


GWsat (with tabu for 02 cnf)-

Walk prob is 0.1(48.79 mean iterations)



Walk prob is 0.9(282.8 mean iterations)

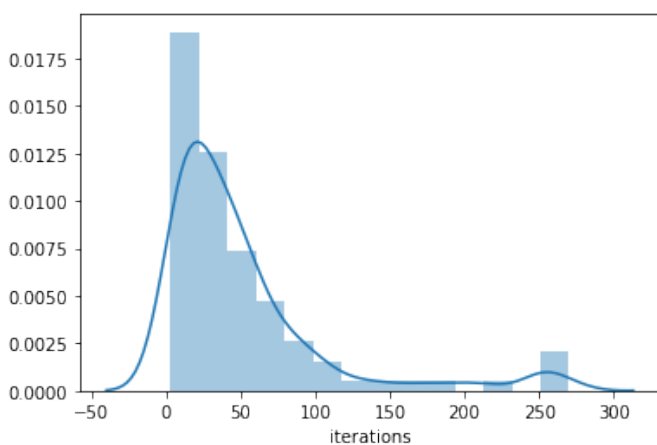


With lower walk probability we can get results faster result as the result is deterministic to flipping the variable with usually the min negative or highest net gain. This is the reason why we converge to the solution faster.

Varying the tabu space(only for walks) - this includes the size of the variables that can be inserted into the que, if there already exists variables then we would make more number of iterations to get new variable inside meaning we can actually run out of iterations if we keep it too large and thus resulting in no solution.

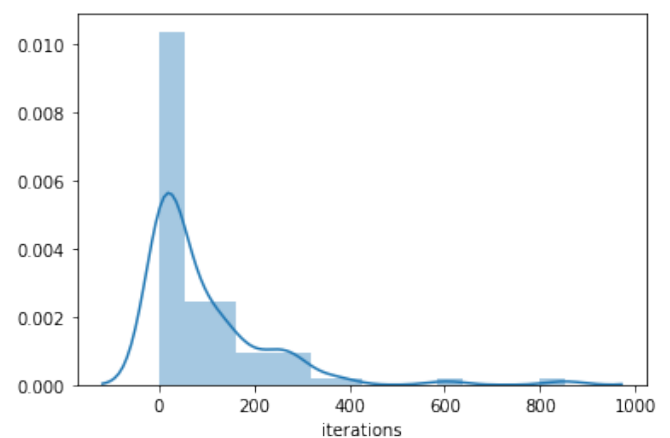
01 cnf-

Tabu space is 2 (51mean iterations)



Execution time- 0.12sec/execution

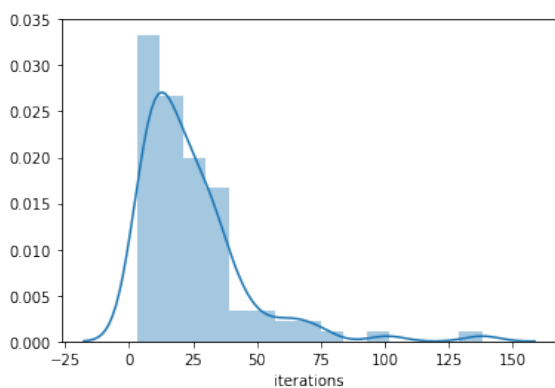
Tabu space is 10 (94.3 mean iterations)



Execution time-4.22sec/execution

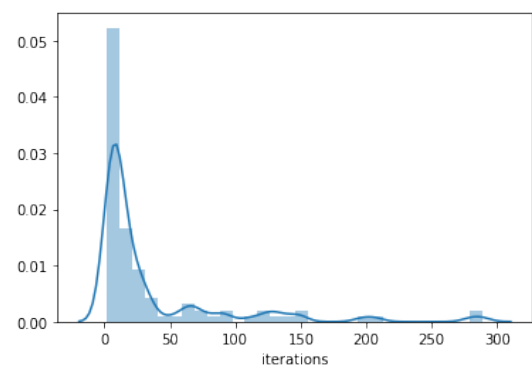
02 cnf-

Tabu space is 2 (24.3mean iterations)



Execution time-0.060sec/execution

Tabu space is 10 (35.6 mean iterations)



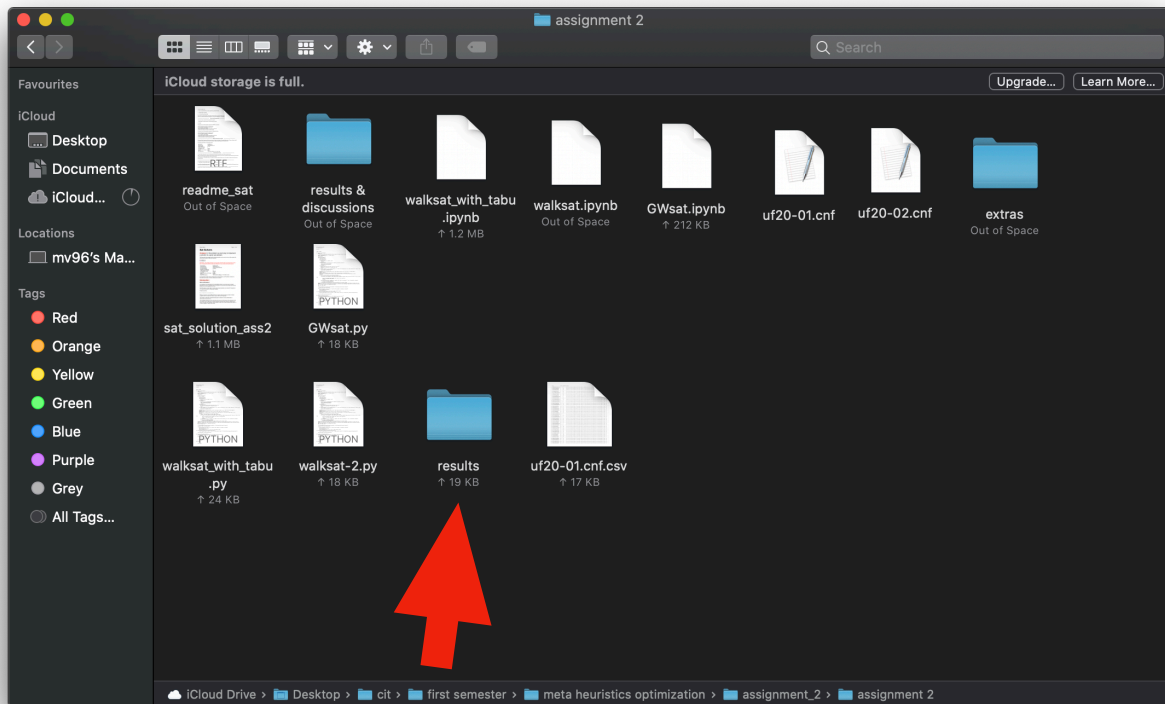
Execution time- 2.74sec/execution

Final conclusion Table for varying parameters-

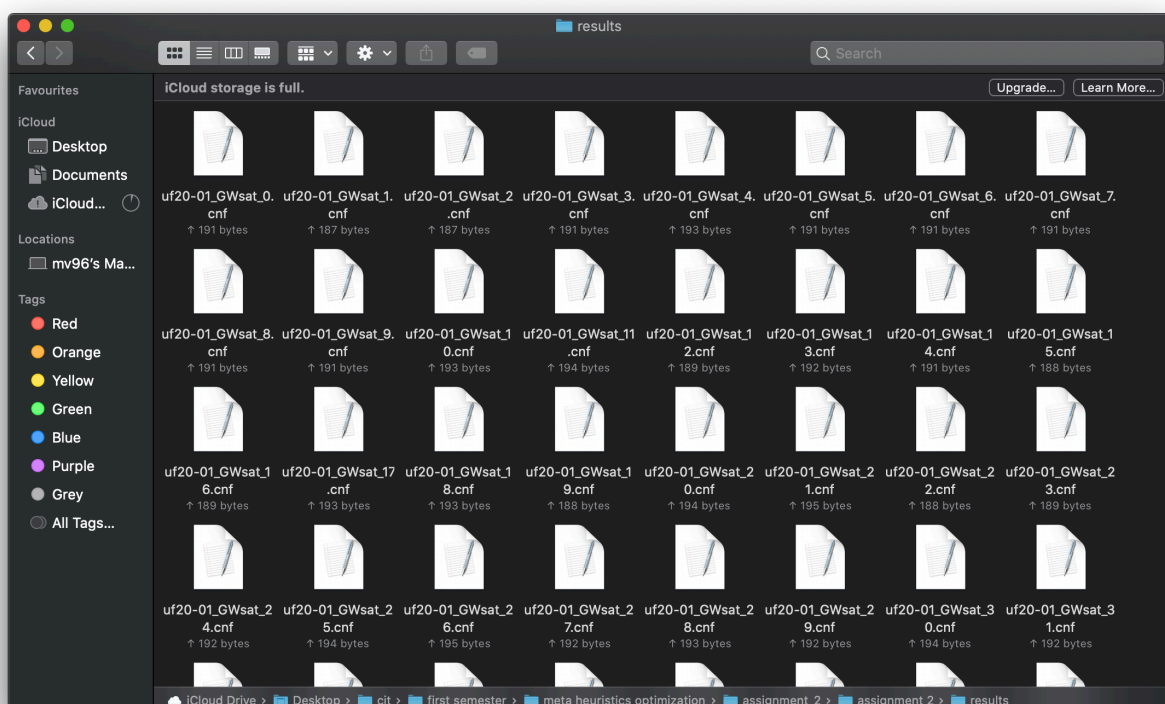
	Iterations	Restarts	Walk probability	Tabu space or step(for walk sat only)
Increasing the parameter	More iterations ensure that we get to reach the solution	More number of restarts allows our algorithms to get out of the local minima which can help in finding the solutions	Increasing the walk probability will allow our solution to fall in the less deterministic region meaning making more explorations by picking random literals hence taking more time	Can take extremely long time to find solutions for the execution as the new flip is dependent upon the literals inserted in the tabu list, However very less number of update or flips are actually required
Decreasing the parameter	Lesser iterations can yield in no solution sometimes	Less number of restarts can cause a solution to be stuck in global minima which can yield to no optimum solution	Decreasing the walk probability will allow our solution to fall in the more deterministic region meaning making less explorations by picking random literals hence taking less time to converge	Takes less time to find solutions for the execution as the new flip is dependent upon the literals inserted in the tabu list(which is very low), Hence more of the updates will take place, does a lot of updates or flips because the tabu space is low and hence overall faster achievement of result

How to view the results-

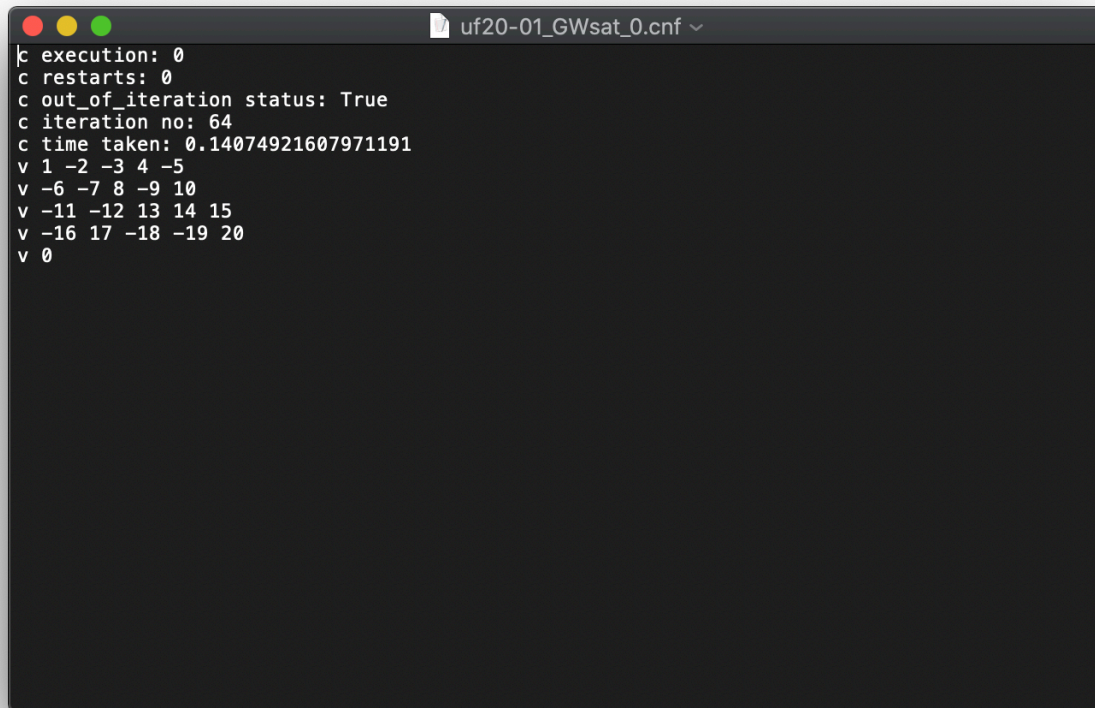
In the same code directory you will see a new folder generated with the name Results



With in that folder you will all the results generated for each execution with the last character before the extension denoting the iteration number.

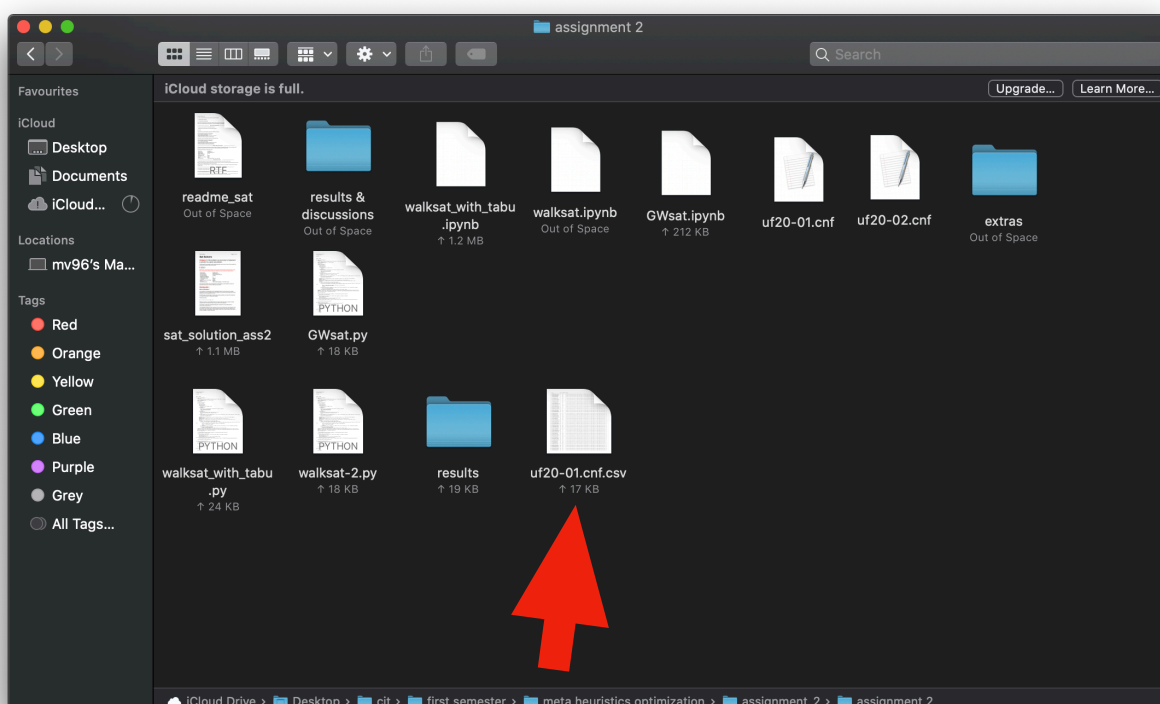


You can view the file and it will be in the standard cnf file format as discussed in Week 8 of the lab.



```
uf20-01_GWsat_0.cnf
c execution: 0
c restarts: 0
c out_of_iteration status: True
c iteration no: 64
c time taken: 0.14074921607971191
v 1 -2 -3 4 -5
v -6 -7 8 -9 10
v -11 -12 13 14 15
v -16 17 -18 -19 20
v 0
```

Along side it is really difficult for all the results to be visualised together in one cnf file hence, the code when run will also generate a csv file with in the drive containing all the information of all the executions.



uf20-01.cnf

	time	execs	iterations	solution	status	restarts
0	0.14074921607971200	0	64	{1: 1, 2: 0, 3: 0, 4: 1, 5: 0, 6: 0, 7: 0, 8: 1, 9: 0, 10: 1, 11: 0, 12: 0, 13: 1, 14: 1, 15: 1, 16: 0, 17: 1, 18: 0, 19: 0, 20: 1}	TRUE	0
1	0.2501029968261720	1	118	{1: 0, 2: 1, 3: 1, 4: 1, 5: 0, 6: 0, 7: 0, 8: 1, 9: 1, 10: 1, 11: 1, 12: 0, 13: 0, 14: 1, 15: 1, 16: 0, 17: 1, 18: 1, 19: 1, 20: 1}	TRUE	0
2	0.06109428405761720	2	24	{1: 0, 2: 1, 3: 1, 4: 1, 5: 0, 6: 0, 7: 0, 8: 1, 9: 1, 10: 1, 11: 1, 12: 0, 13: 0, 14: 1, 15: 1, 16: 0, 17: 1, 18: 1, 19: 1, 20: 1}	TRUE	0
3	0.10018777847290000	3	46	{1: 1, 2: 0, 3: 0, 4: 1, 5: 0, 6: 0, 7: 0, 8: 1, 9: 0, 10: 1, 11: 0, 12: 0, 13: 1, 14: 1, 15: 1, 16: 0, 17: 1, 18: 0, 19: 0, 20: 1}	TRUE	0
4	0.19979596138000500	4	88	{1: 1, 2: 0, 3: 0, 4: 0, 5: 0, 6: 1, 7: 0, 8: 0, 9: 1, 10: 0, 11: 0, 12: 0, 13: 0, 14: 1, 15: 1, 16: 0, 17: 1, 18: 0, 19: 0, 20: 1}	TRUE	0
5	0.02517080307006840	5	8	{1: 1, 2: 0, 3: 0, 4: 0, 5: 0, 6: 1, 7: 0, 8: 0, 9: 1, 10: 0, 11: 0, 12: 0, 13: 1, 14: 1, 15: 1, 16: 0, 17: 1, 18: 0, 19: 0, 20: 1}	TRUE	0
6	0.021818876266479500	6	6	{1: 1, 2: 0, 3: 0, 4: 1, 5: 0, 6: 1, 7: 0, 8: 0, 9: 0, 10: 1, 11: 0, 12: 0, 13: 1, 14: 1, 15: 1, 16: 0, 17: 1, 18: 0, 19: 0, 20: 1}	TRUE	0
7	0.019291162490844700	7	7	{1: 1, 2: 0, 3: 0, 4: 1, 5: 0, 6: 1, 7: 0, 8: 0, 9: 0, 10: 1, 11: 0, 12: 0, 13: 1, 14: 1, 15: 1, 16: 0, 17: 1, 18: 0, 19: 0, 20: 1}	TRUE	0
8	0.05171990394592290	8	22	{1: 1, 2: 0, 3: 0, 4: 1, 5: 0, 6: 0, 7: 0, 8: 1, 9: 0, 10: 1, 11: 0, 12: 0, 13: 1, 14: 1, 15: 1, 16: 0, 17: 1, 18: 0, 19: 0, 20: 1}	TRUE	0
9	0.09645605087280270	9	42	{1: 1, 2: 0, 3: 0, 4: 1, 5: 0, 6: 0, 7: 0, 8: 1, 9: 0, 10: 1, 11: 0, 12: 0, 13: 1, 14: 1, 15: 1, 16: 0, 17: 1, 18: 0, 19: 0, 20: 1}	TRUE	0
10	0.026294708251953100	10	7	{1: 1, 2: 0, 3: 0, 4: 1, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0, 10: 1, 11: 0, 12: 0, 13: 1, 14: 1, 15: 1, 16: 0, 17: 1, 18: 0, 19: 0, 20: 1}	TRUE	0
11	0.03942513465881350	11	18	{1: 1, 2: 0, 3: 0, 4: 0, 5: 0, 6: 1, 7: 0, 8: 0, 9: 1, 10: 0, 11: 0, 12: 0, 13: 0, 14: 1, 15: 1, 16: 0, 17: 1, 18: 0, 19: 0, 20: 1}	TRUE	0
12	0.038995981216430700	12	17	{1: 0, 2: 1, 3: 1, 4: 1, 5: 0, 6: 0, 7: 0, 8: 1, 9: 1, 10: 1, 11: 1, 12: 0, 13: 0, 14: 1, 15: 1, 16: 0, 17: 1, 18: 1, 19: 1, 20: 1}	TRUE	0
13	0.07352399826049810	13	26	{1: 1, 2: 0, 3: 0, 4: 1, 5: 0, 6: 0, 7: 0, 8: 1, 9: 0, 10: 1, 11: 0, 12: 0, 13: 1, 14: 1, 15: 1, 16: 0, 17: 1, 18: 0, 19: 0, 20: 1}	TRUE	0
14	0.0625462532043457	14	30	{1: 1, 2: 0, 3: 0, 4: 1, 5: 0, 6: 0, 7: 0, 8: 1, 9: 0, 10: 1, 11: 0, 12: 0, 13: 1, 14: 1, 15: 1, 16: 0, 17: 1, 18: 0, 19: 0, 20: 1}	TRUE	0
15	0.06212615966796880	15	33	{1: 0, 2: 1, 3: 1, 4: 1, 5: 0, 6: 0, 7: 0, 8: 1, 9: 1, 10: 1, 11: 1, 12: 0, 13: 0, 14: 1, 15: 1, 16: 0, 17: 1, 18: 1, 19: 1, 20: 1}	TRUE	0
16	0.051358938217163100	16	18	{1: 0, 2: 1, 3: 1, 4: 1, 5: 0, 6: 0, 7: 0, 8: 1, 9: 1, 10: 1, 11: 1, 12: 0, 13: 0, 14: 1, 15: 1, 16: 0, 17: 1, 18: 1, 19: 1, 20: 1}	TRUE	0
17	0.21567487716674800	17	95	{1: 1, 2: 0, 3: 0, 4: 1, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0, 10: 1, 11: 0, 12: 0, 13: 1, 14: 1, 15: 1, 16: 0, 17: 1, 18: 0, 19: 0, 20: 1}	TRUE	0
18	0.3132898807525640	18	162	{1: 1, 2: 0, 3: 0, 4: 1, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0, 10: 1, 11: 0, 12: 0, 13: 1, 14: 1, 15: 1, 16: 0, 17: 1, 18: 0, 19: 0, 20: 1}	TRUE	0
19	0.05600619316101070	19	29	{1: 0, 2: 1, 3: 1, 4: 1, 5: 0, 6: 0, 7: 0, 8: 1, 9: 1, 10: 1, 11: 1, 12: 0, 13: 0, 14: 1, 15: 1, 16: 0, 17: 1, 18: 1, 19: 1, 20: 1}	TRUE	0
20	0.014393091201782200	20	4	{1: 1, 2: 0, 3: 0, 4: 0, 5: 0, 6: 1, 7: 0, 8: 0, 9: 0, 10: 0, 11: 0, 12: 0, 13: 1, 14: 1, 15: 1, 16: 0, 17: 1, 18: 0, 19: 0, 20: 1}	TRUE	0
21	0.038993120193481400	21	12	{1: 1, 2: 0, 3: 0, 4: 0, 5: 0, 6: 1, 7: 0, 8: 0, 9: 1, 10: 0, 11: 0, 12: 0, 13: 0, 14: 1, 15: 1, 16: 0, 17: 1, 18: 0, 19: 0, 20: 1}	TRUE	0
22	0.09996485710144040	22	42	{1: 0, 2: 1, 3: 1, 4: 1, 5: 0, 6: 0, 7: 0, 8: 1, 9: 1, 10: 1, 11: 1, 12: 0, 13: 0, 14: 1, 15: 1, 16: 0, 17: 1, 18: 1, 19: 1, 20: 1}	TRUE	0

As we can see the time, execution number, iterations it took , the solution resulted, the status (wether the solution was able to solve the expression or not), number of restarts it took to generate the solution will all be present as the column headers in this csv file.