15-281 Archive — Practice Midterm 1A

INSTRUCTIONS

- Exam length: 80 minutes
- You are permitted to have one handwritten page of notes, double-sided
- No calculators or other electronic devices allowed

Name	
Andrew ID	

For staff use only

Q1. Heuristics	/ 35
Q2. Adversarial Search	/ 20
Q3. CSPs	/ 25
Q4. Local Search	/ 10
Q5. Linear Prog.	/ 10
Total	/100

Q1. [35 pts] Heuristics

For parts a, b, c below, consider now the CornersProblem from programming assignment 1: there is a food pellet located at each corner, and Pacman must navigate the maze to find each one. The step cost is one for each action.

(a) For each of the following heuristics, say whether or not it is admissible. If a heuristic is inadmissible, give a concrete counterexample (i.e., draw a maze configuration in which the value of the heuristic exceeds the true cost).

(i) [3 pts] h_1 is the maze distance to the nearest food pellet (if no food pellets remain, $h_1 = 0$).

● Admissible ○ Inadmissible

Eating pellets requires reaching them, which means moving at least this far.

(ii) [3 pts] h_2 is the number of uneaten food pellets remaining.

● Admissible ○ Inadmissible

Eating pellets requires reaching them all, which means at least this many moves.

(iii) [3 pts] $h_3 = h_1 + h_2$

○ Admissible ● Inadmissible

Doesn't quite work because if the first pellet is one step away this heuristic counts 2. Counterexample: In a 2x2 board with pellets in 3 corners, $h_3 = 4$ but $h^* = 3$.

(iv) [3 pts] $h_4 = |h_1 - h_2|$

● Admissible ○ Inadmissible

Since $h_2 \geq 0$, this is never bigger than h_5.

(v) [3 pts] $h_5 = \max\{h_1, h_2\}$

● Admissible ○ Inadmissible

Since both h_1 and h_2 are lower bounds on h^*, their maximum is also.

(b) [3 pts] Pick one heuristic from part (a) that you said was inadmissible; call it h_k. Give the smallest constant $\epsilon > 0$ such that $h' = h_k - \epsilon$ is an admissible heuristic. Briefly justify your answer.

ϵ:

Justification:
$\epsilon = 1$ works. It must be at least 1, to fix the counterexample given above. And 1 works, because the total cost must be at least the cost to get to any pellet and the cost of eating the remaining $n - 1$ pellets.

(c) [5 pts] We will say that for two heuristics h_a and h_b, $h_a \leq h_b$ if for all possible states x, $h_a(x) \leq h_b(x)$. In this case, we say h_b dominates h_a. Fill in the boxes below with all of the heuristics that you said were admissible in part (a) so that all heuristics are to the right of any heuristics they dominate. If two heuristics share no dominance relationship, put them in the same box. You may not need to use all the boxes.

□ ≤ □ ≤ □ ≤ □ ≤ □

h_1, h_2, and h_4 belong in the leftmost box because they are all incomparable (for each pair, there is some state where one has a higher value than the other and vice versa; h_5 in the next box because it dominates all three.

(d) [3 pts] Let h_i and h_j be two admissible heuristics and let $h_\alpha = \alpha h_i + (1-\alpha)h_j$. Give the range of values for α for which h_α is guaranteed to be admissible.

Min value of range: 0	**Max value of range:** 1

[0,1] works; this heuristic is dominated by h_5 and is therefore admissible.

(e) Consider an arbitrary search problem in a graph with start state S and goal state G. Let h^* be the "perfect heuristic," so $h^*(x)$ is the optimal distance from x to the goal G. Let ϵ be some positive number.

For each of the heuristics h_A, h_B, and h_C, give expressions for the range of possible values for the total cost of a path from S to G that A* tree search could return when using that heuristic. You may write your answers in terms of $h^*(S)$, and ϵ.

Let h_0 be an arbitrary admissible heuristic.

(i) [3 pts] h_A: $h_A = h_0$ for all states

Min value of range: $h^*(S)$	**Max value of range:** $h^*(S)$

$[h^*(S), h^*(S)]$; This is an admissible heuristic, so it will return the optimal solution

(ii) [3 pts] h_B: $h_B = h_0$ for all states except at one unspecified state y, where $h_B(y) = h_0(y) + \epsilon$

Min value of range: $h^*(S)$	**Max value of range:** $h^*(S) + \epsilon$

$[h^*(S), h^*(S) + \epsilon]$; if y is on the optimal path, it makes the optimal path look ϵ worse than it is, allowing nodes on some suboptimal path to be selected for expansion if they are no more than ϵ worse.

(iii) [3 pts] h_C: $h_C(x) = h_0(x) + \epsilon$ for all states x.

Min value of range: $h^*(S)$	**Max value of range:** $h^*(S)$

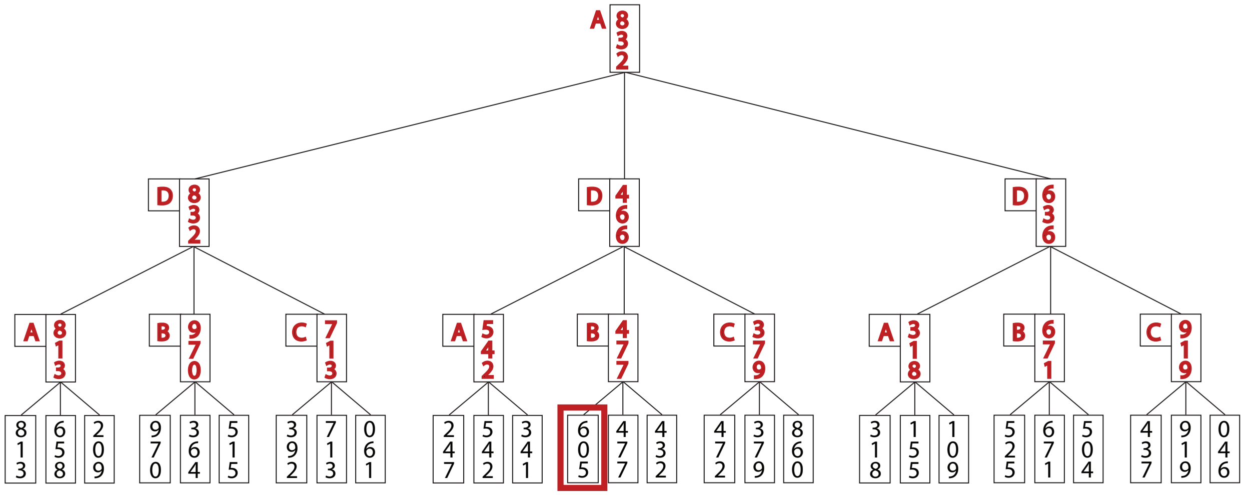
$[h^*(S), h^*(S)]$: as A^* expands states in order of $f = g + h$, adding a constant to h for all nodes doesn't change the order of expansion, so it still returns an optimal solution as before. This might seem counter-intuitive because h_C is "worse" than h_B almost everywhere!

Q2. [20 pts] Adversarial Search

Consider a game with three players A, B, and C. in which, *before every move*, a fair 3-sided die, D, is rolled to determine which player gets to make a move. (The sides of the die are marked A, B, and C.) The first die has been rolled, and it is A's turn to start the game. In every nonterminal state, the player whose turn it is has a choice of three moves. In a terminal state s, each player receives their own payoff from a payoff tuple $[U_A(s), U_B(s), U_C(s)]$; the aim of each player is to maximize the expected payoff he or she receives.

(a) [8 pts] The game tree below corresponds to two turns of the game, after which the game ends and the values shown are attained.

Fill in the three utility values in each dotted-line box in the game tree. Assume each player plays optimally and the die is fair.



(b) [6 pts] Suppose you know in advance that each value must be in the range 0-9, as shown above. Using this knowledge, in the above game tree, circle the first leaf node that need not be evaluated if the tree is explored left-to-right.

(c) [6 pts] Consider a version of this tree without specific leaf values. Circle all leaf nodes that will always be explored, i.e. the leaf nodes that will never skipped due to pruning regardless of the values in the tree. Again, assume that each value must be in the range 0-9, and that the tree is explored left-to-right.

First, fourth, and seventh leaves. Because we know that the values are bounded to be at most 9, and A, B, and C are all maximizers, we are guaranteed to visit the left-most child of A, B, and C in the left-most branch of this tree (if they are all 9, you don't have to visit anything else).

Q3. [25 pts] CSPs

(a) Pacman's new house

After years of struggling through mazes, Pacman has finally made peace with the ghosts, Blinky, Pinky, Inky, and Clyde, and invited them to live with him and Ms. Pacman. The move has forced Pacman to change the rooming assignments in his house, which has 6 rooms. He has decided to figure out the new assignments with a CSP in which the variables are Pacman **(P)**, Ms. Pacman **(M)**, Blinky **(B)**, Pinky **(K)**, Inky **(I)**, and Clyde **(C)**, the values are which room they will stay in, from 1-6, and the constraints are:

i) No two agents can stay in the same room
ii) **P** > 3
iii) **K** is less than **P**
iv) **M** is either 5 or 6
v) **P** $>$ **C**
vi) **B** is even
vii) **I** is not 1 or 6
viii) $|\mathbf{I}\text{-}\mathbf{C}| = 1$
ix) $|\mathbf{P}\text{-}\mathbf{B}| = 2$

(i) [3 pts] **Unary constraints** On the grid below cross out the values from each domain that are eliminated by enforcing unary constraints.

P	~~1~~	~~2~~	~~3~~	4	5	6
B	~~1~~	2	~~3~~	4	~~5~~	6
C	1	2	3	4	5	6
K	1	2	3	4	5	6
I	~~1~~	2	3	4	5	~~6~~
M	~~1~~	~~2~~	~~3~~	~~4~~	5	6

The unary constraints are ii, iv, vi, and vii. ii crosses out 1,2, and 3 for P. iv crosses out 1,2,3,4 for M. vi crosses out 1,3, and 5 for B. vii crosses out 1 and 6 for I. K and C have no unary constraints, so their domains remain the same.

(ii) [2 pts] **MRV** According to the Minimum Remaining Value (MRV) heuristic, which variable should be assigned to first?

○ P ○ B ○ C ○ K ○ I ● M

M has the fewest value remaining in its domain (2), so it should be selected first for assignment.

(iii) [6 pts] **Forward Checking** For the purposes of decoupling this problem from your solution to the previous problem, assume we choose to assign P first. If we consider values 4, 5, and 6 for P, what are the resulting domains after enforcing unary constraints (from part i) and running forward checking for each of these three possible assignments? Again, cross off values on the grids below.

P				4		
B	~~1~~	2	~~3~~	~~4~~	~~5~~	6
C	1	2	3	~~4~~	~~5~~	~~6~~
K	1	2	3	~~4~~	~~5~~	~~6~~
I	~~1~~	2	3	~~4~~	5	~~6~~
M	~~1~~	~~2~~	~~3~~	~~4~~	5	6

P					5	
B	~~1~~	~~2~~	~~3~~	~~4~~	~~5~~	~~6~~
C	1	2	3	4	~~5~~	~~6~~
K	1	2	3	4	~~5~~	~~6~~
I	~~1~~	2	3	4	~~5~~	~~6~~
M	~~1~~	~~2~~	~~3~~	~~4~~	~~5~~	6

P						6
B	~~1~~	~~2~~	~~3~~	4	~~5~~	~~6~~
C	1	2	3	4	5	~~6~~
K	1	2	3	4	5	~~6~~
I	~~1~~	2	3	4	5	~~6~~
M	~~1~~	~~2~~	~~3~~	~~4~~	5	~~6~~

(iv) [2 pts] **LCV** Building on your answer from the previous problem, according to the Least Constraining Value (LCV) heuristic, which value for P should we try?

○ 4 ○ 5 ● 6

Assigning 6 removes the least number of options.

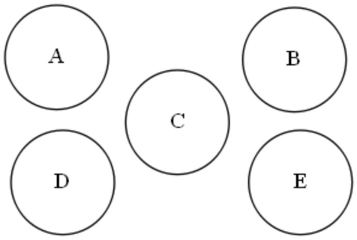
(b) All Satisfying Assignments

Now consider a modified CSP in which we wish to find **every possible satisfying assignment**, rather than just one such assignment as in normal CSPs. In order to solve this new problem, we use a new algorithm, which is the same as the normal backtracking search algorithm, except that when it sees a solution, instead of returning it, the solution gets added to a list, and the algorithm backtracks. Once there are no variables remaining to backtrack on, the algorithm returns the list of solutions it has found.

For each graph below, select which techniques do NOT help reduce the number of nodes explored in the search tree in this new situation.

The remaining parts all have a similar reasoning. Since every value has to be checked regardless of the outcome of previous assignments, the order in which the values are checked does not matter, so LCV has no effect.
In the general case, in which there are constraints between variables, the size of each domain can vary based on the order in which variables are assigned, so MRV can still have an effect on the number of nodes explored for the new "find all solutions" task.
The one time that filtering and MRV are guaranteed to not have any effect is when the constraint graph is completely disconnected, as is the case for part i. In this case, the domains of each variable do not depend on any other variable's assignment. Thus, the ordering of variables does not matter, and MRV cannot have any effect on the number of nodes explored.

(i) [4 pts]



Select ALL that do NOT reduce nodes explored:

☒ Forward checking

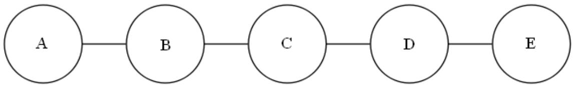
☒ AC-3

☒ MRV

☒ LCV

□ None of the above, i.e. all are helpful

(ii) [4 pts]



Select ALL that do NOT reduce nodes explored:

□ Forward checking

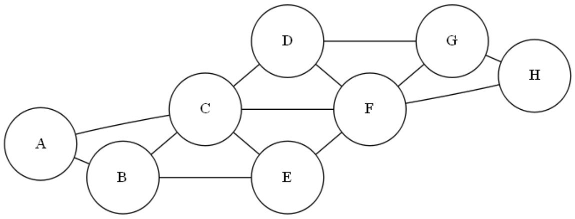
□ AC-3

□ MRV

☒ LCV

□ None of the above, i.e. all are helpful

(iii) [4 pts]



Select ALL that do NOT reduce nodes explored:

□ Forward checking

□ AC-3

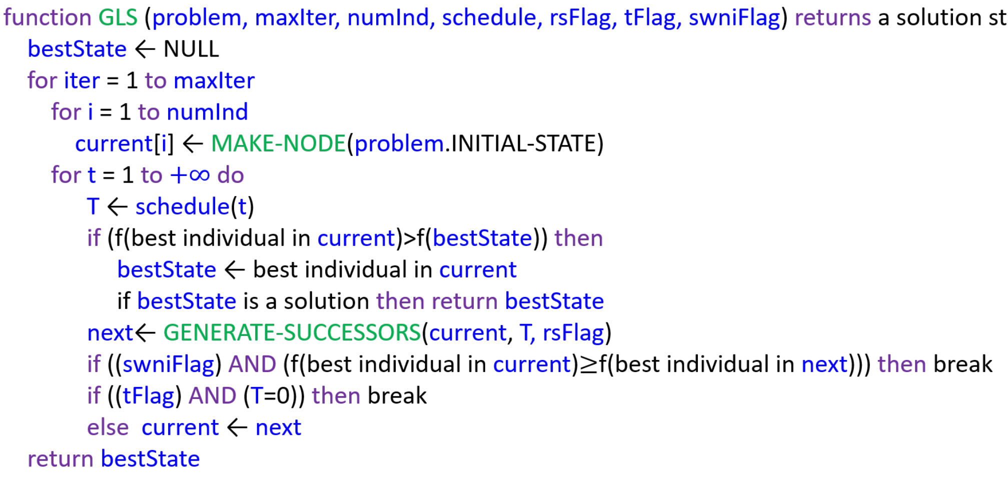
□ MRV

☒ LCV

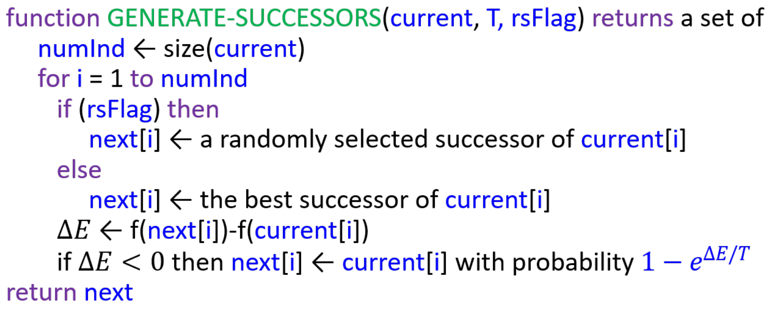
□ None of the above, i.e. all are helpful

Q4. [10 pts] Generic Local Search

You are given a generic local search algorithm GLS as shown below. f is a known function that provides value or fitness scores for an state. $f(\text{NULL}) = -\infty$. schedule(t) is the temperature at time step t. You can call GLS with appropriate parameter values to get a specific local search algorithm.



```
function GLS (problem, maxIter, numInd, schedule, rsFlag, tFlag, swniFlag) returns a solution state
  bestState ← NULL
  for iter = 1 to maxIter
    for i = 1 to numInd
      current[i] ← MAKE-NODE(problem.INITIAL-STATE)
    for t = 1 to +∞ do
       T ← schedule(t)
       if (f(best individual in current)>f(bestState)) then
          bestState ← best individual in current
          if bestState is a solution then return bestState
       next← GENERATE-SUCCESSORS(current, T, rsFlag)
       if ((swniFlag) AND (f(best individual in current)≥f(best individual in next))) then break
       if ((tFlag) AND (T=0)) then break
       else  current ← next
  return bestState
```



```
function GENERATE-SUCCESSORS(current, T, rsFlag) returns a set of states
   numInd ← size(current)
   for i = 1 to numInd
     if (rsFlag) then
        next[i] ← a randomly selected successor of current[i]
     else
        next[i] ← the best successor of current[i]
     $\Delta E$ ← f(next[i])-f(current[i])
     if $\Delta E < 0$ then next[i] ← current[i] with probability $1 - e^{\Delta E/T}$
return next
```

(a) [5 pts] Complete the table with appropriate parameter values to get the required local search algorithm. For flags, use F to represent false and T to represent true. If multiple values can make the algorithm work, just fill in one possible value.

Algorithm Name	maxIter	numInd	schedule(t)	rsFlag	tFlag	swniFlag
Vanilla Hill Climbing	1	1	0	F	F	T
Random-restart Hill Climbing with M random restarts	M	1	0	F	F	T
Random Walk	1	1	$+\infty$	T	T/F	F
Simulated Annealing with schedule $s(t)$	1	1	$s(t)$	T	T	F

Hint: rsFlag is short for randomSuccessorFlag. tFlag is short for terminationFlag. swniFlag is short for stopWhenNoImprovementFlag.

(b) [5 pts] If you rewrite the GENERATE-SUCCESSOR function, GLS can also be called for Beam Search and Genetic Algorithm. Complete the following table with appropriate parameter values and provide a **short description (1-2 sentences)** of the GENERATE-SUCCESSOR function for each of them.

Algorithm Name	maxIter	numInd	swniFlag
Beam Search with beam width K	1	K	F
Genetic Algorithm with population size N	1	N	F

GENERATE-SUCCESSOR function for Beam Search with beam width K:

For each state in current, generate all its successors. Select the best K successors among them all.

Hint: This customized function only needs to make use of the "current" parameter. 1-2 sentences will suffice.

GENERATE-SUCCESSOR function for Genetic Algorithm with population size N:

Generate N offsprings from current population through selection, crossover, and mutation.

Hint: Mention the three main operators. This customized function only needs to make use of the "current" parameter. 1-2 sentences will suffice.

Q5. [10 pts] Linear Programming

A chip company produces bags of corn chips and a potato chips. The company needs to make at least 100 bags of corn chips and 80 bags of potato chips each day. However, because of limitations on production capacity, no more than 200 bags of corn chips and 170 bags of potato chips can be made daily. To satisfy a shipping contract, a total of at least 200 bags of either kind of chip much be shipped each day.

If each bag of corn chips sold results in a \$2 loss, but each bag of potato chips produces a \$5 profit, how many of each type should be made daily to maximize net profits?

(a) [5 pts] Write the constraints to solve this problem in inequality form.

Inequality Form:

maximize $-2C + 5P$ s.t.

$$-C \leq -100 \tag{1}$$

$$-P \leq -80 \tag{2}$$

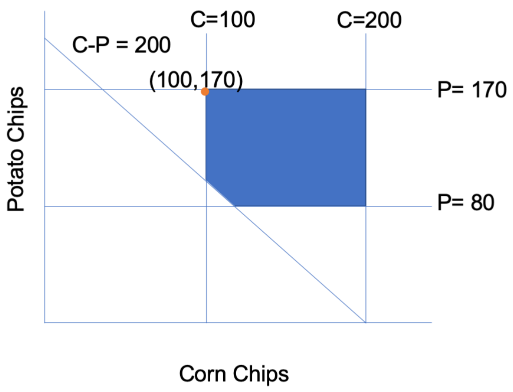
$$C \leq 200 \tag{3}$$

$$P \leq 170 \tag{4}$$

$$-C - P \leq -200 \tag{5}$$

(b) [4 pts] Graph the constraints and shade the feasible region.

Graph:



(c) [1 pt] What is the optimal amount of potato and corn chips to produce and what is the corresponding profit?

Solution:

$C = 100, P = 170$

Profit:

$-2 * (C = 100) + 5 * (P = 170) = 650$

THIS PAGE INTENTIONALLY LEFT BLANK