Rui Xiao
Comp160 Final Project – Satisfiability
Professor : Roni Khardon
TA : Mona Yousofshahi

0. How to run the program

1. g++ board.cpp play.cpp
2. ./a.out CNFn75m325.txt
3. options are 1, 2, 3, 4 who stands for DP, WalkSat, binary assignment(my way), and randomize, respectively.
4. To test different algorithm, terminate one and test a new one will be better because the memory leak is not cleaned up, which will affect the running time a little bit.

1. David Putnam(DP)

Intro

For each step of the assignment, the program works like this:

1. Construct a frequency table to record frequencies of all variables in ***available clauses***. A clause is an ***available clause*** if the truth value of this clause has not been determined in the current step. So an available clause must not have:
   a. Variable assigned with a true (or !variable assigned with a false)
   b. Three fixed variables
2. Take the difference of frequencies of Xi and Not Xi. For example, if there is 13 x1 and 6 !x1, then the difference is 7.
3. Find the highest value by counting sort and the variable it stands for.
4. Assign this variable as true and recursively call the assignment function.
5. Each assignment function will return a false if both of its children are false, return true otherwise. For ***leaves***, where all variables are fixed, will return true if the input is satisfied by all fixed variables, and return false if otherwise.

Heuristics I use and why I choose

I assign a variable true that best satisfies the current input by making as many true assignments as possible. The program will recursively do so until it reaches the end. If the end is not satisfied, it will change the assignment of value from the least frequent one to the most frequent one.

I choose this because I think it is a greedy solution for each step: finding the most influential variable and change it.

Data Structure I use

Define n as number of elements, m as number of clauses.

I used one 3 by (n+1) bool array to store the assignment of variables: first column is the value of xi, and second column is the value of not xi, and the third column is if xi has ever been assigned. Trivially, the First column and the second column is opposite. The third column is false by default. Example: A[1][5]=1 means x5 is assigned as true, not x5 is trivially false, and A[2][5]=1 because x5 is assigned. In this table, index is equal to the variable index for convenience.

I used one m+1 by 3 integer array to store the input (clauses): for example A[2][1]=4, A[3][2]=-4 represent: Line 3, the second variable is x4; Line 4, the third variable is !x4. In this table, index is equal to variable index minus 1.

Results from the experiments

Shown below.

More exploration

1. Binary none recursive assignment
2. Random assignment
3. Top 3 frequent assignment (assuming there are more than 3 variables, didn't code this)

Conclusions

1. Time complexity
   Let's go through the algorithm.

   For those inputs that will never get satisfied, the run time is O(2^n) (n is the number of the elements) because the algorithm will go through all combinations(leaves) and return an answer of false. For those inputs that will potentially get satisfied, the run time is hard to determine.

2. Space complexity
   For those inputs that will never get satisfied, the space used is O(2^n) because the algorithm will go through all combinations(leaves) and return an answer of false. For those inputs that will potentially get satisfied, the run time is hard to determine. All

assignment tables are copied in each step so that space complexity is O(2^n*3*n) =O(n*2^n).

3. Conclusion
   Great algorithm. It will always return the correct answer. However time is relatively slow probably because finding the best solution is space-wise and time-wise time consuming.

2. WalkSat

Intro

For each step of the assignment, the program works like this:

1. Assign all variables with random Truth and false
2. Test if the assignment satisfy the input, return true if it does.
3. Else change one of the variable by
   a. 50% choose a random one and flip.
   b. Find the xi whose frequency of xi-!xi is the most.And flip it
4. Repeat 2 and 3 for 10000 times.
5. Return false if there's no return beforehand

Heuristics I use

Same as the heuristic method I used for David Putnam but with a random variable. Choosing the most influential one to change value.

I choose this because I think it is a greedy solution for each step: finding the most influential variable and change it.

Data Structure I use

Same as the data structure I used for David Putnam.

Results from the experiments

Shown below

Conclusions

1. Time complexity
   If it returns false, the time complexity is simply O(10000) which is a constant. However if there are more than 10000 elements, the run-time could be linear to number of elements
2. Space complexity
   Space is O(3*m)+O(3*n) for the clausetable and assignment table.
3. Conclusion

It's very hard to choose the best value change when it shouldn't do random pick. My version of WalkSat doesn't work for some of the trivial cases. I might dig through it later on.

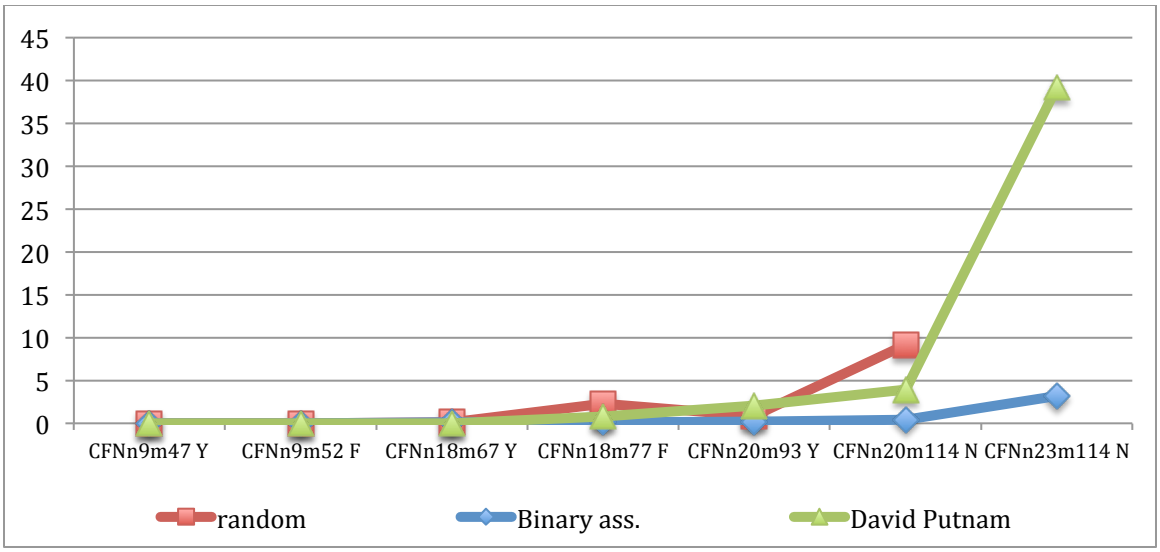
Extra work

1. Binary assignment(BA)
   Variables are assigned according to their importance in the very first input. All frequency of xi-!xi are sorted and variables are assigned according to this sequence
   Run time is O(2^n), space complexity is O(3*n)+O(3*m). All tables don't get copied. For data structure, several arrays are used.
2. Random assignment for each step
   Variables are assigned randomly in each step and thus the whole process builds a binary recursion tree.
   Run time is O(2^n) , space complexity is O(n*2^n) because all assignment tables get copied. For data structure, several arrays are used.

3. Comparison & Conclusion

Time comparison

(Second)	David Putnam	WalkSat	Binary ass.	random
CFNn9m47 Y	0	0 Wrong result	0	0
CFNn9m52 F	0	0	0	0
CFNn18m67 Y	0	0 Wrong result	.11	.12
CFNn18m77 F	.82	.02	.09	2.31
CFNn20m93 Y	2.03	.02 Wrong result	.23	.86
CFNn20m114 N	3.93	.02	.39	9.18
CFNn23m114 N	39.2	Killed	3.22	killed



A few comments:
1. According to the chart we find that David Putnam might have spend too much time and space on finding the best answer for each step and thus make itself slower. Binary assignment method performs well and does not cost too much space!

2. Binary ass. Fails when n is large, because it's strict exponential run time. (especially when the most influential variable is supposed to be false)

3. For DP and BA, their run times are all clearly O(2^n), which is shown from CFNn20m114, and CFMn23m114. Run time for both algorithms for CFN23m114 are approximately 8 times of CFNn20m114, matching that 2^(23-20)=8

4. Random method is somehow O(2^n) as well but with larger variance. I have more test data for CFNn20m93 for randomize, there are: 4.71, 4.88, .86, .87, .27 seconds, with great variation.

4. To do in the future

1. I did not clean up memory leak
2. I will fix the WalkSat to make it more usable
3. I will try to make DP, BA faster when running with larger number of variables
4. I will make DP's greedy algorithm easier