# 16.410-13: Principles of Automated Reasoning and Decision Making

## Midterm Solutions

***October 20th, 2003***

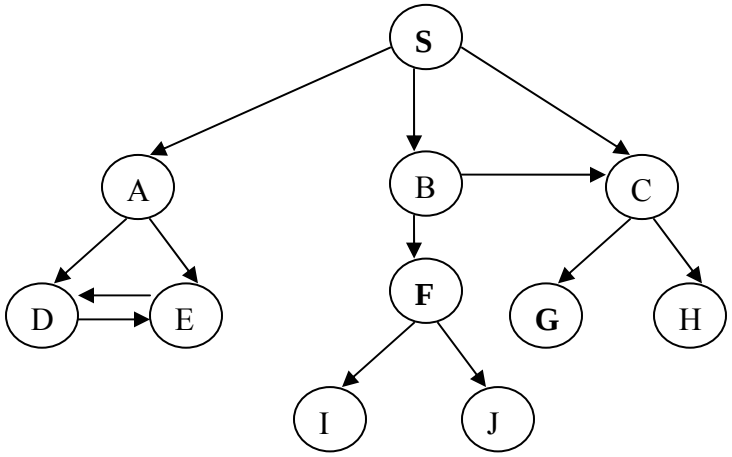
| | |
|---|---|
| Name | |
| E-mail | |

**Note: 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 | 26 | | |
| Problem 2 | 22 | | |
| Problem 3 | 24 | | |
| Problem 4 | 22 | | |
| Total | 94 | | |

# Problem 1 Uninformed Search (26 points)

You are searching for a path from S to G in the following graph using depth-first, breadth first and iterative deepening search:




Make the following assumptions:

- Each search algorithm explores a node's children in **alphabetical order**.
- Unless otherwise stated, the search algorithms **use a visited list** to prune nodes.
- The search stops as soon as the goal is expanded.

Write the sequence of nodes **expanded** by the specified search methods. A node is **expanded** when the search algorithm takes it off the queue, and attempts to create its children.

## Part A Depth-first Search (4 points)

Show the depth-first expansion sequence (we have started it for you):

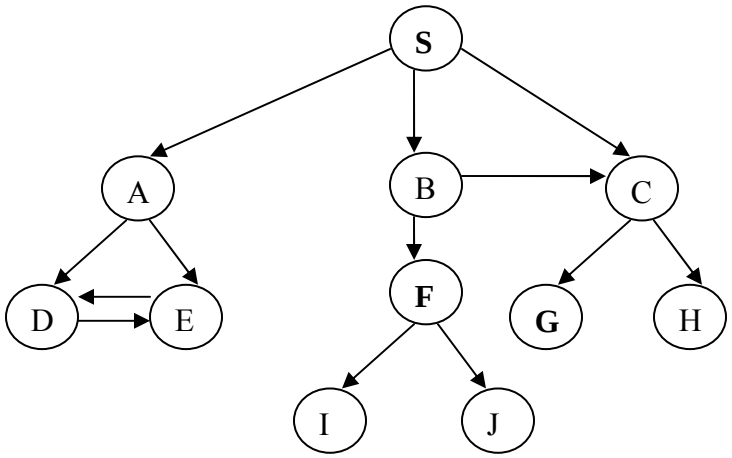
| |
|---|
| S-A-D-E-B-F-I-J-C-G |

## Part B Breadth-first Search (4 points)

Show the breadth-first expansion sequence (we have started it for you):

| |
|---|
| S-A-B-C-D-E-F-G |

Graph repeated for your convenience:




## Part C Iterative Deepening (4 points)

Show the iterative-deepening expansion sequence (we have started it for you):

| |
|---|
| S-S-A-B-C-S-A-D-E-B-F-C-G |

## Part D Visited List (4 points)

Now assume that the depth first search algorithm **does not use a visited list**. Show the search expansion, terminating either when the goal is reached or after the sequence reaches length 15.

Show the depth-first expansion sequence (we have started it for you):

| |
|---|
| S-A-D-E-D-E-D-E-D-E-D-E-D-E-D-E-D |

## Part E Analysis – Lower Bounds on Breadth First Search (10 points)

Assume you have a tree with branching factor b and depth d. Furthermore, assume that the goal node appears on level m, where m < d. Specify the exact cost of the search algorithm as a function of m, d and b.

What is the minimum number of nodes that might be generated by breadth-first search in order to reach the goal?

(b^m+b-2)/(b-1)

**For full credit** you must **show your derivation** below, including the solution to any recurrence:

The minimum number of nodes generated occurs when the goal node is placed at the far left, on level m. In this case, all the nodes above level m are generated, in addition to the one goal node. There is 1 node at level 0, and the number of nodes generated at each successive level is multiplied by b.

Level 0: 1
Level 1: b
Level 2: b^2
…
Level m-1: b^(m-1)
Level m: 1

The total number of nodes generated is:
Nodes=2+b+b^2+…+b^(m-1)

Solving the recurrence:
b*nodes=2b+b^2+b^3…+b^m
-nodes=2+ b+b^2+…+b^(m-1)

hence,

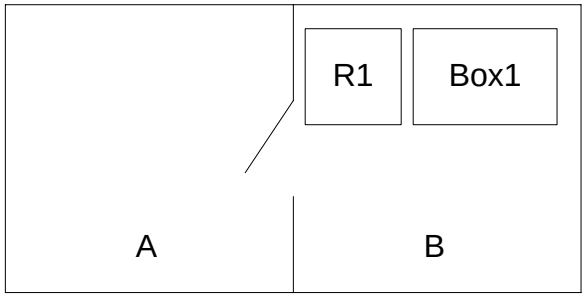
(b-1)*nodes=b^m+b-2

or

nodes=(b^m+b-2)/(b-1)

# Problem 2 Planning (22 points)

The Smith residence is a small house consisting of two rooms, A and B, connected by a door, as shown below:




Dr. Smith is away, but he has left a robot, R1, in charge of the house. Dr. Smith has asked R1 to move the box in room B to room A, then to stay in room B with the door closed. Let's help R1 figure out how to do this (R1, not developed at MIT, is lacking in many basic cognitive abilities, and needs help with such things).

Suppose the problem is represented in the STRIPS language in the following way:

Operators:

```
(operator move-box-to-A
        (preconditions
                in-B
                box-in-B
                door-open)
        (effects
                (not in-B)
                (not box-in-B)
                in-A
                box-in-A))
(operator close-door
        (preconditions
                door-open)
        (effects
                (not door-open)
                door-closed))
(operator go-to-B
        (preconditions
                in-A
                door-open)
        (effects
                (not in-A)
                in-B))
```

Initial state facts:

in-B
box-in-B
door-open

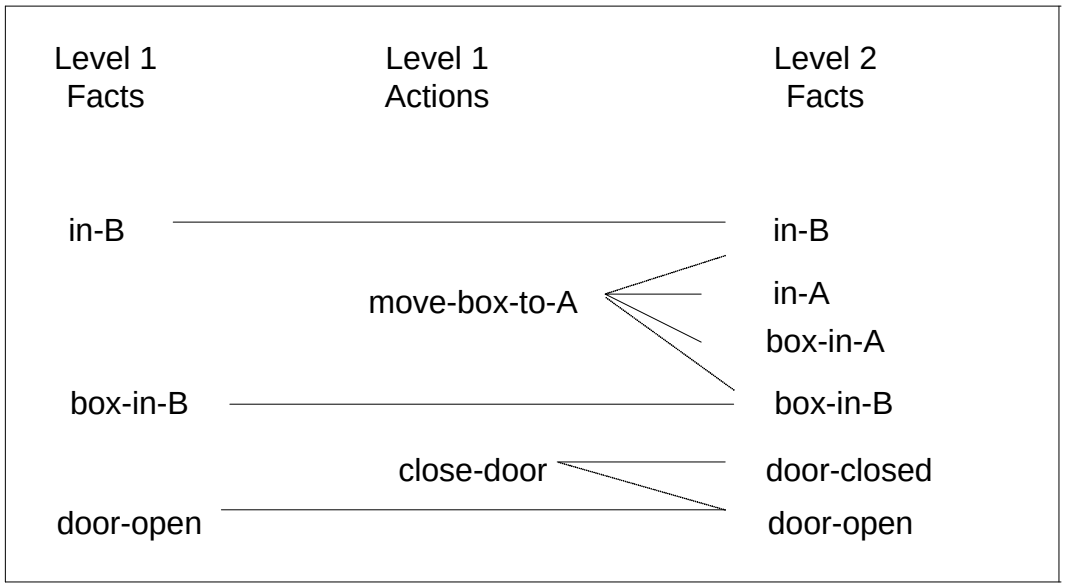
Goal state facts:

in-B
box-in-A
door-closed

## Part A First Level Plan Graph Without Mutexes (5 points)

Fill in the following plan graph for the first level. Show level 1 operators and level 2 facts. Do not show mutex relations.

Answer:



## Part B Mutexes for First Level Plan Graph (5 points)

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

Answer:

| **Action Mutex Pair** | **Mutex Type** |
|---|---|
| door-open noop – close-door | deletes effect |
| close-door – move-box-to-A | deletes precondition |
| move-box-to-A – in-B noop | deletes effect |
| move-box-to-A – box-in-B noop | deletes effect |
| | |
| | |

| **Fact Mutex Pair** |
|---|
| door-open – door-closed |
| in-A – in-B |
| in-A – box-in-B |
| box-in-A – in-B |
| box-in-A – box-in-B |
| door-closed – in-A |
| door-closed – box-in-A |

## Part C Second Level Plan Graph Without Mutexes (5 points)

Fill in the following plan graph for the second level. Show level 2 operators and level 2 and 3 facts (you should use the level 2 facts from part A). Do not show mutex relations.

Answer:

## Part D Mutexes for Second Level Plan Graph (7 points)

In the following table, give all pairs of mutex actions for the Level 2 actions in Part C. For each, specify the type of mutex relation ("deletes precondition", "deletes effect", or "inconsistent preconditions").

Answer:

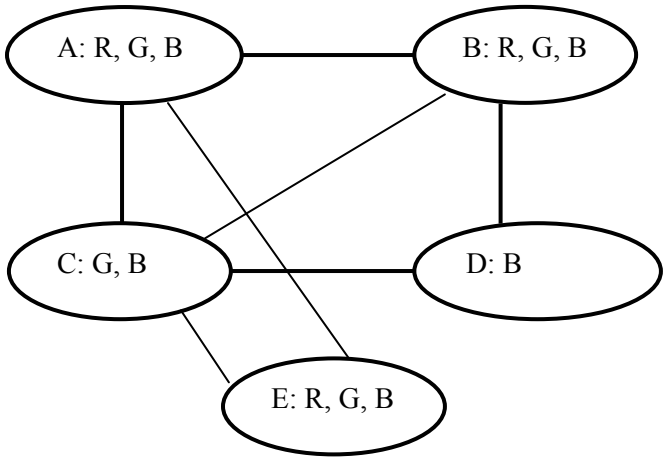
| **Action Mutex Pair** | **Mutex Type** |
|---|---|
| door-open noop – close-door | deletes effect |
| close-door – move-box-to-A | deletes precondition |
| move-box-to-A – in-B noop | deletes effect |
| move-box-to-A – box-in-B noop | deletes effect |
| go-to-B – in-A noop | deletes effect |
| go-to-B – close-door | deletes precondition |
| go-to-B – move-box-to-A | deletes effect |
| | |
| in-A noop – in-b noop | competing needs |

| **Fact Mutex Pair** |
|---|
| door-open – door-closed |
| in-A – in-B |
| in-A – box-in-B |
| |
| box-in-A – box-in-B |
| |
| |
| |
| |

Because all goal facts appear non-mutex in Level 3, an attempt could be made to extract a solution. This attempt would fail, however, because it would only successfully find either box-in-A and door-closed, or box-in-A, and in-B. Thus, the graph needs to be expanded one level further. Intuitively, this should be clear. The mutexes on actions require three separate actions: move-box-to-A, go-to-B, close-door, to get to the goal state. These actions cannot be performed in parallel, so the graph has to be expanded three full levels.

# Problem 3 Constraints (24 points)

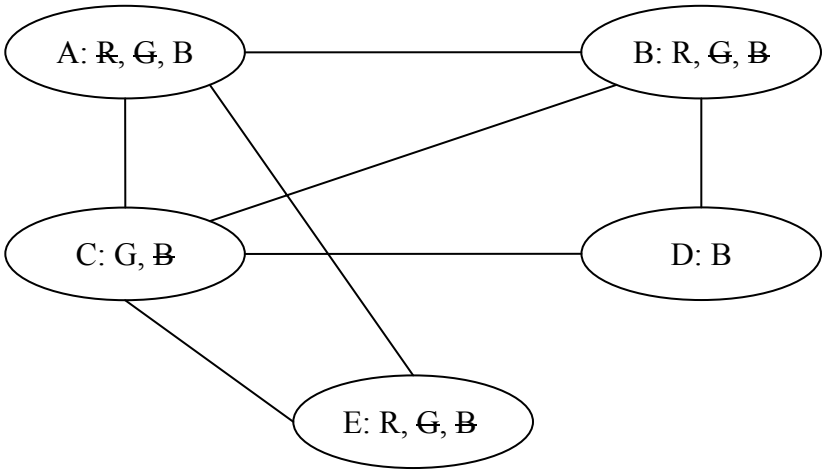
Consider a map coloring problem consisting of five regions, A, B, C, D and E. Your task is the standard map coloring problem. You are to assign each region a color such that no two adjacent regions have the same color. The legal colors are a subset of R, G and B, standing for Red, Green and Blue, respectively. The set of regions are depicted in the following constraint graph as ovals. The legal colors for each region is specified within its corresponding oval, and each pair of adjacent regions is denoted by a line between the two ovals corresponding to the two regions.




## Part A Pruned Domains (6 points)

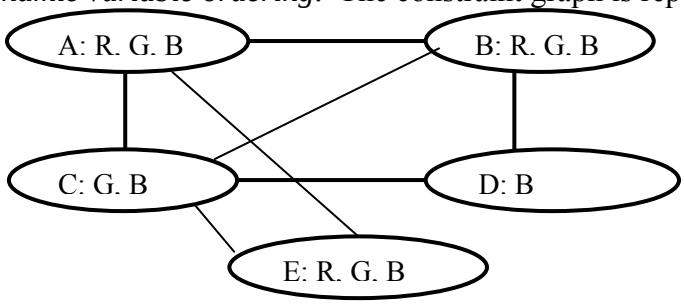
First, consider what domain elements can be eliminated using constraint propagation. Show your results on the copy of the constraint graph shown in the box below. On the graph, cross out each domain element that is eliminated by constraint propagation. For example, to start you off, we have eliminated **B**(lue) from region **B**, which is eliminated using constraint **B-D**:

Answer -




## Part B Backtrack Search with Forward Checking (18 points)

In this part you will use three CSP search methods to **search** for the **first** consistent solution to the map coloring problem described above. The three search methods you will use are *backtrack search*, *backtrack search with forward checking*, and *backtrack search with dynamic variable ordering*. The constraint graph is repeated here:

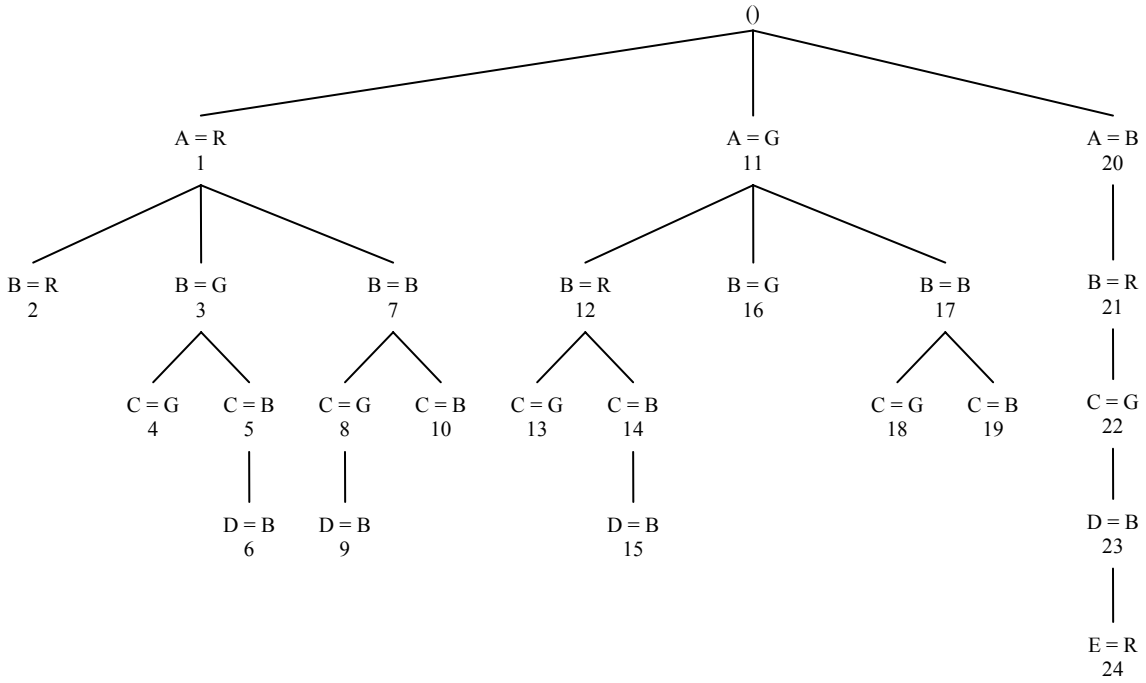


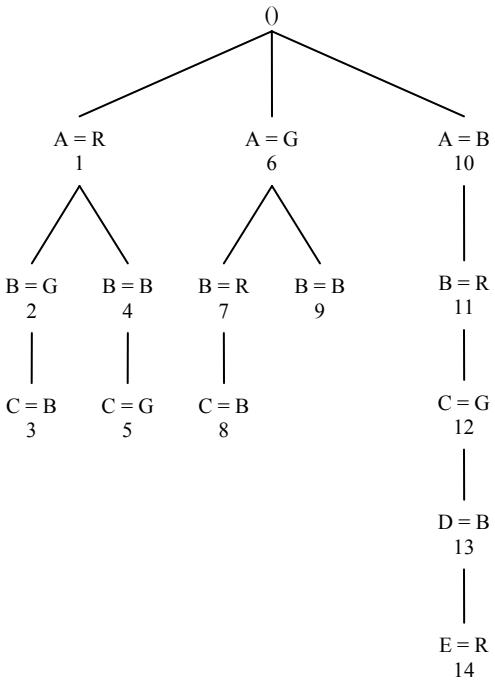

Your task is to draw the search tree resulting from the application of each method, stopping when you reach the first solution, or the end of the search tree if no solution exists. In your tree you will label each node (except the root) with the variable/value assignment made at that point in the search (e.g., A = R). In addition, indicate the order in which assignments are made by writing a number under each assignment in your search tree. To get you going we have started your first search tree below by indicating the root, and the first child expanded (labeled with assignment A=R and step number **1**).

## Part B.1 Simple Backtrack Search (6 points)

Draw the search tree for simple backtrack search (no forward checking) in the box below. In your search, variables, denoting regions, are to be ordered **alphabetically**, and colors are to be **ordered R**, then **G**, and finally **B**. Note that the full constraint propagation from part A is **not** to be performed here.

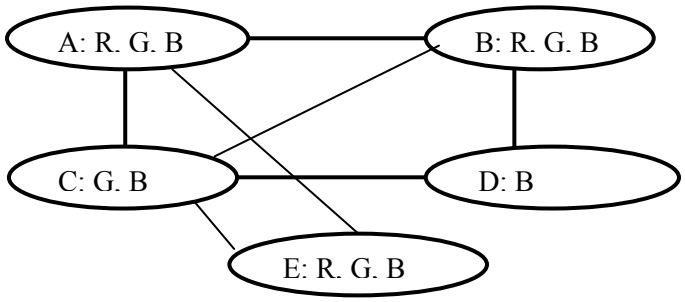
Two acceptable answers:



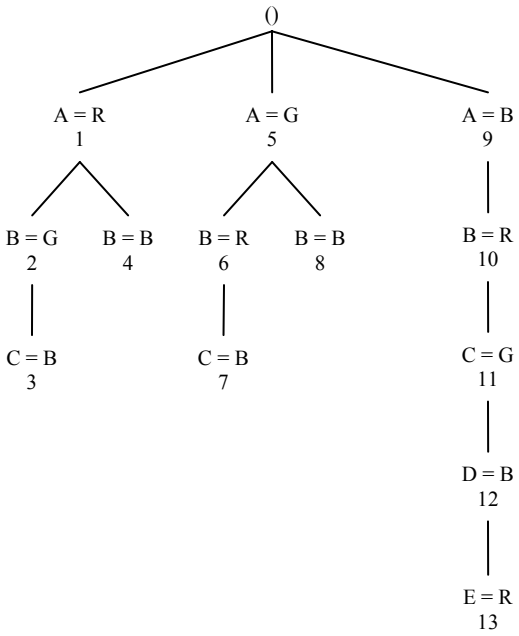
## Part B.2 Backtrack Search with Forward Checking (6 points)

The constraint graph is repeated here for your convenience:



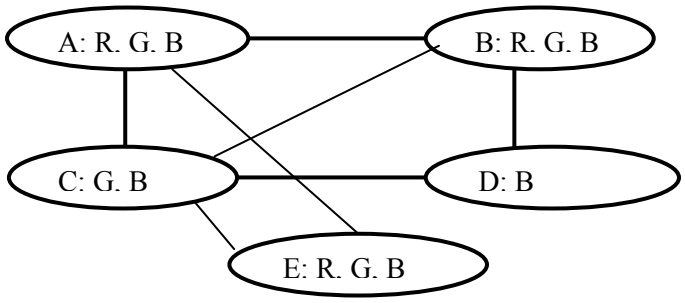

Draw the search tree for backtrack search **with forward checking** in the box below. In your search, variables, denoting regions, are to be ordered **alphabetically**, and colors are to be **ordered R**, then **G**, and finally **B**. Note once again that the full constraint propagation from part A is **not** to be performed here.

Answer


()
A = R
1
A = G
5
A = B
9
B = G
2
B = B
4
B = R
6
B = B
8
B = R
10
C = B
3
C = B
7
C = G
11
D = B
12
E = R
13

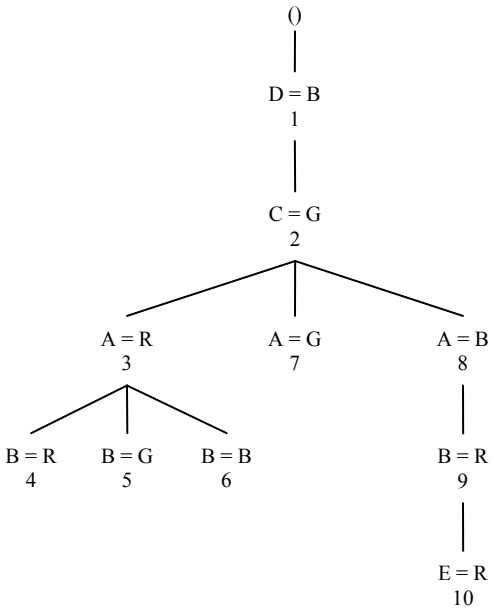
## Part B.3 Backtrack Search with Dynamic Variable Ordering (6 points)

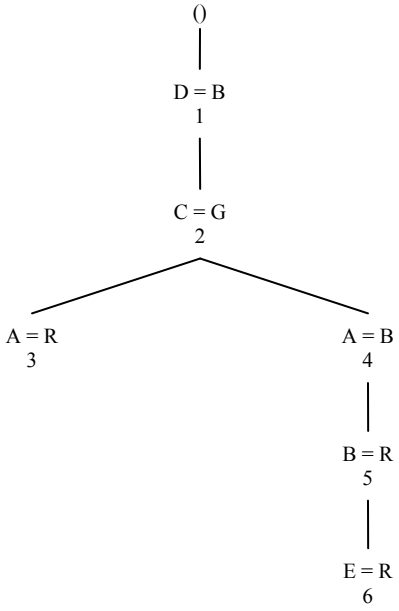
The constraint graph is repeated here for your convenience:




Draw the search tree for backtrack search **with dynamic variable ordering** in the box below. **DO NOT** perform forward checking or full constraint propagation. Select the variable to assign at each step based on the **most constrained variable heuristic**. Select the value of the variable to be assigned based on the **least constrained value heuristic.** We have not specified the assignment for the first step in this tree.

Two acceptable answers:


()
D = B
1
C = G
2
A = R
3
A = G
7
A = B
8
B = R
4
B = G
5
B = B
6
B = R
9
E = R
10


()
D = B
1
C = G
2
A = R
3
A = B
4
B = R
5
E = R
6

# Problem 4 - Linear and Integer Programming (22 points)

## Part A Formulating Linear Programs (6 points)

The Marginal Motors Company cuts sheet metal to stamp into car parts. It requires two basic shapes, plates and bars. The width of the shapes is given in the table below. Both basic shapes are *1 m* long. The sheet metal arrives at the plant in rolls of width *0.9 m*.

| Metal shape | Plates | Bars |
|---|---|---|
| *Width* | *0 .6 m* | *0.25 m* |

Objective: The Marginal Motors Company wants to minimize the amount of metal wasted when the rolls are cut into shapes for the factory's processing.

| Metal shape | Plates | Bars |
|---|---|---|
| *Required per week* | *1000 pieces* | *4600 pieces* |

Write a linear program, not requiring integer variables, to minimize the amount of wasted sheet metal per week. Define each variable in terms of the original problem, and write the objective and constraints in terms of those variables. HINT: What are the possible patterns for cutting for each meter of steel roll?

Provide your formulation in the following box:

Solution:

Let the variables represent the length in meters of rolled steel that Marginal Motors Company will cut in each specified pattern. Other patterns than those below are possible, but all other patterns are dominated by these (provide fewer parts and more waste), so these two are sufficient.

| **Pattern variable** | **Plates** | **Bars** | **Waste** |
|---|---|---|---|
| **x1** | **1** | **1** | ***.05m*** |
| **x2** | **0** | **3** | ***.15m*** |

Minimize: $0.05x1 + 0.15x2$
Subject to: $x1 \geq 1000$
$1x1 + 3x2 \geq 4600$
$x1 \geq 0,\ x2 \geq 0$

## Part B  Simplex method and graphical interpretation

### Part B.1 (8 points)

Solve the following 2-variable LP problem graphically by plotting the constraints and examining all feasible corner points.

Maximize $z = x_1 + x_2$

Subject to $2x_1 + 3x_2 \leq 8$
$3x_1 + 2x_2 \leq 8$
$x_1 \geq 0$
$x_2 \geq 0$

Draw the feasible region on the following graph:

- Label each constraint.
- Label each corner point (feasible and infeasible) with its $x_1$ and $x_2$ values.
- Circle the point that is the optimal feasible solution.

**Part B.2 (8 points)**

Below are the constraints in standard form of the linear program in Part C.1.

Subject to $2x1 + 3x2 + x3 = 8$
$3x1 + 2x2 + x4 = 8$
$x1 \geq 0, x2 \geq 0, x3 \geq 0, x4 \geq 0$

What would be the objective in a Phase I linear program, whose solution provides a feasible basis for the original problem?