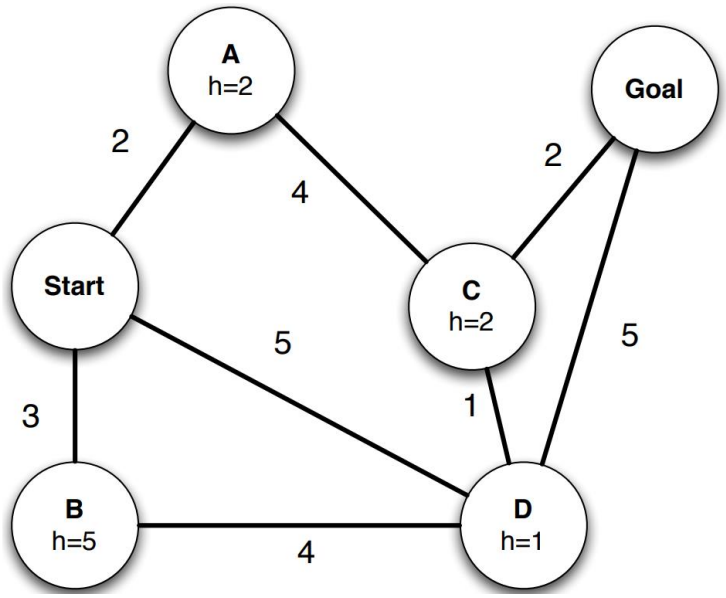
Contents

informed and uninformed search

1) (Graph search algorithms)

For each of the following graph search strategies, work out the order in which states are expanded, as well as the path returned by graph search. In all cases, assume ties resolve in such a way that states with earlier alphabetical order are expanded first. The start and goal state are S and G, respectively. Remember that in graph search, a state is expanded only once.



a) Depth-first search.

States Expanded: Start, A, C, D, B, Goal

Path Returned: Start-A-C-D-Goal

b) Breadth-first search.

States Expanded: Start, A, B, D, C, Goal

Path Returned: Start-D-Goal

c) Uniform cost search.

States Expanded: Start, A, B, D, C, Goal

Path Returned: Start-A-C-Goal

d) Greedy search with the heuristic h shown on the graph.

States Expanded: Start, D, Goal

Path Returned: Start-D-Goal

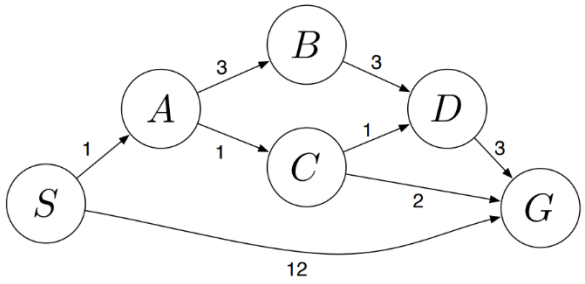
e) A* search with the same heuristic.

States Expanded: Start, A, D, C, Goal

Path Returned: Start-A-C-Goal

2) (Evaluation of heuristics)

Answer the following questions about the search problem shown above. Break any ties alphabetically. For the questions that ask for a path, please give your answers in the form 'S – A – D – G.'



(a) What path would breadth-first graph search return for this search problem?

S – G

(b) What path would uniform cost graph search return for this search problem?

S – A – C – G

(c) What path would depth-first graph search return for this search problem?

S – A – B – D – G

(d) What path would A* graph search, using a consistent heuristic, return for this search problem?

S – A – C – G

(e) Consider the heuristics for this problem shown in the table below.

State	h_1	h_2
S	5	4
A	3	2
B	6	6
C	2	1
D	3	3
G	0	0

i. Is h1 admissible? Yes No

ii. Is h1 consistent? Yes No

iii. Is h2 admissible? Yes No

iv. Is h2 consistent? Yes No

i. NO

ii. NO

iii. YES

iv. NO

An admissible heuristic must underestimate or be equal to the true cost. A consistent heuristic must satisfy h(N) − h(L) ≤ path(N → L) for all paths and nodes N and L. h1 overestimates the cost S → G as 5 when it is 4, so it is inadmissible. h1 is not consistent because h(S) − h(A) ≤ path(S → A) is violated as 5 − 3 ≤ 1. h2 does not overestimate costs and is admissible. h2 is not consistent because h(S) − h(A) ≤ path(S → A) is violated as 4 − 2 ≤ 1.

3) (Status space search)

Pacman and Ms. Pacman are lost in an NxN maze and would like to meet; they don‟t care where. In each time step, both simultaneously move in one of the following directions: {NORTH, SOUTH, EAST, WEST, STOP}. They do not alternate turns. You must devise a plan which positions them together, somewhere, in as few time steps as possible. Passing each other does not count as meeting; they must occupy the same square at the same time.

a. Formally state this problem as a single-agent state-space search problem.
States:

The set of pairs of positions for Pacman and Ms. Pacman: {((x1, y1), (x2, y2)) | x1, x2, y1, y2 are in {1, 2, …, N}}

Goal test:

isGoal((x1, y1), (x2, y2)): return x1 == x2 and y1 == y2

Legal actions (given a state):

legalActions((x1, y1), (x2, y2)): If not blocked from their current positions, both pacman and mrs pacman can move north, south, east, west. They can always stop.

Successor function (given a state and an action):
successor(((x1, y1), (x2, y2)), action): Move pacman and mrs pacman from their current state in the direction they both moved (respectively). Return the new x1, y1, x2, y2 positions.

b. Give a non-trivial admissible heuristic for this problem.

Answer: Manhattan distance between Pacman and Ms. Pacman DIVIDED BY 2 (since both take a step simultaneously)

c. Circle all of the following graph search methods which are guaranteed to output optimal solutions to this problem:
(i) DFS (ii) BFS (iii) UCS

(iv) A* (with a consistent and admissible heuristic)

(v) A* (with heuristic that returns zero for each state)

Everything but DFS.

d. If h1 and h2 are admissible, which of the following are also guaranteed to be admissible? Circle all that apply:
(i) h1 + h2
(ii) h1 * h2

(iii) $\max(h_1, h_2)$

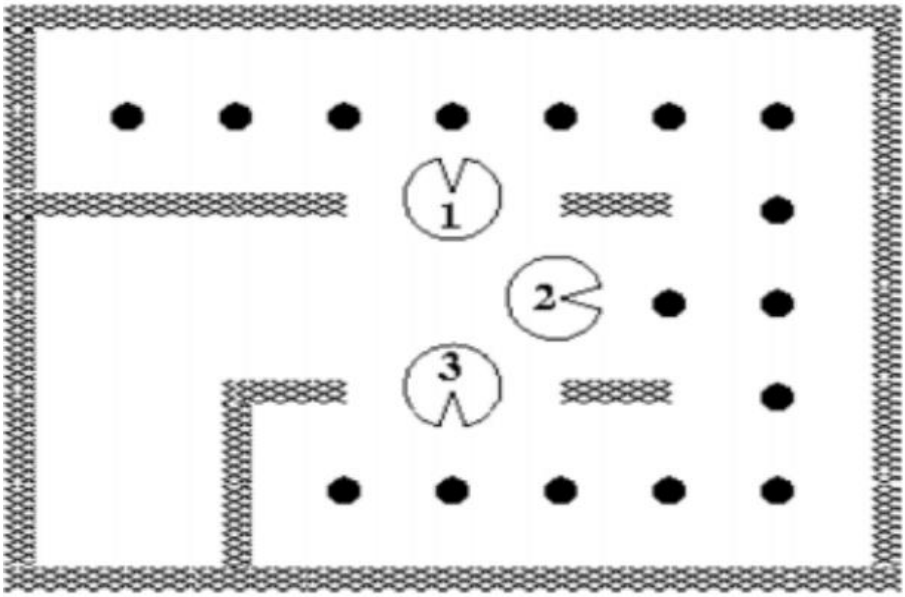
(iv) $\min(h_1, h_2)$

(v) $(\alpha)h_1+(1-\alpha)h_2$ for any value α between 0 and 1

Answer: (iii), (iv), (v)

4) (Multifactor search)

Pacman is trying eat all the dots, but he now has the help of his family! There are initially k dots, at positions (f1, . . . , fk). There are also n Pac-People, at positions (p1, . . . , pn); initially, all the Pac-People start in the bottom left corner of the maze. Consider a search problem in which all Pac-People move simultaneously; that is, in each step each Pac-Person moves into some adjacent position (N, S, E, or W, no STOP). Note that any number of Pac-People may occupy the same position.



(a) Define the state space of the search problem.

The state space consists of the following: a k-tuple of boolean variables E, where Ei = 1 if the ith food has been eaten and 0 otherwise, and the n-tuple of Pac-People positions P = (p1, . . . , pn). We assign a cost of 2 for a movement of Pac-People which does not result in a food being eaten, and a cost of 1 for a movement which does.

(b) Give a reasonable upper bound on the size of the state space for a general r by c grid.

To represent just the tuple E we need 2k states. In addition, there will be a state for each possible arrangement of the Pac-People, and since each Pac-Person can be in one of rc positions, we need (rc) n states just for the Pac-People positions, so all in all we have an upper (and lower) bound of 2k (rc) n states in our state space.

(c) What is the goal test?

The goal test in this case is whether or not the tuple E consists entirely of 1's, representing that all of the food has been eaten.

(d) What is the maximum branching factor of the successor function in a general grid?

The maximum branching factor of the successor function in a general grid will be 4n, since there are n Pac-People and each of them can go in one of four directions, N, S, E, or W. (Note that the minimum may be lower, since different combinations of directions may result in the same state, for example if a Pac-Person is in the upper-left corner then going left and going up results in the same outcome for that Pac-Person).

(e) Circle the admissible heuristics below (−1/2 point for each mistake.)

- h1(s) = 0

This is admissible since the cost of getting to the goal state will never be below 0. (Even with one food left, we will still need to pay a cost of at least 1 to move a Pac-Person to that food.)

- h2(s) = 1

This is admissible in this case, since the cost of getting to the goal state will always be at least 1. However, if we set our costs differently this may become non-admissible, i.e., if the cost of eating a food was changed to 0.5 then this function would overestimate the cost of reaching the goal state in some cases.

- h3(s) = number of remaining food/ n

This is admissible in our case since number of remaining food on its own will never overestimate the cost of reaching the goal (since it costs at least 1 to eat each food piece), and dividing by n will only increase the margin by which we underestimate.

- h4(s) = maxi maxj manhattan(pi , fj)

This function intuitively corresponds to "the distance between the farthest food/Pac-Person pair". This is not admissible since, for example, there could be two Pac-People, one of whom can eat the final food in one move and another who would require 5 moves, and in this case the function will overestimate the cost for reaching the goal as 5 when it is actually 1.

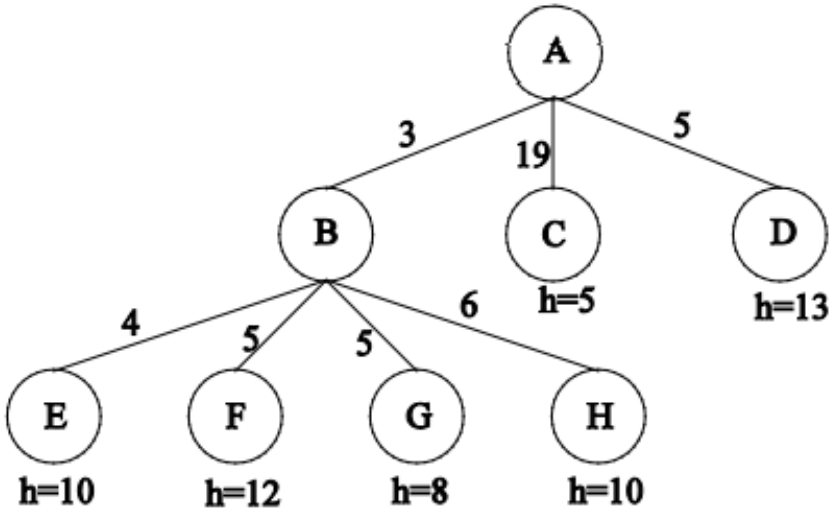
- h5(s) = maxi minj manhattan(pi , fj)

This function intuitively corresponds to "the farther distance any Pac-Person has to their closest food". This is still not admissible, since it could be the case that one Pac-Person is far

away from the final food (say 5 moves away) while another Pac-Person is directly adjacent to it, and this function would predict 5 rather than the actual cost of 1.

5) (Graph search algorithms)

Consider the following search tree produced after expanding nodes A and B, where each arc is labeled with the cost of the corresponding operator, and the leaves are labeled with the value of a heuristic function, h. For uninformed searches, assume children are expanded left to right. In case of ties, expand in alphabetical order.



Which one node will be expanded next by each of the following search methods?

(a) Depth-First search

E

(b) Greedy Best-First search

C (smallest h value)

(c) Uniform-Cost search

D (smallest g value)

(d) A* search

G (smallest g+h value)

6) (search algorithms)

True or False:

(a) Greedy Best-First search using an admissible heuristic is guaranteed to find an optimal solution.

False

(b) Algorithm A search using the heuristic h(n) = c for some fixed constant c > 0 is guaranteed to find an optimal solution.

True since each node in the Frontier has value defined by f(n) = g(n) + c. Since only the ordering of the nodes in Frontier matters, adding a constant to all nodes will not change their order. So, this is equivalent to Uniform-Cost search, which always finds an optimal solution.

(c) If a heuristic is consistent, it is also admissible.

True

(d) If h1 and h2 are both admissible heuristics, it is always better to use the heuristic h3(n) = max(h1(n), h2(n)) rather than the heuristic h4(n) = min(h1(n), h2(n)).

True

(e) Beam search with a beam width W = 3 and an admissible heuristic is not guaranteed to find a solution (optimal or not) when one exists.

True

(f) Say we have a state space where all arc costs are 1 but we don't have a heuristic function. We want to use a space-efficient search algorithm (in terms of the maximum number of nodes stored at any point during the search in Frontier) but also want to guarantee that we find an optimal solution. Which one of the following search methods would be best to use in this situation?

(i) Breadth-First Search
(ii) Depth-First Search
(iii) Iterative-Deepening Search
(iv) Uniform-Cost Search

(iii) Iterative-Deepening

CSP

1) (Preliminaries and tree design)

You are in charge of scheduling for computer science classes that meet Mondays, Wednesdays and Fridays. There are 5 classes that meet on these days and 3 professors who will be teaching these classes. You are constrained by the fact that each professor can only teach one class at a time.

The classes are:

1. Class 1 - Intro to Programming: meets from 8:00-9:00am
2. Class 2 - Intro to Artificial Intelligence: meets from 8:30-9:30am
3. Class 3 - Natural Language Processing: meets from 9:00-10:00am
4. Class 4 - Computer Vision: meets from 9:00-10:00am 5. Class
5 - Machine Learning: meets from 10:30-11:30am

The professors are:
1. Professor A, who is qualified to teach Classes 1, 2, and 5.
2. Professor B, who is qualified to teach Classes 3, 4, and 5.
3. Professor C, who is qualified to teach Classes 1, 3, and 4.

1. Formulate this problem as a CSP problem in which there is one variable per class, stating the domains, and constraints. Constraints should be specified formally and precisely, but may be implicit rather than explicit.

Variables Domains (or unary constraints)

C1 {A, C}

C2 {A}

C3 {B, C}

C4 {B, C}

C5 {A, B}

Binary Constraints

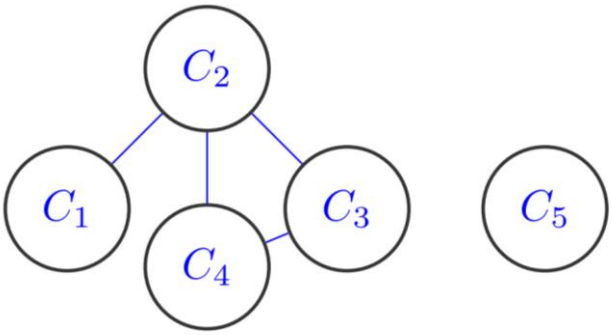
C1 ≠ C2

C2 ≠ C3

C2 ≠ C4

C3 ≠ C4

2. Draw the constraint graph associated with your CSP.



3. Your CSP should look nearly tree-structured. Briefly explain (one sentence or less) why we might prefer to solve tree-structured CSPs.

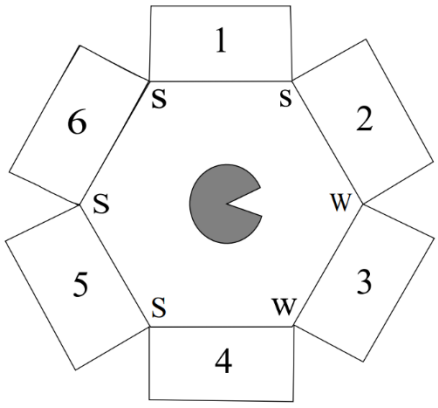
Minimal answer: we can solve them in polynomial time. If a graph is tree structured (i.e. has no loops), then the CSP can be solved in O(nd2) time as compared to general CSPs, where worst-case time is O(d n). For tree-structured CSPs you can choose an ordering such that every node's parent precedes it in the ordering. Then after enforcing arc consistency you can greedily assign the nodes in order, starting from the root, and will find a consistent assignment without backtracking.

2) (Circular structure and backtracking)

Pacman is trapped! He is surrounded by mysterious corridors, each of which leads to either a pit (P), a ghost (G), or an exit (E). In order to escape, he needs to figure out which corridors, if any, lead to an exit and freedom, rather than the certain doom of a pit or a ghost.

The one sign of what lies behind the corridors is the wind: a pit produces a strong breeze (S) and an exit produces a weak breeze (W), while a ghost doesn't produce any breeze at all. Unfortunately, Pacman cannot measure the strength of the breeze at a specific corridor. Instead, he can stand between two adjacent corridors and feel the max of the two breezes. For example, if he stands between a pit and an exit he will sense a strong (S) breeze, while if he stands between an exit and a ghost, he will sense a weak (W) breeze. The measurements for all intersections are shown in the figure below.

Also, while the total number of exits might be zero, one, or more, Pacman knows that two neighboring squares will not both be exits.



Pacman models this problem using variables Xi for each corridor i and domains P, G, and E.

1. State the binary and/or unary constraints for this CSP (either implicitly or explicitly).

Binary:

X1 = P or X2 = P, X2 = E or X3 = E,

X3 = E or X4 = E, X4 = P or X5 = P,

X5 = P or X6 = P, X1 = P or X6 = P,

∀i, j s.t. Adj(i, j) ¬(Xi = E and Xj = E)

Unary:

X2 ≠ P,

X3 ≠ P,

X4 ≠ P

2. Cross out the values from the domains of the variables that will be deleted in enforcing arc consistency.

X_1	P	G	E
X_2	P	G	E
X_3	P	G	E
X_4	P	G	E
X_5	P	G	E
X_6	P	G	E

X_1	P		
X_2		G	E
X_3		G	E
X_4		G	E
X_5	P		
X_6	P	G	E

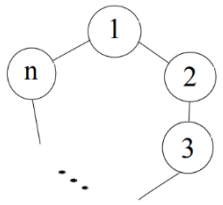
3. According to MRV, which variable or variables could the solver assign first?

X1 or X5 (tie breaking)

4. Assume that Pacman knows that X6 = G. List all the solutions of this CSP or write none if no solutions exist.

(P, E, G, E, P, G)

(P, G, E, G, P, G)



5. The CSP described above has a circular structure with 6 variables. Now consider a CSP forming a circular structure that has n variables (n > 2), as shown below. Also assume that the domain of each variable has cardinality d. Explain precisely how to solve this general class of circle-structured CSPs efficiently (i.e. in time linear in the number of variables), using methods covered in class. Your answer should be at most two sentences.

We fix Xj for some j and assign it a value from its domain (i.e. use cutset conditioning on one variable). The rest of the CSP now forms a tree structure, which can be efficiently solved without backtracking by enforcing arc consistency. We try all possible values for our selected variable Xj until we find a solution.

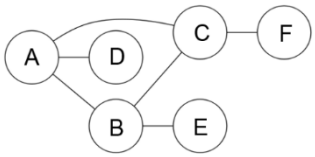
6. If standard backtracking search were run on a circle-structured graph, enforcing arc consistency at every step, what, if anything, can be said about the worst-case backtracking behavior (e.g. number of times the search could backtrack).

A tree structured CSP can be solved without any backtracking. Thus, the above circle-structured CSP can be solved after backtracking at most d times, since we might have to try up to d values for Xj before finding a solution.

3) (Filtering and stability)

(a) The graph below is a constraint graph for a CSP that has only binary constraints. Initially, no variables have been assigned.

For each of the following scenarios, mark all variables for which the specified filtering might result in their domain being changed.



(i) A value is assigned to A. Which domains might be changed as a result of running forward checking for A?

☐ A ☐ B ☐ C ☐ D ☐ E ☐ F

(ii) A value is assigned to A, and then forward checking is run for A. Then a value is assigned to B. Which domains might be changed as a result of running forward checking for B?

☐A ☐ B ☐ C ☐ D ☐ E ☐ F

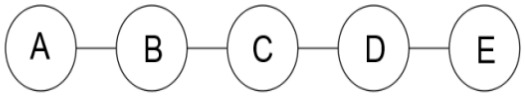
(iii) A value is assigned to A. Which domains might be changed as a result of enforcing arc consistency after this assignment?

☐ A ☐ B ☐ C ☐ D ☐ E ☐ F

(iv) A value is assigned to A, and then arc consistency is enforced. Then a value is assigned to B. Which domains might be changed as a result of enforcing arc consistency after the assignment to B?

☐ A ☐ B ☐ C ☐ D ☐ E ☐ F

(b) You decide to try a new approach to using arc consistency in which you initially enforce arc consistency, and then enforce arc consistency every time you have assigned an even number of variables. You have to backtrack if, after a value has been assigned to a variable, X, the recursion returns at X without a solution. Concretely, this means that for a single variable with d values remaining, it is possible to backtrack up to d times. For each of the following constraint graphs, if each variable has a domain of size d, how many times would you have to backtrack in the worst case for each of the specified orderings?

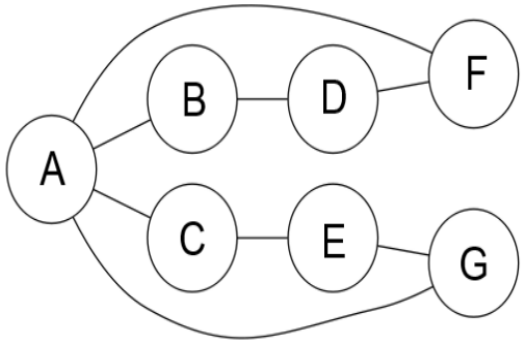


A-B-C-D-E: _____

A-E-B-D-C: _____

C-B-D-E-A: _____

(ii) [6 pts]



A-B-C-D-E-F-G: _____

F-D-B-A-C-G-E: _____

C-A-F-E-B-G-D: _____

Answer:

(a)

(i) A value is assigned to A. Which domains might be changed as a result of running forward checking for A?

☐ A ■ B ■ C ■ D ☐ E ☐ F

Forward checking for A only considers arcs where A is the head. This includes B → A, C → A, D → A. Enforcing these arcs can change the domains of the tails.

(ii) A value is assigned to A, and then forward checking is run for A. Then a value is assigned to B. Which domains might be changed as a result of running forward checking for B?

☐ A ☐ B ■C ☐ D ■ E ☐ F

Similar to the previous part, forward checking for B enforces the arcs A → B, C → B, and E → B. However, because A has been assigned, and a value is assigned to B, which is consistent with A or else no value would have been assigned, the domain of A will not change.

(iii) A value is assigned to A. Which domains might be changed as a result of enforcing arc consistency after this assignment?

☐ A ■B ■ C ■ D ■ E ■ F

Enforcing arc consistency can affect any unassigned variable in the graph that has a path to the assigned variable. This is because a change to the domain of X results in enforcing all arcs where X is the head, so changes propagate through the graph. Note that the only time in which the domain for A changes is if any domain becomes empty, in which case the arc consistency algorithm usually returns immediately and backtracking is required, so it does not really make sense to consider new domains in this case.

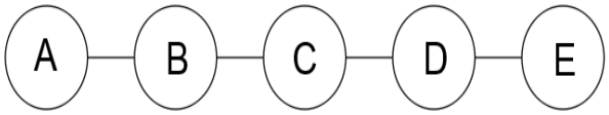
(iv) A value is assigned to A, and then arc consistency is enforced. Then a value is assigned to B. Which domains might be changed as a result of enforcing arc consistency after the assignment to B?

☐ A ☐B ■ C ☐ D ■ E ■ F

After assigning a value to A, and enforcing arc consistency, future assignments and enforcing arc consistency will not result in a change to As domain. This means that Ds domain won't change because the only arc that might cause a change, D → A will never be enforced.

(b)

(i) [6 pts]



A-B-C-D-E: 0

A-E-B-D-C: 2*d*

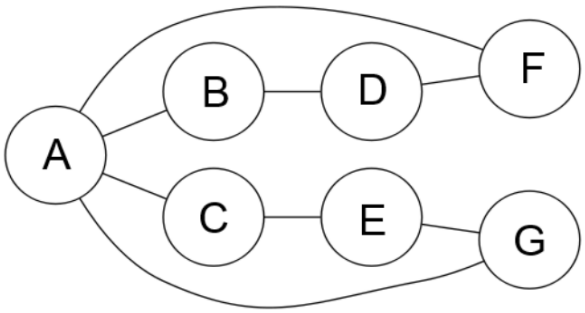
C-B-D-E-A: 0

If no solution containing the current assignment exists on a tree structured CSP, then enforcing arc consistency will always result in an empty domain. This means that running arc consistency on a tree structured 7 CSP will immediately tell you whether or not the current assignment is part of a valid solution, so you can immediately start backtracking without further assignments.

A – B – C – D – E and C – B – D – E – A are both linear orderings of the variables in the tree, which is essentially the same as running the two pass algorithm, which will solve a tree structured CSP with no backtracking.

A – E – B – D – C is not a linear ordering, so while the odd assignments are guaranteed to be part of a valid solution, the even assignments are not (because arc consistency was not enforced after assigning the odd variables). This means that you may have to backtrack on every even assignment, specifically E and D. Note that because you know whether or not the assignment to E is valid immediately after assigning it, the backtracking behavior is not nested (meaning you backtrack on E up to d times without assigning further variables). The same is true for D, so the overall behavior is backtracking 2d times.

(ii) [6 pts]



A-B-C-D-E-F-G: d^2

F-D-B-A-C-G-E: $d^4 + d$

C-A-F-E-B-G-D: d^2

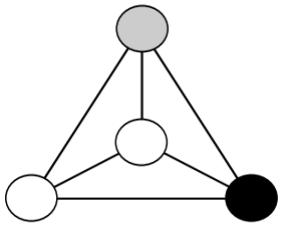
A–B –C – D –E –F –G: The initial assignment of A, B might require backtracking on both variables, because there is no guarantee that the initial assignment to A is a valid solution. Because A is a cutset for this graph, the resulting graph consists of two trees, so enforcing arc consistency immediately returns whether the assignments to A and B are part of a solution, and you can begin backtracking without further assignments.

F – D – B – A – C – G – E: Until A is assigned, there is no guarantee that any of the previous values assigned are part of a valid solution. This means that you may be required to backtrack on all of them, resulting in d 4 times. Furthermore, the remaining tree is not assigned in a linear order, so further backtracking may be required on G (similar to the second ordering above) resulting in a total of d 4+d.

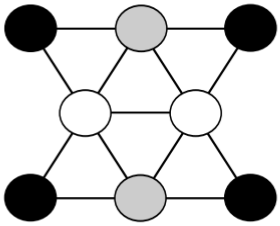
C – A – F – E – B – G – D: This ordering is similar to the first one. except that the resulting trees are not being assigned in linear order. However, because each tree only has a single value assigned in between each run of arc consistency, no backtracking will be required (you can think of each variable as being the root of the tree, and the assignment creating a new tree or two where arc consistency has been enforced), resulting in a total of d 2 times.

4) (Local search and objective function evaluation)

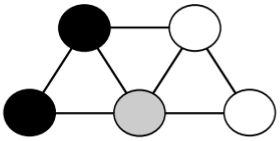
In this question we are considering CSPs for map coloring. Each region on the map is a variable, and their values are chosen from {black, gray, white}. Adjacent regions cannot have the same color. The figures below show the constraint graphs for three CSPs and an assignment for each one. None of the assignments are solutions as each has a pair of adjacent variables that are white. For both parts of this question, let the score of an assignment be the number of satisfied constraints (so a higher score is better).



(1)



(2)



(3)

Consider applying Local Search starting from each of the assignments in the figure above. For each successor function, indicate whether each configuration is a local optimum and whether it is a global optimum (note that the CSPs may not have satisfying assignments).

Successor Function	CSP	Local optimum?		Global Optimum?	
Change a single variable	(1)	Yes	No	Yes	No
	(2)	Yes	No	Yes	No
	(3)	Yes	No	Yes	No

Change a single variable, or a pair of variables	(1)	Yes	No	Yes	No
	(2)	Yes	No	Yes	No
	(3)	Yes	No	Yes	No

Answer)

Successor Function	CSP	Local optimum?		Global Optimum?	
Change a single variable	(1)	Yes	No	Yes	No
	(2)	Yes	No	Yes	No
	(3)	Yes	No	Yes	No
Change a single variable, or a pair of variables	(1)	Yes	No	Yes	No
	(2)	Yes	No	Yes	No
	(3)	Yes	No	Yes	No

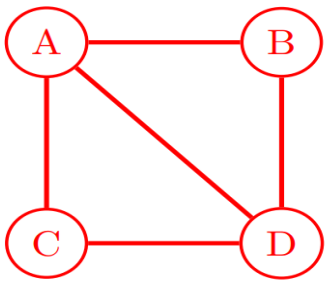
5) (Processing communication processes)

Four people, A, B, C, and D, are all looking to rent space in an apartment building. There are three floors in the building, 1, 2, and 3 (where 1 is the lowest floor and 3 is the highest). Each person must be assigned to some floor, but it's ok if more than one person is living on a floor. We have the following constraints on assignments:

- A and B must not live together on the same floor.
- If A and C live on the same floor, they must both be living on floor 2.
- If A and C live on different floors, one of them must be living on floor 3.
- D must not live on the same floor as anyone else.
- D must live on a higher floor than C.

We will formulate this as a CSP, where each person has a variable and the variable values are floors.

(a) Draw the edges for the constraint graph representing this problem. Use binary constraints only. You do not need to label the edges.



(b) Suppose we have assigned C = 2. Apply forward checking to the CSP, filling in the boxes next to the values for each variable that are eliminated:

A | 1☐ 2☐ 3☐
B | 1☐ 2☐ 3☐
C | 2☐
D | 1☐ 2☐ 3☐

(c) Starting from the original CSP with full domains (i.e. without assigning any variables or doing the forward checking in the previous part), enforce arc consistency for the entire CSP graph, filling in the boxes next to the values that are eliminated for each variable:

A | 1☐ 2☐ 3☐
B | 1☐ 2☐ 3☐
C | 1☐ 2☐ 3☐
D | 1☐ 2☐ 3☐

(d) Suppose that we were running local search with the min-conflicts algorithm for this CSP, and currently have the following variable assignments.

A | 3
B | 1
C | 2
D | 3

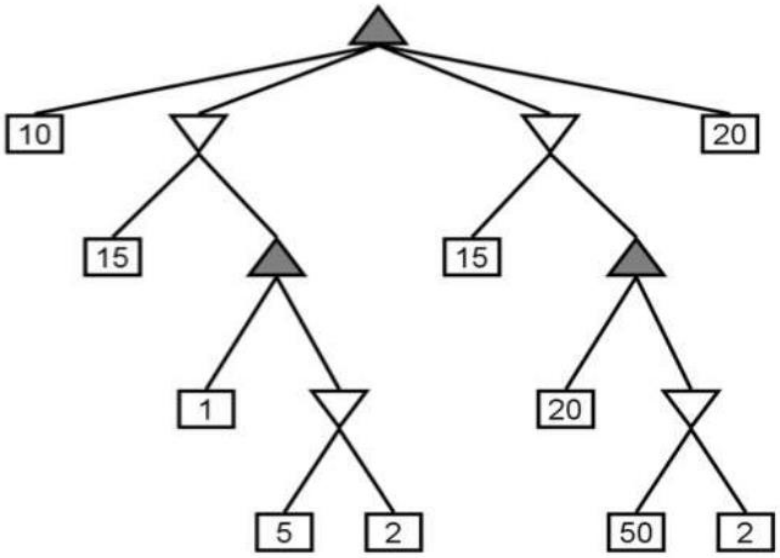
Which variable would be reassigned, and which value would it be reassigned to? Assume that any ties are broken alphabetically for variables and in numerical order for values.

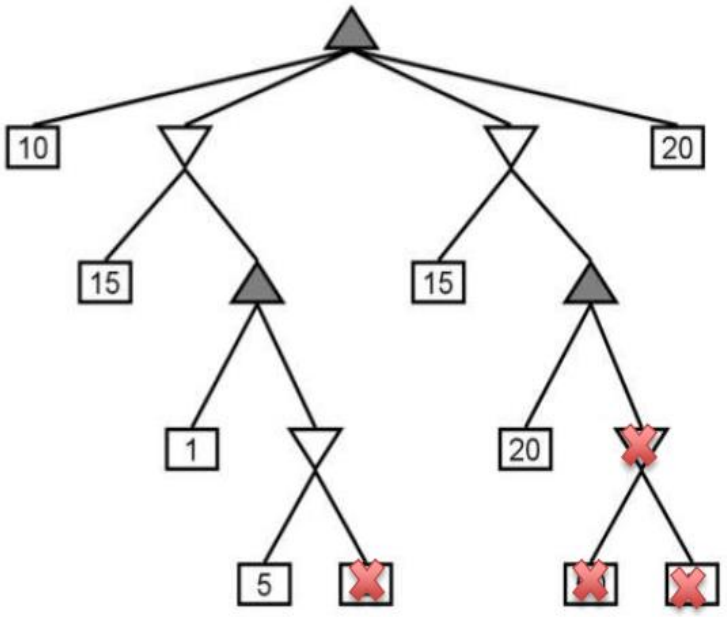
A is reassigned value 2.

Adversarial Search

1) (minimax tree and alpha-beta pruning)

Consider the following minimax tree:





a. What is the minimax value for the root?
20

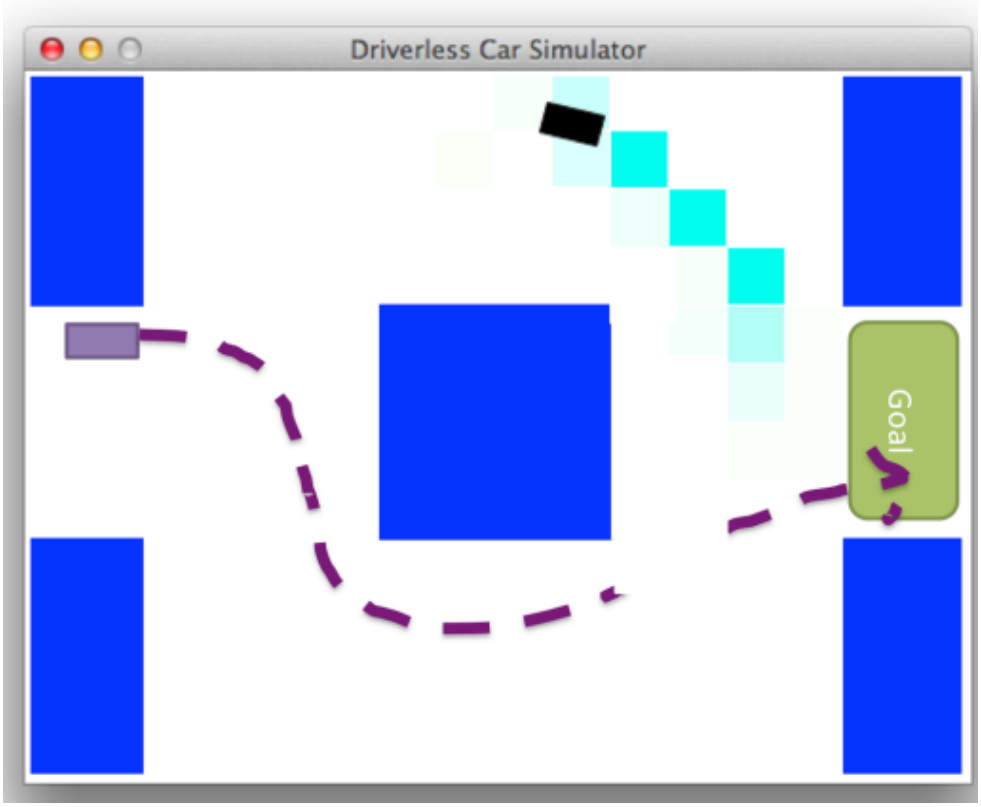
b. Draw an X through any nodes which will not be visited by alpha-beta pruning, assuming children are visited in left-to-right order.
See pic above.

c. Is there another ordering for the children of the root for which more pruning would result? If so, state the order.
Yes, if we had the children ordered as 20, 15, 10, 2.

d. Propose a general, practical method for ordering children of nodes which will tend to increase the opportunities for pruning. You should be concise, but clearly state both what to do about min nodes and max nodes.
In general we would want to use an evaluation function to estimate the values of the children and then order them so that at max nodes we order the children with larger estimated values first and at min nodes we order the children with larger estimated values first. For those who did not include the evaluation function: ordering nodes by their true values is not practical.

2) (Expectimax and Markov decision(Out of midterm topics))

The goal of this problem is to extend the self driving car to reason about the future and use that reasoning to make a utility maximizing decision on how to act



In this problem we are going to assume that the world is the same as in the Driverless Car programming problem. There is a single agent that exists in a closed world and wants to drive to a goal area. For this problem we are going to assume that there is only one other car.

Each heartbeat (there are twenty per second) each car can perform one of four actions Accelerate, TurnLeft, TurnRight, Brake, None. Assume that for each tile we have a probability distribution that the other car will take each of those actions. A car always has a wheel position and a velocity. Even though the car might not be taking the Accelerate action, if it has positive velocity it may continue to move forward.

[important] For this problem assume that cars are manufactured so precisely that if they take an action there is no uncertainty as to how the car will respond. Also assume that you know the start state of both your car and the other.

Formalize the problem of choosing an action as a Markov decision problem:

(a) What variables make up a state?

We will need to include both cars' current velocity, wheel position, what direction it is facing (up, down, left, or right), and location (what tile it is in).

(b) What is the start state?

The start state is the state in which the location of both cars is set to their known start positions, their velocities are set to zero, and their wheel positions are set to "forward", and the direction of the controlled car is set to "right" and of the other car is set to "left".

(c) For each state what are the legal actions?

In a given state, the legal actions are Accelerate, TurnLeft, TurnRight, Brake, and None. (Note that these correspond to the actions for the car we have control over. The "action" taken by the other car will actually just be represented by the probability distribution on the edges in the MDP.)

(d) What is a terminal condition for a given state S?

For all terminal conditions specify a utility. If G is the range of locations comprising the goal area, then one terminal condition is whether the controlled car's location is in G. The utility for this goal should be very large, say 100. Another terminal condition is whether the location of the controlled car is the same as the location of the other car. This

corresponds to the cars crashing, and thus should have a very low utility, say 1.

(e) Given a state S from which your agent took action A, what is the successor state distribution?

If A = Accelerate, the successor is a state wherein the controlled car's velocity is one higher than the velocity in the current state and its location is determined by location(), which is defined below.
If A = T urnLef t, the successor is a state wherein the controlled car's wheel position is the position to the left of its wheel position in the current state ("right" → "forward", "forward" → "left"). If the wheel position is already "left", then the next state has the same wheel position as the current state ("left" → "left"). Location is set by location().

If A = T urnRight, the opposite of TurnLeft happens ("left" → "forward", "forward" → "right", "right" → "right"). Location is set by location().

If A = Brake, the opposite of Accelerate happens, except that if the controlled car's velocity in the current state is zero then its velocity in the next state remains zero. Location is set by location(). If A = None, the controlled car's properties in the next state are the same as its properties in the current state, except its location which is set by location().

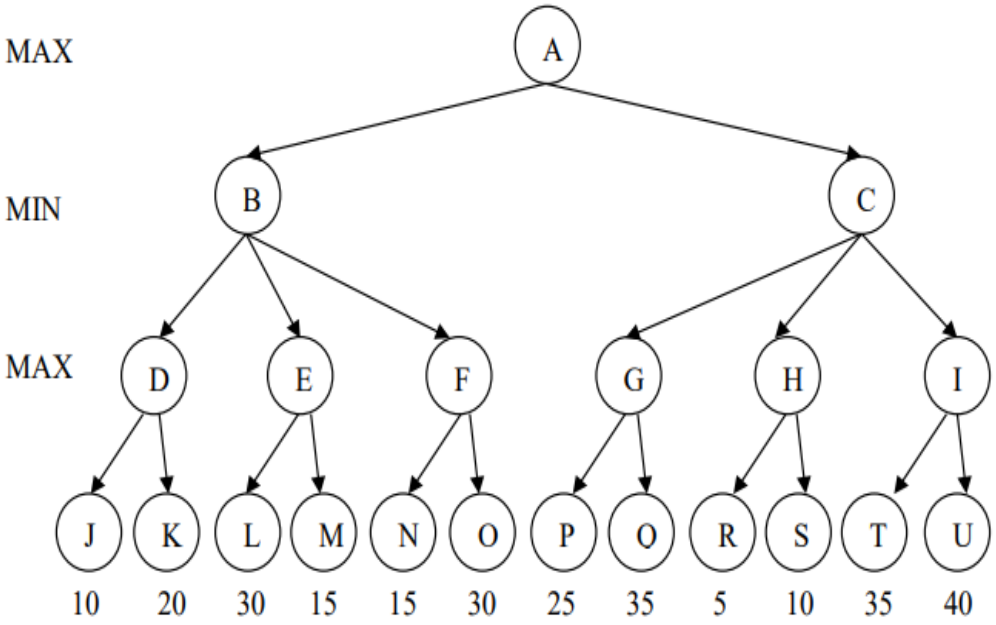
location() is set as follows: if the wheel position is "forward" then the controlled car's position is set to be v units in the direction it is facing, where v is its velocity. If the wheel position is "left" or "right" then its position is set to be v/√ 2 units (rounded to the nearest integer) in the direction it is facing and v/√ 2 units (again rounded to the nearest integer) in its relative "left" or "right" direction (respectively). (This is based on me handwaving over some physics things with trig functions that 3 I don't exactly remember, but you can throw in some cosines and sines and stuff to make it more accurate...)

(f) We are going to solve this problem using expectimax. However, we may not want to expand the entire tree. Give a reasonable heuristic utility for a given state S.

One heuristic could be just the negative of the maximum of (a) the minimum manhattan distance from the controlled car to the locations comprising the goal area minus the manhattan distance from the controlled car to the other car and (b) zero, in case (a) is below zero (we don't really want a heuristic function to have negative values, even though we technically wouldn't be breaking any rules if it did have them).

3) (Expectimax and Alpha-Beta Pruning)

Consider the following game tree. The root is a maximizing node, and children are visited left to right.



(a) Ignoring the values given above at the leaf nodes, could there possibly exist some set of values at the leaf nodes in the above tree so that Alpha-Beta pruning would do the following. Answer either Yes or No.

(i) Pruning occurs at the arc from D to K.

No

(ii) Pruning occurs at the arc from E to M.

Yes

(iii) Pruning occurs at the arc from C to G.

No

(b) In the tree above assume that the root node is a MAX node, nodes B and C are MIN nodes, and the nodes D, …, I are not MAX nodes but instead are positions where a fair coin is flipped (so going to each child has probability 0.5). What is the Expectimax value at node A?

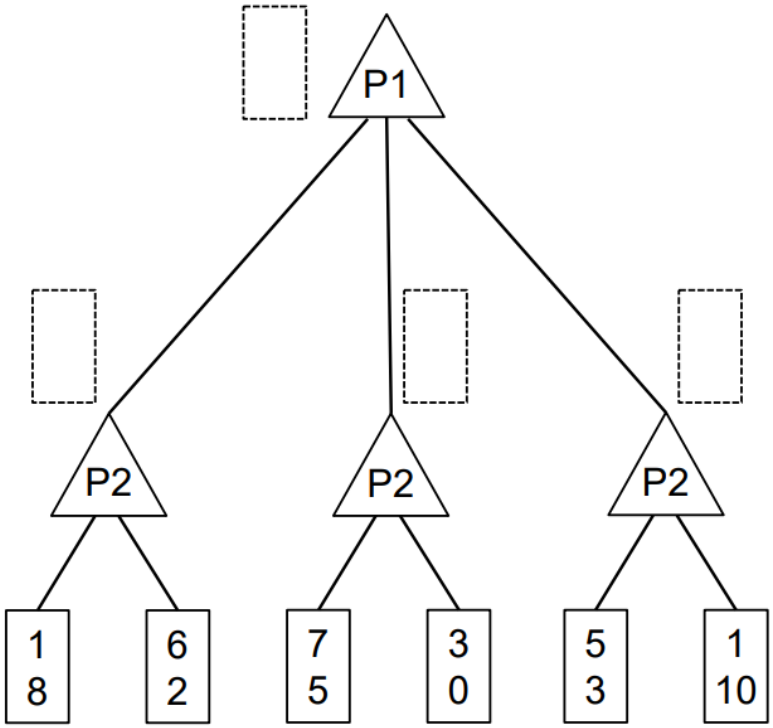
15

(c) True or False: When using Expectimax to compute the best move at the root, rescaling the values at all leaf nodes by multiplying them all by 10 may result in a different move at the root.

False

4) (MiniMax , Max-First Search and Expectimax)

For the following game tree, each player maximizes their respective utility. Let x, y respectively denote the top and bottom values in a node. Player 1 uses the utility function U1(x, y) = x.



(a) Both players know that Player 2 uses the utility function U2(x, y) = x − y.

(i) Fill in the rectangles in the figure above with pair of values returned by each max node.

From top-down, left-right: (6, 2), (6, 2), (3, 0), (5, 3)

(ii) You want to save computation time by using pruning in your game tree search. On the game tree above, put an 'X' on branches that do not need to be explored or simply write 'None'. Assume that branches are explored from left to right.

None

(b) Now assume Player 2 changes their utility function based on their mood. The probabilities of Player 2's utilities and mood are described in the following table. Let M, U respectively denote the mood and utility function of Player 2.

P(M = happy)	P(M = mad)
a	b

	M = happy	M = mad
P(U2(x, y) = −x | M)	c	f
P(U2(x, y) = x − y | M)	d	g
P(U2(x, y) = x 2 + y 2 | M)	e	h

Calculate the maximum expected utility of the game for Player 1 in terms of the values in the game tree and the tables. It may be useful to record and label your intermediate calculations. You may write your answer in terms of a max function.

We first calculate the new probabilities of each utility function as follows.

P (U2 (x, y) = −x)	P (U2 (x, y) = x − y)	P (U2 (x, y) = x 2 + y 2)
ac + bf	ad + bg	ae + bh

EU (Left Branch) = (ac + bf) (1) + (ad + bg) (6) + (ae + bh) (1)

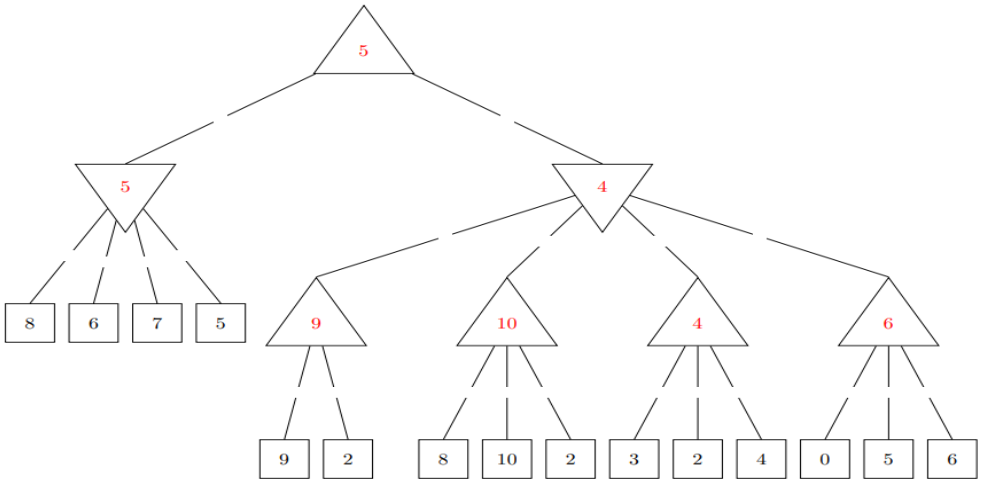
EU (Middle Branch) = (ac + bf) (3) + (ad + bg) (3) + (ae + bh) (7)

EU (Right Branch) = (ac + bf) (1) + (ad + bg) (5) + (ae + bh) (1)

MEU(φ) = max ((ac + bf) (1) + (ad + bg) (6) + (ae + bh) (1), (ac + bf) (3) + (ad + bg) (3) + (ae + bh) (7), (ac + bf) (1) + (ad + bg) (5) + (ae + bh) (1))

5) (MiniMax)

Minimax The first part is based upon the following tree. Upward triangle nodes are maximizer nodes and downward are minimizers. (small squares on edges will be used to mark pruned nodes in part (ii))



(i) Complete the game tree shown above by filling in values on the maximizer and minimizer nodes.

(ii) Can any edges be pruned? Explain.

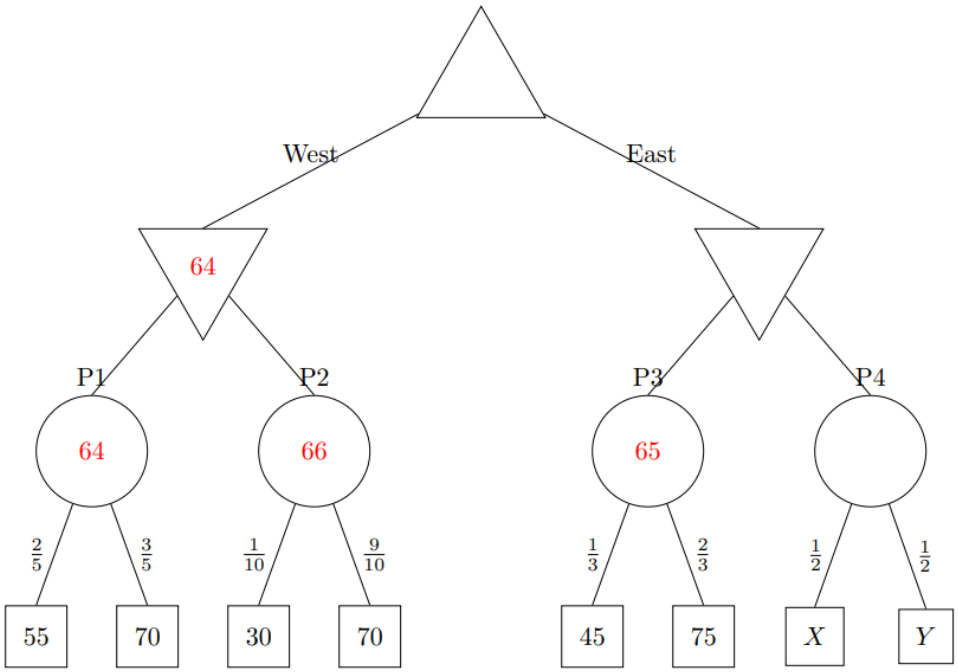
Edges that can be pruned: 10-2, 4-6

6) (Game theory and game tree (MiniMax))

Pacman is playing a tricky game. There are 4 portals to food dimensions. But, these portals are guarded by a ghost. Furthermore, neither Pacman nor the ghost know for sure how many pellets are behind each portal, though they know what options and probabilities there are for all but the last portal.

Pacman moves first, either moving West or East. After which, the ghost can block 1 of the portals available.

You have the following gametree. The maximizer node is Pacman. The minimizer nodes are ghosts and the portals are chance nodes with the probabilities indicated on the edges to the food. In the event of a tie, the left action is taken. Assume Pacman and the ghosts play optimally.



(i) Fill in values for the nodes that do not depend on X and Y .

(ii) What conditions must X and Y satisfy for Pacman to move East? What about to definitely reach the P4? Keep in mind that X and Y denote numbers of food pellets and must be whole numbers: X, Y ∈ {0, 1, 2, 3, . . . }.

To move East: X + Y > 128

To reach P4: X + Y = 129

The first thing to note is that, to pick A over B, value(A) > value(B).

Also, the expected value of the parent node of X and Y is $\frac{X+Y}{2}$.

$\Longrightarrow \min(65, \frac{X+Y}{2}) > 64$

$\Longrightarrow \frac{X+Y}{2} > 64$

So, X + Y > 128 $\Longrightarrow$ value(A) > value(B)

To ensure reaching X or Y, apart from the above, we also have $\frac{X+Y}{2} < 65$

$\Longrightarrow 128 < X + Y < 130$

So, X, Y $\in$ N $\Longrightarrow$ X + Y = 129