354 Project

Tian Luo
tianl1
15-354

Algorithm Design:

Here is my algorithm:

1. Find the smallest clause. Pick a literal and satisfy it.
2. Recursively solve SAT. If it returns true, return true. If it returns false, then set the literal to false and then pick a new literal that has no assignment and satisfy it. Repeat this step until the recursive call returns true or if all literals have an assignment to it (in which case return false).

Notice that this algorithm automatically handles unit-propagation (as specified by the DPLL algorithm) by consideration the smallest clause first; thus, a unit clause will always be considered first.

Before the algorithm runs, the algorithm removes any duplicate variables in the CNF clauses, i.e. $(x \vee x)$ is instead encoded as (x). Clauses that are tautologies are automatically considered satisfied and not included to improve runtime.

The idea behind the algorithm comes from an algorithm I learned in 15-859 that solves 3-SAT in $O(1.83^n)$ time. If the input to my program is a 3-SAT instance, then the recurrence would be $T(n) \leq T(n-1) + T(n-2) + T(n-3) \in O(1.83^n)$. However, for the general SAT case, I don't really have a good bound for my algorithm other than $O(2^n \cdot poly(n, m))$ where n is the number of variables and m is the number of clauses. The $poly(n, m)$ term is actually $O(n + m)$.