Assignment 2

Classical and Local Search, Adversarial Search
Constraint Satisfaction Problems and Logic

Deadline: March 10, 11:59pm.
Perfect score: 100 points.

Assignment Instructions:

Teams: Assignments should be completed by pairs of students. No additional credit will be given for students working individually. You are strongly encouraged to form a team of two students. This can be the same team as in previous assignments or a different team. If you have not done so already, please inform the TAs as soon as possible about the members of your team so they can update the scoring spreadsheet (find the TAs' contact info under the course's website: `http://www.pracsyslab.org/cs440`).

Submission Rules: Submit your reports electronically as a PDF document through Sakai (`sakai.rutgers.edu`). Do not submit Word documents, raw text, or hard-copies etc. Make sure to generate and submit a PDF instead. Each team of students should submit only a single copy of their solutions and indicate all team members on their submission. Failure to follow these rules will result in lower grade in the assignment.

No demos will take place for this assignment. Just submit a PDF of your final report.

Late Submissions: No late submission is allowed. 0 points for late assignments.

Extra Credit for LaTeX: You will receive 10% extra credit points if you submit your answers as a typeset PDF (using LaTeX). Resources on how to use LaTeX are available on the course's website. There will be a 5% bonus for electronically prepared answers (e.g., on MS Word, etc.) that are not typeset. If you want to submit a handwritten report, scan it and submit a PDF via Sakai. We will not accept hard-copies. If you choose to submit handwritten answers and we are not able to read them, you will not be awarded any points for the part of the solution that is unreadable.

Precision: Try to be precise. Have in mind that you are trying to convince a very skeptical reader (and computer scientists are the worst kind...) that your answers are correct.

Collusion, Plagiarism, etc.: Each team must prepare its solutions independently from other teams, i.e., without using common notes, code or worksheets with other students or trying to solve problems in collaboration with other teams. You must indicate any external sources you have used in the preparation of your solution. Do not plagiarize online sources and in general make sure you do not violate any of the academic standards of the course, the department or the university (the standards are available through the course's website: `http://www.pracsyslab.org/cs440`). Failure to follow these rules may result in failure in the course.

Problem 1: [25 points] Given a Boolean formula, the satisfiability problem (SAT) is to determine whether or not there is an assignment of truth values to its variables that makes the formula true according to the rules of Boolean algebra. SAT is a prototypical NP-complete problem with applications in hardware and software verification, planning, scheduling, and circuit synthesis.

Research [1] into the average case complexity of random Boolean formulas in a normal form called 3-CNF, has revealed that the median running time for complete SAT solvers peaks at density 4.26 (the density of a 3-CNF formula with l clauses over n variables is $\frac{l}{n}$). See the corresponding graph in Figure 1. It is believed that some of the hardest SAT problems occur at this density where the probability that a randomly generated formula is satisfiable is exactly one half.

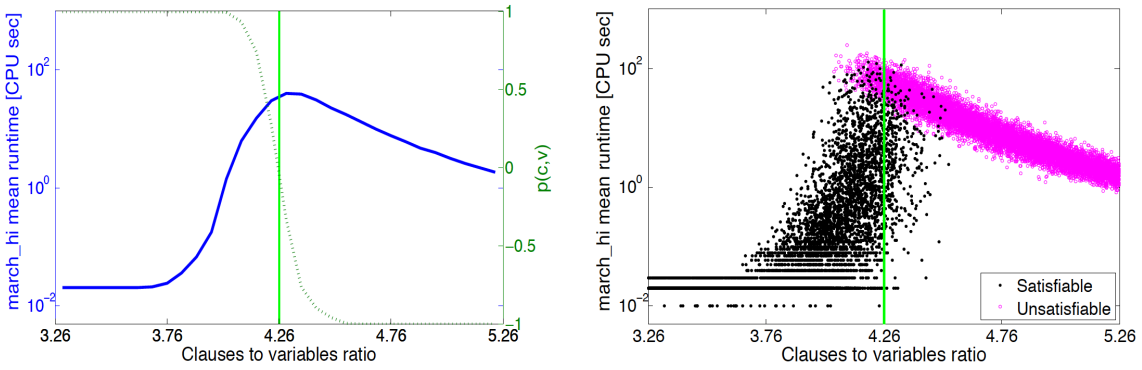

Figure 1: (left) In blue: The Runtime of a SAT solver on uniform-random 3-SAT instances with 400 variables and a changing ratio $\frac{l}{n}$, i.e., number of clauses per variables. The dotted line corresponds to the probability of an instance to be satisfiable. At density 4.26, the probability is 50%. (right) The runtime to solve a problem per instance, colored by satisfiability status.

Your task is to code a genetic algorithm variance so as to solve SAT problems. This means that you will code a local search algorithm that generate off-spring states through genetic operators and improves them subsequently by means of local search. Each individual state corresponds to an assignment of true or false values to the variables. Here, a variation to the vanilla genetic algorithms is described for your implementation. Employ a population size of 10 and the following steps during each iteration:

1. **Elitism:** The best two individuals in the population according to the evaluation function are propagated directly to the next population (they still participate as candidates in the selection process but not in the mutation process). The evaluation function corresponds to the number of clauses satisfied given the assignment.

2. **Probabilistic selection:** Based on the evaluation function the individuals are selected for reproduction (this is the same with the probabilistic selection with normalization process described in the class). Note that due to the elitism you are going to select 8 states out of all the 10 available in the previous population (the elit still participates in reproduction).

3. **Uniform crossover:** This is a different version of the crossover operation. Given parents x and y, the i-th bit of the offspring is from x with probability 0.5 and from y with probability 0.5. Apply the crossover operation only to the 8 states, which resulted from the probabilistic selection process and propagate the elit forward intact.

4. **Disruptive mutation:** A string is mutated with probability 0.9. If a string is to be mutated then the mutation operator flips each bit with probability 0.5. Apply the mutation operation only to the 8 states that are the result of the crossover operation.

5. **Flip heuristic:** After performing crossover and mutation the algorithm scans the bits of the assignment in random order. Each bit is flipped, and the flip is accepted if the gain

(the number of clauses that are satisfied after the flip minus the number that are satisfied before the flip) is greater than or equal to zero. When all the bits in an assignment have been considered and if the process improves the assignment's fitness, the flipping process is repeated for that assignment until no additional improvement can be achieved. Do not apply the flip heuristic on the elit states.

This algorithm uses standard genetic operators to make broad changes in an assignment and the flipping process to locally optimize assignments. Implement the method in a language of your choice. As input for your algorithm use satisfiable SAT instances with density 4.26 available from: `http://people.cs.ubc.ca/~hoos/SATLIB/benchm.html`

Use the instances in the first bulleted list titled, uniform random-3-sat. Solve 100 satisfiable instances with 20, 50, 75, and 100 variables. The files are in .tar.gz format, so you will have to uncompress and then unpack them. The SAT problems are in a standard `CNF` format (see `http://people.cs.ubc.ca/~hoos/SATLIB/Benchmarks/SAT/satformat.ps` for details).

Plot the success rate of your algorithm as a function of the number of variables *n* in the formula. Success rate is the percentage of the benchmark problems for which solutions are found by the solver. Recall that all the benchmark problems are satisfiable. Computational efficiency is measured in two ways: running time (which is machine and code dependent), and the average number of flips in finding a solution (which is machine and code independent). Plot the median running time and the average bit flips as a function of *n*. Explain the performance results.

Problem 2: [15 points] Two players, the MAX and the MIN are playing a game where there are only two possible actions, "left" and "right". The search tree for the game is shown in Figure 2. The leaf/terminals nodes, which are all at depth 4, contain the evaluation function at the corresponding state of the game. The MAX player is trying to maximize this evaluation function, while the MIN player is trying to minimize this evaluation function.

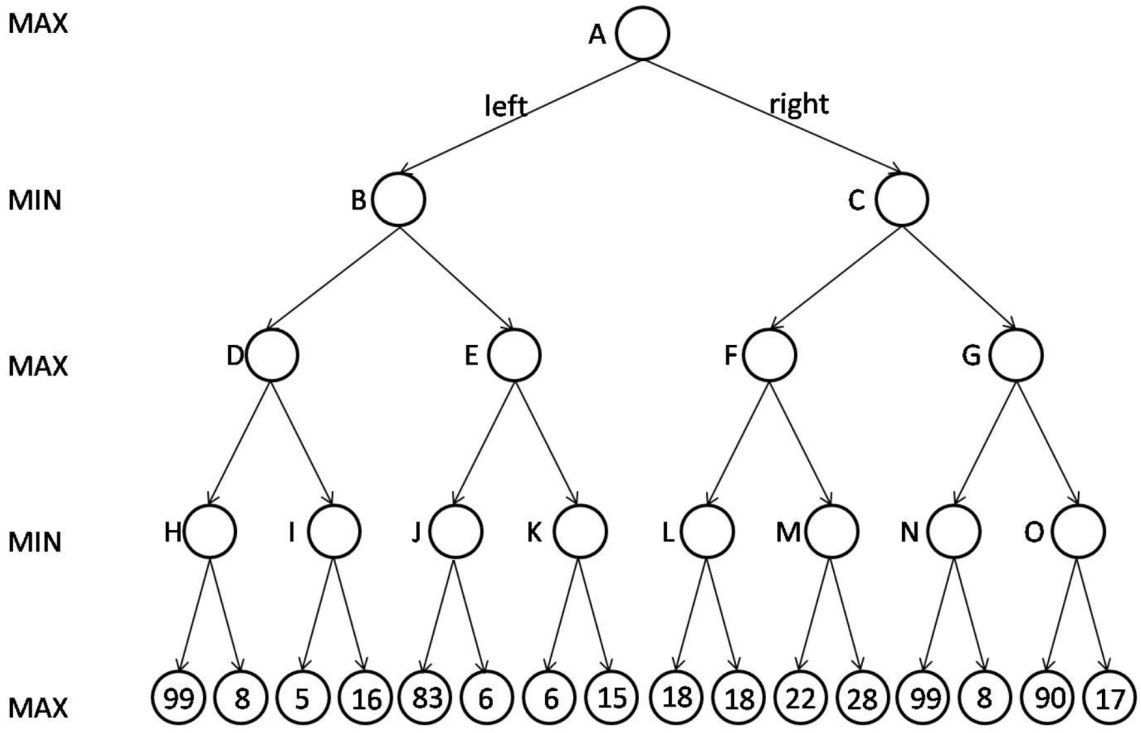

Figure 2: The adversarial search tree.

a. Consider the operation of the Minimax algorithm over the above search tree and indicate the value of each intermediate node.

b. Consider the operation of the Minimax algorithm with alpha-beta pruning over the above search tree. Indicate which nodes will not be consider by this algorithm (you can reuse the figure and mark them visibly or mention which intermediate and terminal nodes are not visited in text). Furthermore, indicate the alpha and beta values on each intermediate node that is visited.

c. What action will the MAX player choose at the root state according to the exhaustive Minimax algorithm? What action will the MAX player choose at the root state according to the Minimax algorithm employing alpha-beta pruning? In general, is the best move computed by the two versions of the algorithm guaranteed to be the same or not?

d. Consider a version of the algorithm that heuristically evaluates the quality of nodes at depth 2 so as to guide the ordering of nodes considered during the alpha-beta pruning.

   In particular, assume that the heuristic value at the depth 2 nodes are: $h(D) = 9$, $h(E) = 7$, $h(F) = 24$ and $h(G) = 16$, and Minimax backs-up these values at nodes B and C as well.

   Now, the MAX player executes the alpha-beta pruning algorithm down to depth 4. This time, however, it uses the heuristic values at nodes B through G to reorder the nodes of the depth-4 game tree at depths 1 and 2. The objective is to maximize the number of nodes that will not be examined by the alpha-beta pruning process.

   How will the alpha-beta pruning algorithm reorder the nodes, given the information generated up to depth 2? Show the corresponding new tree considered by the alpha-beta pruning algorithm using the labels of the nodes from the above figure. In addition, inside each leaf node give the value of the evaluation function.

   How many nodes will not be examined by the alpha-beta pruning algorithm in this case?

e. Consider now a variation of the game, where the opponent is known to play randomly, i.e., instead of trying to minimize the evaluation function, it selects with 50% the left action and with 50% the right action. Provide the value of each node on the original tree in this case. What will be the choice of the MAX player in this case at the root state?

   Can you apply alpha-beta pruning in this case? If yes, show how. If not, explain why.

Problem 3: [15 points] Consider the game Sudoku, where we try to fill a 9×9 grid of squares with numbers subject to some constraints: every row must contain all of the digits $1, \ldots, 9$, every column must contain all of the digits $1, \ldots, 9$, and each of the 9 different 3×3 boxes must also contain all of the digits $1, \ldots, 9$. In addition, some of the boxes are filled with numbers already, indicating that the solution to the problem must contain those assignments. Here is a sample board:
Each game is guaranteed to have a single solution. That is, there is only one assignment to the empty squares which satisfies all the constraints. For the purposes of this homework, let's use $n_{i,j}$ to refer to the number in row i, column j of the grid. Also, assume that M of the numbers have been specified in the starting problem, where $M = 29$ for the problem shown above.

	1		4	2				5
		2		7	1		3	9
							4	
2		7	1					6
				4				
6					7	4		3
	7							
1	2		7	3		5		
3				8	2		7	

Figure 3: Sample Sudoku board

a. This is an instance of a Constraint Satisfaction Problem. What is the set of variables, and what is the domain of possible values for each? How do the constrains look like?

b. One way to approach the problem, is through an incremental formulation approach and apply backtracking search. Formalize this problem using an incremental formulation. What are the start state, successor function, goal test, and path cost function?

Which heuristic for backtracking search would you expect to work better for this problem, the degree heuristic, or the minimum remaining values heuristic and why?

What is the branching factor, solution depth, and maximum depth of the search space? What is the size of the state space?

c. What, is the difference between “easy” and “hard” Sudoku problems? [Hint: There are heuristics which for easy problems will allow to quickly walk right to the solution with almost no backtracking.]

d. Another technique that might work well in solving the Sudoku game is local search. Please design a local search algorithm that is likely to solve Sudoku quickly, and write it in pseudocode. You may want to look at the WalkSAT algorithm for inspiration. Do you think it will work better or worse than the best incremental search algorithm on easy problems? On hard problems? Why?

Problem 4: [10 points] Consider the following sequence of statements, which relate to Batman’s perception of Superman as a potential threat to humanity and his decision to fight against him.

For Superman to be defeated, it has to be that he is facing an opponent alone and his opponent is carrying Kryptonite. Acquiring Kryptonite, however, means that Batman has to coordinate with Lex Luthor and acquire it from him. If, however, Batman coordinates with Lex Luthor, this upsets Wonder Woman, who will intervene and fight on the side of Superman.

a. Convert the above statements into a knowledge base using the symbols of propositional logic.

b. Transform your knowledge base into 3-CNF.

c. Using your knowledge base, prove that Batman cannot defeat Superman *through an application of the resolution inference rule* (this is the required methodology for the proof).

Problem 5: [25 points] Consider the Traveling Salesman Problem (TSP), which corresponds to the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?" It is one of the known NP-hard problems in Computer Science.

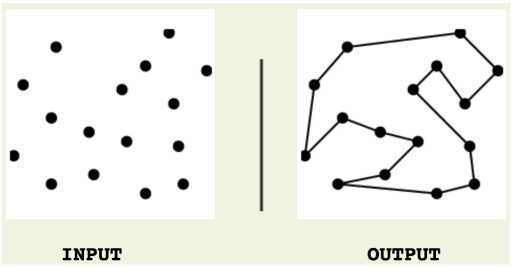

Figure 4: An example solution for a TSP.

- Generate a process for specifying a TSP challenge with a text file by employing the following format: The problem instances that are in the files are in the following format:
  <number of cities>
  <city id> <X> <Y>
  <city id> <X> <Y>
  .
  .
  .
  <city id> <X> <Y>
  Generate 25 files for each of the following number of cities: 10, 25, 50, 100 (i.e., 100 files total). For each city, select its coordinates randomly in the range $[0, 100]$.

- First try to approach the problem using heuristic search. Write a program that uses A^* to solve the Traveling Salesman Problem (TSP). Let the cost to move between cities be Euclidean distance, i.e., straight-line distance.

  Propose and use a good admissible heuristic h function, and describe it in detail in your report. Think carefully about how to construct the h function since some choices of heuristics may not help you to execute the search in practice.

  Always start your TSP route from the first city in the text file.

  Apply your algorithm to the generated input files and use a time limit of 10 minutes for each challenge. Then, report: a) number of problems solved within the time threshold, b) average solution time, c) number of nodes generated, and d) average solution quality, as the size of challenge increases (i.e., average for the 10-city challenges, then for the 25-city ones, etc.)

- Write a program that solves the TSP using simulated annealing. Explain what local search operators you use so as to make sure that a route that starts from the first city ends in the same city and goes through all other cities.

  Experiment with different annealing schedules, and explain what schedule you choose to use.

  Similar to A*, apply your algorithm to the generated input files and use a time limit of 10 minutes for each challenge. Then, report: a) number of problems solved within the time threshold, b) average solution time, c) number of nodes generated, and d) average solution quality, as the size of challenge increases (i.e., average for the 10-city challenges, then for the 25-city ones, etc.)

  Plot how the cost of the solution changes during example runs of the algorithm on some of the 25-size examples.

- Discuss the results you got with A* and with simulated annealing.

  Furthermore, discuss for what types of problems is simulated annealing a useful technique. What assumptions about the shape of the value function are implicit in the design of simulated annealing?

Problem 6: [10 points] Assume that you have two heuristic functions for a specific problem h_1, h_2. Provide formal arguments to answer the following questions.

a. If you use the *minimum* value of $h_1(n)$ and $h_2(n)$ at every state, will the resulting heuristic be admissible? Is it a consistent one?

b. If you use the *maximum* value of $h_1(n)$ and $h_2(n)$ at every state, will the resulting heuristic be admissible? Is it a consistent one?

c. If you define the heuristic function $h_3(n) = w \cdot h_1(n) + (1 - w) \cdot h_2(n)$, where $0 \leq w \leq 1$, will this be admissible? Will it be consistent?

d. Consider an informed, best-first search algorithm in which the objective function is $f(n) = (2 - w) \cdot g(n) + w \cdot h(n)$. For what values of w is this algorithm guaranteed to be optimal when h is admissible (for the case of TREE-SEARCH)? What kind of search does the resulting algorithm perform when $w = 0$? When $w = 1$? When $w = 2$?

References

[1] K. Leyton-Brown, H. H. Hoos, F. Hutter, and L. Xu. Understanding the Empirical Hardness of NP-Complete Problems. *Communications of the ACM*, 57(5):98–107, 2014.