CORK INSTITUTE OF TECHNOLOGY

INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Department of Computer Science

Assignment 2 – GWSAT and WALKSAT/SKC with tabu

Module Name: Metaheuristic Optimization – COMP9058

Student Name: Sreekanth Palagiri - R00184198

Table of Contents

Introduction 2
GWSAT 2
Evaluation 2
Conclusion 3
WalkSAT/SKC with Tabu 4
Evaluation 4
Conclusion 5
Run-Time Distribution Evaluation 6
Conclusion 7

Introduction

GSAT and WalkSAT are stochastic local search algorithms to solve Boolean satisfiability problems. Both the algorithms take formula in CNF format as input and try to find a solution. Both algorithms are not complete i.e. not guaranteed to find a solution. At high level they work as below:

1) Initialize a random value to each of the variables to form a solution.
2) If the solution satisfies the problem return the solution.
3) Else, flip one variable by using selection method.
4) Repeat 2-3 till as solution is reached or max. no of tries are exhausted.
5) Restart at 1 (with a new random assignment) till max no. of restarts are exhausted.

These algorithms vary in the way we select variable to be flipped. As part of the assignment we will looking in GWSAT, a variation of GSAT and WalkSAT/SKC with tabu, a variation of WalkSAT. Further sections describe and evaluate these algorithms. We evaluate on three instances: uf20-01.cnf, uf20-02.cnf and uf50-01.cnf. Below are details of the instances.

Instance	Variables#	Clauses#
uf20-01.cnf	20	91
uf20-02.cnf	20	91
uf50-01.cnf	50	281

GWSAT

In GWSAT, variable to be flipped at each try is selected by random walk step with probability wp, i.e. select a variable in the solution randomly among unsatisfied clauses or GSAT step i.e. select a variable with max net gain where net gain is $B_0 – B_1$, B_0 being current no. of unsatisfied clauses and B_1 no. of unsatisfied clauses after flipping the variable.

GWSAT takes below parameters:

Instance – File Name

Executions – No. of execution for each program run.

Restarts – No. of restarts per execution.

Iterations – No. of iterations for each restart.

wp – probability for Random Walk step.

Evaluation

In GWSAT algorithm, selecting variable is done via one of two different methods. One is GSAT method, in which we evaluate net gain for all the variables in the problem to select a variable and another is random walk component where a random variable is selected form the all the unsatisfied clauses. GSAT step's computation is depended on the number of variables times, no. of clauses and random walk step is of constant time. With increase in no. of variables and clauses, time for each step will increase proportional to no. of clauses and variables. We evaluated GWSAT on all three instances for 30 executions, 10 restarts, 1000 iterations and 0.4 wp. Below table specifies average no. of steps, computation time and no. of executions that found solution:

Instance	Average Steps per execution	Average Time (s) per execution	Average Time Per Step (ms)	Successful Executions#
uf20-01.cnf	67.47	0.05	.74	30

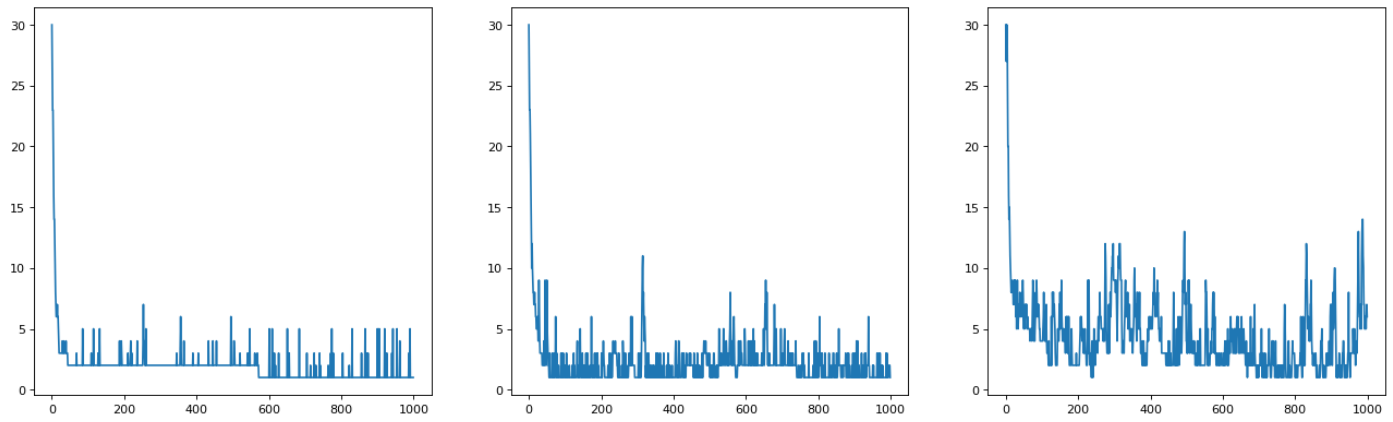
uf20-02.cnf	25.73	0.02	.77	30
uf50-01.cnf	1269.57	3.25	2.56	30

From above table we can infer that when clauses increase 3-fold and variables more than double, average step execution time also increased more than 3-fold. Computational time is in the order of N * M, where n is no. of variables and m is no. of clauses. If we consider only clauses *c* which are affected by variable v in GSAT step, then we can reduce the complexity to N * (M-*c*). Please note c varies with each iteration. With increase in number of variables, complexity will continually increase.

Parameter wp controls GSAT steps against Random walk step. With wp=0, random walk step is completely ignored and GWSAT behaves like a GSAT algorithm. With wp=1, algorithm behaves like a random search algorithm and finding a solution will be very much upon chance. It is important to setup wp such that random walk is used to break local optimum. Below table represents average no. of steps taken to find a solution for 30 executions for different wp values:

P	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
uf20-01.cnf	64.63	40.63	46	67.47	54.63	52.93	74.43	87.5	140.77
uf20-02.cnf	28.33	29.07	33.83	25.73	34.73	41.37	65.37	58.4	79.43
uf50-01.cnf	2382.43	1684.13	1510	1269.57	1066.43	1252.6	1188.47	2156.47	4864.23

From table above, we can see that lower and higher values of wp usually resulted in more no. of steps. For lower wp values, steps are more as program will be struck in local minima while for higher values steps are high because algorithm is most times randomly checking for solution. Value of wp is always best in the range of 0.3-0.5. This is demonstrated using below plot. Plot shows total no. of unsatisfied clauses at each step for wp 01.,0.4 and 0.8 for a single execution that didn't result in solution. First plot shows program struck in local optima at most times. Increase in no. of unsatisfied clauses is occurring with random walk step. For wp 0.4, we see algorithm breaking local optimum consistently. For wp 0.8, we see unstable and random nature of the algorithm.



Plot of No. of unsatisfied clauses at each step for wp 0.1,0.4 and 0.8

Parameters executions, iterations and restarts should be set according to the no. of variables and clauses. Setting lower values might not result in solution for high dimensional sat problems. These values should be high enough to allow algorithm to find a solution. Quality of initial solution will also help in quick convergence to a local optimum, we can also try to include heuristic approach to select initial solution based on problem though it is not possible for SAT problems.

Conclusion

For suitably chosen settings, GWSAT finds solution in reasonable time and doesn't get struck in local optima. However, with increase in problem size, time complexity increases substantially.

WalkSAT/SKC with Tabu

In WalkSAT/SKC, variable to be flipped at each try is selected in three steps from an unsatisfied clause:

1) Select a variable with net negative gain zero using random tie breaking.
2) With probability p, select a variable having least negative gain.
3) With probability 1-p, select a variable randomly.

We also make sure variable selected is not tabu list i.e. last few flipped variables in line with rules of tabu search.

WalkSAT/SKC takes below parameters:

Instance – File Name

Executions – No. of execution for each program run.

Restarts – No. of restarts per execution.

Iterations – No. of iterations for each restart.

wp – probability for Random Walk step.

tl – length of tabu list

Evaluation

WalkSAT/SKC selects a variable to flip in three steps as described in above section. WalkSAT unlike GWSAT has constant computation time for the variable selection step. This is because at any point of time for a 3-SAT problem GWSAT is evaluating only 3 variables. Computational complexity of WalkSAT is in the order of M, M being no. of clauses. If we consider only clauses *c* which are affected by variable v in GSAT step, then we can reduce the complexity to M-*c*. We evaluated WalkSAT/SKC with tabu on all three instances for 30 executions, 10 restarts, 1000 iterations, 0.4 wp and tabu list length 5. Below table specifies average no. of steps, computation time and execution that found solution:

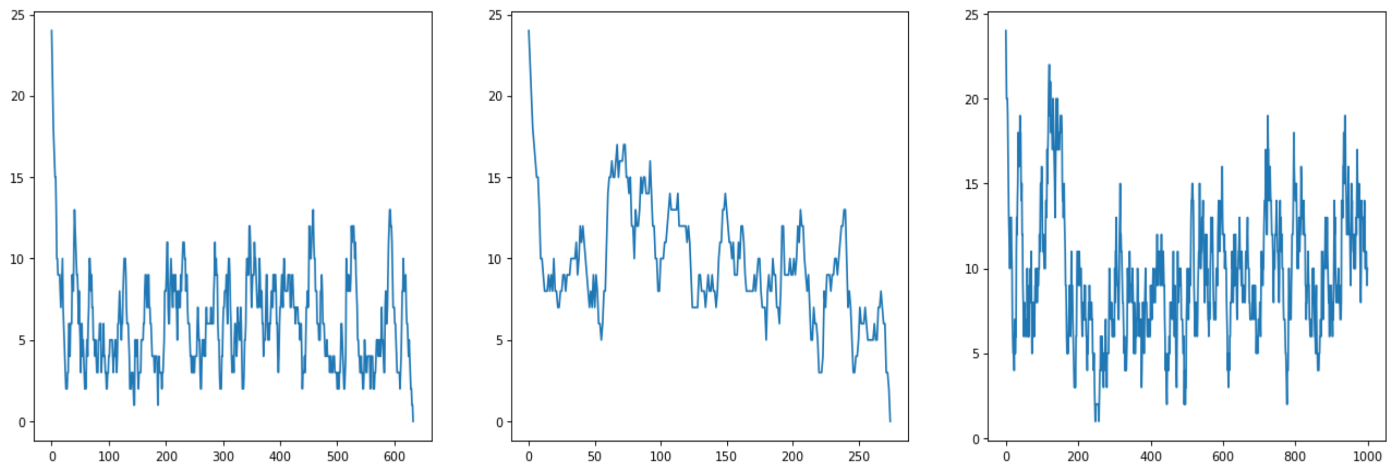
Instance	Average Steps#	Average Time (s) per execution	Average Time Per Step (ms)	Successful Executions#
uf20-01.cnf	104.5	0.09	.25	30
uf20-02.cnf	23.8	0.05	.24	30
uf50-01.cnf	716.43	0.49	.64	30

From table above we can see that for large problem instance uf50-01.cnf, each step in WalkSAT takes around one-fourth of GWSAT time. Also, increase in clauses 3-fold times has not resulted in 3-fold increase in time. WalkSAT also takes considerably less no. of steps. This is due to heuristic nature of the WalkSat step where in we are considering variables in only unsatisfied clauses for modification.

Parameter p controls whether WalkSAT selects a random variable from clause or one with minimum negative gain. Since we check this condition only when there is no net negative gain move, parameters p doesn't have same amount of importance as in GWSAT and plays less role in breaking local optima. Below table represents no. of steps for different p values:

P	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
uf20-01.cnf	65.27	71.63	46.47	104.5	81.77	116.4	122.43	57.03	102.43
uf20-02.cnf	56.3	21.1	20.97	23.8	24.53	97.23	38.77	74.17	110.9
uf50-01.cnf	368.07	353.77	681.63	716.43	682.37	742.57	1209.87	2316.67	2189.4

For P=0, we are always selecting variable with lower negative gain and for P=1, we always select a variable randomly. From above table we can infer that best range of P is range of 0.2 and 0.4. At higher p values search becomes more random and algorithm takes more steps. Below plot demonstrates the same:



Plot of No. of unsatisfied clauses at each step for wp 0.1,0.4 and 0.8

From above we can see that at p=0.8, program is more random, and solution is not found at the end of execution. At p=0.1, we see that WalkSAT unlike GWSAT is not struck in local optima for many iterations. However, still it takes considerable time to converge to solution. At p=0.4, we see program gradually converging to solution.

Parameter tl, specifies the size of tabu list. Below tables show average no. of steps for parameter tl. We have kept all other parameters at constant.

tl	2	3	5	8	10
uf50-01.cnf	465.67	472.13	716.43	1501.33	2374.17

We can see that no. of steps increase with increase in tl. This is because with bigger tabu list, we don't have enough variables to consider for the best move which means selected solution doesn't not always result in maximum gain. With smaller tabu list we circumvent this problem and mitigate repetitive selection of the same variables.

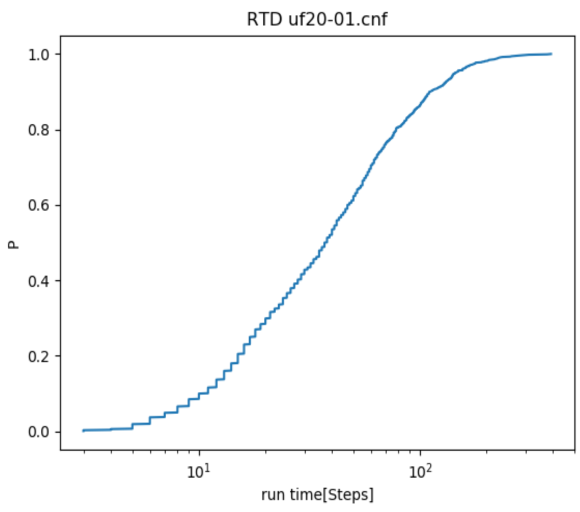
Parameters executions, iterations and restarts should be set according to the no. of variables and clauses. Setting lower values might not result in solution for high dimensional sat problems. These values should be high enough to allow algorithm to find a solution. At the same time, we should find a balance between restarts and iteration by detailed RTD analysis. Quality of initial solution will also help in quick convergence to a local optimum, we can also include one or more heuristic approach to select initial solution based on problem though it is not possible for SAT problems.

Conclusion

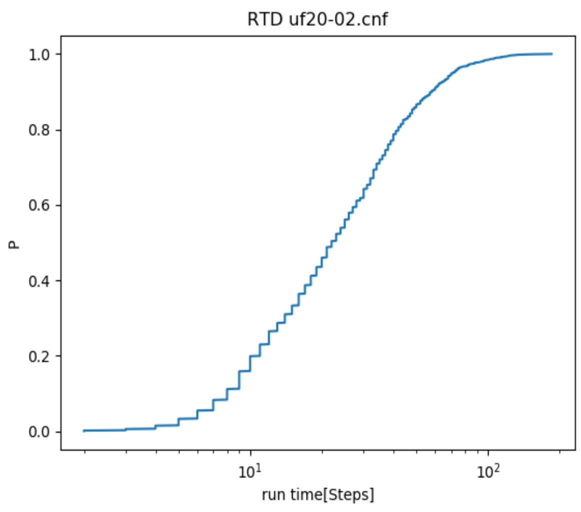
For suitably chosen settings, WalkSAT/SKC with tabu outperforms GWSAT. WalkSAT/SKC by very nature doesn't get struck in local optimum. Compared to GWSAT with increase in problem size, increase in time complexity is not substantial. At tl=3 and p=0.3, WalkSAT/SKC with tabu is very good algorithm to search SAT problems.

Run-Time Distribution Evaluation

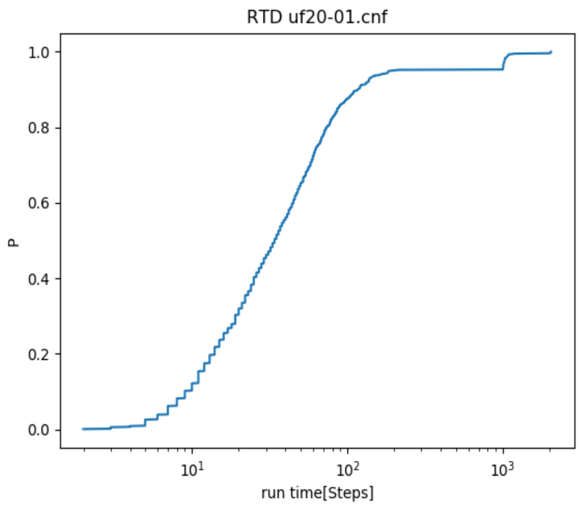
Run-Time distributions, RTD in short, is detailed analysis of an algorithm. For search problems, runtime (either steps or CPU time) to reach solution are treated as random variable. As part of this assignment we study RTDs of WalkSAT/SKC with tabu and GWSAT. We treat no. of steps to reach the solution as random variable. We plot cumulative probability graph of no. of steps on log scale. Below are RTD plots for instances uf20-01.cnf and uf20-02.cnf. First two plots are of GWSAT and third and fourth are of WalkSAT/SKC with tabu. Base line parameters used are 10 restarts, 1000 iterations and 0.4 wp for GWSAT over 1000 executions and 10 restarts, 1000 iterations, 5 tl and 0.4 wp for WalkSAT/SKC over 1000 executions.



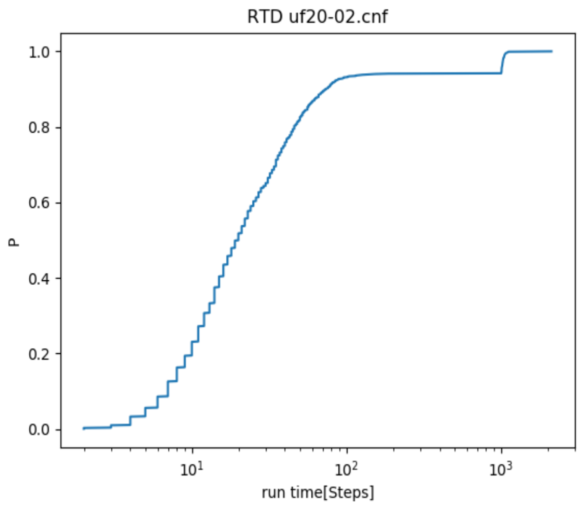
GWSAT



GWSAT

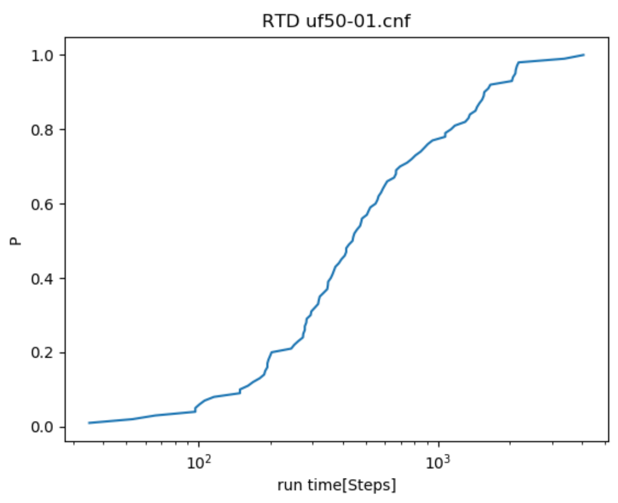


WALKSAT/SKC with Tabu

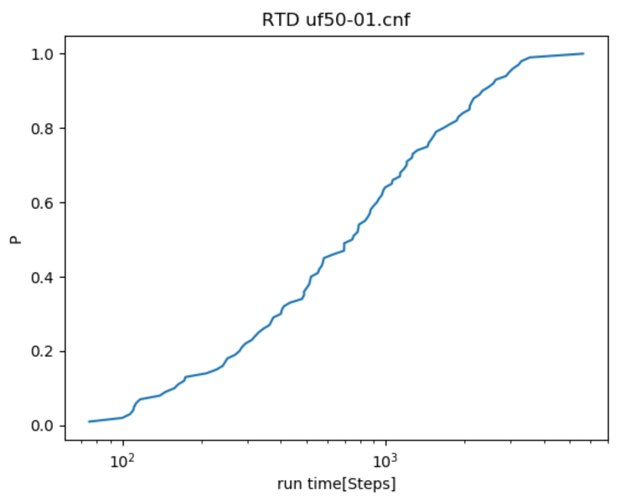


WALKSAT/SKC with Tabu

For small 3SAT problems like our data instances we see that both the algorithms perform well, we see that 90% of the times the solution is arrived with in 100 steps. However, if solution is not found with-in 100 steps WalkSAT takes more steps than GWSAT to arrive at the solution. This also can be noticed in the tables in algorithm evaluation section with avg. no of steps. This is because degree of randomness is more in WalkSAT than in GWSAT. However, same randomness acts in the favour of WalkSAT/SKC in large problems. Plot in page 7 shows comparison on instance uf50-01.cnf over 100 executions, as you can see about 80% of times WalkSAT/SKC is able to find solution within 1000 steps where as GWSAT is able to find only in 50% of executions.



WalkSAT/SKC with Tabu



GWSAT

Below table enlists descriptive statistics for instance uf50-01.cnf, below table confirms that for large data instance WalkSAT/SKC with tabu outperforms GWSAT.

	Mean	Median	ST. Dev	CV	Max	Min	q0.25; q0.1	q0.75; q0.9	q0.75/q0.25	q0.9/q0.1
WalkSAT	706.31	441.5	703.22	.996	4054	35	277; 159.8	884.5; 1568.2	3.19	9.81
GWSAT	1040.4	753.5	970.03	.932	5645	55	340.5; 162.5	1450.75; 2349.9	4.26	14.4

Conclusion

Based on RTD’s, we can confirm that WalkSAT/SKC with tabu outperforms GWSAT as complexity of the problem increases. We can similarly compare RTD of same algorithm across different parameter setting to choose best hyper parameters.