Author Eric Vance
Comp 610

Evolution of project:

To start the project I began with brute force. This method worked but became infeasible once the input grew.

The next iteration was using a greedy algorithm. This algorithm counted the number of clauses both negated and not negated. It would choose the variable with the highest count and assign that variable the truth statement to make the clause it belongs to true. Those clauses would be removed and the processes would be repeated.

To improve the result of the algorithm I then added a brute force mod that would help refine the answer. This strategy did not end up being feasible with the way the brute force was written. I was not able to improve the algorithm.

The algorithm I ended up with is Local Search. I came up with a few different variants:

1. Local Search Constant Bit
   Solution: Will be represented as a string of bits
   Closeness: A neighbor is close if they differ by 1 bit. ie. 0000 1000 0100 0010 0001 are all neighbors. Only trues will be flipped followed by false.
   Movement: Calculate all neighbors. A move will be made if there is a solution that is at least as good as the current one. (in other terms if the neighbor increases or is equal to the total clauses satisfied move to that neighbor.
   Stop Condition: If no moves were made, ie all neighbors satisfy less clauses. The algorithm will terminate
2. *Local Search One Flip Bit*
   Solution: Will be represented as a string of bits
   Closeness: A neighbor is close if they differ by 1 bit. ie. 0000 1000 0100 0010 0001 are all neighbors. True or false will be flipped.
   Movement: Calculate all neighbors. A move will be made if there is a solution that is at least as good the current one. (in other terms if the neighbor increases or is equal to the total clauses satisfied move to that neighbor.
   Stop Condition: If no moves were made, ie all neighbors satisfy less clauses. The algorithm will terminate
3. *Local Search Flip Bit Random Start*
   Solution: Will be represented as a string of bits
   Closeness: A neighbor is close if they differ by 1 bit. ie. 0000 1000 0100 0010 0001 are all neighbors. True or false will be flipped. Two rounds with a random start.
   Movement: Calculate all neighbors. A move will be made if there is a solution that is at least as good as the current one. (in other terms if the neighbor increases or is equal to the total clauses satisfied move to that neighbor.

Stop Condition: If no moves were made, ie all neighbors satisfy less clauses. The algorithm will terminate

4. *Local Search Flip Bit Random Start Constant Bit*
   Solution: Will be represented as a string of bits
   Closeness: A neighbor is close if they differ by 1 bit. ie. 0000 1000 0100 0010 0001 are all neighbors. True or false will be flipped. Two rounds with a random start.
   Movement: Calculate all neighbors. A move will be made if there is a solution that is at least as good as the current one. (in other terms if the neighbor increases or is equal to the total clauses satisfied move to that neighbor.
   Stop Condition: If no moves were made, ie all neighbors satisfy less clauses. The algorithm will terminate
5. *Local Search Random Solution*
   Solution: Will be represented as a string of bits
   Closeness: A neighbor is random.
   Movement: Randomize each round. A round is the length of possible solutions. Move will occur as soon as a random solution increases the value.
   Stop Condition: If no moves were made after trying len(solution), The algorithm will terminate

**NOTE: I added a clause where the stop condition occurred the solutions kept moving to a neighbor that satisfied an equal number of clauses. The only way to continue would be to find a solution during the iteration of neighbors that would increase the number of trues.

Interesting notes:

I found that using a random truth assignment significantly sped up the answer. The Local search with Random Solution string would run extremely fast. However, this caused the answer to not be as accurate. By combining the flipping of bits for each neighbor and randomization I was able to increase the accuracy of getting the most truths.

Finally the randomization does find a very accurate solution, however, it does not do it every time. I decided to run three different algorithms, one flip bit, random start one flip bit, and random start constant bit flip. I did this because all algorithms performed well but they did not always give the most accurate answer. The three run concurrently and the largest solution is chosen.

To account for variables that were not there, I mapped all of the variables that were discovered during the file read. If the variable was not in the set, I skipped that iteration of flipping or processing. To further account for missing variables, I started any processing or flipping at the smallest variable that was read into the set.

I attempted to introduce the branch and bound technique. I was unable to finish it. This would have significantly improved my solution, but I ran out of time because I was not familiar enough with coding it.