CS 182 FALL 2022, PROBLEM SET 2

Due: October 11, 2022 11:59pm

This problem set covers Lectures 5 through 9. The topics include multi robot systems, constraint satisfaction problems, convex optimization, integer programming, and game theory.

1. *Constraint Satisfaction Problem Warmup.* (10 points) You are in charge of scheduling classes. There are 5 courses that meet on these days and 3 instructors who will be teaching these classes. Each instructor is only able to teach one class at a time. The class times are as follows:

- Class A runs from 9:00am-10:00am.
- Class B runs from 9:30am-10:30am.
- Class C runs from 10:00am-11:00am.
- Class D runs from 10:00am-11:00am.
- Class E runs from 10:30am-11:30am.

Each instructor is capable of teaching some subset of the course:

- Instructor X is able to teach classes C and D.
- Instructor Y is able to teach classes B, C, D, E
- Instructor Z is able to teach classes A, B, C, D, and E.

Here are the questions:

(1) In order to assign each class an instructor, formulate the problem as a CSP. What are the variables, domains, and constraints?

(2) Draw the constraint graph of the CSP that you just formulated.

(3) What is the new CSP after enforcing arc consistency? Why is this new CSP easier to solve? Provide a solution to the new arc-consistent CSP, and verify that it is also a solution to the CSP in (a).

2. *Sudoku Backtracking Search Algorithm.* (25 points)

(1) (4 points) Describe how a standard 9x9 Sudoku game board can be thought of as a constraint satisfaction problem (CSP)? What are the decision variables? What are the domains of those decision variables? What are the constraints that must be satisfied? Are the constraints unary or binary? How does backtracking search differ from simple brute-force guess-and-check? (**Note:** Sudoku was discussed in lecture, and you can read more about the game and its rules here.)

(2) (8 points) Complete the method for implementing forward-checking in `pset2.py` by filling in the `fwd_checking()` method and its helper method `get_related_coords()`. Forward-checking should enforce arc-consistency and look for instances of foreseeable incompatibility.

(3) (8 points) Complete the function for implementing the minimum remaining value (MRV) heuristic in `pset2.py` by filling in `MRV_heuristic()` in `pset2.py`. This will be used in the select-unassigned-variable heuristic method abbreviated as `SUV_heuristic()` that selects the next unassigned decision variable to be assigned during backtracking search. (See the instructions and hints detailed in `pset2.py` for more information.)

(4) (5 points) How does the average amount of backtracking search required to find a solution change as a function of the number of values provided on the starting board? You can assume a valid solution exists with the values provided. Consider the case where there is just 1 blank square and when the entire board is blank. Explain your answer. (High-level intuition is fine, no need for a formal proof)

Hint: Does having more decision variables make the solution easier or harder to find? It can be shown that a Sudoku puzzle must have at least 17 filled in values for the solution to be unique. Does the existence of multiple solutions make the solution easier or harder to find?

(5) *Bonus problem.* (5 bonus points) Complete the `ODV_heuristic_bonus()` function for a chance to win 5 bonus points on this problem set. Your function should select a value from the list of candidates to assign to the decision variable identified and return `None` if there are no candidates to choose from. Selecting a value that is more likely to lead to a solution will result in an overall faster average runtime across many test cases. The top 5 submissions by total number of iterations across a set of hidden test cases will earn 5 bonus points on this problem set. Your heuristic has access to the `SudokuGameBoard` data structure, but you may not solve the board or perform searching within your heuristic. Your heuristic should be intuitive to understand, please include a description and justification for what you chose to use in your write up to be eligible for bonus points.

3. *Convex Optimization.* (25 Points)

(1) *Convex Functions.* (10 Points) Are the following functions are convex? Provide proof.

(a) (3 Points) $f : \mathbb{R}^n \to \mathbb{R}$, $f(\mathbf{x}) = \max_i x_i$. (For example, in $\mathbb{R}^2$, $f((x, y)) = \max(x, y)$.)

(b) (3 Points) $f : \mathbb{R}^2_+ \to \mathbb{R}$, $f((x, y)) = x/y$, where $\mathbb{R}^2_+$ denotes the set $\{(x, y) \in \mathbb{R}^2 | y > 0\}$.

(c) (4 Points) $f : \mathbb{R} \to \mathbb{R}$. Suppose $f_1, \ldots, f_n$ are all convex functions $\mathbb{R} \to \mathbb{R}$, and let $f(x) = \max\{f_1(x), \ldots, f_n(x)\}$.

(2) *Convex Sets.* (15 Points)

(a) (5 Points) Let $C \subseteq \mathbb{R}^n$ be a convex set, with $x_1, \ldots, x_k \in C$, and let $\theta_1, \ldots, \theta_k \in \mathbb{R}$ satisfy $\theta_i \geq 0$, $\theta_1 + \cdots + \theta_k = 1$. Show that $\theta_1 x_1 + \cdots + \theta_k x_k \in C$ for all $k \geq 1$. **Hint:** When is this definition equivalent to the definition of convexity? Try induction.

(b) (10 Points) Define the convex hull of a set of points S in $\mathbb{R}^n$ as the set of all convex combinations of points in S:

$$CH(S) = \{\theta_1 x_1 + \cdots + \theta_k x_k | x_i \in S, \theta_i \geq 0, i = 1, \ldots, k, \sum_{i=1}^{k} \theta_i = 1\}$$

(i) (5 Points) Show that $CH(S)$ is indeed a convex set.

(ii) (5 Points) Show that the convex hull of a set S is the intersection of all convex sets that contain S.

4. *Game Theory.* (20 points)

(1) (10 points) The following payoff matrix shows a game between Paul and Fiona. There are three cards on a table: a red card, a green card, and a blue card. Each player picks one card, and both players pick their card without knowing the other player's choice. Both players also know the entire payoff matrix.

	Paul: red	Paul: green	Paul: blue
Fiona: red	F = 7, P = 1	F = 9, P = 2	F = 6, P = 3
Fiona: green	F = 8, P = 10	F = 7, P = 5	F = 7, P = 1
Fiona: blue	F = 7, P = 2	F = 8, P = 2	F = 8, P = -3

Find a pure Nash equilibrium for this game, and then find a mixed-strategy Nash equilibrium where both players have strictly positive probability of playing each action.

(2) (10 points) Recall from lecture the ice cream wars setting. Suppose that there are k ice cream sellers on a beach which can be modeled as the interval $[0, 1]$ and that each ice cream seller chooses a real number within that interval where they will set up their cart to sell ice cream. The ice cream being sold from all carts is identical, i.e. customers are indifferent about which cart's ice cream they buy and will always buy from the seller closest to them. Customers are uniformly distributed along the beach i.e. the interval $[0, 1]$ (assume distance only matters to customers in the horizontal direction along the beach's length, the width of the beach is negligible).

(a) (5 points) Let $k = 3$, is there an arrangement of locations that the 3 ice cream sellers can choose that will result in a pure strategy Nash equilibrium? If so, describe the locations and why. If not, explain why.

(b) (5 points) Let $k = 4$, is there an arrangement of locations that the 4 ice cream sellers can choose that will result in a pure strategy Nash equilibrium? If so, describe the locations and why. If not, explain why.

5. *Integer Programming.* (15 points)

(1) *Sudoku IP.* (6 points) As shown in lecture, Sudoku puzzles can also be formulated as integer programs rather than CSPs. Follow the instructions and template in the starter code of `pset2.py` and use the `cvxpy` Python package to solve this problem via integer programming. The Gradescope autograder will verify the correctness of your formulation. Compare the the efficiency of the backtracking-based and IP-based solutions. Report which one is faster experimentally based on the set of test cases provided and report the approximate runtimes for each to complete the test cases.
Hint: Refer back to the lecture slides on integer programming for details of how one can formulate a Sudoku puzzle as an integer program.

(2) *Minimum Makespan Problem.* (10 points) At a factory there are N tasks to complete, each of which takes a potentially different amount of processing time denoted by $p_1, p_2, \ldots, p_N$. Each task is worked on by only 1 machine and each machine can only work on 1 task at a time. There are K identical machines in the factory that work equally efficiently on each of the tasks. We wish to find an assignment of the N tasks to the K machines such that the overall processing time to complete all N tasks (i.e. the makespan) is minimized. Mathematically, let $X_1, \ldots, X_K$ denote the collection of tasks assigned to each machine, then our objective is to minimize the max over machines j of the sum of p_i over jobs i assigned to j:

$$\min_{X_1,\ldots,X_K} \max_j \sum_{i \in X_j} p_i$$

(a) (5 points) Formulate this problem as an integer program.
Hint: Refer back to the maximin share problem in the lecture slides on integer programming for a related example.

(b) (5 points) Follow the instructions in the starter code and use `cvxpy` to code a solution framework for the minimum makespan problem. Provide a lower bound for the objective function based on K and the input vector of processing times for each task, justify your answer.

6. Collaboration, Calibration and References (5 points).

(1) With whom did you work on this problem set? What (if any) references and/or resources did you use beyond the course lecture slides and textbook?

(2) (5 points) Approximately how long did it take you to complete this problem set? Please complete this brief survey worth 5 points, graded for completion.