16.410-13: Principles of Automated Reasoning and Decision Making

Sample Midterm

October 20th, 2010

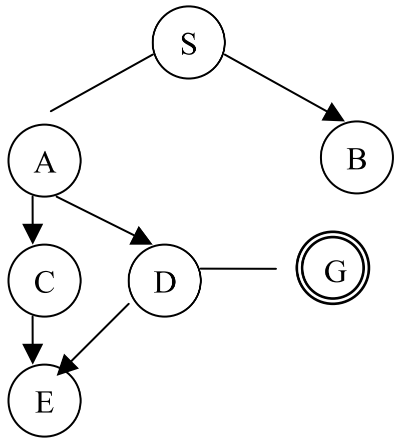
Name	
E-mail	

<u>Note:</u> Budget your time wisely. Some parts of this quiz 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	16.41x 37 16.413 15		
Problem 2	25		
Problem 3	35		
Problem 4	15		
Total	16.410 112 16.413 127		

Problem 1 Uninformed Search (X points)

*You are given the task of finding a path from state **S** to state **G** in the following directed graph, using the three uninformed search algorithms introduced in class – depth-first, breadth-first and iterative-deepening search:*



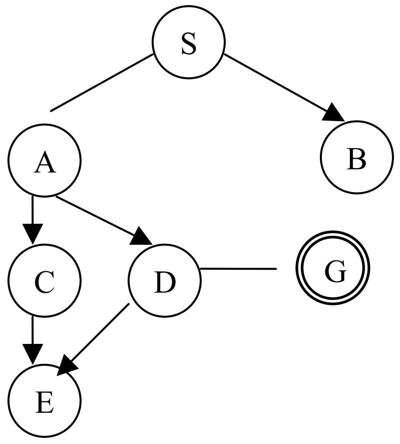
Make the following assumptions:

- *Depth and breadth-first search explore a node's children in **alphabetical order**.*
- *The depth-limited search called by iterative deepening search also explores a node's children in **alphabetical order**.*
- *Each search algorithm **uses a visited list** to prune nodes, **unless** otherwise stated.*
- *The search stops as soon as the goal is **expanded** (not when it is first visited).*

*In the following, recall that a node is **expanded** when the search algorithm takes a path to that node off the queue, and attempts to create its children. A node is **visited** when a path to that node is first placed on the queue.*

Part 1.A Depth-first Search with a Visited List (12 points)

The above graph is repeated here for your convenience:



For each iteration of depth-first search, fill in the path being expanded, the state if the search queue and the visited list after expansion. Search begins with the path (S) on the queue and node S on the visited list. Remember to ***use*** *a* ***visited list****. Stop if search does not terminate after 6 steps.*

	Path being expanded	Queue after expansion	Visited list after expansion
1		**(S)**	**S**
2	**(S)**	**(A,S)(B,S)**	**SAB**
3	**(A,S)**	**(C,A,S)(D,A,S)(B,S)**	**SABCD**
4	**(C,A,S)**	**(E,C,A,S)(D,A,S)(B,S)**	**SABCDE**
5	**(E,C,A,S)**	**(D,A,S)(B,S)**	**SABCDE**
6	**(D,A,S)**	**(G,D,A,S)(B,S)**	**SABCDEG**

Part 1.B Search Issues (6 Points)

In each of the following, circle T if the statement is True, and F, otherwise:

a. [✓ T] F *Assuming an edge cost of 1, breadth-first search* ***with a visited list*** *is guaranteed to find a shortest path to the goal if one exists.*

b. T [✓ F] *Assuming an edge cost of 1, depth-first search* ***with a visited list*** *is guaranteed to find a shortest path to the goal if one exists.*

c. [✓ T] F *Breadth-first search* ***without a visited list*** *is guaranteed to find a path to the goal if one exists.*

d. T [✓ F] *Depth-first search* ***without a visited list*** *is guaranteed to find a path to the goal if one exists.*

The following are all statements about Iterative Deepening search. For each statement, circle T if the statement is True, and F, otherwise.

Iterative Deepening Search…

e. [✓ T] F *will typically require much less memory than breadth-first search.*

f. T F *has a smaller worst case run-time overhead for search trees with a small branching factor, when compared to search trees with a larger branching factor.*

This is true if just the run-times of the over head is compared. If the ratio of (overhead runtime of iterative deepening)/(breadth-first runtime) is

Part 1.C Complexity Analysis (12 points)

*In this problem we consider the complexity of searching a tree of **depth d**, in which the **branching factor** is not constant, but **varies** as a function of **node level i**. In particular, the branching factor b_i of each node at level i is $b_i = d - i$, with the **root** node appearing at **level 0**. For example, the root has **d** children, all nodes at level 2 have **d-1** children, nodes at level 3 have **d-2** children, and so on; at level d, the tree dies out and no node at level d has a child node).*

Part 1.C.1 Number of nodes in the tree (6 points)

*Given a tree of depth **d**, write down an expression for the number of nodes at each level of the tree, up to **d**. This expression must be a function of the level number **m** and the depth of the tree **d**.*

Number of nodes in level **m** is $d(d-1)(d-2) \ldots (d-(m-1)) = d! / (d-m)!$

*Write an expression for the total number of number of nodes in the tree for a tree with depth **d**. This expression must be a function of the depth of the tree **d**. (You do **not** need to solve the recursion in this step).*

Total number of nodes in the tree is the sum:

$T(d) = 1 + d + d(d-1) + d(d-1)(d-2) + \ldots + d!$

*Let T(d) denote the total number of nodes in the tree. Write an equation for the total number of nodes in the tree in a recursive form. That is, write an equation for T(d) in terms of T(d-1) (the number of nodes in a tree that has depth **d-1**). (You do **not** need to solve the recursion in this step).*

Note that:
$T(d) = 1 + d + d(d-1) + d(d-1)(d-2) + \ldots + d!$
$T(d-1) = 1 + (d-1) + (d-1)(d-2) + (d-1)(d-2)(d-3) + \ldots + (d-1)!$

Then, write T(d) as
$T(d) = 1 + d\,(1 + (d-1) + (d-1)(d-2) + \ldots + (d-1)!) = 1 + d\,T(d-1)$.

Hence, $T(d) = 1 + d\,T(d-1)$.

Part 1.C.2 Worst Case Run-time of Breadth First Search (6 points)

Next consider runtime. What is the ***maximum*** *number of* ***nodes*** *that might be* ***generated (visited)*** *during* ***breadth-first search****, in order to reach a* ***goal*** *that is placed on level* ***d****? You can write your answer in terms of the depth of the tree (denoted as* ***d)*** *and the total number of nodes in the tree (denoted as* ***T(d))****.*

In the worst case, all the nodes are visited. The answer is T(d).

For full credit *you must* ***explain your derivation*** *below:*

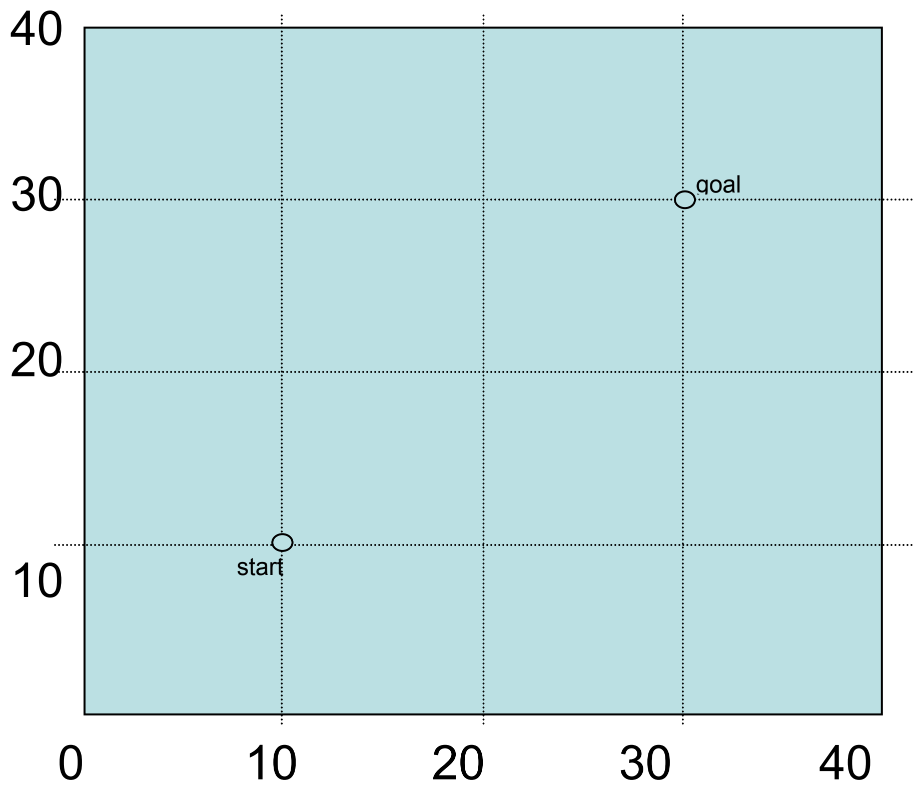
Must draw that all the nodes are visited, and show where the goal is located in the worst case.

Problem 2 Formulating Roadmaps (20 points)

In this problem we considered the mapping of a path-planning problem with obstacles to a road map, using the concepts of configuration spaces and visibility graphs. A robot has dimensions 2 feet by 4 feet (it's a rectangle) and is positioned so that its center is at location [10, 10] (all dimensions in feet). It must reach a goal position at location [30, 30], while avoiding all obstacles. There are two rectangular obstacles, which are five by ten foot, and centered at [20, 20] and [20, 10]. The robot can translate along any vector, as long as it doesn't collide with an obstacle, but can not rotate.

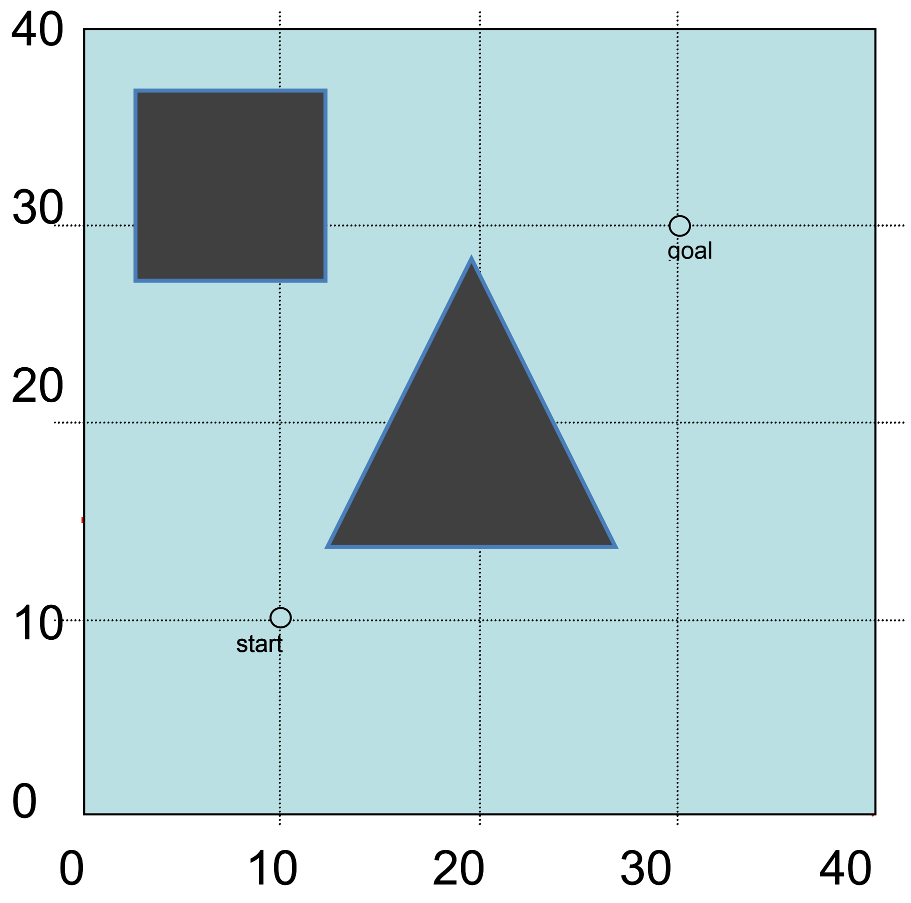
Part 2.A – Drawing the Configuration Space (10 points)

First consider mapping the problem above to an equivalent C-space. In the space below, please draw the C-space map for the problem. As stated earlier, when the robot changes direction, assume it ***does not rotate****, hence the orientations of the robot's boundaries do* ***not change****.*



Part 2.B Visibility Graph (10 points)

Consider the problem instance with two obstacles in the following figure. Draw the corresponding visibility graph on the figure.



Problem 3 Constraint Programming (35 points)

In this problem we consider the two essential elements of constraint programming, constraint modeling, and constraint satisfaction.

Part 3.A Constraint Modeling (12 points)

Betty, John and Alfredo are about to order their dinner at a fine restaurant. Your job is to design a robot Waiter who will recommend what they should order. You will accomplish this be formulating the decision problem as a constraint satisfaction problem.

You know that the menu is comprised of the following:

Main Courses		*Wines*	
Beef Bourguignon (BB)		*Red*	*Cabernet (Ca)*
Chicken Kiev (CK)			*Merlot (M)*
Glazed Salmon (GS)	*(Fish)*	*White*	*Chardonnay (Ch)*
Pasta Primavera (PP)	*(Vegetarian)*		*Pinot Grigio (PG)*
Mushroom Risotto (MR)	*(Vegetarian)*		

In addition, Betty, John and Alfredo tell you that your recommendation must satisfy the following constraints:

- *Each of the three diners will order a main course and a suitable wine.*
- *Betty is a vegetarian, and will not eat a course with chicken, beef, or fish.*
- *John wants to impress Betty, and will order the same main course.*
- ~~*To show individuality, John will order a different wine from Betty.*~~
- ~~*White wine can be ordered with chicken, fish and vegetarian food, but not beef.*~~
- *Red wine can be ordered with beef or fish, but not chicken or vegetarian courses.*
- *Alfredo is rebellious and will not select a wine that either Betty or John has ordered.*

Please formulate this problem as a constraint satisfaction problem.

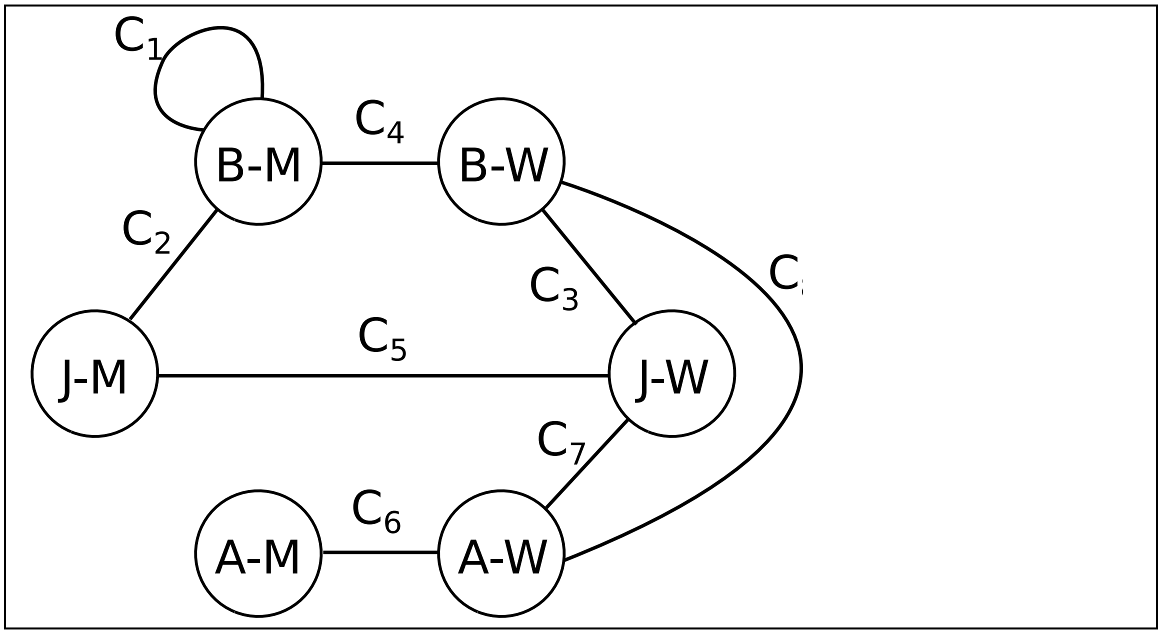
List all variables and their corresponding domains. Denote the menu items by using their short form (e.g., BB for Beef Bourguignon).

Define 6 variables, two for each person denoting their choice of the main course and the wine: B-M and B-W for Betty; J-M and J-W for John; and A-M and A-W for Alfredo.

The domains of B-M, J-M, A-M are {BB, CV, GS, PP, MR}.

The domains of B-W, J-W, A-W are {Ca, M, Ch, PG}.

Draw a constraint graph for your encoding of this problem. Label the vertices and edges with a symbol for each corresponding variable and constraint.



Write a description for each constraint, consisting of its scope and relation. Describe the relation by enumerating allowed assignments to the variables in the scope.

2) Betty is a vegetarian: C1 constrains the domain:
D_{B-W} = {PP, MR}

3) John wants to impress Betty: C2 constrains B-M J-M:
R_{B-M, J-M} = {(BB,BB), (CV,CV), (GS,GS), (PP,PP), (MR,MR)}

~~**4) John has individuality:** C3 constrains B-W and J-W:~~
~~R_{B-W, J-W} = {(Ca, M), (Ca, Ch), (Ca, PG),~~
~~(M, Ca), (M, Ch), (M, PG),~~
~~(Ch, Ca), (Ch, M), (Ch, PG),~~
~~(PG, Ca), (PG, M), (PG, Ch)}.~~

~~**5) White goes well with chicken, fish, and veggies**: C4 constrains B-M and B-W, C5 constrains J-M and J-W, C6 constrains A-M and A-W.~~
~~R_{B-M, B-W} = {(CV, Ch), (CV,PG), (GS, Ch), (GS, Ch),~~
~~(PP, Ch), (PP, PG), (MR, Ch), (MR, PG)}. (C5 and C6 are same).~~

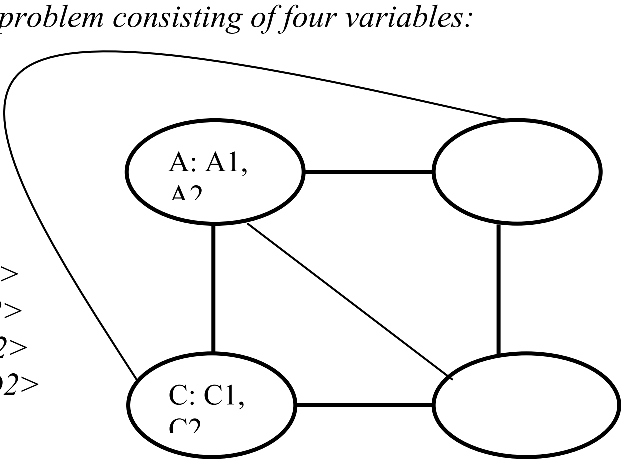
6) Red goes well with beef and fish: will be merged with constraints C4, C5, and C6.
R_{B-M, B-W} = {(BB, Ca), (BB, M), (GS, Ca), (GS, M)} (C5 and C6 are same).

7) Alfredo is rebellious: C7 constrains A-W and J-W, C8 constrains A-W and B-W
R_{A-W, J-W} = R_{A-W, B-W} = R_{B-W, J-W} (from item 4).

Part 3.B Constraint Satisfaction (23 points)

Consider a constraint satisfaction problem consisting of four variables:

A: {A1, A2}
B: {B1, B2}
C: {C1, C2}
D: {D1, D2}



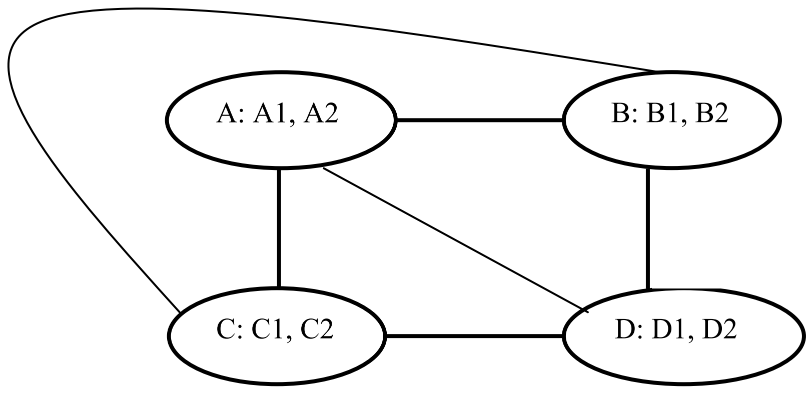
And constraints:

- ***A-B**: <A1, B1> or <A2,B1>*
- ***A-C**: <A1, C1> or <A2,C2>*
- ***B-D**: <B1, D1> or <B1,D2>*
- ***C-D**: <C2, D1> or <C2, D2>*
- ***B-C**: <B1, C2>*
- ***A-D**: <A2, D2>*

Part 3.B.1 Constraint Propagation (10 points)

3.B.1.a Pruned Domains (4 points)

On the following copy of the constraint graph, cross out each domain element that is eliminated by constraint propagation:



3.B.1.b Examined Arcs (6 points)

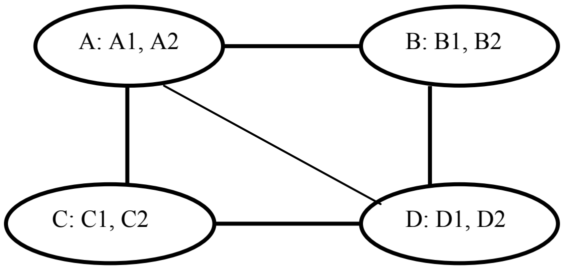
List the arcs examined by constraint propagation, in the order in which they are examined. For each arc, list its pair of variables, and use ">" to indicate the direction in which arc consistency is tested (similar to the lecture notes):

Part 3.B.2 Backtrack Search and Forward Checking (13 points)

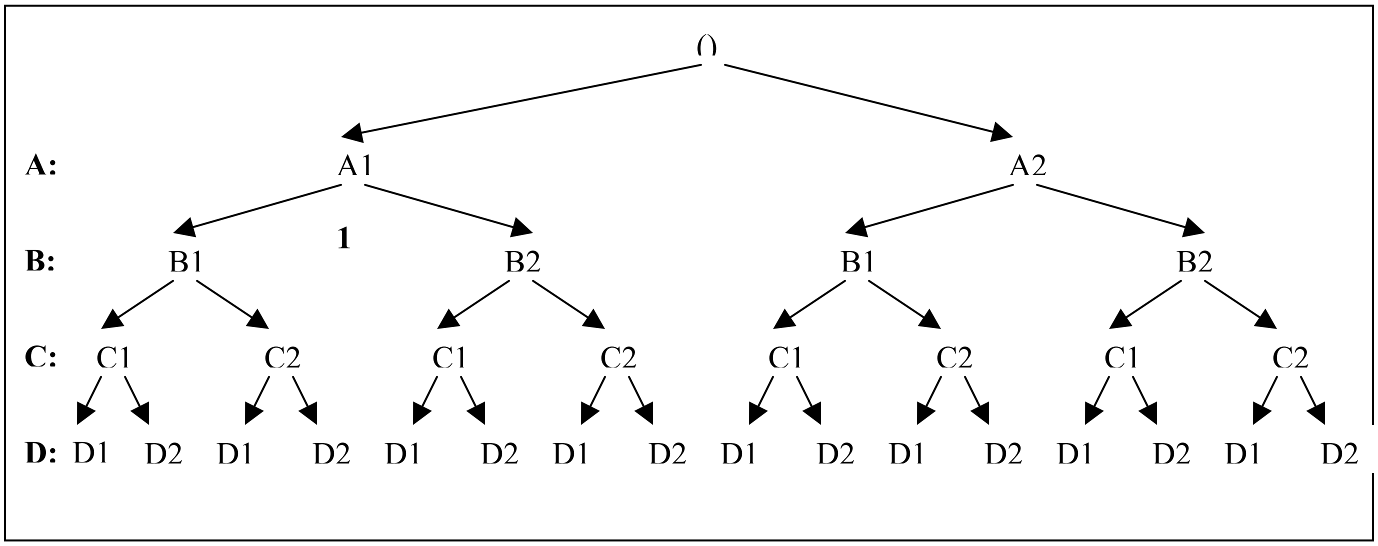
*In this part we **search** for the **first** consistent solution to the constraint problem described above, stopping when found. Variables are ordered alphabetically, and values are ordered numerically.*

3.B.2.a Backtrack Search (no Forward Checking) (5 points)

- ***A-B***: *<A1, B1> or <A2,B1>*
- ***A-C***: *<A1, C1> or <A2,C2>*
- ***B-D***: *<B1, D1> or <B1,D2>*
- ***C-D***: *<C2, D1> or <C2, D2>*
- ***B-C***: *<B1, C2>*
- ***A-D***: *<A2, D2>*



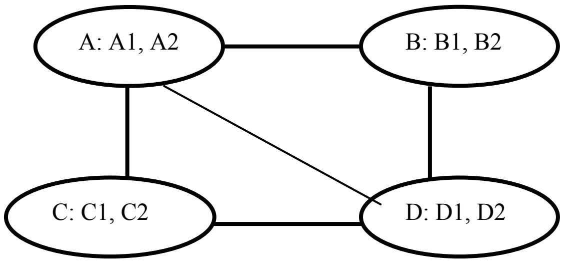
*Indicate which assignments are made and the order of those assignments, by writing a number under each assignment made, on the copy below. We have written "**1**" under the first assignment. Circle the first consistent assignment. Cross-out assignments pruned.*



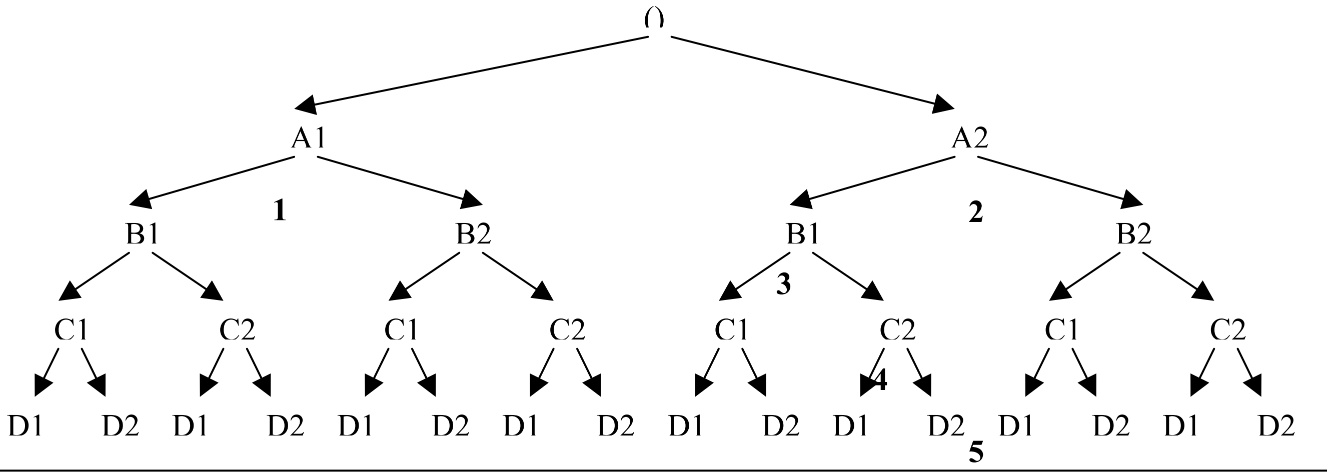
Space available to make comments or show your work:

Part 3.B.2 Backtrack Search with Forward Checking (8 points)

- ***A-B***: *<A1, B1> or <A2,B1>*
- ***A-C***: *<A1, C1> or <A2,C2>*
- ***B-D***: *<B1, D1> or <B1,D2>*
- ***C-D***: *<C2, D1> or <C2, D2>*
- ***B-C***: *<B1, C2>*
- ***A-D***: *<A2, D2>*



Indicate which assignments are made and the order of those assignments, by writing a number under each assignment made on, the copy of the search tree below. We have written 1 under the first assignment. Circle the first consistent assignment. Cross out the assignments that are pruned.



Space available to make comments or show your work:
(Please indicate the pruned domain values for each step)

Problem 4 – Planning (35 points)

In this problem we consider the formulation of activity planning problems as atomic actions in PDDL, the construction of planning graphs, and properties of solution extraction from the planning graph.

Part 4.A Formulating Planning Problems (10 points)

Recently the elevators in the Stata center have been running very slow. Sertac, who sits on the seventh floor, is busy delivering his paper on time, but needs to get the exam papers to Brian's office, which is on the 1st floor. In order not to waste time, Sertac has decided to use one of his robots to carry the exam papers from the 7th floor to the 1st floor. Luckily, his robot already has a built-in activity planner, which takes a problem description in the PDDL format, and generates and executes a plan that satisfies the PDDL description. Please help Sertac formulate this problem using atomic operators in PDDL.

Part 4.A.1 (3 points)
List and describe your predicates, actions, initial condition, and goal predicates.

```
(:objects robot, elevator, floor1, floor7)

(:predicate (IN ?robot ?elevator) (ELEV-AT ?elevator ?floor) (ROBOT-AT ?robot ?floor))

(:action GET-IN :parameters (?robot ?elevator ?floor)
        :precondition (and (ROBOT-AT robot floor) (ELEV-AT elevator floor)
        :effect (IN robot elevator) )

(:action GET_OUT :parameters (?robot ?elevator ?floor)
        :precondition (and (IN robot elevator) (AT elevator floor))
        :effect (not (IN robot elevator)) )

(:action PUSH-CALL-BUTTON :parameters (?robot ?elevator ?floor)
        :precondition (and (not (IN robot elevator))
                          (ROBOT-AT robot floor) (not (ELEV-AT elevator floor)))
        :effects (ELEV-AT elevator floor) )

(:action PUSH-GO-BUTTON :parameters (?robot ?elevator ?floorAt ?floorTo)
        :precondition (and (IN robot elevator) (ELEV-AT elevator floorAt))
        :effect (and (ELEV-AT elevator floorTo) (ROBOT-AT robot floorTo)) )

(:init (ROBOT-AT robot floor7), (ELEV-AT elevator floor1))

(:goal (and (ROBOT-AT robot floor1) (not (IN robot elevator)) )
```

Part 4.A.2 (3 points)
State any assumptions that you make, and justify why your representation is at the right level of abstraction.

We had assumed that the robot can

i) push the call button and wait for the elevator until it gets to the floor that the robot is at and the elevator door opens
ii) get inside the elevator, once the elevator door is open
iii) push the button to its destination and wait until the elevator gets to the destination and the elevator door opens
iv) get outside the elevator

This level of abstraction provides the basic set of activities as the primitives.

Part 4.A.3 (4 points)
Point out one limitation of atomic actions in PDDL (as presented in class) that limits the effectiveness of your solution. Augment PDDL with a new syntax that remedies this limitation. Explain the meaning of this augmentation and give one example action description.

One limitation is the inability to handle metric time. For instance, we may want the goal predicate to be true at some time interval. PDDL can be extended to support propositions with metric time intervals, e.g.,

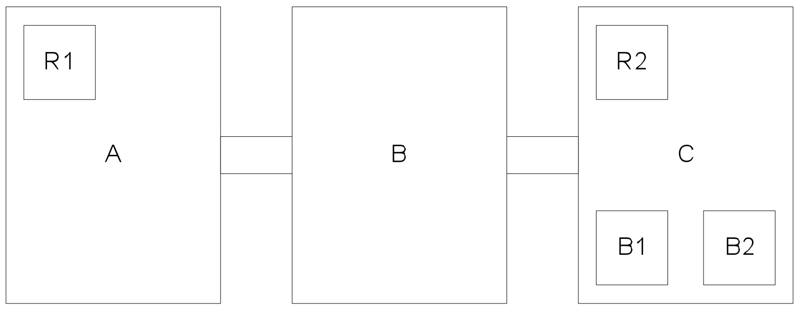
```
(and (or (IN robot elevator1):[10, 20] (IN robot elevator2):[30, 50])
          (ELEV-AT elevator1 floor2)):[40,50]
```

This formula states that either the robot has to be in elevator1 in the interval [10,20] time units, or it should be in elevator2 in the interval [30,50]. Moreover, elevator1 should be in floor2. And all these should be true during the interval [40,50].

A second limitation is the inability to handle sensing, and nondeterministic action. To remedy this one can introduce a conditional effect operator, in which the effect depends on the sensor readings after execution of the particular action.

Part 4.B Constructing Planning Graphs (15 points)

Consider the plan graph for the following problem. The Smith residence is a small house consisting of three rooms, A, B, and C, arranged as shown below:



Dr. Smith is away, but he has left two robots, R1 and R2, in charge of the house. Dr. Smith asked them to move two boxes, B1 and B2, from room C to room A. R2 is initially in C, and R1 is initially in A. Let's help the robots figure out the best way to do this.

To generate a plan for the robots, we formulate the problem in PDDL and give the description to Graph Plan, as follows (note that for the :parameters field of :action, an expression like (Robot ?R) specifies that there is a parameter ?R that is of type "Robot"):

Operators:

```
(:action move-left
        :parameters ((Robot ?R) (Room ?From) (Room ?To))
        :preconditions (:and
                         (left-of ?To ?From)
                         (in ?R ?From))
        :effects     (:and
                         (in ?R ?To)
                         (del in ?R ?From)))

(:action move-right
        :parameters ((Robot ?R) (Room ?From) (Room ?To))
        :preconditions (:and
                         (right-of ?To ?From)
                         (in ?R ?From))
        :effects      (:and
                         (in ?R ?To)
                         (del in ?R ?From)))

(:action carry-left
        :parameters ((Robot ?R) (Box ?Box) (Room ?From) (Room ?To)
        :preconditions (:and
                         (left-of ?To ?From)
                         (in ?R ?From)
                         (in ?Box ?From))
        :effects (:and
                         (in ?R ?To)
                         (in ?Box ?To)
                         (del in ?R ?From)
                         (del in ?Box ?From)))
```

```
(:action carry-right
        :parameters  ((Robot ?R) (Box ?Box) (Room ?From) (Room ?To)
        :preconditions (:and
                         (right-of ?To ?From)
                         (in ?R ?From)
                         (in ?Box ?From))
        :effects      (:and
                         (in ?R ?To)
                         (in ?Box ?To)
                         (del in ?R ?From)
                         (del in ?Box ?From)))
```

Initial State Facts:

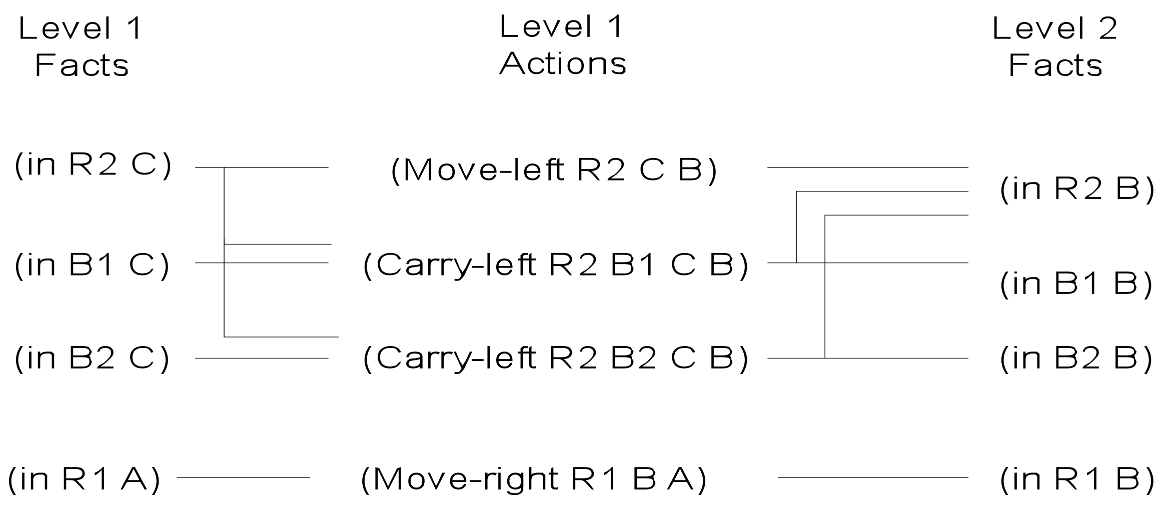
```
   (:init
     (Robot R1)            (in R1 A)
     (Robot R2)            (in R2 C)
     (Room A)              (in B1 C)
     (Room B)              (in B2 C)
     (Room C)              (left-of A B)
     (Box B1)              (left-of B C)
     (Box B2)              (right-of B A)
                           (right-of C B)      )
```

Goal State Facts:

```
  (:goal
     (in B1 A)             (in B2 A)           )
```

Part 4.B.1 Plan Graph Minus Mutex (10 points)

For the PDDL description above, fill in the following plan graph for the first level. Show Level 1 operators and Level 2 facts. Do not show mutex relations:



Part 4.B.2 Mutex for the Plan Graph (5 points)

In the following table, list ***three pairs*** *of mutex actions for the Level 1 actions in Part 4.B.1. For each pair, specify the type of mutex relation ("deletes precondition," "deletes effect," or "inconsistent preconditions"). Note that you may not need all of the entries shown in this table:*

Available mutex types are:

1. *Inconsistent effects*
2. *Effect interferes with precondition*
3. *Competing needs*

Action Mutex Pair	*Mutex Type*
1	
2	
3	

Mutex operators – (move-left R2 C B) (carry-left R2 B1 C B) (carry-left R2 B2 C B)

are all mutex with each other (three mutex relations). They are all mutex because they undo each other's precondition that R2 is in C. This makes up three pairs.

Part 4.C Proving completeness of Graph Plan (10 points)

Give an inductive proof for the fact that the Graph Plan algorithm finds ***any*** *plan of length* ***N*** *that is* ***complete*** *and* ***consistent****.*

What is the induction on?

The number of levels in the plan graph.

Formulate and prove the base case:

In the base case, the number of levels in the graph is zero. Hence, we only have the propositions that are initially true. In this case, trivial (and only) plan of length zero is to do nothing. Hence, the algorithm will find only this plan guaranteeing completeness for plan graphs of length zero.

Formulate and prove the induction step:

Assume that the hypothesis holds for plan graphs with number of levels N. Let us show that it holds for plan graphs with number of levels equal to N+1.

Note that if there exists a consistent plan of length N+1, then the prefix of length N appears in the previous plan graph level, i.e., at level N. Then, by the induction hypothesis, our algorithm can find this prefix, since it is a consistent plan of length N.

When investigating level N+1, all consistent plans of length N will be completed to all its consistent extensions of length N+1, since the search at level is complete (for instance, carried out with a complete CSP algorithm).

Hence, assuming that all consistent plans of length N can be found, all consistent plans of length N+1 can also be found by the algorithm.

MIT OpenCourseWare
http://ocw.mit.edu

16.410 / 16.413 Principles of Autonomy and Decision Making
Fall 2010

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.