1. **Game tree search (15 pts.)** Consider a game where each move is dependent on the roll of a single die. Let’s consider how one would extend standard minimax game search to account for this. More specifically, consider how we evaluate a given node in the tree. Let us say that at some point in the game tree player 'max' has three possible move choices. Now depending on the roll of the die following the move, each move will be scored differently according to the table below.

For example, if 'max' makes move #1 and then a 2 is rolled 'min' will be able to make a move resulting in a board with score -0.1. Similarly, if 'max' makes move #3 and 'min' rolls a 6, then the best that 'min' could do results in a board with a score of 0.3. (A +1 scored state is an immediate win for Max, and a -1 scored state is an immediate loss for Max.)

<table>
<thead>
<tr>
<th></th>
<th>Move</th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
</tr>
</thead>
<tbody>
<tr>
<td>die</td>
<td></td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>1</td>
<td>0.4</td>
<td>-0.1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-0.1</td>
<td>0.1</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-0.1</td>
<td>-0.1</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-0.1</td>
<td>-0.1</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>-0.1</td>
<td>0.1</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>-0.1</td>
<td>0.1</td>
<td>0.3</td>
<td></td>
</tr>
</tbody>
</table>

(a) (10 pts.) Draw the portion of the search tree representing this point in the game. (Your tree should have the 3 possible moves by max, followed by a level with possible outcomes of the die.)

Assuming max wants to make the move with the highest possible *expected* value, which move is that?

(b) (5 pts.) What is a possible drawback of using expected value? Suggest an alternative. (Consider the “risk” of a move.)

2. **Local Search (30 points total)**

(a) i. (2 pts.) Briefly describe the process of local search.

ii. (4 pts.) Describe the main difference(s) between simulated annealing and hill climbing.

iii. (4 pts.) Considering simulated annealing, explain the effect of having the starting temperature (a) too high or (b) too low.
(b) (10 pts.) Consider the state space in Figure 1, with 4 states, $s_1$, $s_2$, $s_3$, and $s_4$. The edges indicate possible transitions a simulated annealing (SA) procedure can make. With each node, we give the value of a function, $E(s)$, which is to be minimized.

We will move around the state space following the standard SA process. That is, when in state $s$, SA considers a randomly selected neighbor $s'$. If $E(s') \leq E(s)$, then SA moves to state $s'$. If $E(s') > E(s)$, then with probability $e^{-(E(s') - E(s))/T}$, SA makes moves to $s'$, else SA stays in place. ($T$ is a parameter $> 0$.)

Starting at a random state, after a long series of moves (sufficient for SA to come arbitrarily close to the stationary distribution), let $P(s)$ be the probability of finding SA in state $s$.

i. If $T \rightarrow \infty$, describe how SA moves around the space. What is $P(s)$ for $s \in \{s_1, s_2, s_3, s_4\}$?

ii. If $T = 1$, is it the case that $P(s_1) > P(s_2)$? What are these probabilities? (Show calculation.)

(c) (10 pts.) Consider a problem ONES, which maximizes the number of ones in a bit-string of length $n$. We use a genetic algorithm with a bit-string representation and evaluate the individuals in the population by computing the sum of bits set to one.

i. (5 pts.) Given the two individuals $I_1 = 01010$ and $I_2 = 10101$ and a cross-over at the second position, i.e., $I_1 = 01|010$ and $I_2 = 10|101$, what would be the subsequent offspring?

ii. (5 pts.) For $n = 8$, the global optimal is 11111111. The corresponding fitness value is 8. What is the probability of attaining a globally optimal individual in the offspring, when applying the a one-point cross-over operator to the following individuals $I_3$ and $I_4$? (Assume cross-over point is selected uniformly at random.)

- $I_3$: 01111111
- $I_4$: 10111111

Figure 1: State space with function $E(s)$ to be minimized.
3. Propositional Encodings — SAT — Problem (25 pts. total) Consider the following rules and definitions for a sports league scheduling problem:

- N (even) teams, and every two teams play each other exactly once during season.
- The season lasts (N-1) weeks.
- Every team plays one game in each week of the season.
- There are N/2 periods or slots per week; every slot is scheduled for one game.

For example, a valid schedule for 8 teams named 0,1,2,3,4,5,6,7 would be given by filling in the 4 slots for each week as follows:

<table>
<thead>
<tr>
<th></th>
<th>Slot1</th>
<th>Slot2</th>
<th>Slot3</th>
<th>Slot4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Week1</td>
<td>0 vs 1</td>
<td>2 vs 3</td>
<td>4 vs 5</td>
<td>6 vs 7</td>
</tr>
<tr>
<td>Week2</td>
<td>0 vs 2</td>
<td>1 vs 7</td>
<td>3 vs 5</td>
<td>4 vs 6</td>
</tr>
<tr>
<td>Week3</td>
<td>4 vs 7</td>
<td>0 vs 3</td>
<td>1 vs 6</td>
<td>2 vs 5</td>
</tr>
<tr>
<td>Week4</td>
<td>3 vs 6</td>
<td>5 vs 7</td>
<td>0 vs 4</td>
<td>1 vs 2</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Encode the Sports League Scheduling problem as a Boolean satisfiability problem. Hints:
- In order to model that two different teams play each other in a given slot, divide each slot in two subslots. For each week, we have N subslots. Adopt the convention that two teams that play consecutive sublots — an odd numbered subslot followed by an even subslot — in fact play each other.
- Variable $X_{ijk}$ is assigned True iff team $i$ plays in subslot $j$ in week $k$
- Variable $Y_{ijk}$ is assigned True iff team $i$ plays team $j$ in week $k$

(a) (5 pts.) Give the clauses that state that exactly one team plays in each subslot. How many clauses are there? (Note: big-O notation is fine.)
(b) (5 pts.) Give the clauses that state that a team plays at most once a week. How many clauses are there? Given these clauses and the previous set, does this imply that each team plays exactly once per week? Explain.
(c) (5 pts.) Give the clauses that relate the $X$ variables with the $Y$ variables.
(d) (5 pts.) Give the clauses that state that each pair of teams plays each other exactly once during the season. How many clauses are there?
(e) (5 pts.) Of the above sets of clauses, which ones are redundant, if any? Explain.

4. SAT — Erdös Discrepancy Problem (40 pts. total) Consider a sequence $S_n$ of length $n$ of +1s and −1s, for example +1, −1, −1, +1, +1, +1 for a sequence of length 6. In general, we have a sequence $x_1, x_2, x_3, \ldots x_n$ with $x_i \in \{+1, -1\}$. We will consider the process of summing up the elements of certain subsequences. The subsequences are based on so-called arithmetic progressions. An arithmetic progression is a sequence of numbers in which each differs from the preceding by a constant quantity (e.g., 3, 6, 9, 12, etc.; 5, 10, 15, 20, etc.). Or, in terms of our $S_n$, we would have $x_3, x_6, x_9, x_{12}$ (for stepsize 3) and $x_5, x_{10}, x_{15}, x_{20}$ (for steps of 5). Let’s denote a subsequence of $S_n$ with stepsize $l$ by $S^l_n$. 
Erdős proposed to consider the sum of the terms of such subsequences. So we have
\[
\text{sum}(S_n^l) = \sum_{i \in \{1, 2, \ldots\}} x_{il} \quad 1 \leq il \leq n
\]

For our example sequence, we get \(\text{sum}(S_{6}^1) = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 = 2\), \(\text{sum}(S_{6}^2) = x_2 + x_4 + x_6 = 1\), and \(\text{sum}(S_{6}^3) = x_3 + x_6 = 0\).

Over 80 years ago, Erdős, one of the most prominent mathematicians of the 20th century, conjectured that given any fixed value \(l\) for a sufficiently long initial sequence the sum of at least one of the subsequences will rise above \(l\) or below \(-l\) during the summation process, no matter what initial sequence you construct. Informally, the “discrepancy” (from zero) of the sequence grows above \(l\). Using exhaustive search, one can show that even for relatively short sequences, the discrepancy cannot be kept at 1. However, somewhat incredibly, the conjecture for the case \(l = 2\) was unknown up to 2014. That is, it was unknown whether or not one could construct an arbitrarily long sequence of +1s and -1s for which none of the sums of the subsequences would grow above 2 or below -2 at any point. The problem was that any direct search of sequences up to a certain length always found some sequence with discrepancy bounded by 2, but direct search cannot check very long sequences.

It took a SAT solver in 2014 to show that there exists a sequence of length 1160 with discrepancy bounded by 2. However, the SAT solver was also able to show that no such sequence exists of length 1161. That is, all sequences of 1161 +1s and -1s will have a subsequence with a sum that rises above +2 or below -2 at some point. (Convince yourself that that means that any sequence longer than 1161 will also have a discrepancy larger than 2.)

We will encode the discrepancy problem for \(l = 2\) as a SAT instance and run some experiments. Our encoding will be such that for a given value of \(n\), if the encoding has satisfying assignment then there exists a sequence of length \(n\) stays within the discrepancy bound of 2; otherwise, the set clauses will be unsatisfiable, which means that there is no sequence of length \(n\) that stays within the discrepancy bound of 2.

Use Boolean variables \(X_i\) with \(i = 1, 2, 3, \ldots n\) to represent the original sequence \(S_n\) of length \(n\). Let \(X_i\) equal to True iff \(x_i = +1\). We need auxiliary variables, \(Y_{t,j,l}\), to keep track of the sums of subsequences. Specifically, we want \(Y_{t,j,l}\) equal to True iff the sum of the first \(j\) terms of the subsequence with stepsize \(l\) is equal to \(t\), where \(-2 \leq t \leq 2\).

Let \(Y_{0,l}\) represent the sum of subsequence with stepsize \(l\) before we start summing up values of the sequence. This sum equals 0, for all subsequences. So, \(Y_{0,l}\) should be set to True for all stepsizes \(l\) and \(Y_{0,l}\) should be set to False for \(u \in \{-2, -1, +1, +2\}\).

We now want to define clauses to get the correct behavior for the \(Y\) variables given a setting of the \(X\) values. To do this we want to step one by one through each subsequence and make sure that the partial sum of the terms in a subsequence (a) represents the correct partial sum as given by the values of \(X_i\), and (b) stays within the range of -2 to +2.

(a) (10 pts.) Give the implication that represents “If the partial sum of the first \(j\) terms of stepsize \(l\) subsequence is \(t\) (assume \(t < 2\)) and \(X_{(j+1)/l} = +1\), then the sum of the \(j + 1\) terms equals \(t + 1\).”

Also, give the clausal form of the implication.
(b) (10 pts.) Give the logical statement that captures the reverse direction: if the value of sum of the first \( j + 1 \) terms of the stepsize \( l \) subsequence equals \( t \) (\( t > -2 \)) and \( X_{(j+1)l} = +1 \), then the sum of the first \( j \) terms equals \( t - 1 \).

Also, give the clausal form of the implication.

(c) (10 pts.) Give the clauses corresponding to the items a) and b) for the case that \( X_{(j+1)l} = -1 \).

(d) (5 pts.) We now want to make sure that our discrepancy stays within the bounds +2 and -2. That is, we want our incremental sums of subsequences to stay within the bounds. Specifically, consider what happens if a subsum equals 2 (or -2), as given by the \( Y \) variables. What does that imply about the next and the previous \( X \) value in the subsequence? Give clauses that capture the right behavior.

(e) (2 pts.) Now give clauses that define the initial \( Y \) values. That is, the clauses that give the correct \( Y_t^0 \) settings.

(f) (3 pts.) Make sure you you have indicated the correct ranges for various indices used in the clauses above. With each set of clauses state how many there are. Big Oh notation is fine.

5. EXTRA CREDIT OPTION: Erdős Discrepancy Problem (15 pts. extra credit option)

See description of Erdos discrepancy problem in previous question.

Write code (in your favorite language) that generates the clauses in the DIMACS CNF format. See http://www.satcompetition.org/2009/format-benchmarks2009.html.

You need to number the variables from 1 to the maximum number of variables you are using.

(Actually, the file header gives the highest occurring variable number, \( V_{max} \). It’s ok to have some numbers in the range 1 through \( V_{max} \) that do not occur in any clause. In other words, variables don’t need to be numbered in strictly consecutive order.)

You should use the numbers 1 through \( n \) to correspond to the variables \( X_1, X_2, ... X_n \), i.e., the variables representing the original sequence.

Your program should take as input the length \( n \) of the original sequence of +1/-1s. It should generate a set of clauses, as you developed above, that encode the constraints on the +1/-1s sequence, \( x_1 \) through \( x_n \), such that a satisfying assignment to the clauses will correspond to a valid sequence of +1/-1s that has discrepancy at most 2.

What stepsizes for your subsequences should you consider? You start with \( l = 1 \) (the original sequence) and encode up to a maximum value stepsize of \( l = \lceil n/3 \rceil \). So, for \( n = 100 \), you should encode subsequences with stepsize up to 33. All in one set of clauses. (Why? Each partial sum starts at 0. So, the first way to get into potential trouble is after three +1s or three -1s. In other words, a subsequence of 2 or fewer steps can never create a problem.)

After generating a file with the clauses in DIMACS CNF format. (Small trick: Your program can generate the clauses and header line, but you can add the clause count to the header by hand after doing a linecount on the file.)

First, generate an encoding for a relatively small \( n \), e.g., 10. Then use the SAT solver Lingeling or the multi-threaded version Plingeling (see http://fmv.jku.at/lingeling/) to solve the instance. The solver will print out a satisfying assignment if it exist. (Format example:
+9 means variable 9 is set to True; -9 means the variable is set to False.) Now, check that you have a valid sequence! (It may be helpful to write some code that checks that all partial sums of all subsequences stay within bounds of +2 and -2 during the summation process.)

You can try \( n = 1160 \), to get the longest sequence possible. When you try \( n = 1161 \), Plingeling will tell you UNSAT. Both cases take about 5,000 seconds on a MacBook Pro. You have now solved that first non-trivial case of the Erdos discrepancy conjecture!

To demonstrate that you have everything working, generate the clauses for the case \( n = 400 \). Now do the following: add to your encoding the following eleven unit clauses: 1, 20, 40, 50, 70, 90, 115, 130, 150, 170, and 200.

E.g. for the first two unit clauses, you add the lines
\[
1 \ 0 \\
20 \ 0
\]
where the first clause forces \( X_1 \) to True and the second clause forces \( X_{20} \) to True. This means that our original sequence now needs to start with a +1 and also has a +1 in the 20th position, etc. Don’t forget to increment the clause number in the header.

Now, find a satisfying assignment of this set of clauses, using Plingeling, and look at the satisfying assignment.

(a) What are the truth values of \( X_{100} \), \( X_{210} \), \( X_{220} \), \( X_{240} \), \( X_{250} \), and \( X_{330} \)?

(b) Why were you asked to first add the unit clauses to your original clause set? Given the resulting set of clauses, what is special about the variables \( X_{100} \), \( X_{210} \), \( X_{220} \) etc.? Use Lingeling to confirm your answer.

(c) Finally, submit your code (source) to Gradescope. Include a README.txt explaining how to compile and run. In README.txt, include output from Lingeling obtained for answering part (a). (Bundle files in .zip.)