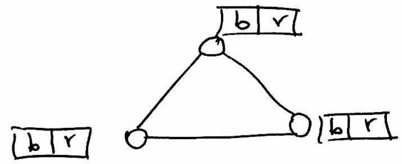
In the name of God, the Merciful, the Compassionate

AI Midterm Solution

1) a) False. It could fail to do so if there is a loop in the constraint graph. For instance, consider graph coloring problem and the following constraint graph:



where b stands for blue and r stands for red. The graph is completely consistent but it is obviously a dead-end.

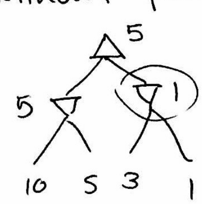
b) False. It depends on the step length α. If it is small enough, then we could guarantee a decrease in the function value in the next iteration. For example consider the function $f(x,y) = x^2 + y^2$ and the initial point $(1,1)$.
The gradient is $\nabla f = (2x, 2y)$. So in each iteration

$$\begin{cases} x_{n+1} \leftarrow x_n - 2\alpha x_n \\ y_{n+1} \leftarrow y_n - 2\alpha y_n \end{cases}, \quad \text{Let } \alpha = 2. \text{ We get } \begin{cases} x_2 = -3 \\ y_2 = -3 \end{cases}.$$

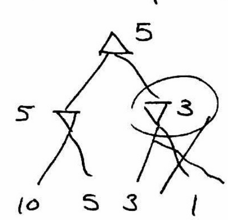
The function value would increase to $3^2 + 3^2 = 18$.

c) False. The values of internal nodes (i.e. the ones other than the root) could be inaccurate if their successors are pruned.
For example:

without pruning



with pruning



d) <u>True</u>. If the function value increases in a candidate next move, the probability of acceptance would be $p = e^{\frac{-(y_{new} - y_{old})}{T}}$. Note that if $y_{new} > y_{old}$, $\lim_{T \to 0} p = 0$ and the algorithm would not accept the candidate next move.

2) a) Let each state of the search tree be a set of selected nodes in the given graph. In each action, we add a node that is connected to all already selected nodes in the given graph. Search would terminate if we reach a state with K nodes in it. We could use DFS, for example, to search in this tree along with some heuristics to speed up the search.

b) Let each state be a subset of nodes in the given graph, with size K. Define the heuristic function as $h(s) = \binom{K}{2} - |E_s|$. where E_s is the set of edges between nodes in s in the given graph. We seek a state s^* with $h(s^*) = 0$. We could run a random search such as simulated annealing to optimize $h(s)$ by removing a node and replacing it with another node at each iteration.

c) Let x_i be a binary variable indicating whether or not the i-th node belongs to the clique.

Obviously $\sum_{i=1}^{n} x_i = K$ should be satisfied. Also,

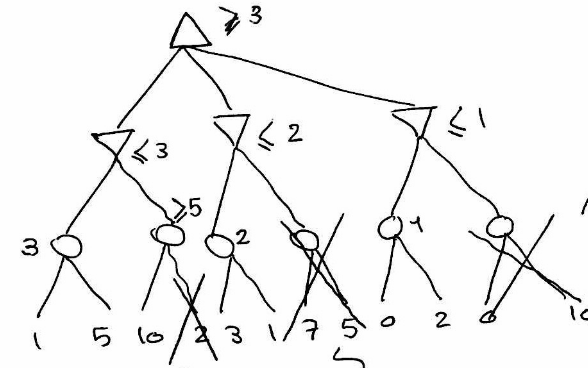
$$\forall i,j : \quad x_i = 1 \wedge x_j = 1 \Rightarrow W_{ij} = 1$$

where W_{ij} is the i,j entry of the graph adjacency matrix.

These are the heuristics that could guide the search to assign values to the variables:

- Minimum Remaining Value:
  If a node doesn't have at least K neighbors in the graph, we could immediately assign $\emptyset$ to it. Also, if a node is not connected to all already nodes in the cliques, we can safely ignore it, and assign $\emptyset$ to it.
- Degree Constraint:
  If a node has a higher degree in the given graph, assign its value first.
- Least Constraining Value:
  If assigning $\underline{1}$ to a variable causes a lot of other variables to be $\underline{0}$, (that is the neighbors of that variable are not connected to all of already selected nodes), we should first try 0.

3)



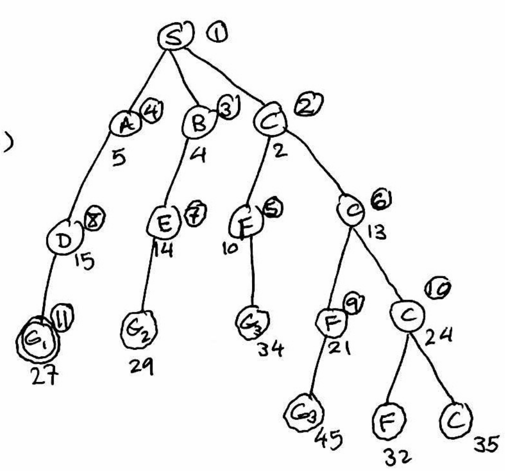
MIN would be less than or equal to 1, which is less than $\alpha=3$.

Note that once 10 is observed we would conclude the expectation node value is going to be ≥ 5. As we know the predecessor MIN node ≤ 3 there is no point in evaluating the next child.

MIN would be less than or equal to 2, which is less than 3, the best choice of the predecessor MAX.

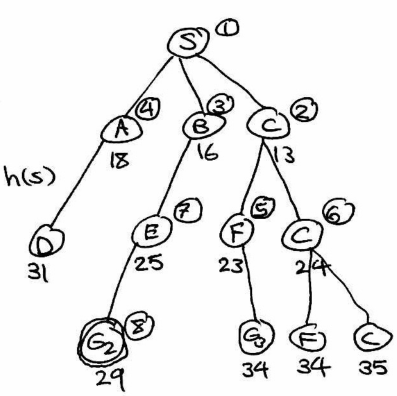
4) a) Uniform Cost Search

Value = $g(s)$



A* Search

Value $= g(s) + h(s)$



S, C, B, A, F, C, E, D, F, C, G_1

S, C, B, A, F, C, E, G_2

b) Because h is not admissible, for node D, $h(D)=16$ while there is only cost 12 to reach the goal state. So h overestimates the remaining cost-to-goal in state D and hence A* is not necessarily optimal.