COS 402 – Program 2 Report

The SAT-solver is an extension of the WalkSat algorithm discussed in class and in the course book. The algorithm first generates a random model for the given CNF. Additionally, it checks to see if any of the clauses contain one literal; if so, make the clause true. While the timer has not run out, get an ArrayList<Integer> that holds the indexes of unsatisfied clauses with the current model. If the ArrayList is empty (all clauses are satisfied) then return the model, otherwise select a random unsatisfied clause. With some probability p, flip a random symbol in the clause, otherwise find the symbol which increases a heuristic and flip it. The heuristic is calculated by taking the clauses that become satisfied as a result from a symbol flip subtracted by the clauses that become unsatisfied from the flip. This means we prefer symbol flips that increase the number of unsatisfied clauses. If we find that no clauses became unsatisfied and some became satisfied from a symbol flip, immediately flip the symbol and go through the iteration again (since it was strictly better to flip this symbol). In the cases where the heuristic is equal to the current best, choose the one which had a lower break value (this means we prefer flips that make less clauses become unsatisfiable). If it happens that no symbols that increased the number of satisfiable clauses exist (the heuristic for all symbols in the clause is negative), the algorithm has found a local min, to help counter this, the next iteration of the loop will flip a random bit from an unsatisfied clause.

This algorithm is similar to WalkSat, but differs in that it will initially satisfy one-symbol clauses and will immediately flip a symbol that improves the model if it finds one. Additionally, it uses a heuristic that considers ties when there a multiple symbols that improve the number of satisfiable clauses.

The generator is based on the dinner party planning problem. The problem consists of seating a number of guests at a large table such that pairs of guests that have indicated they must sit together are seated next to each other and guests pairs that have indicated they refuse to sit next to each other must not be seated next to each other. For this problem, the assumptions are that the table is a long rectangular one (the host and hostess are seated at the end) and therefore the guests are seated on either side of the table. Additionally, the number of guests is equal to the number of seats (this assumption can be removed by introducing a "dummy" guest of which no constraints are put on it) and guests sitting across from each other, even at the end of the table, are not considered sitting next to each other.

The problem is reduced to CNF by imagining two sets of connected nodes for each side of the table (e.g. P1-P2-P3-P4 and P5-P6-P7-P8). Given there are n chairs there will be n2 symbols; P11 means person 1 in chair 1, P12 means person 2 in chair 1, etc. First, each seat must have one person sitting in it (P11 v P12 v P13…, where P11 is person 1 sitting in chair 1). This is one clause for each chair. Second, each seat can only have one person sitting in it ((!P11 v !P12) ^ (!P11 v !P13) ^…); each chair has a set of clauses that is all unique pairings of guests. Next we need to clauses that ensure that pairs that have indicated they want/refuse to sit together do/do not. Let's say person 4 and person 18 have indicated they want to sit together. Then for each pair of seats next to each other (1-2, 2-3, 3-4, etc), P14 ⇔P218, where this means person 4 sitting in seat 1 mutually implies person 18 sitting in seat 2. This is reduced to CNF by (!P14 v P218) ^ (!P118 v P24); we do this for each pairs of seats. Let's say instead, person 4 and 18 have indicated that they refuse to sit next to each other. Then P14 ⇔!P218 which reduced to (!P14 v !P218) ^ (!P118 v !P24) for each pairs of seats.

In order to generate random dinner-planning problems we first start with a max number of possible seats. For memory/time reasons I've selected 50 to be the maximum number of guests for this problem and 4 to be the minimum number. The generator selected a random number between 4 and 50 and takes this to be the table size and number of guests. To ensure that only satisfiable problems are generated we first generate a solution by creating a table and assigning a person to a random chair. Next we randomly select the number of complementary pairs and randomly select the number of opposing pairs. These numbers are in the range of 0 to the maximum number of such pairs in a given problem. Once these numbers are selected the pairs are chosen systematically based on the model to make the problem most interesting. As such, unique pairs will be selected first (1-2, 3-4), before crossover occurs (1-2, 2-3) for complementary pairs. This is the same for opposing pairs (1-3, 2-4, etc would be selected first before crossover occurs such as 1-3, 1-4).

To add the clauses that encode complementary/opposing pairs the algorithm first generates a model then from that model selects pairs and puts them in their respective ArrayList<Pair>. Once the pairs have been selected, for each pair add its clauses to the CNF. At the end return the CNF.