SAT Solvers

Jordan Turley

For this project, I implemented the DPLL and WalkSAT algorithms. You give the program a file in the DIMACS standard, and it will read it in and attempt to solve it, eventually telling you a solution or that a solution could not be found.

I used Java for my implementation of these algorithms. I created a SAT class which reads the data from the given DIMACS file, and then if you call solve() on that object, it will solve the problem and return a satisfying model, or throw an UnsatisfiableException, which means that a satisfying model could not be found. I created a SymbolValue class that stores a symbol and a value, which is used in the Clause class and the Model class. The Clause class stores a List of SymbolValues for the literals of that clause, and the Model class extends from ArrayList and is an ArrayList of SymbolValues. I did not use any third party libraries or packages, but I did use Java's built in ArrayList.

For DPLL, I check if all clauses are true with respect to the model, and if they are, return the model. Then check if any clause is false with respect to the model, and if at least one is, return a failure. Then, I look for any pure symbols or unit clauses, and if there is one, add it to the model. If none of this happens, I take the first symbol from the symbols variable and remove it, and pass it recursively as true or false, as going left and right on the tree. DPLL can take a long time if there are no pure symbols or unit clauses, but it will always find a solution or find that the problem is unclassifiable, with 100% accuracy.

For WalkSAT, I create a model and randomly assign a value to each symbol. Then, in a loop, check if every clause is satisfied, and if it is, return the model. If it isn't, a false clause is found, and about 50% of the time (or whatever probability you define), a random symbol in the clause is flipped in the model, or the other 50% of the time, the symbol is found that it's flip would maximize the number of satisfied clauses, and that symbol is flipped in the model. This is done until the loop has gone through the number given by maxSteps, or until a solution is found. WalkSAT is not guaranteed to find a solution, but depending on the number of steps you allow it to take, it will find one or determine that the problem is unsatisfiable. For WalkSAT to determine a problem is unsatisfiable, it must go through the max number of steps, which can take a long time.

Overall, for me, WalkSAT was much, much faster than DPLL. I think this is because with DPLL, you build the model as you go and try true or false for each variable if there are no unit clauses or pure symbols. However, with WalkSAT, you start with every variable with a value, and try randomness and greed. It is possible (extremely unlikely) that even for a very hard problem, you wouldn't even have to go through the loop once, it would just be solved. Randomness was much faster than searching through the tree. However, DPLL is guaranteed to give you the correct answer, either a satisfying model or that the problem is unsatisfiable. Depending on the number of max steps for WalkSAT, it could say that the problem is unsatisfiable when it actually is. So, for harder problems, the max number of steps must be very high.

I implemented the Strategy design pattern for choosing which algorithm to use to solve the problems. We just recently learned about this pattern in CSC 300, and I thought that this would be a good place to implement it, because we are trying to accomplish the same goal of solving the SAT problem, but with different ways. To use it, you can either use the defaults created in the SATStrategy class, or create your own objects of type DPLLAlgorithm or

WalkSATAlgorithm, and just pass it to the SAT class when creating an object. This makes it very easy to select which algorithm you use to solve the problem.

I have attached an excel document with data and graphs for my experiment. For the experiment, for DPLL I ran the solver only once because there is no randomness and the only variation would be on my computer and the load on the processor. For WalkSAT, I ran the solver on each file 10 times, with a maximum number of steps at 100,000. For problems that are any harder than this, the maximum number of steps would probably have to be higher, but for these problems, there was never any problem in solving it within that number of steps. For DPLL, except for the first problem for 20 variables and 40 variables, it always took much longer to determine the problem was unsatisfiable then to find a solution. For WalkSAT, it always took much longer to determine there was no solution. This is because it had to go through the total steps before giving up. For both DPLL and WalkSAT, it was able to solve all of the 20 variable problems in well under a second. For the 40 variable problems, WalkSAT solved them all, except for the very last one, in under a second also, but DPLL took very random amounts of time. For one, it took over 5 minutes, but then for another, it only took a couple seconds, and for all others, it took less than a minute to solve. I don't really understand this, because all of the 40 variable problems had the same number of clauses, and none of them had a big number of unit clauses or pure symbols. However, even when it took a very long time to find a solution, it still did find a solution eventually, which is what really matters.

One thing I did think about after implementing both algorithms is, what if you tried to combine the two? For example, Take the unit clause heuristic and pure symbol heuristic from DPLL, but use randomness and greed from WalkSAT. Then, when you find a unit clause or pure symbol, you could set that variable in the model, and have some property that says that it cannot be changed, so that when you randomly change a variable in WalkSAT, it would not change one of those variables.

In this assignment, I learned a lot. It was very interesting to see how a SAT solver would work. I think that these are a very powerful tool, and if you can break a problem down into a series of booleans, it makes the problem very easy to solve. Before actually implementing and testing them, I thought that DPLL would be faster and more reliable than WalkSAT, but I found that for problems that are at least a little bit hard, WalkSAT was much faster than DPLL, and still found a solution every time.