Graduate AI – Midterm SOLUTIONS

March 6, 2012.
10:30–11:50am.

Name: _____

Andrew ID: _____

Read all of the following information before starting the exam:

- Please clearly write your **name** and **Andrew ID** in the spaces above.
- For full credit, please show all work *clearly* and *in order*.
- The test consists of five questions:
  - One multi-part short answer question.
  - Four long-answer questions.
- Good luck!

Problem	Possible	Awarded
1	20	20
2	20	20
3	20	20
4	20	20
5	20	20
Total	100	

1. (*20 points*)
Please clearly and concisely respond to all of the following short-answer problems. When appropriate, explain your answer or show your work.

a. (*3 pts*) *Explain why the number of misplaced tiles heuristic for n-puzzle is dominated by the Manhattan distance heuristic.*

Both heuristics are clearly admissible, but the Manhattan distance heuristic is a more accurate measure of the distance to the goal. In fact, the Manhattan distance heuristic is always at least as large (typically larger) than the misplaced tiles heuristic, and thus dominates it. (A misplaced tile has Manhattan distance at least 1, while the misplaced tiles heuristic always reports 1.)

b. (*3 pts*) *We know that in a planning graph operations monotonically increase, conditions monotonically increase, and mutexes monotonically decrease. Explain in 1-2 lines why it follows that the graph levels off.*

Let o_t, c_t, m_t be the number of operations, conditions, and mutexes in the planning graph at time step t. Then we know $o_{t+1} \geq o_t$, $c_{t+1} \geq c_t$, and $m_{t+1} \leq m_t$. Furthermore, let O be the maximum number of operations and C the maximum number of conditions; both are defined by the domain and are finite.
For times t and $t+1$, we have two cases. Either none of the counts change—i.e., $o_{t+1} = o_t$, $c_{t+1} = c_t$, and $m_{t+1} = m_t$—or at least one of the counts changes. In the former case, the graph has leveled off. In the latter, at least one of the following holds: $o_{t+1} > o_t$, $c_{t+1} > c_t$, or $m_{t+1} < m_t$. In future time steps, either nothing will change or these counts will hit O, C, and 0 respectively—which levels off.

c. (*3 pts*) *Iterative deepening search never expands more nodes than breadth first search before finding the goal. T/F? Explain.*

False. Iterative deepening search expands many nodes more than once (a cost that's outweighed by having to store a much smaller fringe than BFS).

d. (*3 pts*) *Suppose h_1 and h_2 are two admissible heuristics for a search problem such that for all nodes n, $h_1(n) < h_2(n)$. Then there do not exist any starting nodes for which A^* using h_1 expands fewer nodes than A^* using h_2. T/F? Explain.*

True by the definition of heuristic h_2 dominating heuristic h_1. It was presumed in the problem to be a minimization context (this was also announced during the midterm). However, credit was given if the answer indicated that this statement would be false for a maximization context.

e. (*3 pts*) *A^* uses $O(b^d)$ space while IDA* uses $O(bd)$ space, where b is the branching factor of the search tree and d is the depth of the shallowest goal node. T/F? Explain.*

True. IDA* uses $O(bd)$ space, A* uses $O(b^d)$ (unless $|h(n) - h^*(n)| \leq O(\log h^*(n)$ for each node n).

f. (*3 pts*) *Depth-first search uses $O(bd)$ space, where b is the branching factor of the search tree and d is the depth of the shallowest goal node. T/F? Explain.*

False. Not necessarily. DFS uses $O(bm)$, where m is the depth of the search space—not $O(bd)$, where d is the depth of the shallowest goal node. DFS can "miss" a shallow goal node and continue searching deep into the search space. If the depth is infinite, DFS might not even finish! In fact, it's not a complete search.

g. (*2 pts*) *What does it mean for a search algorithm to be complete? Optimal?*

Complete: if any feasible solutions to a search problem exist, the algorithm will find one.
Optimal: the solution returned by the algorithm has value at least as good as all other possible solutions, where "good" is defined by some objective function (e.g., path cost).

2. (*20 points*)
Consider the following CNF SAT clauses:

$$(x_4 \vee x_5 \vee \neg x_6)$$
$$(x_1 \vee x_4 \vee \neg x_5)$$
$$(x_3 \vee \neg x_4 \vee x_6)$$
$$(x_2 \vee \neg x_3 \vee x_6)$$
$$(x_1 \vee \neg x_2 \vee \neg x_5)$$
$$(x_3 \vee \neg x_4 \vee x_5)$$
$$(\neg x_1 \vee x_7 \vee \neg x_8)$$
$$(\neg x_1 \vee x_8 \vee \neg x_7)$$

A DPLL algorithm tries to make the following assignments sequentially:

$$x_6 \mapsto \mathbf{F}$$
$$x_5 \mapsto \mathbf{T}$$
$$x_1 \mapsto \mathbf{F}$$
$$x_2 \mapsto \mathbf{T}$$
$$x_4 \mapsto \mathbf{T}$$
$$x_3 \mapsto \mathbf{F}$$
$$x_8 \mapsto \mathbf{T}$$
$$x_7 \mapsto \mathbf{T}$$

However, at some point during these assignments, a conflict is discovered.

a. (*7 pts*) *Which decision induces the conflict?*

When $x_6 \mapsto \mathbf{F}$:

$$(x_4 \vee x_5)$$
$$(x_1 \vee x_4 \vee \neg x_5)$$
$$(x_3 \vee \neg x_4)$$
$$(x_2 \vee \neg x_3)$$
$$(x_1 \vee \neg x_2 \vee \neg x_5)$$
$$(x_3 \vee \neg x_4 \vee x_5)$$
$$(\neg x_1 \vee x_7 \vee \neg x_8)$$
$$(\neg x_1 \vee x_8 \vee \neg x_7)$$

When $x_5 \mapsto \mathbf{T}$

$$(x_1 \vee x_4)$$
$$(x_3 \vee \neg x_4)$$
$$(x_2 \vee \neg x_3)$$
$$(x_1 \vee \neg x_2)$$
$$(\neg x_1 \vee x_7 \vee \neg x_8)$$
$$(\neg x_1 \vee x_8 \vee \neg x_7)$$

When $x_1 \mapsto \mathbf{F}$

$$(x_4)$$
$$(x_3 \vee \neg x_4)$$
$$(x_2 \vee \neg x_3)$$
$$(\neg x_2)$$

Here, we have a decision point. Either DPLL chooses x_4 to unit propagate, or DPLL chooses $\neg x_2$ to unit propagate. I'll work through the former (so $x_4 \mapsto \mathbf{T}$):

$$(x_3)$$
$$(x_2 \vee \neg x_3)$$
$$(\neg x_2)$$

Again, DPLL makes a choice between unit propagating x_3 or $\neg x_2$. I'll work through the former (so $x_3 \mapsto \mathbf{T}$):

$$(x_2)$$
$$(\neg x_2)$$

This is clearly a contradiction, but DPLL doesn't know that yet. It chooses to unit propagate either x_2 or $\neg x_2$. I'll work through the former (so $x_2 \mapsto \mathbf{T}$):

$$()$$

We have an empty clause; a conflict has been induced. If any of the other unit propagation paths were taken, the answer remains the same (although the conflict graph will change a bit). Setting x_1 to **F** induced the conflict.

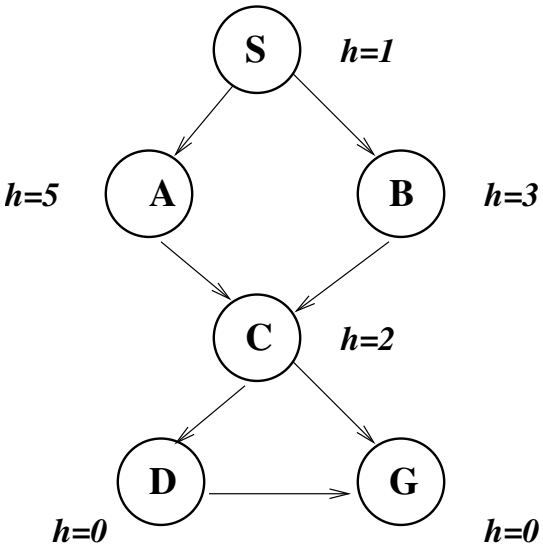
b. (*7 pts*) *Draw the conflict graph for when the conflict is discovered.*
The conflict graph changes based on the order of unit propagations chosen above. Many people did not get the above question correct, so the conflict graph is a function of how many decisions were made as well. See `http://www.msoos.org/2011/05/understanding-implication-graphs/` for a walkthrough on conflict graphs.

c. (*6 pts*) *Draw a conflict cut on the above graph that involves the* fewest *decision variables, and write out what the corresponding learned clause is.*
See Mate Soos' site referenced above; this explains cuts like 1UIP, the first unique implication point in the conflict graph.

3. (*20 points*)

a. (*8 pts*) Consider the search problem below with start state S and goal state G. The heuristic values are shown. Unfortunately we do not know the transition costs, and we really would like to know them. However we know that the given heuristic is admissible. Furthermore we know what was the priority queue of A^* after each node expansion, namely:

1. { (S, f=1) }
2. { (B, f=5), (A, f=6) }
3. { (A, f=6), (C, f=7) }
4. { (C, f=6) }
5. { (D, f=5), (G, f=7) }
6. { (G, f=6) }



Fill in the transition costs for all the edges.

$S \to A : 1$
$S \to B : 2$
$A \to C : 3$
$B \to C : 3$
$C \to D : 1$
$C \to G : 3$
$D \to G : 1$

b. (*6 pts*) Chris claims that: In general, if we are given a search problem, for which we know: (i) the heuristic value of all the nodes; (ii) that the heuristic is admissible; (iii) the solution found by A^* is given; and (iv) all the values of the prioriy queue of A^*'s search performance, then the transition values of all the edges in the search problem can be uniquely determined. Is Chris correct? If yes, then prove it. Otherwise, give an example that shows that Chris is incorrect.

No. A* may not visit all the nodes, so the A* search values would not include any information about the unvisited nodes.

c. (*6 pts*) Would A^* still be guaranteed to find the minimal path to the goal if there are negative transition costs?

No. If there is a cycle, negative costs would create a path of negative infinite cost, leading to an infinite search for A*.

4. (*20 points*)
No-op actions are ones that have one precondition and one effect, and both are the same. In Graphplan, no-op actions are added, at every level, for each proposition that appears in the previous level.

a. (*5 pts*) Explain why no-op actions are needed in Graphplan.

There are a couple ways to see this. First, Graphplan needs to maintain all the possible states at each planning graph level because conditions at a previous level may be useful for achieving the goal but could not be used earlier. Thus, if Graphplan did not propagate non-changing literals via no-ops, it would not be maintaining all the possible states at each level and might not reach the goal. Second, Graphplan needs no-op actions in order for convergence. Remember that one of the key conditions for Graphplan's guaranteed convergence is that literals monotonically increase. No-op actions make it so that once a literal appears at some level of Graphplan, it will appear at all subsequent levels, guaranteeing that literals monotonically increase. Without No-op actions, this guarantee is not met.

b. (*5 pts*) In Graphplan, no-op actions are included when determining mutex relationships. Why is this done? Would Graphplan still work if it did not find those types of mutexes?

Graphplan needs to maintain mutexes involving no-op actions to meet the mutex requirement for "inconsistent effects" between possible states. Graphplan would not work if it did not find these types of mutexes.

c. (*2 pts*) A CoBot needs to deliver mail between GHC, NSH, and the UC. Let's use planning with GraphPlan to help the CoBot deliver mail to the different mail centers.

$Init(At(M1, GHC) \wedge At(M2, NSH) \wedge At(M3, UC) \wedge Mail(M1) \wedge Mail(M2) \wedge Mail(M3) \wedge MailCenter(GHC) \wedge MailCenter(NSH) \wedge MailCenter(UC) \wedge Robot(CoBot) \wedge At(Cobot, GHC)) \wedge \neg GripperFull(Cobot)$
$Goal(At(M1, NSH) \wedge At(M2, UC) \wedge At(M3, GHC) \wedge At(Cobot, GHC))$

Action (Go(b,s,e),
$PRECOND : Robot(b) \wedge At(b, s) \wedge MailCenter(s) \wedge MailCenter(e)$
$EFFECT : \neg At(b, s) \wedge At(b, e))$

Action(PickUp(b,l,m),
$PRECOND : Robot(b) \wedge Mail(m) \wedge MailCenter(l) \wedge At(b, l) \wedge At(m, l) \wedge \neg GripperFull(b)$
$EFFECT : GripperFull(b) \wedge GripperCarries(m) \wedge \neg At(m, l)$

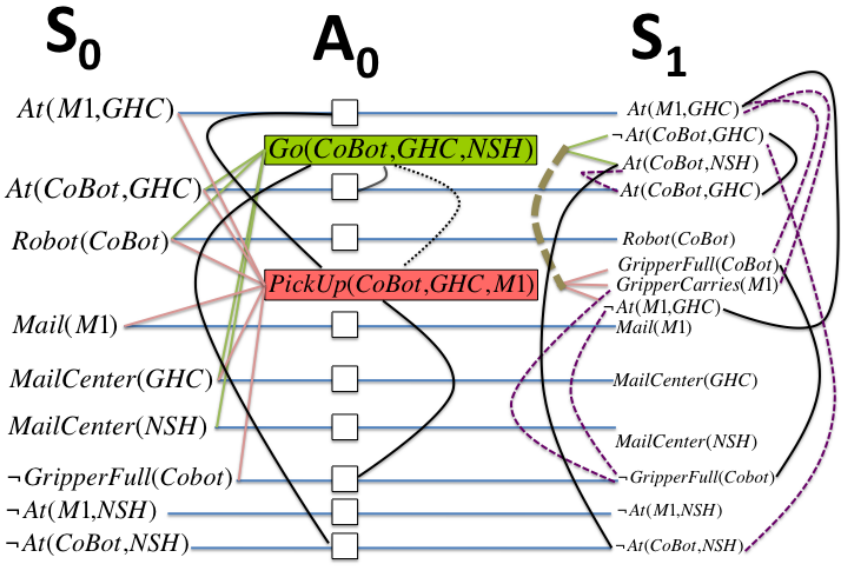
Given ONLY the predicates given in the other two actions (i.e. DON'T introduce any new predicates), write a definition for the PutDown(b,l,m) function that specifies preconditions and effects. Use the predicates Robot(b), MailCenter(l), and Mail(m) in your solution.

Action(PutDown(b,l,m),
$PRECOND : Robot(b) \wedge MailCenter(l) \wedge Mail(m) \wedge At(b, l) \wedge GripperFull(b) \wedge GripperCarries(m)$
$EFFECT : \neg GripperFull(b) \wedge \neg GripperCarries(m) \wedge At(m, l)$

d. (*2 pts*) Given the set of actions, formulate a plan that will reach the goal state from the start state.

Here is one possible plan: [PickUp(CoBot,GHC,M1), Go(CoBot,GHC,NSH), PutDown(CoBot,NSH,M1), PickUp(CoBot,NSH,M2), Go(CoBot,NSH,UC), PutDown(CoBot,UC,M2), PickUp(CoBot,UC,M3), Go(CoBot,UC,GHC),PutDown(CoBot,GHC,M3)]

e. (*6 pts*) Say the initial state is ONLY $At(M1, GHC) \wedge At(Cobot, GHC) \wedge Robot(CoBot) \wedge Mail(M1) \wedge MailCenter(GHC) \wedge MailCenter(NSH) \wedge \neg GripperFull(Cobot)$ and the goal is ONLY $Goal(At(M1, NSH) \wedge At(Cobot, NSH))$. Draw ONLY the first level of the planning graph including the states, actions, and necessary mutexes.



First notice that at state level S_0 we must also include $\neg At(M1, NSH) \wedge \neg At(CoBot, NSH)$ due to the Closed World Assumption (CWA). At the action level A_0, the solid black arrows represent mutexes due to inconsistent effects. The dotted black arrow represents a mutex due to interference. At state level S_1, the solid black arrows represent mutexes due to two literals conflicting (one being the negation of the other). The dotted magenta arrows represent mutexes due to the inconsistent support condition (every possible pair of actions that could achieve the two literals is mutex). The two true actions in A_1 are mutex, and their effects are the only way to achieve those effects. Thus, all pairs of effects between these actions are mutex. A thick gold dashed line represents this effect without cluttering up the plot.

Scoring rubric:
+2 points for drawing a mostly graph structure, ignoring CWA
+2 points for drawing correct action mutexes
+2 points for drawing an example of inconsistent support.

5. (*20 points*)
This problem deals with linear and integer linear programming. Assume the following linear program (LP) $A\mathbf{x} \leq \mathbf{b}$:

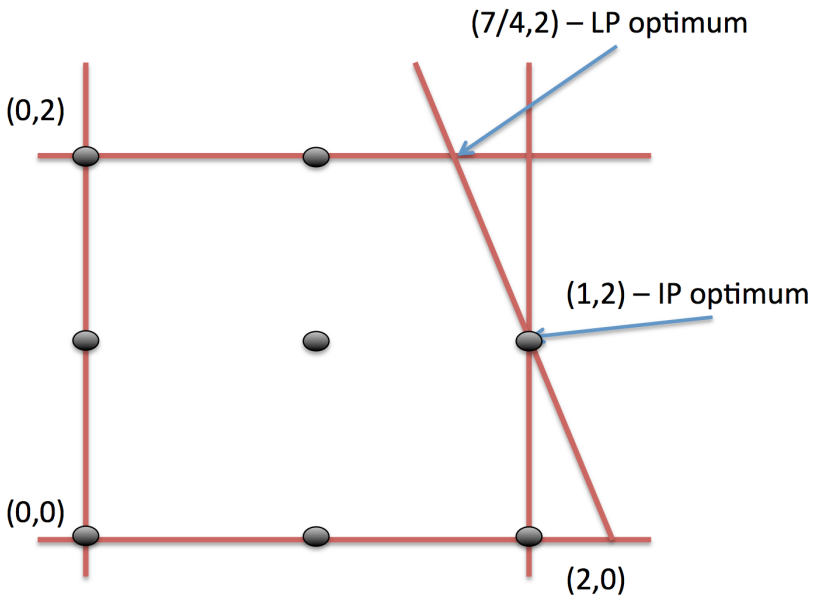
$$A = \begin{bmatrix} 4 & 1 \\ 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} 9 \\ 2 \\ 2 \\ 0 \\ 0 \end{bmatrix}$$

$$x, y \in \mathbb{R}$$

with linear objective function

$$\max x + y$$

a. (*3 pts*) *Draw the polytope representing the set of all feasible solutions to the LP.*



b. (*3 pts*) *Trace out the Simplex method on this polytope, starting at the origin.*

I was pretty lenient here, since people were rushed for time. The canonical tableau for the problem is:

$$\left[\begin{array}{ccc|ccc||c} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 4 & 1 & 1 & 0 & 0 & 9 \\ 0 & 1 & 0 & 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 0 & 0 & 1 & 2 \end{array}\right]$$

Either x (column 2) or y (column 3) can be made basic here; since we didn't specify variable selection rules, I allowed either. The Simplex method then either bounces from $(0,0) \to (0,2) \to (\frac{7}{4}, 2)$ or from $(0,0) \to (2,0) \to (2,1) \to (\frac{7}{4}, 2)$; both were accepted.

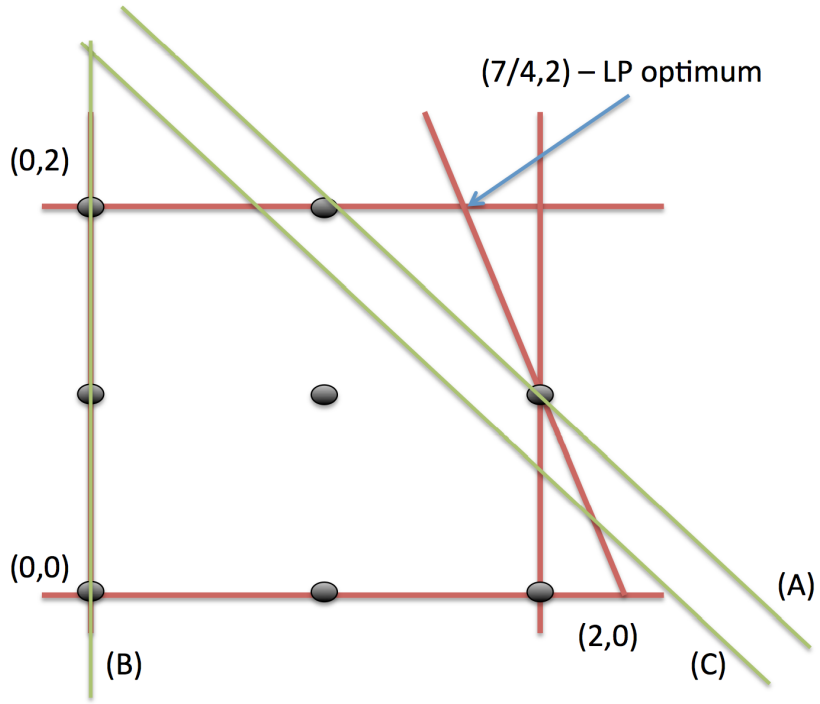
c. (*2 pts*) *What is the optimal value (i.e., what maximizes the objective) of the LP?*
The optimal value is $\frac{15}{4}$, and occurs when $x = \frac{7}{4}$ and $y = 2$. I did not take off points if only half of this answer was given, since the question is a little vague on what exactly it wants.

d. (*6 pts*) *Restrain the decision variables x and y to integer points. That is, let $x, y \in \mathbb{Z}$. Draw the branch and bound tree for the problem such that:*

- *Nodes are expanded in DFS order*
- *Branch first on x, with the left branch $x \geq 2$ and the right $x \leq 1$*
- *If necessary, branch on y with the left branch $y \geq 2$ and the right $y \leq 1$*

Branching on x first, we restrict $x \geq 2$. Then, the LP optimum for this restricted problem is $(x = 2, y = 1)$ for a value of 3. This is an integer solution, so we're done at this branch (i.e., we don't branch on y here at all).
Now we've backtracked to the root and branch on $x \leq 1$, with a lower bound of 3. Here again the LP optimum for the restricted problem is $x = 1, y = 2$ for a value of 3, proving optimality. Again, we don't branch on y here at all.

e. (*6 pts*) *On the original polytope (but with $x, y \in \mathbb{Z}$), draw* and label *three cuts, one of each type: (A) separating and valid, (B) not separating but still valid, (C) invalid.*



There are many (infinitely many) correct answers to this problem. We've drawn one of each on Figure e above. For (A), any linear inequality that separates the LP optimum from the (convex hull of the) feasible integer set is good. For (B), any cut that leaves the (convex hull of the) feasible integer set untouched is good. For (C), anything that isn't (A) or (B) is good.