# Assignment 2

Local and Classical Search - Adversarial Search -
Constraint Satisfaction Problems - Logic-based Inference and Satisfiability

Deadline: October 17, 11:59pm.
Available points for undergraduate students: 120. Perfect score: 100.
Available points for graduate students: 110. Perfect score: 100.

**Assignment Instructions:**

**Teams:** Assignments should be completed by teams of students - three members for undergraduates and two for graduate students. Mixed teams between undergraduates and graduates are discouraged. If formed, they will be graded as graduate teams. No additional credit will be given for students that complete an assignment with fewer members than the recommended. **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`). For programming questions, you need to also submit a compressed file via Sakai, which contains your code. 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.

**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, in which case you should also submit electronically your source code). 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:** Answer the following questions on informed search and heuristics.

a) Which of the following are admissible, given admissible heuristics $h_1$, $h_2$? Which of the following are consistent, given consistent heuristics $h_1$, $h_2$?

- $h(n) = min\{h_1(n), h_2(n)\}$ [4 points]
- $h(n) = wh_1(n) + (1-w)h_2(n)$, where $0 \leq w \leq 1$ [4 points]
- $h(n) = max\{h_1(n), h_2(n)\}$ [4 points]

b) Consider an informed, best-first search algorithm in which the objective function is $f(n) = (2-w)g(n) + wh(n)$. For what values of $w$ is this algorithm guaranteed to be optimal when $h$ is admissible (consider the case of TREE-SEARCH)? What kind of search does this perform when $w = 0$? When $w = 1$? When $w = 2$? [8 points]

Make sure you provide formal proofs to the above questions.

(20 points total)

**Problem 2:** Simulated annealing is an extension of hill climbing, which uses randomness to avoid getting stuck in local maxima and plateaux.

a) For what types of problems will hill climbing work better than simulated annealing? In other words, when is the random part of simulated annealing not necessary?

b) For what types of problems will randomly guessing the state work just as well as simulated annealing? In other words, when is the hill-climbing part of simulated annealing not necessary?

c) Reasoning from your answers to parts (b) and (c) above, 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?

d) As defined in your textbook, simulated annealing returns the current state when the end of the annealing schedule is reached and if the annealing schedule is slow enough. Given that we know the value (measure of goodness) of each state we visit, is there anything smarter we could do?

(e) Simulated annealing requires a very small amount of memory, just enough to store two states: the current state and the proposed next state. Suppose we had enough memory to hold two million states. Propose a modification to simulated annealing that makes productive use of the additional memory. In particular, suggest something that will likely perform better than just running simulated annealing a million times consecutively with random restarts. [Note: There are multiple correct answers here.]

(10 points)

**Problem 3:** *Approximately Optimal Search.* The two objectives of finding a solution as quickly as possible during informed search and finding an optimal solution are often conflicting. In some problems, one may design two heuristic functions $h_A$ and $h_N$, such that $h_A$ is admissible and $h_N$ is not admissible, with $h_N$ resulting in much faster search most of the time. Then, one may try to take advantage of both functions.

(a) A best-first search algorithm called $A^*_\epsilon$ uses the evaluation function $f(N) = g(N) + h_A(N)$. During each iteration, $A^*_\epsilon$ expands a node $N$ such that $f(N) \leq (1+\epsilon) \times min_{N \in FRINGE}\{f(N)\}$, where $\epsilon$ is any strictly positive number. Formally argue about the cost of the solution returned by $A^*_\epsilon$. [10 points for graduates - 15 points for undergraduates]

(b) Explain briefly how $A^*_\epsilon$ can use the second heuristic function $h_N$ to reduce the time of the search. What tradeoff is being made in choosing $\epsilon$? [5 points]

(15 points for graduates - 20 points for undergraduates)

**Problem 4**: Consider the two-player game described in Figure 1.

a. Draw the complete game tree, using the following conventions: [3/4 points]

- Write each state as $(s_A, s_B)$ where $s_A$ and $s_B$ denote the token locations.
- Put each terminal state in a square box and write its game value in a circle.
- Put *loop states* (states that already appear on the path to the root) in double square boxes. Since it is not clear how to assign values to loop states, annotate each with a "?" in a circle.

b. Now mark each node with its backed-up minimax value (also in circle). Explain how you handled the "?" values and why. [3/4 points]

c. Explain why the standard minimax algorithm would fail on this game tree and briefly sketch how you might fix it, drawing on your answer to (b). Does your modified algorithm give optimal decisions for all games with loops? [3/4 points]

d. This 4-square game can be generalized to $n$ squares for any $n > 2$. Prove that A wins if $n$ is even and loses if $n$ is odd. [6/8 points]

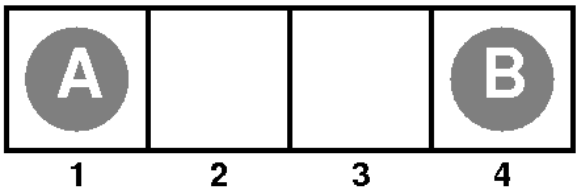


Figure 1: The start position of a simple game. Player A moves first. The two players take turns moving, and each player must move his token to an open adjacent space in *either direction*. If the opponent occupies an adjacent space, than a player may jump over the opponent to the next open space if any. (For example, if A is on 3 and B is on 2, then A may move back to 1.) The game ends when one player reaches the opposite end of the board. If player A reaches space 4 first, then the value of the game to A is +1; if player B reaches space 1 first, then the value of the game to A is -1.

(15 points for graduates - 20 points for undergraduates)

**Problem 5:** Consider the problem of constructing (not solving) crossword puzzles: fitting words into a rectangular grid. The grid, which is given as part of the problem, specifies which squares are blank and which are shaded. Assume that a list of words (i.e., a dictionary) is provided and that the task is to fill in the blank squares using any subset of the list. Formulate the problem in two ways: [Hint: There might be multiple correct answers here.].

a) As a general search problem. Choose an appropriate algorithm, and specify a heuristic function, if you think is needed. Is it better to fill in blanks one letter or one word at a time?

b) As a constraint satisfaction problem. Should the variables be words or letters?

Which formulation do you think will be better? Why?

(10 points)

**Problem 6**: Consider the following sequence of statements, which describe the operation of a security system.

*If the system is armed and motion is detected, then the alarm will sound. If the alarm sounds, then the system has been armed or there has been a fire. Regardless of whether the system is armed, the alarm should go off when there is a fire. Motion is constantly detected.*

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

b. Using the resolution rule and your knowledge base, derive the following theorem: *The alarm will sound if and only if the system is armed or there is a fire.*

(10 points)

**Problem 7**: 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.

Recent research 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 $\frac{l}{n}$). 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.

Your task is to code a genetic algorithm variance, called the FlipGA genetic algorithm to solve SAT problems. FlipGA is an evolutionary local search algorithm that generates off-springs by genetic operators and improves them subsequently by means of local search. FlipGA employs 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.

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.

FlipGA uses standard genetic operators to make broad changes in an assignment and the flipping process to locally optimize assignments. Implement FlipGA 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.

(30 points)