Adrian Melendez
Short Explanation of Findings.

Introduction

The goal of this project was to make a version of MiniSat in Java. MiniSat is a “minimalistic, open-source SAT solver, developed to help researchers and developers alike to get started on SAT” according to http://minisat.se/. It is a vast improvement over the naive approach because it uses learning and the DPLL algorithm to reduce the size of the space that must be searched. This paper assumes the reader has a basic understanding of satisfiability solving and formulas in CNF format.

Naive Approach

When I started working on this project I decided to make my own Sat solver using a naive approach to use as a comparison against better methods. Here are some code snippets that show how my original approach worked:

```
public int solveSat() {
    for (int x = 0; x < combNum; x++) {
        for (int y = 0; y < claNum; y++) {
            if (!clauseList.get(y).testClause(x)) {
                break;
            }

            if (y == claNum - 1) {
                return x;
            }
        }
    }

    return -1;

}
```

```
public boolean testClause(int x) {
    for (int i : ors) {
        if (i < 0) {
            if (MyUtils.getBit(x, (Math.abs(i) -1)) == 0) {
                return true;
            }
        } else {
            if (MyUtils.getBit(x, (i - 1)) == 1) {
                return true;
            }
        }
    }

    return false;
}
```

In the above code, combNum is equal to 2^n where n is the number of variables and uses the binary representation of the number to set which variables are true and false. The variable claNum is the number of clauses in the formula. The code tests each combination of truth values over every clause in the formula and returns -1 if the formula isn't satisfiable or an integer representing the first found answer if it is. This is obviously bad and slow because it iterates over every possible combination of truth values without skipping over any irrelevant or incorrect answers.

DPLL Approach

The DPLL algorithm is a vast improvement over the naive approach. These improvements are made by unit propagation and pure literal elimination, which I explained in my mid-term presentation slides as seen below:

Unit propagation - If a clause is made of a single variable, this clause can only be True if the correct value is assigned to that variable. Therefore, there is no other option for all other instances of that variable.

Example:
(not a) AND (a OR not b) AND (not a OR c) AND (b OR c)
(not b) AND (b OR c)
(c)
TRUE

Pure Literal Elimination - If a variable only appears with one polarity (i.e. either the plain variable or the negation of the variable) there is only one option for the assignment of that variable and all clauses with it can be made true.

Example:
(not a OR not b) AND (not a OR c) AND (not a OR b)
TRUE

After using the above rules to prune the search space to something reasonable, we can assign truth values to the variables which are still relevant to the problem. Making a variable true removes that clause from the formula and when all the clauses are removed, we know we have solved the problem. Making a variable false removes that variable from the clause and when a clause has no variables left, we know that clause can't be made true under the current assignments so we must backtrack and try again. The above rules will be applied each time an assignment is made to see if the formula was made false and to see if the search space can be reduced further.

Instructions For Running the Program
For ease of use, I put the whole program into a single Java file. It can be opened in your IDE of choice and run without changing anything or setting anything up. The program uses System.in as the input so after starting the program, paste a CNF formated formula into the place in your IDE that handles System.in and System.out. You may need to hit enter after pasting in your formula to make sure the last line ends and can be read by the program.

Here are some example formulas that are known to work correctly:
http://people.sc.fsu.edu/~jburkardt%20/data/cnf/aim-50-1_6-yes1-4.cnf
http://people.sc.fsu.edu/~jburkardt%20/data/cnf/hole6.cnf

Conclusion
My second program was a vast improvement over the first. The first program can't handle formulas with more than 31 variables. Although it could easily be changed to use Longs instead of Integers, the running time is so slow that it isn't measurable in a human lifetime. My version of MiniSat can easily handle formulas of 50 or more variables and finds answers in a matter of seconds.The improvement is truly astounding.