16.410: Principles of Autonomy and Decision Making

Sample Final Exam

December 3rd, 2004

Name	
E-mail	

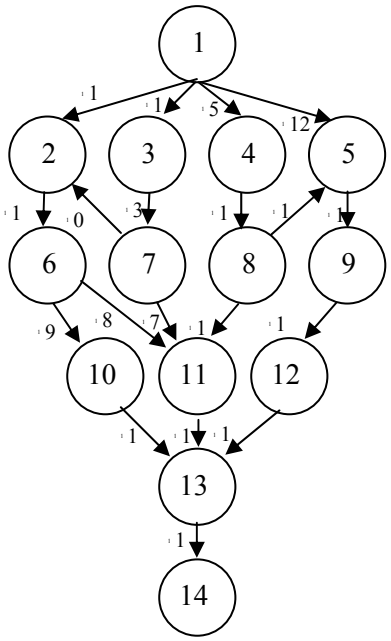
Note: Budget your time wisely. Some parts of this exam could take you much longer than others. Move on if you are stuck and return to the problem later.

Problem Number	Max	Score	Grader
Problem 1	20		
Problem 2	20		
Problem 3	20		
Problem 4	27		
Problem 5	30		
Problem 6	30		
Problem 7	14		
Problem 8	15		
Problem 9	15		
Total	191		

Problem 1 – Search (20 points)

Part A – Dijkstra's Algorithm (6 points)

Consider the following graph, with node 1 as the start, and node 9 as the goal.



Part A-1 – (2 points)

How many times does the value at node 11 change?

Part A-2 – (2 points)

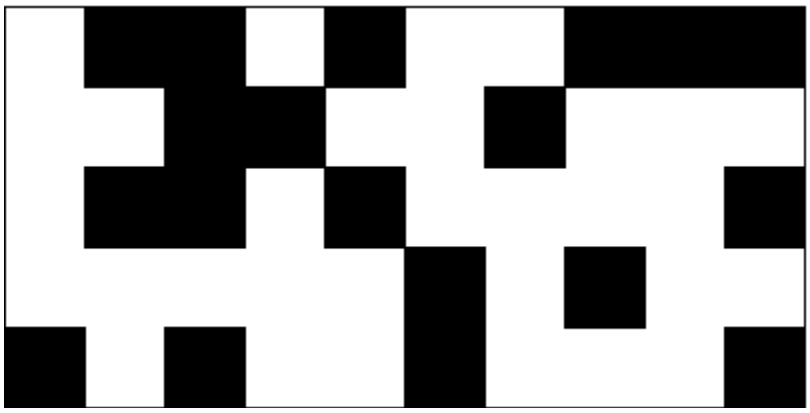
What is the length of the shortest route from start to goal?

Part A-3 – (2 points)

In what order are the nodes expanded?

Part B – A* Search (8 points)

Consider the following maze. Actions are moves to the 8 neighboring squares. Each such move involves a move over a distance, which is the cost of the move. Let's say that the distance to the left and right neighbors, and to the upper and lower neighbors is 1, and that the distance to the corner neighbors is $\sqrt{2}$.



Part B-1 – (4 points)

The Manhattan distance is not an admissible heuristic. Can it be made admissible by adding weights to the x and y terms? (The Manhattan distance between two points <x1,y1> and <x2,y2> is [|x2 – x1| + |y2 – y1|]. A weighted version would be [α|x2 – x1| + β|y2 – y1|].) .

Part B-2 – (4 points)

Is the L-∞ distance an admissible heuristic? Why or why not? (The L-∞ distance between two points <x1,y1> and <x2,y2> is (max(|x2 – x1|, |y2 – y1|)).

Part C – Properties of Search (6 points)

Part C-1 – (1 point)

How can A* be made to behave just like breadth-first search?

Part C-2 – (1 point)

How can depth-first search be made to behave just like breadth-first search?

Part C-3 – (2 points)

Is A* always the fastest search method? Explain your answer.

Part C-4 – (2 points)

Is breadth first always slower than depth first search? Explain your answer.

Problem 2 – Planning with Graphplan (20 points)

Consider the following set of initial facts and operators.

Initial facts:

(Item Brian)
(Item Laptop)
(City Boston)
(City PaloAlto)
(City Ames)
(Plane USAir-1)
(Plane USAir-2)

(in Boston Brian)
(in PaloAlto Laptop)
(in Boston USAir-1)
(in PaloAlto USAir-2)

Operators:

```
(operator board
        (parameters (Item x) (Plane y) (City z))
        (preconditions (in z x) (in z y))
        (effects (on y x))
)

(operator fly
        (parameters (Plane x) (City y) (City z))
        (preconditions (in y x))
        (effects (in z x))
)

(operator deplane
        (parameters (Item x) (Plane y) (City z))
        (preconditions (on y x) (in z y))
        (effects (in z x))
)
```

Part A – Get Brian from Boston to his laptop in Palo Alto (5 points)

What is the goal fact?

How many different propositional symbols result from instantiating fly(x, y, z) for the first ten levels of the plan graph, given that the initial facts have already been instantiated?

Part B – Get Brian's laptop from Palo Alto to Boston using Graphplan (15 points)

Fill in the plan graph, showing level 1 operators and level 1 and 2 facts (but no mutexes).

Problem 3 – Propositional Logic and Inference (20 points)

Consider a theory comprised of the following six clauses:

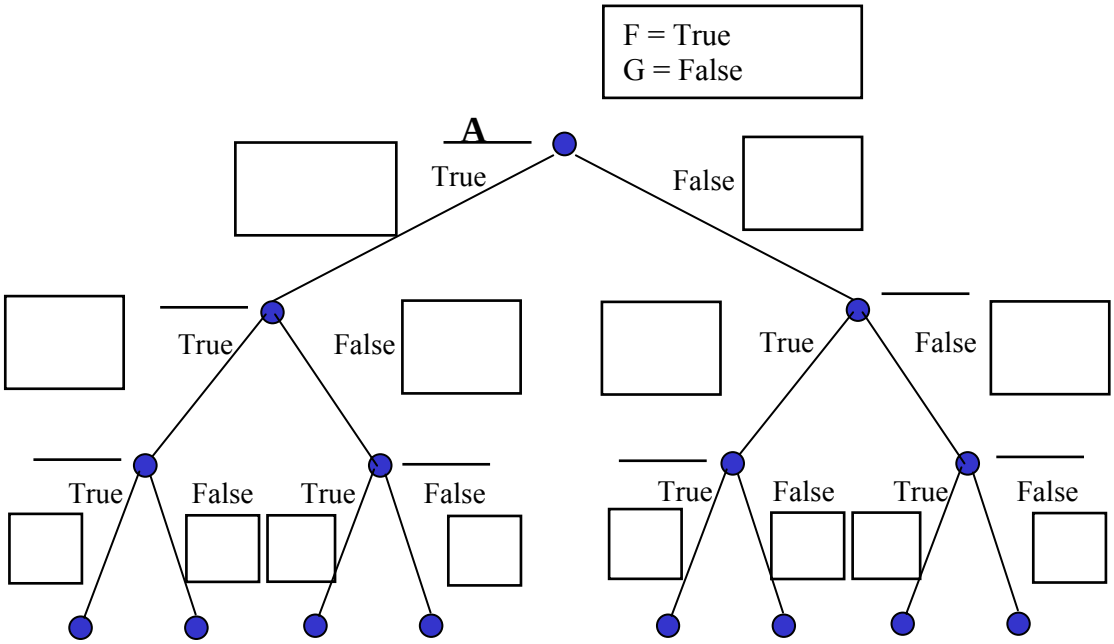
not A or B;
not B or C;
not C or not D or E;
not D or not E;
not F or not G;
F.

Part A – Satisfiability Using DPLL (15 points)

Use the DPLL algorithm (backtrack search plus unit propagation) to **find a truth assignment** to propositions A, B, C, D, E, F and G, that is consistent with the theory. **Fill out the search tree supplied below, stopping at the first consistent assignment found.**

- **Search** the propositions **in alphabetical order** (no other order please!).
- For each proposition P, **assign** the value **True before** trying **False**.
- On the line next to each node in the tree, **write the proposition being assigned** a truth value at that point in the search.
- In the box next to each branch**, list all propositions** whose truth value is **determined by unit propagation** based on the assignment to the proposition at that branch.
- **Indicate** the **truth value derived** for each of these propositions.
- **Draw an X** at each node that is immediately below the branch **where at least one clause becomes false**; this is where the search backtracks.
- **Circle each** node that denotes a **complete and consistent assignment** to the propositions A **–** G.

We filled out the result of the initial propagation in the box above the tree. In addition, we filled out the first variable to be assigned (A), next to the root.



Part B – Satisfiability with Backtracking + Forward Checking (5 points)

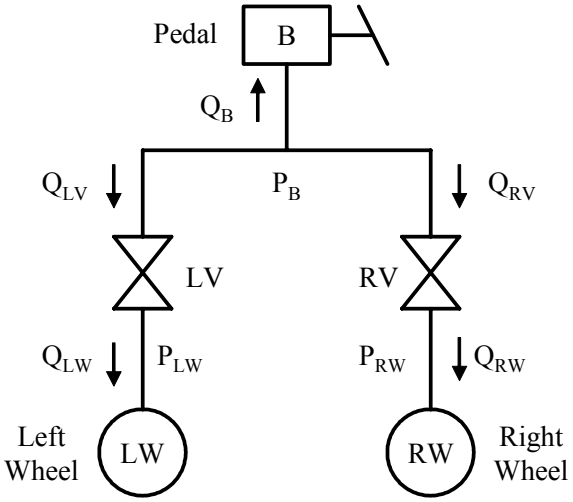
Consider solving the same problem (finding the first consistent assignment for the theory) using **backtrack search plus forward checking**. Would the depth of the search tree be different, that is, would backtrack search plus forward checking assign the **same** number of variables, **fewer** variables, or **more** variables than the DPLL algorithm? Give a brief explanation for your answer.

Circle one of Fewer, Same or More
Why?

Problem 4 – Model-based Diagnosis (27 points)

You are having trouble with your car. Each time you brake, the car drags to the right (the left wheel is underbraked and the right wheel is overbraked), while the brake pedal feels harder than normal. Let's see if what you have learned in 16.410 can help you find the cause.

The reference manual of the car states that the hydraulic circuit consists of the pedal brake cylinder B, the left and right wheel brake cylinders LW and RW, and two valves LV and RV:



The fluid flow in the pipes (in the direction indicated by the arrows) is denoted Q_B, Q_{LV}, Q_{RV}, Q_{LW}, and Q_{RW}. The pressure in the pipes is denoted P_B, P_{LW}, and P_{RW}.

If a valve is working correctly, then the flow across it is proportional to the pressure difference in the adjacent pipes. If a pedal or wheel brake cylinder is working correctly, then the flow into it is proportional to the pressure in the adjacent pipe. The brake fluid is incompressible, so the flow sums up to zero at the junction of the pipes to the left and right branches.

The next page shows a constraint-based model of the hydraulic circuit as described above. Each variable can assume one of the three values "low", "nominal", and "high", abbreviated as "-", "0", and "+". Each constraint lists the possible combinations of values if the component works correctly (denoted "G"). In this representation, the observations of the car's strange behavior can be expressed as P_B="+", P_{LW}="-", P_{RW}="+".

B:

P_B	Q_B
-	-
0	0
+	+

LW:

P_{LW}	Q_{LW}
-	-
0	0
+	+

RW:

P_{RW}	Q_{RW}
-	-
0	0
+	+

LV:

Q_{LV}	Q_{LW}	P_B	P_{LW}
-	-	-	-
-	-	-	0
-	-	-	+
-	-	0	+
-	-	+	+
0	0	-	-
0	0	0	0
0	0	+	+
+	+	-	-
+	+	0	-
+	+	+	-
+	+	+	0
+	+	+	+

RV:

Q_{RV}	Q_{RW}	P_B	P_{RW}
-	-	-	-
-	-	-	0
-	-	-	+
-	-	0	+
-	-	+	+
0	0	-	-
0	0	0	0
0	0	+	+
+	+	-	-
+	+	0	-
+	+	+	-
+	+	+	0
+	+	+	+

Pipe Junction:

Q_{LV}	Q_B	Q_{RV}
+	-	-
-	+	-
+	0	-
0	+	-
+	+	-
+	-	0
0	0	0
-	+	0
+	-	+
-	+	+
-	0	+
0	-	+
-	-	+

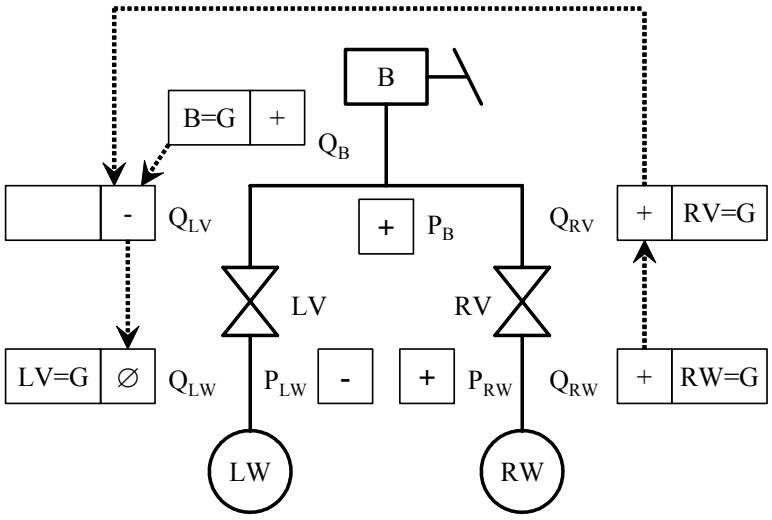
In the lecture, you learned how to find symptoms by applying unit propagation to a set of clauses, and how to extract conflicts from inconsistent clauses by tracing support. Now you will **generalize** propagation and conflict extraction **from clauses to constraints**.

Recall that in unit propagation, we look for clauses where **all literals except one** are false. We then **assign** true to the remaining literal, and **record the clause as a support** for this literal.

Constraint propagation generalizes this in the following way: In constraint propagation, we look for constraints where **all values of a variable except one** are excluded. We then **assign** this remaining value to the variable, and **record the constraint as a support** for this assignment.

For example, consider the constraint for LV. If P_B = "+" and P_{LW} = "-", then Q_{LV} is restricted to the single value "+" and Q_{LW} is restricted to the single value "+". We assign these values to the variables and record "LV=G" as a support for Q_{LV} = "+" and as a support for Q_{LW} = "+".

In the diagram below, we applied constraint propagation to the model above, given the observations P_B="+", P_{LW}="-", P_{RW}="+". The support for each predicted value is shown next to that value. The pipes are assumed to be ok, so the pipe junction is not included as a support.



Part A – Conflict Extraction from Support [4 points]

As shown in the diagram above, the model and the observations are **inconsistent** with the assertion that every component is correct: constraint LV has become empty, as there is no tuple in the constraint LV that allows for Q_{LV} ="-", P_B = "+", and P_{LW} = "-". **Extract the conflict** by tracing back the support for the predictions (you should be able to do this by inspection):

Part B – Constraint Suspension for Fault Localization [20 points]

Let's now see what single failures can account for the symptoms you observed in your car. Find this out by successively **suspending constraints**.

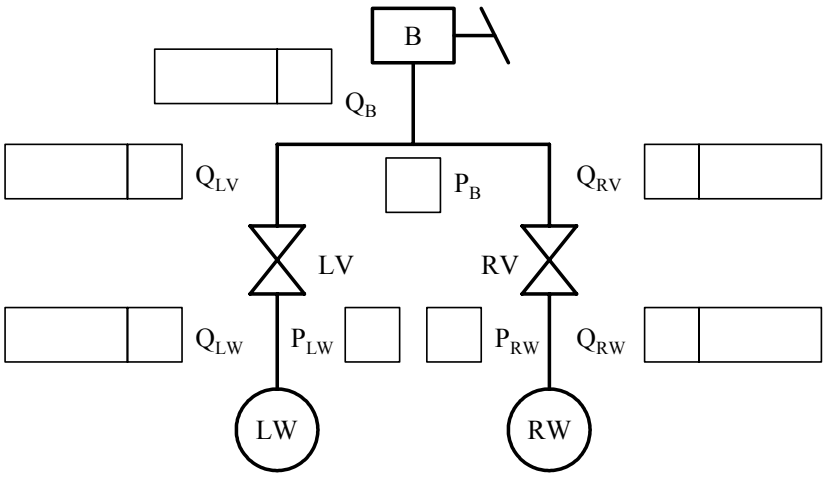
Show your results on the following diagrams, one for each candidate. For each diagram, write at the top if the candidate **is subsumed by (can be pruned by) conflicts** that have been discovered so far. **If the candidate is not subsumed by conflicts discovered so far**, then:

- **Cross out** the suspended constraint.
- Perform constraint propagation and write the **predicted values** in the box next to the corresponding variable. You can stop propagation as soon as you have found that the candidate is inconsistent.
- Write the **support** in the box next to the predicted value. Do not record the pipe junction as support, we assume it to be fault-free.
- Write at the top of each diagram whether or not the candidate is **consistent**.
- If the candidate is inconsistent, extract a **conflict** and write it on top of the diagram.

Recall, the observations are P_B="+", P_{LW}="-", P_{RW}="+".

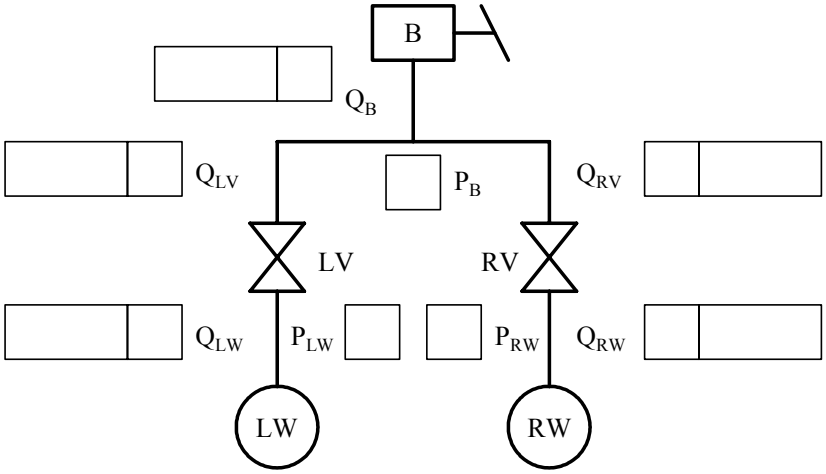
Candidate B:

Subsumed by conflicts (yes/no)? _____ Consistent (yes/no)? _____
Conflict: _____



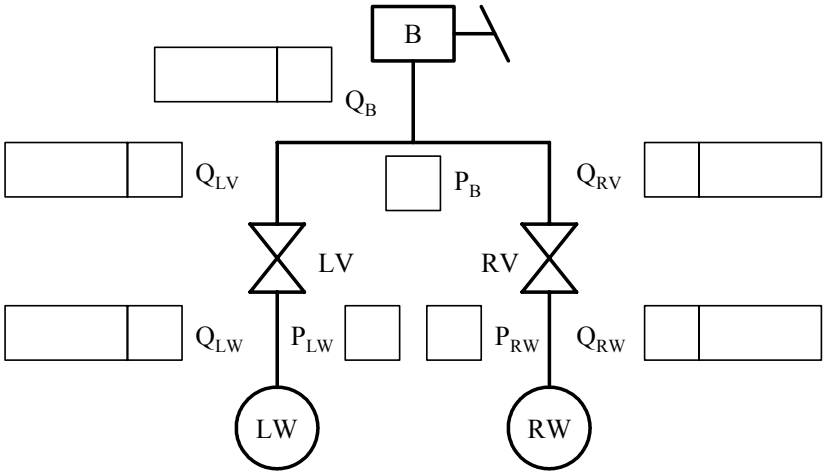
Candidate LV:

Subsumed by conflicts (yes/no)? _____ Consistent (yes/no)? _____
Conflict: _____



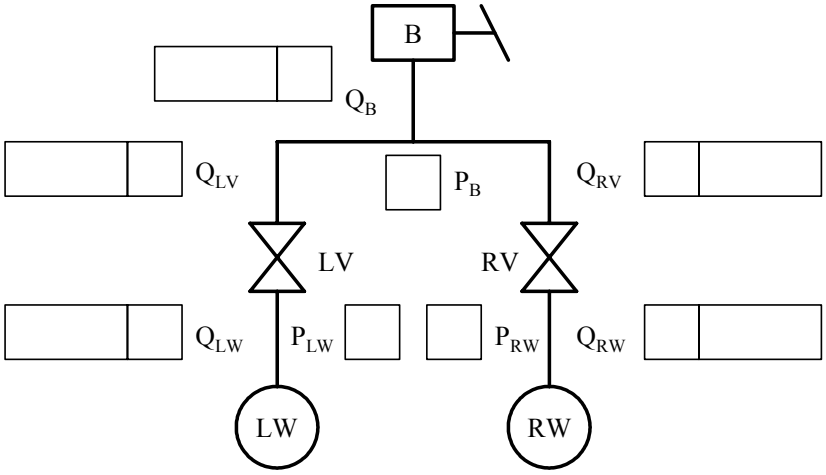
Candidate RV:

Subsumed by conflicts (yes/no)? _____ Consistent (yes/no)? _____
Conflict: _____



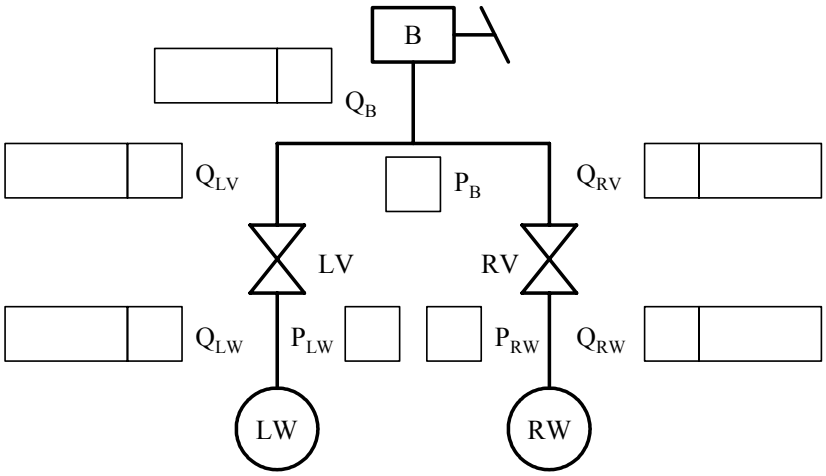
Candidate LW:

Subsumed by conflicts (yes/no)? _____ Consistent (yes/no)? _____
Conflict: _____



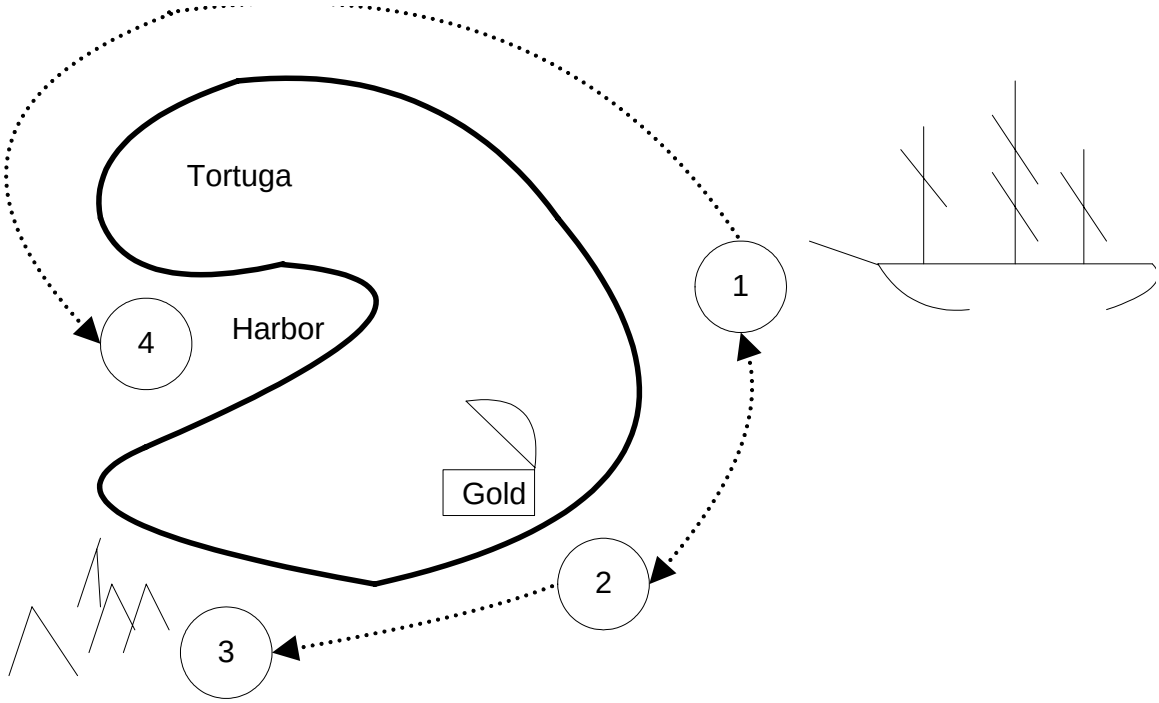
Candidate RW:

Subsumed by conflicts (yes/no)? _____ Consistent (yes/no)? _____
Conflict: _____



Problem 5 – Navigation by MDP (30 points)

Captain Jack Sparrow, infamous pirate, has sailed his ship to the eastern side of the island of Tortuga (see chart below).



Captain Jack would like to anchor in the harbor on the western side. Let's help him by using an ancient navigation technique that is known to all sailors worth their salt: value iteration.

First, let's consider some details shown on the chart. There are four locations, with the dotted arrows indicating valid moves between them. We will assume that all moves are deterministic. Location 3 represents rocks that will sink the ship, so there are no actions that lead out of this state. Similarly, location 4 represents the goal, and there are no actions that lead out of location 4 (Captain Jack wants to relax after he has anchored). In order to relax, Captain Jack needs some gold. Fortunately, he remembers that he has previously stashed some at location 2.

Let's assume that any time the ship reaches location 2, and the ship is not carrying the gold, the gold is automatically loaded onto the ship.

To use an MDP formulation, we need some notion of reward. Let's assume that location 4, the goal, has a reward of 1000. Also, let's assume that location 3 has a reward of

-1000 (because the ship sinks). Finally, let's assume that location 2 has a reward of 200 if the ship has not yet picked up the gold. Rewards for any actions in all other states are 0.

Part A - Modeling

In this part, you will design an MDP model for this problem. Assume that the state vector consists of the following two variables (with corresponding possible values):

Location (1, 2, 3, 4)
Ship-has-gold (true, false)

Part A-1 (5 points)

Write the transition function for this problem. Note that the system is deterministic.

Current state	Action	Next state

Part A-2 (5 points)

Write the reward function for this problem.

Current state	Action	Reward

Part B – Value Iteration (10 points)

Use value iteration to determine the values of each state. Perform two iterations using a discount factor of $\gamma = 0.9$. Assume that the initial value for all states corresponding to location 3 is -1000, the initial value for all states corresponding to location 4 is 1000, and the initial value for all other states is 0.

State	Initial value	Iteration 1 value	Iteration 2 value

Part C – Optimal Policy (5 points)

What is the optimal policy for each state?

State	Action

Part D – Effect of discount factor (5 points)

What would be the effect of changing the discount factor from 0.9 to 0.5?

Problem 6 - Scheming with CNF (30 points)

For this problem, we'll use a subset of propositional logic. In particular, we won't worry about negation (`not`), and the `and` and `or` operators are binary (as opposed to n-ary). Well-formed formulas (WFF's) are either symbols (e.g. `a`), or a conjunction of two WFF's, (e.g. (`and` ... ...)), or a disjunction of two WFF's, (e.g. (`or` ... ...)). We can write the grammar for WFF's as:

WFF :: = Symbol
| (`or` WFF WFF)
| (`and` WFF WFF)

The following are WFF's:

```
1. a
2. (or a b)
3. (and a b)
4. (and (or a b) c)
5. (and (or a b) (and c (or d e)))
6. (and a (or (and b c) d))
7. (or (or a (and b c)) (and (or d e) f)))
```

A WFF in *Disjunctive Normal Form* (*DNF*) is a disjunction of clauses, where each clause is either a symbol or a disjunctive normal clause. A single symbol can be considered a degenerate DNF. For example, formulas 1 and 2, above, are in DNF. We can write the grammar for DNF formulas as:

DNF ::= Symbol
| (`or` DNF DNF)

A WFF in *Conjunctive Normal Form (CNF)* consists of a conjunction of WFF's such that each conjunct is either in CNF itself or is in DNF. In the list of formulas above, WFF's 1 through 5 are in CNF. Note that formulas 1 and 2 can be considered to consist of a single conjunct. A grammar for CNF formulas is:

CNF ::= Symbol
| (`and` CNF CNF)
| (`or` DNF DNF)

An important step in converting logical formulas to CNF is to distribute `or` over `and` whenever possible. For example, we can convert (`or` A (`and` B C)) to (`and` (`or` A B) (`or` A C)), and we can convert (`or` (`and` A B) C) to (`and` (`or` A C) (`or` B C))

Part A – Convert to CNF Manually (10 points)

Convert the following WFF's to CNF by repeatedly applying `or` distribution.
Show intermediate steps, so that if you make a mistake we can give you partial credit.

`(and a (or (and b c) d))`
`(or (or a b) (and c d))`
(or (or a (and b c)) (and (or d e) f))

Part B – Scheme Code to Convert to CNF (20 points)

Following is the skeleton of a function to convert a WFF into CNF by distributing `or` over `and`:

```
(define first car)
(define second cadr)
(define third caddr)
;; Assume formulas are well-formed, and conform to the following grammar:
;;   wff ::= symbol | (and wff wff) | (or wff wff)
(define (and-clause? wff) (and (list? wff) (eq? 'and (first wff))))
(define (or-clause? wff) (and (list? wff) (eq? 'or (first wff))))
;; Distribute OR over AND in a well-formed formula (wff).
;; 1. (or (and A B) C) => (and (or A C) (or B C))
;; 2. (or A (and B C)) => (and (or A B) (or A C))
;; Note that in the above, if A, B, or C are not symbols or simple
;; disjunctive clauses, they will need to be further converted, so
;; that we wind up with, in general, wff's of the form
;; cnf  ::= symbol | (and cnf cnf) | (or dnf dnf)
;; dnf  ::= symbol | (or dnf dnf)
(define (distribute-or wff)           ; => CNF
  (cond
   ((symbol? wff)              ; symbol
    wff)
   ((and-clause? wff)                 ; an AND clause
    (list 'and
        ;; [1] construct 1st conjuct here
        ;; [2] construct 1st conjuct here
   ((or-clause? wff)                  ; an OR clause
    (let ((disj1 (distribute-or (second wff))) ; get disjuncts
          (disj2 (distribute-or (third wff)))) ; into CNF
      (cond
       ((and-clause? disj1)           ; case 1, above
       (let ((a (second disj1))
             (b (third disj1))
             (c disj2))
         (list 'and
             ;; [3] construct 1st conjuct here
             ;; [4] construct 1st conjuct here
        ((and-clause? disj2)           ; case 2, above
       (let ((a disj1)
             (b (second disj2))
             (c (third disj2)))
         (list 'and
             ;; [5] construct 1st conjuct here
             ;; [6] construct 1st conjuct here
        (else
       (list 'or
             ;; [7] construct 1st disjunct here
             ;; [8] construct 1st disjunct here
             )))))))
```

Provide the eight Scheme expressions that should replace the eight commented lines in the code of the form "[n] construct ... here".

1.
2.
3.
4.
5.
6.
7.
8.

Problem 7 - Mixed Integer Programming (14 points)

Part A. Formulation with fixed costs (7 points)

Formulate the following mixed integer linear program. Stacy's Subway Company needs to supply 5 subway cars to Manhattan. The alternatives for shipping include transporting them by truck over a bridge (at a cost of $2000 each) or transporting them by ferry (at a cost of $1000 each). However, the permit process in Manhattan is absurd and so there is a one time red-tape cost of $6000 for transporting any subway cars in trucks over a bridge and a one time red-tape cost of $10,000 for transporting any subway cars by ferry.

Formulate a MILP to minimize the transportation costs.

Part B. Branch and Bound Search Tree (7 points)

Solve the following mixed integer linear problem using Branch and Bound.

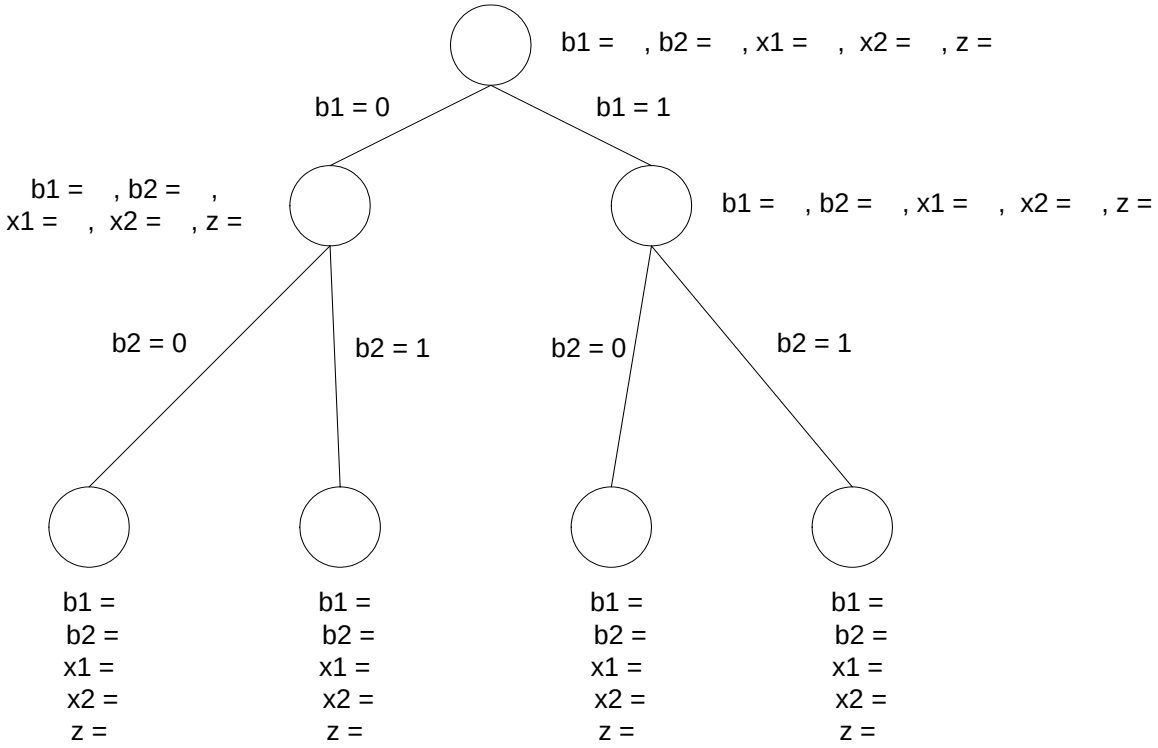
Minimize $z = 10x_1 + 4x_2 + 20b_1 + 25b_2$

subject to
$2x_1 + x_2 \geq 10$
$x_1 \leq 10b_1$
$x_2 \leq 10b_2$
$x1 \geq 0, x2 \geq 0$

$b_1 + b_2 \leq 1$
b_1, b_2 are binary (i.e., $b_i \in \{1,0\}$).

Fill in the branch and bound search tree below. Branch on the binary variables in the **following order**: b_1, b_2. Evaluate the **0** branch **before** the **1** branch. Cross off each node that is infeasible

or fathomed. For feasible, non-fathomed nodes, give the relaxed solution and the value of Z. For fathomed nodes, give the solution and value of Z.



Problem 8 - Decision Trees (15 points)

City planners have accumulated the following data on a variety of threats to urban life. The purpose of this data is to predict, based upon a monster's origin, appearance, and breath attack (if any), whether or not it will practically raze a city before it retires for mid-afternoon tea and crumpets.

Origin	**Appearance**	**Foul Breath**	**City-Destroyer**
Outer Space	Blob	Acid	Yes
Outer Space	Reptile	None	Yes
Outer Space	Reptile	Acid	Yes
Outer Space	Blob	Fire	No
Pacific Ocean	Blob	None	No
Pacific Ocean	Blob	Acid	No
Pacific Ocean	Reptile	Fire	No
Pacific Ocean	Reptile	Acid	No

1. (5 points) With no other information, how many bits on average would you need to transmit City-Destroyer ?

2. (1 point) Which of the three other attributes gives you the highest information gain with respect to City-Destroyer? Show your reasoning.

3. (2 points) Using information gain as your splitting criteria, what decision tree would you get? Do not prune the tree.

4. (2 points) Fill in the following table with your predictions.

Origin	**Appearance**	**Foul Breath**	**City-Destroyer**
Outer Space	Blob	Fire	
Pacific Ocean	Reptile	Acid	

5. (5 points) How might you deal with missing values? For example, what would you predict if all you knew about a visitor was that it was blob-shaped, and why?

Problem 9- Hidden Markov Models (15 points)

Consider an HMM defined by the traditional set of parameters:

$$\lambda = (S,Z,T,O,p_0)$$

Recall that the following array of probabilities can be obtained by dynamic programming in the forward algorithm:

$$\alpha_t(i) = p(z_1,..., z_t \wedge q_t = s_i|\lambda)$$

a) We can compute a similar array of probabilities using dynamic programming in a backward algorithm:

$$\beta_t(i) = p(z_t,..., z_T \wedge q_t = s_i|\lambda)$$

Give a mathematical expression for $\beta_\tau(\iota)$ that can be computed using dynamic programming.

b) In the process of learning an HMM from data we need the following array of probabilities:

$$\gamma_t(i) = p(q_t = s_i|z_1, ..., O_t, O_{t+1}, ..., O_T, \lambda)$$

Your job is to define $\gamma_t(i)$ in a simple finite expression in terms of $\alpha_t(i)$, $\beta_t(i)$ and $p(z_1, ..., O_t, O_{t+1}, ..., O_T, \lambda)$.

$$\gamma_t(i) = \frac{\alpha_t(i) \times \beta_t(i)}{p(z_1, ..., O_t, O_{t+1}, ..., O_T, \lambda)}.$$