AI Planning for Autonomy

Problem Set I: Blind Search

1. Choose **one** of the problems listed below and describe a simple example along with its corresponding *State Model*.

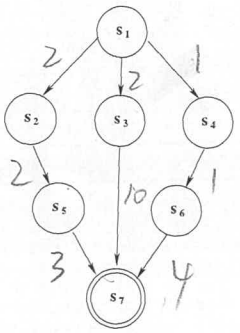
The problems are:

1. 8-Puzzle.
2. Travelling Salesman Problem

Definition should be brief, clear, and compact.[1]

- State space S
- Initial state $s_0 \in S$
- Set of *goal* states $S_G \subseteq S$
- Applicable actions function $A(s)$ for each state $s \in S$
- Transition function $f(s, a)$ for $s \in S$ and $a \in A(s)$
- Cost of each action $c(a, s)$ for $s \in S$ and $a \in A(s)$

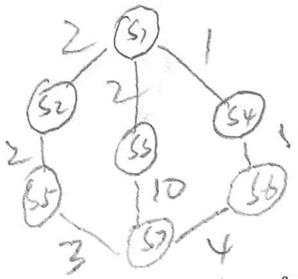
2. Consider the following state space S, where $s_0 = s_1$ and $S_G = \{s_7\}$



where actions changing a state s into another state s' are given by the edges. The cost to transition from state s to s' is given by the following table:

s	s'	$c(s, s')$	s	s'	$c(s, s')$
s_1	s_2	2	s_3	s_7	10
s_1	s_3	2	s_4	s_6	1
s_1	s_4	1	s_5	s_7	3
s_2	s_5	2	s_6	s_7	4

[1] *Compact* means using mathematical notation to define sets, i.e. $S = \{x | x \in V\}$ to define that there are as many states as elements in the set V, and pseudo-code, i.e. to define the transition function.



Describe the execution of Breadth First Search ($BrFS$), Depth First Search (DFS) and Iterative Deepening (ID) in this problem by filling in a table like the one below. Show the order in which nodes are expanded. Each node must be named, e.g. $n_3 = \langle s_3, g(n), n_{parent}\rangle$. The node should contain all the relevant information for the search: current state s_i, the accumulated cost of the path from the initial state s_0 to s_i, and a pointer to the parent node.

$s_1 \leftarrow s_3 \leftarrow s_7$

can't grantee optimality

	Breadth First Search
Ordered Sequence of States expanded Example: $\langle n_1 =< s_1, 0, - >, n_2 =< s_2, 2, n_1 >, \ldots \rangle$	$n_3 = < s_3, 2, n_1 >$, $n_4 = < s_4, 1, n_1 >$ $n_5 = < s_5, 4, n_2 >$, $n_6 = < s_7, 12, n_3 >$ stop

two much memory

can't grantee

	Depth First Search
Ordered Sequence of States expanded Example: $\langle n_1 =< s_1, 0, - >, n_2 =< s_2, 2, n_1 >, \ldots \rangle$	$n_3 = < s_5, 4, n_2 >$, $n_4 = < s_7, 7, n_3 >$ $s_1 \leftarrow s_2 \leftarrow s_5 \leftarrow s_7$

	Iterative Deepening
Ordered Sequence of States expanded Example: $\langle n_1 =< s_1, 0, - >, n_2 =< s_2, 2, n_1 >, \ldots \rangle$	

stuck

- Which is the solution found by each algorithm?
- Which is the optimal solution?
- Explain under which conditions the algorithms guarantee optimality? same cost → BFS, ID
- Adapt any of the previous algorithms to account for $g(n)$. Explain properties: optimality, complete, sound.

solutions can be achieved

not sound

BFS: 跑到goal，其它都 不管

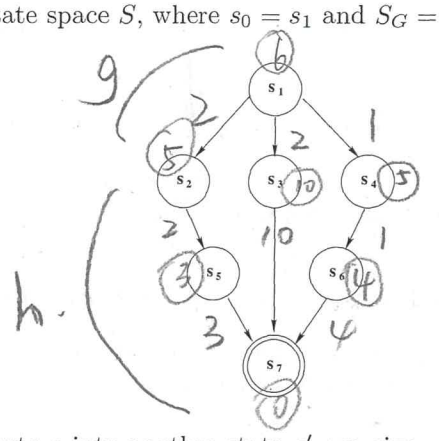
ID: DFS
$n=0$ $n_1 = < s_1, 0, - >$
$n=1$ $n_1 = < s_1, 0, - >$, $n_2 = < s_2, 2, n_1 >$, $n_3 = < s_3, 2, n_1 >$, $n_4 = < s_4, 1, n_1 >$
$n=2$ $n_1 = < s_1, 0, - >$, $n_2 = < s_2, 2, 1 >$, $n_3 = < s_5, 4, n_2 >$
$n_4 = < s_3, 2, n_1 >$, $n_5 = < s_7, 12, n_4 >$

BFS和ID: $s_1 \leftarrow s_3 \leftarrow s_7$

AI Planning for Autonomy

Problem Set II: Heuristic Search Continued

1. Consider the following state space S, where $s_0 = s_1$ and $S_G = \{s_7\}$



where actions changing a state s into another state s' are given by the edges. The cost to transition from state s to s' is given by the following table:

s	s'	$c(s,s')$	s	s'	$c(s,s')$
s_1	s_2	2	s_3	s_7	10
s_1	s_3	2	s_4	s_6	1
s_1	s_4	1	s_5	s_7	3
s_2	s_5	2	s_6	s_7	4

and heuristic estimates for each state:

s	$h_1(s)$	$h_2(s)$	$h_3(s)$	h^*
s_1	4	6	6	6
s_2	3	5	1	5
s_3	5	10	1	10
s_4	3	5	5	5
s_5	2	3	3	3
s_6	2	4	4	4
s_7	0	0	0	0

- Which heuristics are admissible?
- Which are consistent?
- Does any heuristic *dominate* any other?

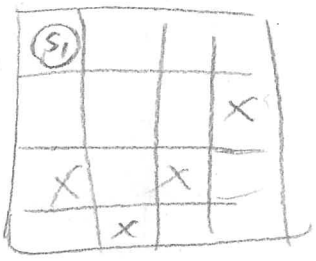
Describe the execution of one of the following algorithms in this problem using one of the heuristics above. Fill in a table like the one below, showing the contents of the OPEN and CLOSED lists at the end of each iteration.

Choose one of: A*, WA* ($w = 5$), or Greedy Best-First Search.

	Iteration 1	Iteration 2
OPEN	$n_1 = \langle s_1, 6, 0, nil \rangle$ *	$n_2 = \langle s_2, 5, 2, n_1 \rangle$ $n_3 = \langle s_3, 10, 2, n_1 \rangle$ $n_4 = \langle s_4, 5, 1, n_2 \rangle$
CLOSED		n_1

- Which is the path returned as a solution?
- Is this the optimal plan? Has the algorithm proved this?

2. Consider an $m \times m$ manhattan grid, and a set of coordinates G to visit in any order.
   - Formulate a state-based search problem to find a tour of all the desired points (i.e. define a state space, applicable actions, transition and cost functions).
   - What is the branching factor of the search?
   - What is the size of the state space in terms of m and G.
   - Define an admissible heuristic function.



AI Planning for Autonomy

Problem Set III: Choosing Heuristics

1. Consider a $m \times m$ manhattan grid, and a set of coordinates V to visit in any order, and a set of inaccessible coordinates (walls) W.
   Using the state space below:

$$
\begin{aligned}
S &= \{\langle x, y, v\rangle \,|\, x, y \in [0..m] \wedge v \subseteq V\} \\
S_0 &= \langle 0, 0, V\rangle \\
A(\langle x, y, v\rangle) &= \{\langle dx, dy\rangle \,|\, dx, dy \in \{-1, 0, 1\} \\
&\qquad \wedge\ |dx| + |dy| = 1 \\
&\qquad \wedge\ \langle x + dx, y + dy\rangle \notin W\} \\
t(\langle dx, dy\rangle, \langle x, y, v\rangle) &= \langle x + dx, y + dy, \\
&\qquad v - \{\langle x + dx, y + dy\rangle\}\rangle \\
c(a, s) &= 1 \\
G &= \{\langle x, y, v\rangle \,|\, \langle x, y, v\rangle \in S \wedge v = \emptyset\}
\end{aligned}
$$

   - Explain the meaning of x, y and v in each state $s \in S$
   - Define 3 different heuristics for this problem.
   - Which of your heuristics is admissible? consistent? dominates the others?
   - Estimate the complexity of calculating each of your heuristics.
   - Which would you use in A*? Why?

2. Reformulate the state-model from Q1 as a STRIPS problem $P = \langle F, O, I, G\rangle$.
3. Write pseudo code for the following search algorithms:
   Feel free to implement these in python in the appropriate places in search.py for assignment 1.
   - Breadth First
   - Depth First
   - A Star
   - Uniform Cost

AI Planning for Autonomy

Problem Set IV: Modeling in STRIPS and PDDL

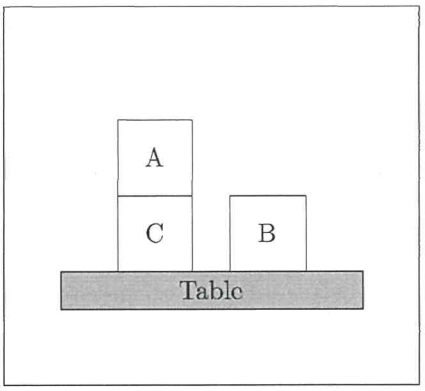


Figure 1: A blocks-world problem.

In blocks-world, the agent's aim is to stack the blocks in one tower with A on B and B on C. The agent can hold up to one block at a time and can put blocks down on the table, or another block.

1. Model Blocks-World as a STRIPS problem $P = \langle F, O, I, G \rangle$: define the set of facts F, the set of operators O, the goal facts G and the initial facts I. You must also define the *pre*, *add*, and *del* functions.

2. Implement your STRIPS model in PDDL. Use `http://editor.planning.domains` to test your model. Remember that a PDDL implementation is split between two files: a domain file (also known as an "operator" file) and a problem file (also known as a "fact" file).

For inspiration, the Australian TSP example from lectures is implemented in PDDL in the figures below. It is also accessible in editor.planning.domains so you can try the solver on the cloud, and edit the TSP: editor.planning.domains link with Domain and Problem file. See `http://www.hakank.org/pddl/` for more examples.

3. Blockworld can be modeled with only 2 actions instead of 4. The robot can pick up a block and put it down on another block (or the table) in a single action. You've got actions `Move(Block,FromTable,ToBlock)` and `Move(Block, FromBlock, ToTable)` You now no longer need to keep track of what the robot is holding or if the hand is empty.

Implement a STRIPS model of this "2-operation" blocks-world in PDDL. Use `http://editor.planning.domains` to write your model. The integrated solver is very limited, so you will want to download docker and install the planners in your computer by typing the command: *docker pull lapkt/lapkt-public*

See `https://hub.docker.com/r/lapkt/lapkt-public/` for more instructions on how to run the available planners

set of Block B

$F = \{on(x,y), onTable(x), clear(x), holding(x), armfree(), | x, y \in B\}$

$A =$

```
(define (domain tsp)
(:requirements :typing)
(:types node)

;; Define the facts in the problem
;; "?" denotes a variable, "-" a type

(:predicates (move ?from ?to - node)
    (at ?pos - node)
    (connected ?start ?end - node)
    (visited ?end - node))

;; Define the action(s)

(:action move
    :parameters (?start ?end - node)
    :precondition (and (at ?start)
        (connected ?start ?end))
    :effect  (and (at ?end)
        (visited ?end)
    (not (at ?start)))))
```

Figure 2: tsp-domain.pddl

```
(define (problem tsp-01)
(:domain tsp)
(:objects Sydney Adelade Brisbane Perth Darwin - node)

;; Define the initial situation
(:init (connected Sydney Brisbane)
    (connected Brisbane Sydney)
    (connected Adelade Sydney)
    (connected Sydney Adelade)
    (connected Adelade Perth)
    (connected Perth Adelade)
    (connected Adelade Darwin)
    (connected Darwin Adelade)
    (at Sydney))
(:goal
    (and (at Sydney)
    (visited Sydney)
    (visited Adelade)
    (visited Brisbane)
    (visited Perth)
    (visited Darwin))))
```

Figure 3: tsp-problem.pddl

AI Planning for Autonomy

Problem Set V: Delete Relaxation

1. Discuss in your group the heuristics you used in project 1. Are any of them related to the domain independent heuristics we have covered in class?

- What is the (optimal) delete relaxation heuristic h^+? How would it be interpreted in pacman?
- What is the relationship between h^{max}, h^+, and h^{add}? What about h^*?

$h^{max} \ll h^+ \ll h^*$. $h^{max} \ll h^+ \ll h^{add}$. h^* dominates admissible heuristic. h^* dominates h^{add}

2. In a blocks-world problem, the agent's aim is to stack the blocks as in Figure 1.

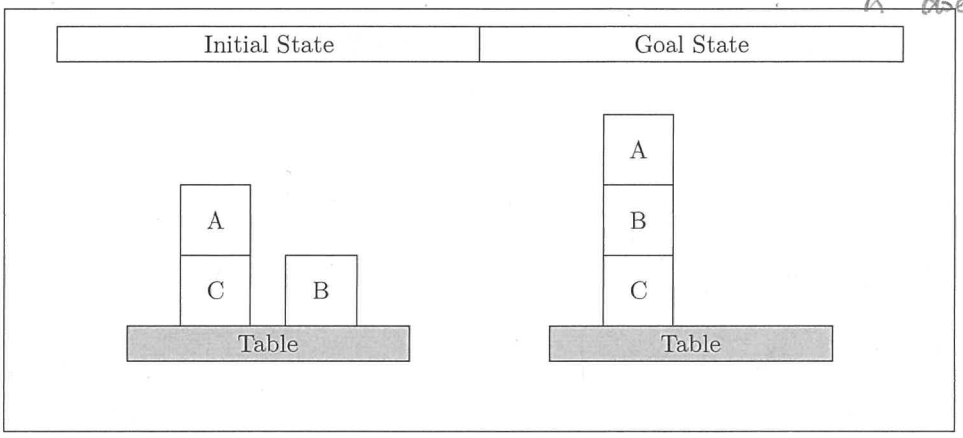


Figure 1: An Initial (Left hand side) and Goal (Right hand side) state of a blocks-world problem.

There are several important classes of domain-independent heuristics. Recall the delete relaxation based heuristics from Lectures:

- Compute $h^{add}(s_0)$ for the 4 operators blocks-world problem.
- Compute $h^{max}(s_0)$ for the 4 operators blocks-world problem.

G = {on(A,B), on(B,C)}

h° = min c(a,s) + pre

Iteration	cl(A)	cl(B)	cl(C)	onTable(A)	onTable(B)	onTable(C)	on(A,C)	on(A,B)	on(B,C)	h(A)	h(B)	h(C)	armFree
0	0	0	∞	∞	0	0	0	∞	∞	∞	∞	∞	0
1	0	0	1	∞	0	0	0	∞	∞	1	1	2	0
2	0	0	1	2	0	0	0	2	3 (add) / 2 (max)	1	1	2	0

F(CC) = unstack(A,C)
↳ pre: on(A,C), armFree, cl(A)
0 0 0

onTable(A): putdown(A)
↳ pre = holding(A), ∞

on(A,B) = stack(A,B)
↳ pre = holding(A), cl(B)
∞ 0

on(B,C) = stack(B,C)
↳ pre = h(B), cl(C)
1 1

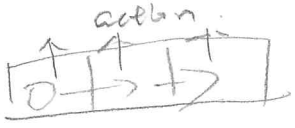
h(A) = unstack(A,C)
↳ pre: on(A,C), armFree, cl(A)

h(B) = pickup(B)
↳ pre = cl(B), onTable(B)

h(C) = pickup(C)
↳ pre = onTable(C), clear(C)
0 1

$\therefore h^{add} = 2+3 = 5$

$h^{max} = \max(2,2) = 2$



have freedom to do all actions.

keep rewards.

$f = \{at(0,0)\}$

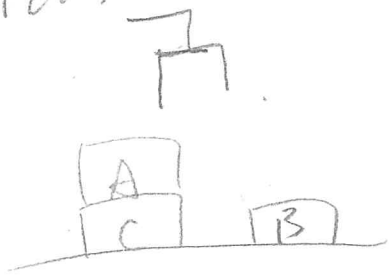
$f_1 = \{at(0,0), at(0,1)\}$

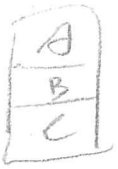
$f_2 = \{at(a,0), at(0,1), at(0,2)\}$

8

6

never overestimated





$G = \{on(A,B), (B,C)\}$

$\downarrow$

stack $(A,B) \Rightarrow c(a) = 1$

$\downarrow$

$pre = \{clear(B), holding(A)\}$

$\downarrow$

unstack (A,C)

$\downarrow$

$pre = \{hF, cl(A), on(A,C)\}$

unfree

$$h^{add}(s,A) = \begin{cases} 0 & , g \subseteq s \\ \min_{a \in A,\ g \in add\ a} c(a) + h^{add}(s, pre) & , |g| = 1 \\ \sum h^{add}(s, g') & |g| > 1 \end{cases}$$

add	cl(A)	cl(B)	cl(C)	ot(A)	ot(B)	ot(C) (on table)	on(A,C)	on(A,B)	on(B,C)	h(A)	h(B)	h(C)	hF
0	0	0	∞	∞	0	0	0	∞	∞	∞	∞	∞	0
1	0	0	1	∞	0	0	0	∞	∞	1	1	∞	0
2	0	0	1	2	0	0	0	2	3	1	1	2	0

$h(A) \rightarrow$ unstack $(A,C) \rightarrow pre = \{cl(A), hF, on(A,C)\}$ $0 + 0 + 0 = 0$

$cl(C) \rightarrow$ unstack $(A,C) \rightarrow c(a) = 1$

$pre = \{on(A,C), cl(A), af\}$

$0 + 0 + 0$ $h^{add}(pre) = 0$ $\rightarrow 1$

put down $(C) \rightarrow h(C) = \infty$

unstack (B,C)

stack (C,A)

stack (C,B)

$\} h^{add}(pre) > 0$

$\rightarrow \{2,3\} = h^{add} =$

$G^{max} = \{2,2\} = h^{max} = 2 \leq h^*$

$ot(A) \rightarrow$ put down $(A) \rightarrow c(a) = 1$

$\hookrightarrow pre = \{h(A)\}$

∞

$on(A,B) \Rightarrow \infty$

$\hookrightarrow$ stack $(A,B) \rightarrow c(a) = 1$

$\hookrightarrow pre = \{cl(B), h(A)\}$

$0, \infty$

AI Planning for Autonomy

Problem Set VI: Relaxed Plan Heuristic and Iterated Width

1. In a blocks-world problem, the agent's aim is to stack the blocks as in Figure 1.

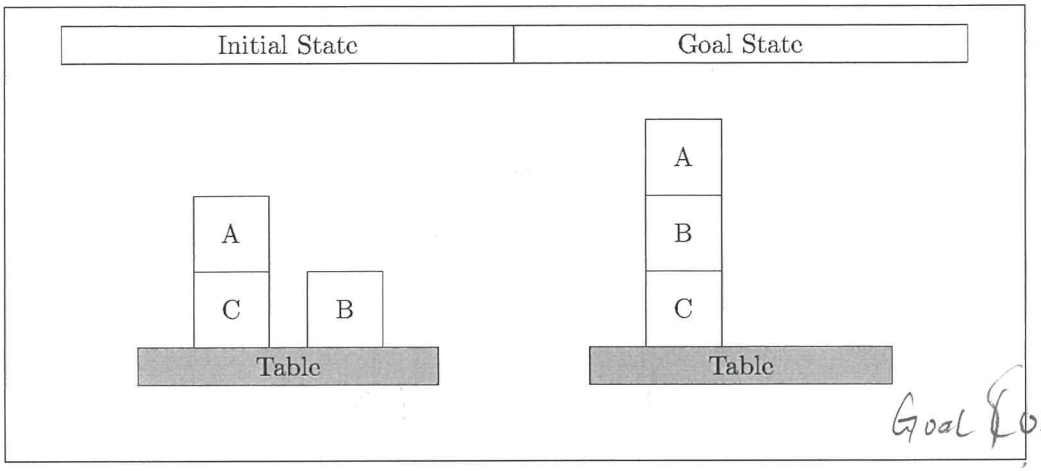


Figure 1: An Initial (Left hand side) and Goal (Right hand side) state of a blocks-world problem.

Compute the values of each of the following heuristics for this problem

- h^{ff}: Use h^{max} for the best-supporters function.
- h^{ff}: Use h^{add} for the best-supporters function.

2. Given the Blocks-World domain, with the initial state where blocks *A B, and C* are on the table, and goal *A on B*:

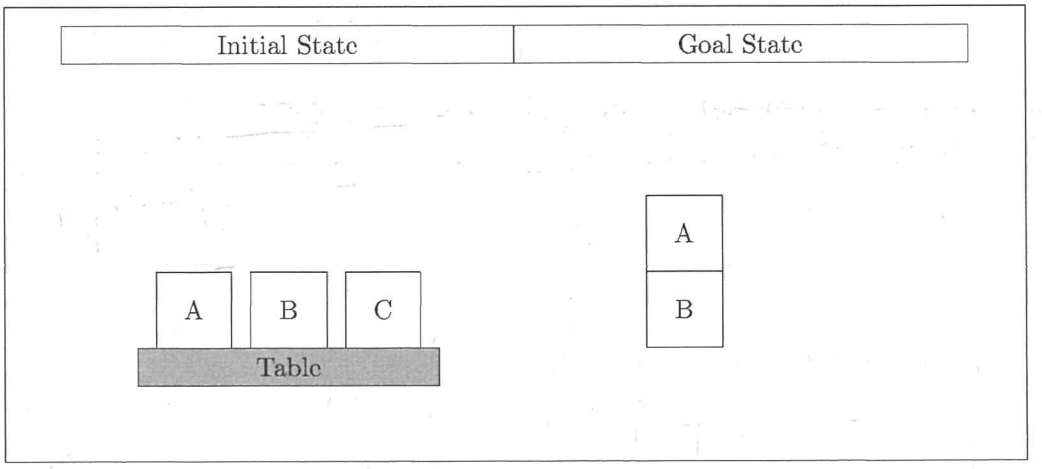


Figure 2: An Initial (Left hand side) and Goal (Right hand side) state of a blocks-world problem.

bs function doesn't change based on h^{max} and h^{add}. Initial = {cl(A), cl(B), on(A,C), onTable(B); on(Table(C), aF

bs(on(A,B)) = ~~unstack(A,B)~~ → pre = cl(B), h(A)

bs(on(B,C)) = ~~unstack(B,C)~~ → pre = cl(C), h(B)

bs(h(A)) = unstack(A) → pre = {on(A,C), clear(A), armFree}

bs(cl(C)) = ~~pickup(C) → pre = {onTable(C), cl(C), armFree}~~

unstack(A)

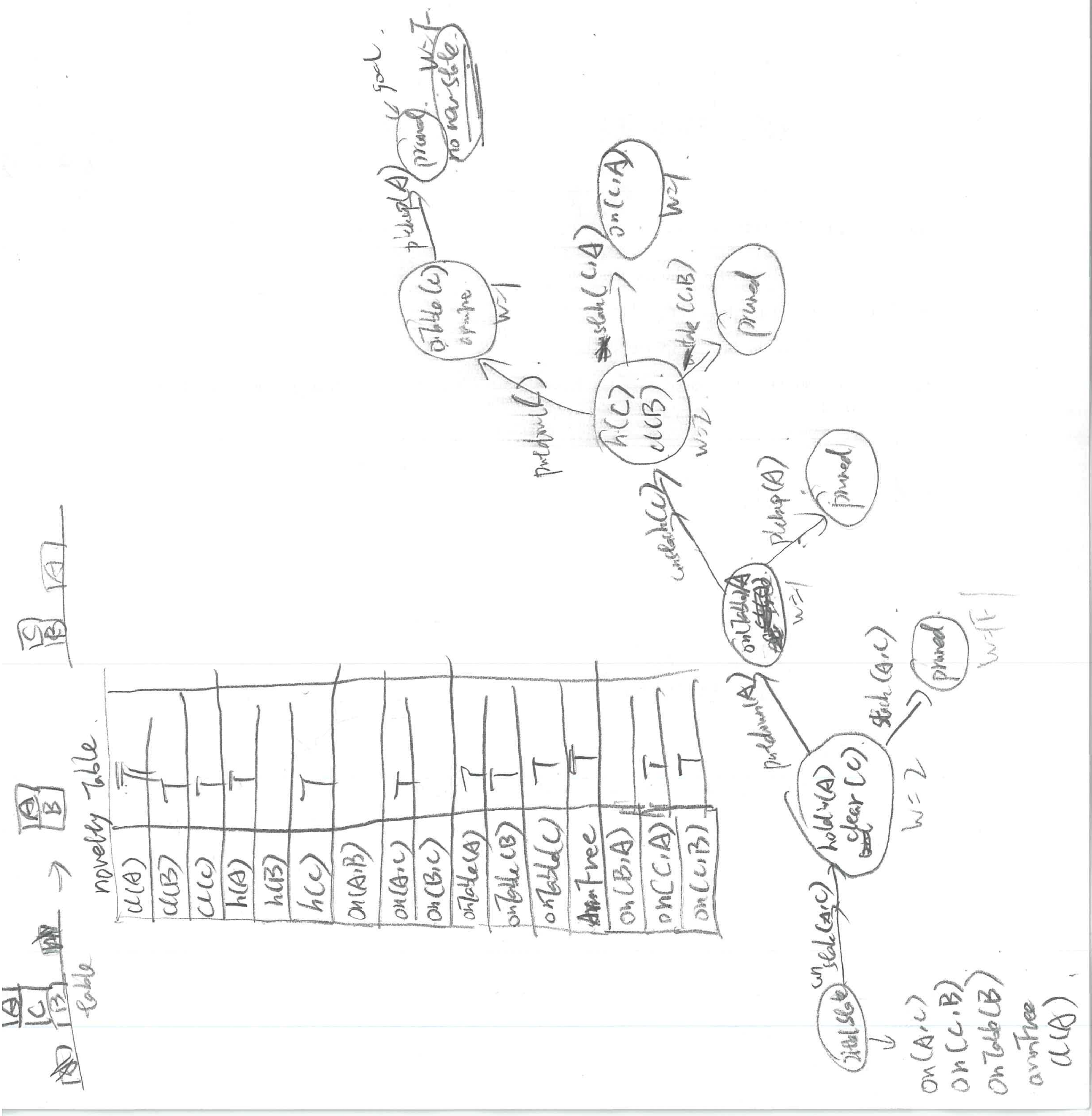
bs(h(B)) = pickup(B) → pre = {onTable(B), clear(B), armFree}

relaxed plan = {stack(A,B), stack(B,C), unstack(A), pickup(B)}

$h^{ff} = 4$

pruned

states that do not add new atom for the first time are pruned.



novelty Table

cl(A)	T
cl(B)	T
cl(C)	T
h(A)	T
h(B)	
h(C)	T
On(A,B)	
On(A,C)	T
On(B,C)	
OnTable(A)	T
OnTable(B)	T
OnTable(C)	T
armFree	T
On(B,A)	
On(C,A)	T
On(C,B)	T

- Show the IW(1) search tree for this problem, highlighting each state why it passes the novelty pruning test or why is being pruned. IW(1) should solve this problem. Stop as soon as you find a state the satisfies the goal condition.
- Can you think of an initial situation where IW(1) cannot find a solution for the goal on(A,B), but IW(2) does, explain your answer?

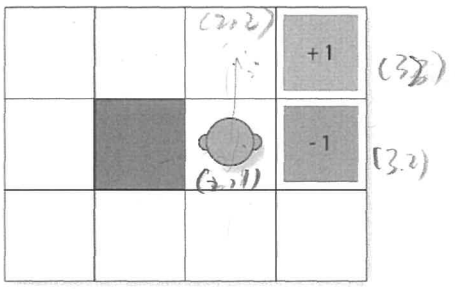
  Methodology: Draw a table for each fluent tuple, i.e subset of fluents of size k, where k is the bound used, and progress the search, marking for each node of Breadth First Search which tuples were made true for the first time. (Initialize the table with the initial state)

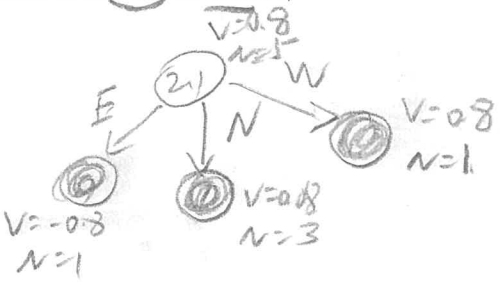
Problem Set VIII: Monte-Carlo Tree Search

Aim The purpose of this workshop is to help you get a better understanding of Monte-Carlo Tree Search for solving MDPs in an online manner.

Tasks

In this workshop, you will consider the example from the lectures of the agent that moves in a 2D grid world. Remember that if the agent tries to move in a particular direction, there is an 80% of success, and a 10% chance of it going to the left or right.





1. The agent is at cell (2,1), in which 2 is the x-coordinate and 1 the y-coordinate (both start from 0). It samples the following 5 iterations of MCTS, where E (East) goes right, W (West) goes left, N (North) goes up, and S (South) goes down:

Iter	Trace	
1	N	$simulate(succ) = -1$
2	$N \xrightarrow{succ} E$	$simulate(slip(S)) = -1$
3	E	$simulate(succ) = -1$
4	W	$simulate(succ) = 1$
5	$N \xrightarrow{succ} S$	$simulate(slip(W)) = 1$

Here, $N \xrightarrow{succ} E$ means that we select N, then select that the outcome of N was successful, and then select E. The notation $N \xrightarrow{slip(E)} S$ means that we select N and the agent went East instead; then select South.

The notation $simulate(X) = r$ means having just expanded a node, simulate from the outcome X, which will return reward r.

Draw the MCTS tree for this, assuming $\lambda = 1.0$. Label the lines on the tree with the actions & outcomes and label the nodes with the value $V(s)$ for each state and N (the number of times this has been visited).

2. Based on your tree, calculate the action with the highest expected return.

3. Based on your tree, which of action, North, South, East, or West, would be more likely to be chosen if we use UCT to probabilistically select the next action? Show your working. Assume that $C_p = \frac{1}{2}$.

$$Q(s,a) = \sum P_a(s'|s)[r(s,a,s') + \gamma V(s')]$$

$$V(s) = \max Q(s,a)$$

$$Q(s,a) = \max P_a(s'|s)[r(s,a,s') + \gamma V(s')]$$

Problem Set VII: Value & Policy Iteration

Aim The purpose of this workshop is to help you get a better understanding of MDPs, value iteration, and policy iteration.

Consider two football-playing robots: Messi and Suarez.

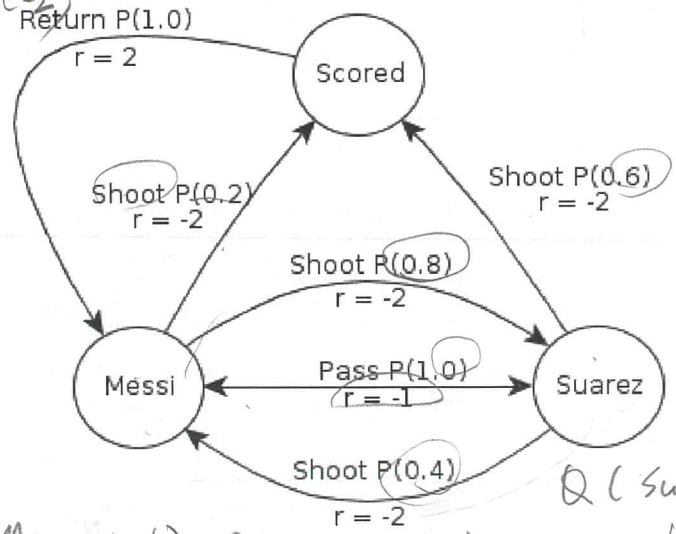
They play a simple two-player cooperate game of football, and you need to write a controller for them. Each player can pass the ball or can shoot at goal.

The football game can be modelled as a discounted-reward MDP with three states: *Messi*, *Suarez* (denoting who has the ball), and *Scored* (denoting that a goal has been scored); and the following action descriptions:

- If Messi shoots, he has 0.2 chance of scoring a goal and a 0.8 chance of the ball going to Suarez. Shooting towards the goal incurs a cost of 2 (or a reward of -2).
- If Suarez shoots, he has 0.6 chance of scoring a goal and a 0.4 chance of the ball going to Messi. Shooting towards the goal incurs a cost of 2 (or a reward of -2).
- If either player passes, the ball will reach its intended target with a probability of 1.0. Passing the ball incurs a cost 1 (or a reward of -1)
- If a goal is scored, the only action is to return the ball to Messi, which has a probability of 1.0 and has a reward of 2. Thus the reward for scoring is modelled by giving a reward of 2 when *leaving* the goal state.

The following diagram shows the transition probabilities and rewards:

$$V(\text{scored}) = P_{\text{return}}(\text{Messi}|\text{scored})[r(\text{scored}, \text{return}, \text{Messi}) + \gamma V(\text{Messi})$$
$$= 1 \times (2 + 1 \times (-2))$$
$$= 0$$



For $V(\text{Messi}) = \max(Q(\text{Messi}, \text{shoot}), Q(\text{Messi}, \text{pass}))$
$= \max(-2.76, -2.2)$
$= -2.2$

$Q(\text{Suarez}, \text{pass})$
$= 1 \times (-1 + 1 \times (-2))$
$= -3$

$V(\text{Suarez}) = \max[Q(\text{Suarez}, \text{shoot}), Q(\text{Suarez}, \text{pass})] = \max(-2.2, -3) = -2.2$

$Q(\text{Suarez}, \text{shoot}) = P_{\text{shoot}}(\text{Messi}|\text{Suarez})[r(\text{Suarez}, \text{shoot}, \text{Messi}) + \gamma V(\text{Messi})]$
$+ P_{\text{shoot}}(\text{Scored}|\text{Suarez})[r(\text{Suarez}, \text{shoot}, \text{Scored}) + \gamma V(\text{Scored})]$
$= 0.4 \times (-2 + 1 \times (-2)) + 0.6 \times (-2 + 1 \times 1) = -1.96 + (-0.6) = -2.2$

① $Q(s,a) = \sum_{s' \in S} P_a(s'|s)[r(s,a,s') + \gamma V(s')]$ ☆

$Q(\text{Messi}, \text{pass}) = P_{\text{pass}}(\text{Suarez}|\text{Messi})[r(\text{Messi}, \text{pass}, \text{Suarez}) + \gamma V(\text{Suarez})]$

$= 1 \times [-1 + 1 \times (-1.2)] = -2.2$

$Q(\text{Messi}, \text{shoot}) = P_{\text{shoot}}(\text{Suarez}|\text{Messi})[r(\text{Messi}, \text{shoot}, \text{Suarez}) + \gamma V(\text{Suarez})]$

$= 0.8 \times [-2 + 1 \times (-1.2)] + 0.2(-2 + 1 \times 1) = -2.76$

$\therefore$ pass

Tasks

1. Assume that we have calculated the following *non-optimal* value function V for this problem using value iteration with $\gamma = 1.0$, after iteration 2 we arrive at the following:

Iteration		0	1	2	3
V(Messi)	=	0.0	-1.0	-2.0	-2.2
V(Suarez)	=	0.0	-1.0	-1.2	-2.2
V(Scored)	=	0.0	2.0	1.0	0

If Messi has the ball (the system is in the Messi state), what action should we choose to maximise our reward in the next state: pass or shoot? Assume we are using the values for V after three iterations.

2. Complete the values of these states for iteration 3 using value iteration. Show your working.

3. Consider the following policy update table and policy evaluation table, with discount factor $\gamma = 0.8$:

Iter	$Q^\pi(Messi, P)$	$Q^\pi(Messi, S)$	$Q^\pi(Suarez, P)$	$Q^\pi(Suarez, S)$	$Q^\pi(Scored)$
0	0	0	0	0	0
1	-5	-5.52	-5	-4.56	-2
2	-4.194	-5.465	-4.355	-3.993	-1.355

Apply two iterations of policy iteration. Finish both tables and show the working for the policy evaluation and policy update.

What is the policy after two iterations?

Iter	$\pi(Messi)$	$\pi(Suarez)$	$\pi(Scored)$
0	Pass	Pass	Return
1	pass	shoot	Return
2	pass	shoot	Return

Additional Tasks for Personal Study

To improve your understanding of value iteration, try completing the first question of Project 3 at `http://ai.berkeley.edu/reinforcement.html#Q1`. You can download all the necessary files to complete this task.

Hints In order to help you complete the task during the workshop, here are some useful hints:

1. The functions that you need to change:
   (a) _init_
   (b) computeQValueFromValue
   (c) computeActionFromValue
2. The files you need to take a look:
   (a) util.py (Counter())
   (b) mdp.py (isTerminal(), getStates() ,getPossibleActions(), etc.)
   (c) other files: (gridworld.py, learningAgent.py) not directly related
3. How to test your code:
   (a) python autograder.py -q q1 (testing by autograde)
   (b) python gridworld.py -a value -i 5 (result after 5 iteration)
   (c) python gridworld.py -a value -i 100 -k 10 (how value iteration works)

AI Planning for Autonomy

Problem Set IX: Reinforcement Learning

Recall the following description from Problem Set VII.

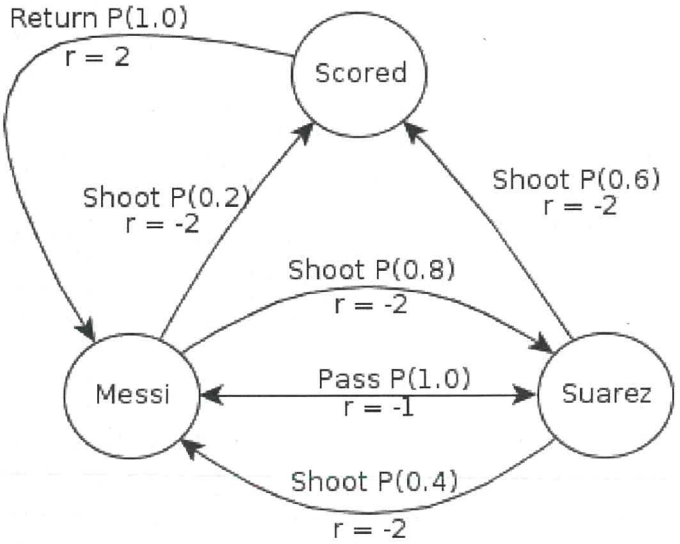
Consider two football-playing robots: Messi and Suarez.

They play a simple two-player cooperate game of football, and you need to write a controller for them. Each player can pass the ball or can shoot at goal.

The football game can be modelled as a discounted-reward MDP with three states: *Messi*, *Suarez* (denoting who has the ball), and *Scored* (denoting that a goal has been scored); and the following action descriptions:

- If Messi shoots, he has 0.2 chance of scoring a goal and a 0.8 chance of the ball going to Suarez. Shooting towards the goal incurs a cost of 2 (or a reward of -2).
- If Suarez shoots, he has 0.6 chance of scoring a goal and a 0.4 chance of the ball going to Messi. Shooting towards the goal incurs a cost of 2 (or a reward of -2).
- If either player passes, the ball will reach its intended target with a probability of 1.0. Passing the ball incurs a cost 1 (or a reward of -1).
- If a goal is scored, the only action is to return the ball to Messi, which has a probability of 1.0 and has a reward of 2. Thus the reward for scoring is modelled by giving a reward of 2 when *leaving* the goal state.

The following diagram shows the transition probabilities and rewards:



Chose actions by same policy.

Tasks

1. What is the difference between Sarsa and Q-learning? on off max/max Q-value
2. Assume the following Q-table, which is learnt after several episodes:

$Q(s,a) = Q(s,a) + \alpha[r(s,a,s') + \gamma \max Q(s',a') - Q(s,a)]$ ☆

$Q(s,a) = Q(s,a) + \alpha[r(s,a,s') + \gamma \max Q(s',a') - Q(s,a)]$

State	Action		
	Pass	Shoot	Return
Messi	-0.4	-0.8	
Suarez	-0.7	-0.2	
Scored			1.2

In the next step of the episode, from the state 'Suarez', Suarez passes the ball to Messi. Show the Q-learning update for this action using a discount factor $\gamma = 0.9$ and learning rate $\alpha = 0.4$.

Note: As this is a reinforcement learning problem, assume that the transition probabilities are not accessible to your algorithm.

3. Consider again being in the state 'Suarez'. Suarez passes the ball to Messi and then Messi decides to shoot. Show the SARSA update for the Pass action using a discount factor $\gamma = 0.9$ and learning rate $\alpha = 0.4$ and assuming a' (the next action to be execute) is Shoot. Compare to the Q-learning update. What is different?

4. Given the following trace from a historical game feed from last season:

   " Suarez passes the ball to Messi. Messi dribbles around all of his opponents, shoots and scores yet another goal! Barcelona F.C 10 - 0 Real Madrid! The ball is returned to Messi for kickoff. After he passes the ball to Suarez, the referee blew the final whistle. End of the game, the ball is taken by Messi to remember the match forever."

   Show the 3-step SARSA update for the above feed. Do you think the 1-step update is more accurate or the 3-step update? Does it indicate more steps is always better? Explain why.

Additional Tasks for Personal Study

To improve your understanding of reinforcement learning and how to work with Q functions, pick up from where you left of last week and complete the *fourth* question of Project 3 at `http://ai.berkeley.edu/reinforcement.html#Q4`. You should already have all the necessary files to complete this task.

Hints In order to help you complete the task, here are some useful hints:

1. The functions that you need to change:
   a) computeValueFromQValues
   b) getQValue
   c) computeActionFromQValues.
2. The files you need to take a look:
   a) util.py (Counter())
   b) featureExtractors.py
3. How to test your code:
   a) python gridworld.py -a q -k 5 -m (to get your agent to learn via keyboard input)
   b) python autograder.py -q q4 (testing by autograde)

AI Planning for Autonomy

Problem Set X: Game Theory

1. Auctions are a classic example of game theory at work. Although the analysis of auctions is more advanced than what we have looked at in these note mainly because the set of strategies (the bids) is so large , we can paint a picture of how they work.

   Probably most of you are familiar with the concept of an *English auction*, in which the bidding starts at a price and bids are incremented until nobody wants to bid any higher. The winner is the person who bids the highest.

   But how many of you have heard of *Dutch auctions*? This is where the bidding starts at a (presumably high value) and the auctioneer *decrements* the bids by a certain amount. The first bidder to signal wins the auction at that price.

   Another common form of auction is a *Vickrey auction*, also known as a *sealed-bid, second-price auction*. In Vickrey auctions, each bidder submits a single bid privately to the auctioneer, and the person with the highest bid wins. However, they only pay what the price of the second-highest bid.

   Why do they only pay the second-highest bid? To eliminate the dreaded "*winner's curse*" which essentially is that if you pay more than anyone else in a market, then you must have overpaid, because you cannot possibly turn around and sell that item back to the same market for what you paid for it (otherwise someone else would have bid for it). However, if you only pay the second-price, then you could (in theory) sell the item to the second-highest bidder for what you paid for it.

   For the first task in the workshop, your tutor will lead you through some auction exercises.

2. Consider a Vickrey auction between two bidders, Bidder A and Bidder B, who each value the item at \$50 and \$51 respectively. Assume that the buyer has no 'reserve' price.

   a) Are there (weakly or strictly) dominant strategies for these two bidders? If so, what are they? To simplify your analysis, assume that bids can only be whole numbers between 49 and 52.

payoff = value - payment

(+1, 49)

		Bidder B (val = 51)			
		49	50	51	52
Bidder A (val = 50)	49	(½, 1)	0, 2	0, 2	0, 2
	50	(1, 0)	0, ½	0, 1	0, 1
	51	(1, 0)	0, 0	-½, 0	0, 0
	52	(1, 0)	0, 0	-1, 0	-½, ½

   hints: Your first task should be to determine the payoffs in matrix. For example, what is the payoff for Bidder A if they bid 51 and win the auction? Assume that if BOTH players win the auction (because they bid the same amount), the item is divided between them evenly (they both receive half of the payoff) and they both pay half of the bid. Note that if they both bid the same amount, the second price is equal to the first price.

If B tied 49, => A (50+1, 52) ... 50 for A

B = 50 => A (49, 50 51 52)

B = 51 => A (49, 50)

B = 52 => A (49, 50, 51)

+1 for B.

For A = 49 => B = 50 +1 +2

50 => B = +1 +2

51 => B = 49, 50, +1 +2

52 => B = 49, 50, +1

b) What are the equilibrium states in this simplified game?

c) **[Challenge question]** (not covered in the workshop): Use the concepts introduced in lectures, can you argue what the (weakly) dominant strategy is for each bidder in a Vickrey auction for N number of players and any value? For simplicity, assume that no two bidders have the same private value (in practice this is not much more difficult to reason about).

3. Consider the following two games, and assume that you are playing as the Row player.

Game 1:

		Column player: L	Column player: R
Row player: T		320, 40	40, 80
Row player: B		40, 80	80, 40

Game 2:

		Column player: L	Column player: R
Row player: T		44, 40	40, 80
Row player: B		40, 80	80, 40

Note that the only difference between the two games is the payoff 320 vs. 44 in the top-left cell.

a) What strategy would you choose in each game?

b) Calculate the mixed strategy that both players should play in both of these games. How close was your intuition to the mixed strategy that you calculated?

For Game 2

(a) For Game 1

For Column player

$E(L) = 40x + 80(1-x) = -40x + 80$

$E(R) = 80x + 40(1-x) = 40x + 40$

$E(L) = E(R) \quad -40x + 80 = 40x + 40$

$80x = 40$

$x = \frac{1}{2}$

$\therefore \begin{cases} x = \frac{1}{2} \\ 1-x = \frac{1}{2} \end{cases}$

For Row player

$E(T) = 320y + 40(1-y) = 280y + 40$

$E(B) = 40y + 80(1-y) = -40y + 80$

$E(T) = E(B)$

$280y + 40 = -40y + 80$

$320y = 40$

$y = \frac{1}{8}$ $\quad \begin{cases} y = \frac{1}{8} \\ 1-y = \frac{7}{8} \end{cases}$