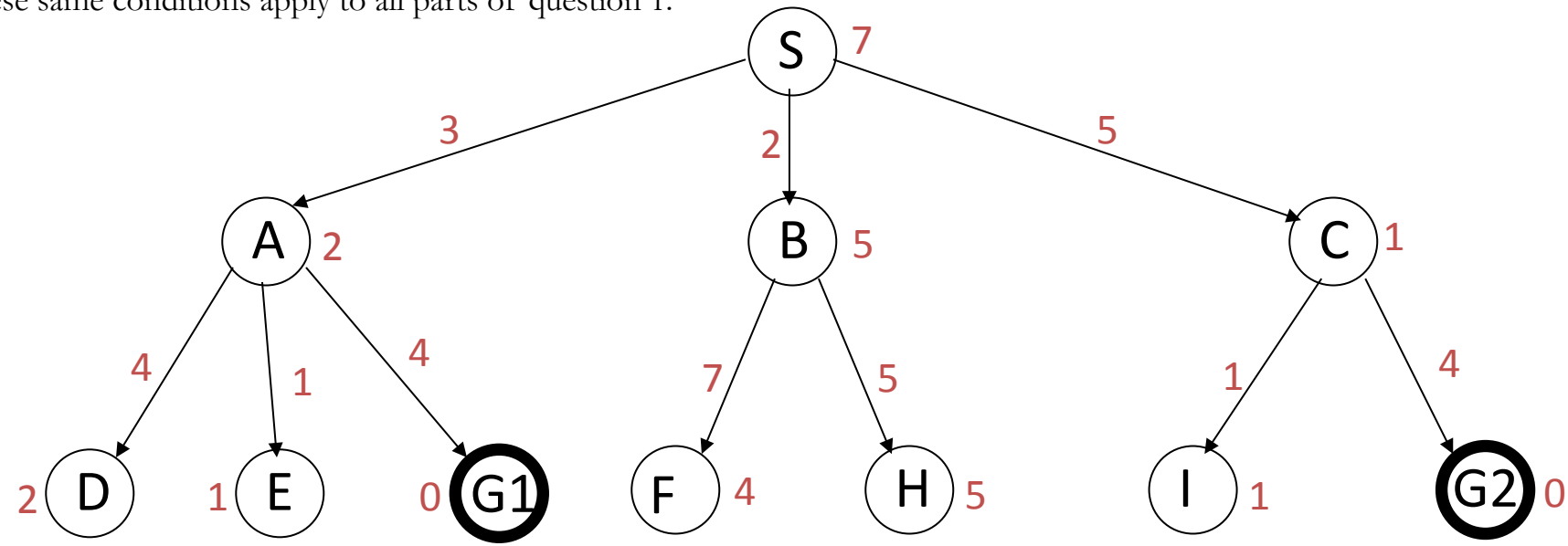
Exam 1 CS x260 Name: _____KEY_____

I will not use a source other than my brain on this exam (not neighbors, not notes, not books, etc):

_____ (please sign)

1. Consider the search graph below. The h value of a node is given adjacent to that node. The actual cost of traversing an arc is given adjacent to that arc. Node S is the start/initial state. Nodes G1 and G2 are goals. Leaf states/nodes have no successors. These same conditions apply to all parts of question 1.



a) (1 pt) Is h consistent? Yes or No. Explain.

No,. To be consistent, the f() value should never decrease along any path, but (for example) the f() value of S is 7, and the f() value of A is 5 (and/or of C is 6).

Grade each component all or nothing

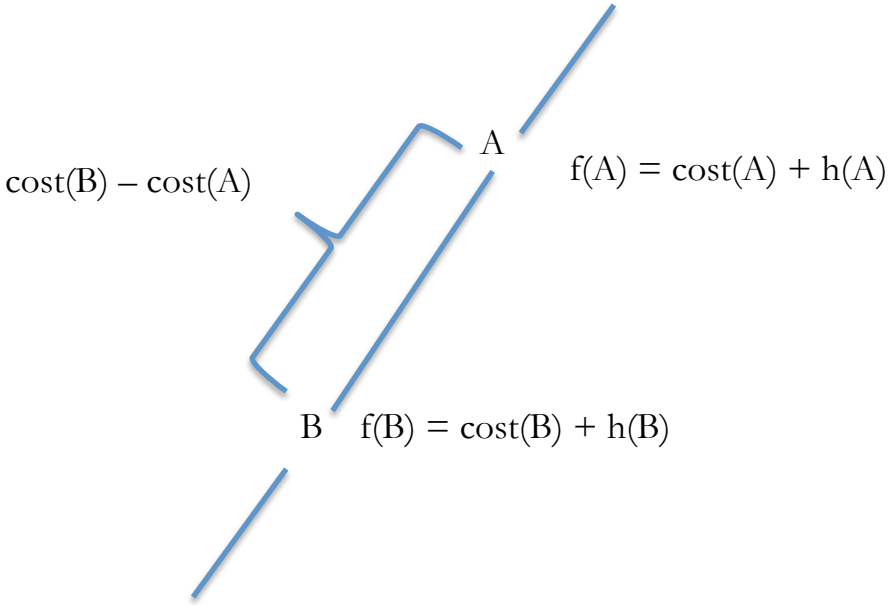
b) (2 pts) Is h admissible? Yes or No. Explain.

Yes. There is NO node at which the h() value overestimates the cost to the nearest goal node. (Technically, to be consistent, h() must be admissible too, but observation probably won't be referenced in the explanation of 1a).

3 minutes

More on Consistency of heurstic

To be consistent, the f() value should never decrease along any path.

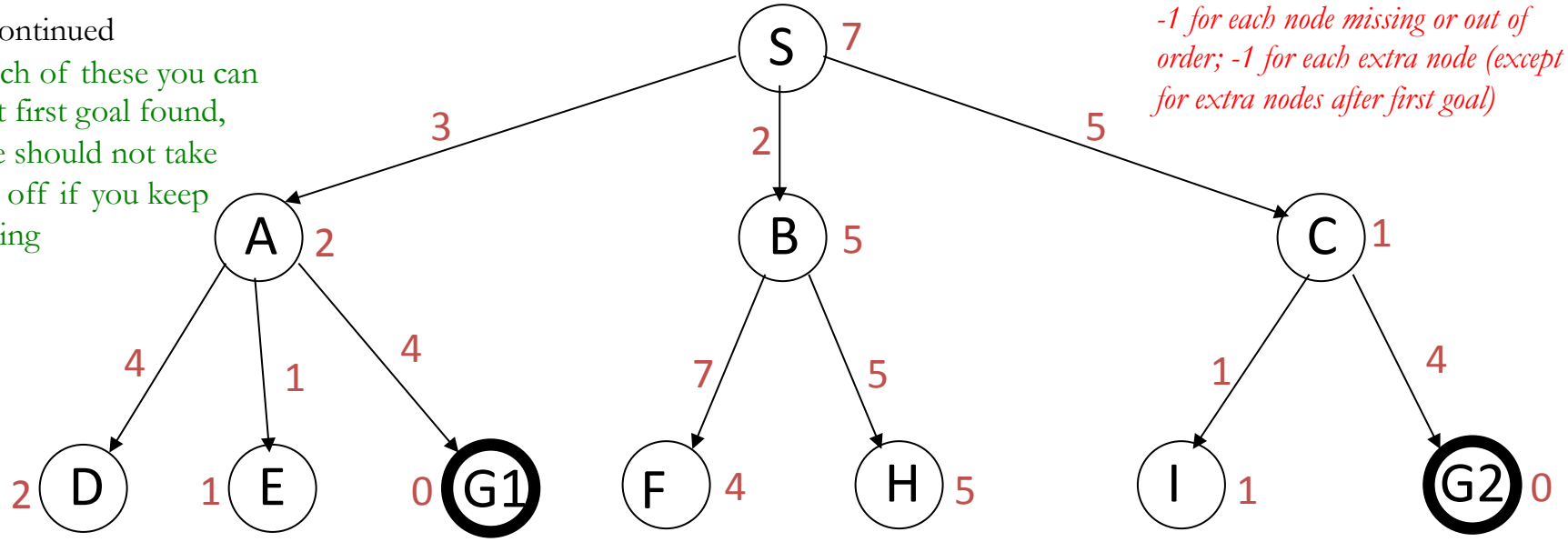


If f(A) > f(B) (i.e., consistency rule is violated), then
cost(A) + h(A) > cost(B) + h(B)
➔ cost(A) + h(A) - h(B) > cost(B)
➔ h(A) - h(B) > cost(B) - cost(A) intuitively, the heuristically-estimated cost between A and B is greater than the actual cost between A and B; the heuristic can is misleading in such places, and can be viewed as "overestimating" the cost between two arbitrary states

1 continued

For each of these you can stop at first goal found, but we should not take points off if you keep searching

-1 for each node missing or out of order; -1 for each extra node (except for extra nodes after first goal)



c) (3 pts) Give the order in which nodes are visited (i.e., checked for goalness) by **heuristic depth first search**. In the case of two or more nodes with the same evaluation score on the frontier, break the tie by visiting the nodes in alphabetical order as labeled above – this same convention applies to the remaining parts of this question.

The frontier is a stack, which each local set of neighbors pushed so that the least h() value local neighbor is popped next. The H() values of each node are shown in parentheses.

S(7), C(1), G2(0)

d) (3 pts) Give the order in which nodes are visited (i.e., checked for goalness) by **lowest cost-first search**.

The frontier is a priority queue, organized by cost() values. The cost() values of each node are shown in parentheses.

S(0), B(2), A(3), E(4), C(5), I(6), D(7), G1(7)

e) (3 pts) Give the order in which nodes are visited (i.e., checked for goalness) by **greedy best-first search**.

The frontier is a priority queue, organized by h() values. The h() values of each node are shown in parentheses.

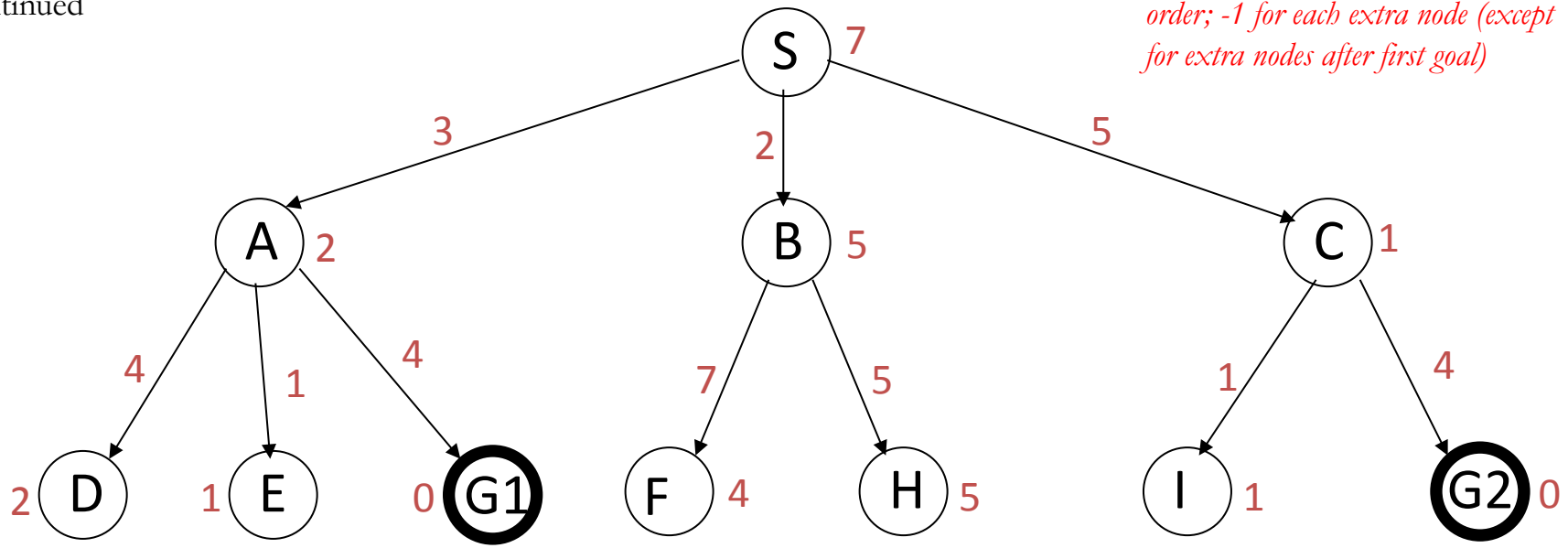
S(7), C(1), G2(0)

In this case, greedy best-first search and heuristic depth first search give the same answer. In the future you would likely see an exam in which they were distinguished

4 minutes

1 continued

-1 for each node missing or out of order; -1 for each extra node (except for extra nodes after first goal)



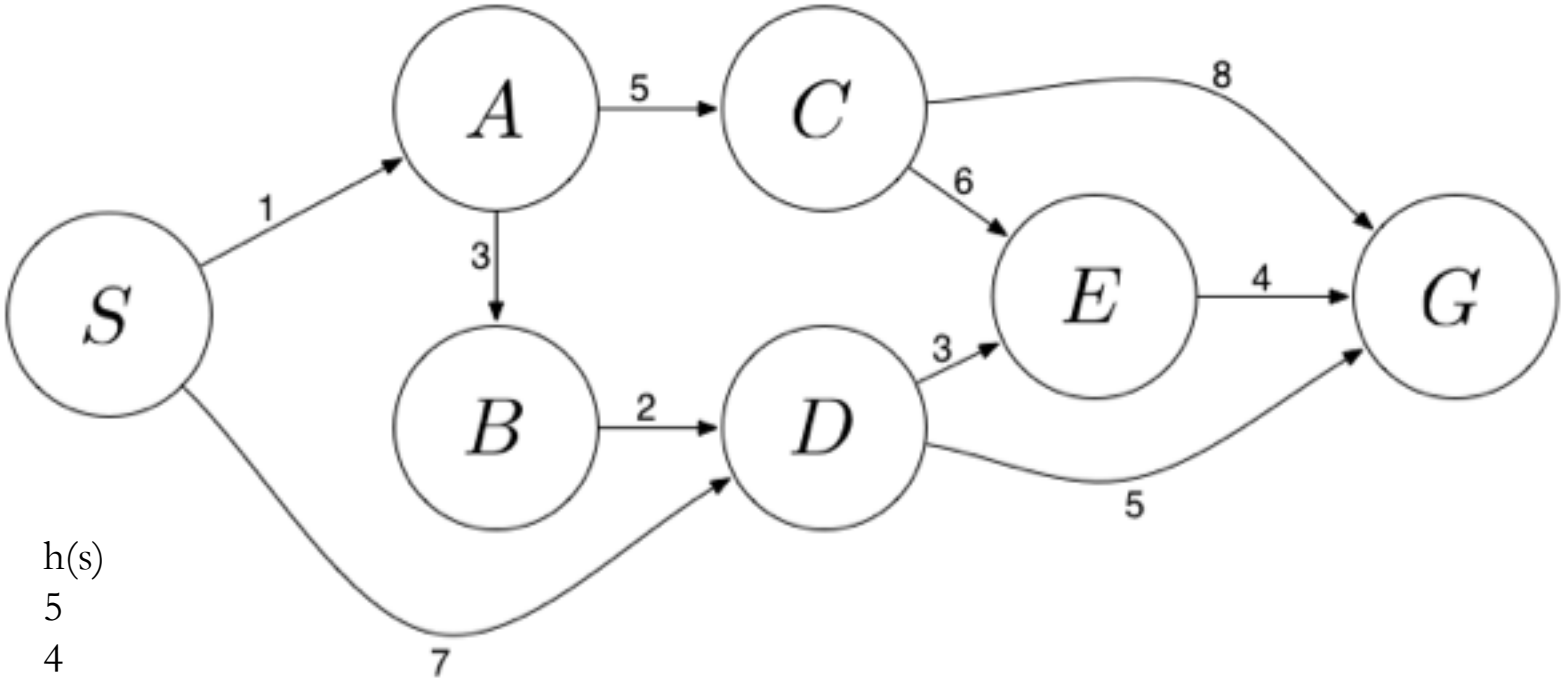
f) (3 pts) Give the order in which nodes are visited (i.e., checked for goalness) by **A***.

The frontier is a priority queue, organized by cost() + h() = f() values. The f() values of each node are shown in parentheses.

S(7), A(5), E(5), C(6), B(7), G1(7)

3 minutes

2. (1 pt) Use this search graph (together with actual costs that label arcs, and the heuristic estimates in the table to the bottom left), as necessary, to answer this question. S is the start state and G is the only goal. This problem is worth 1 point – save it for last.



State s	h(s)
S	5
A	4
B	4
C	7
D	1
E	4
G	0

In question 1 you indicated the order in which states were visited, but the generic search algorithm in the book associates a path with each state removed from the frontier. In this question your answer will be a path. For example, a depth first search would return path S→A→B→D→E→G (assuming we break ties on the frontier alphabetically as labeled).

What path would **iterative deepening A*** return?

To answer this question, you didn't have to understand the details of how IDA* operated. You just had to understand that when the heuristic is admissible, IDA*, like A*, will return the least cost path. In the future you might have to truly simulate IDA*

S→A→B→D→G (cost of 11, which doesn't need to be stated) *all or nothing*

No time estimate was given on this problem, because it might have varied between 15 seconds and several minutes

3. Consider the propositional knowledge base, KB:

p ← q ∧ r ∧ s

q ← y ∧ u

m ← y ∧ z

y ← m

z ← m

z ← r ∧ x

s ← w

r

w

y

u

a) (3 pts) Give a bottom-up proof of **p**, or explain why no such proof exists

consequence set

r

w (could list y and u here too, instead of below)

s follows from, w, s←w

y

u

q follows from y, u, q←y ∧ u

p follows from q, r, s, p← q ∧ r ∧ s

Need not list axioms explicitly in consequence set, but -1 for each inference step missing. Extra inference steps ok, if valid inferences

b) (3 pts) Give a top-down proof of **m**, or explain why no such proof exists

No proof, top down or otherwise

M isn't an axiom, and there is

- only one rule to conclude m,
- which in turn requires z, and
- z requires x (because z ← m would lead to infinite looping)
- but x is not an axiom and there is no rule for concluding x

Explanation can be briefer than this and can be correct (e.g., "no way to prove z", which I would accept")

d) (3 pts) Give a proof of **s** by contradiction using resolution (aka a resolution refutation proof). Convert any elements of the KB into clause form, as needed for this demonstration, and give the converted clauses.

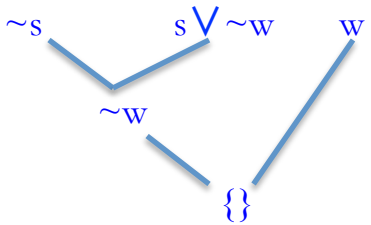
~s (negate the hypothesis)

s ∨ ~w (convert s←w to clause normal form)

~w (follows from resolving ~s with s ∨ ~w)

w (axiom)

{} empty set, contradiction (follows from resolving ~w with w)



Either form

5 minutes

4. Consider the following (feature-based) STRIPS-style operators:

Op1: precondition [p, q];
effects [~p, r]

Op2: precondition [q, r]
effects [~q, ~r, s]

Consider the goal to achieve is [r, q, w] and the **initial state** of the world is [p, s, w].

This doesn't play a role in this particular example, since search only extends one level

(a) (3 pts) Expand the **start state** of a *regression* planner, when run on this problem, giving the IMMEDIATE CHILDREN of the *start state* in the regression planner search:

[r, q, w]

Op1 (will add r, and ~p]

Op2 (effects not consistent with goal)

[p, q, w] only answer

-1 for an extra answer

(b) (2 pts) Of the two planning strategies, progression (forward) and regression (backward), which would you rely on to give you the quickest idea of what the "world" circumstances must be like if the planner is to achieve its goals:

____regression (backward)_____ ?

1 pt each

Which would give you the quickest idea of the way that the "world" circumstances can be in the near term:

____progression (forward)_____ ?

4 minutes

5. Consider the following macro/composite operator in STRIPS notation – this is a block stacking application of the type seen in lecture.

Unstack-A-B → **PutDown-A** → **Pickup-B** → **Stack-B-C**

Unstack-A-B Preconditions	Unstack-A-B Effects	PutDown-A Preconditions	PutDown-A Effects	Pickup-B Preconditions	Pickup-B Effects	Stack-B-C Preconditions	Stack-B-C Effects
on-A-B	~on-A-B	holding-A	~holding-A	onTab-B	~ onTab-B	holding-B	~holding-B
clear-A	~clear-A		clear-A	clear-B	~clear-B	clear-C	~clear-C
handEmpty	~handEmpty		handEmpty	handEmpty	~handEmpty		handEmpty
	holding-A		onTable-A		holding-B		clear-B
	clear-B						on-B-C

The basic operators making up the composite operator are labeled along the top (Unstack-A-B, PutDown-A, Pickup-B, Stack-B-C), with preconditions of each given below the operator name and to the left; effects to the right.

a) (2 pts) Give the preconditions of this macro/composite operator

-0.5 pt for each missing answer, and extra answer

on-A-B, clear-A, handEmpty (preconditions of first operator)
+ **onTab-B** (not added prior to Pickup-B)
+ **clear-C** (not added prior to Stack-B-C)

b) (2 pts) Give the effects of this macro/composite operator. You need only list un-negated effects (because we can build in a KB that allows reason from un-negated propositions to obtain the relevant negated propositions. For example, handEmpty → ~holding-A; handEmpty → ~holding-B; …; on-A-B → ~clear-B; on-B-C → ~clear-C, …)

clear-B, on-B-C (added by last operator and not present at beginning of composite operator; handEmpty not an effect)
+ **onTable-A** (added by PutDown-A, not subsequently removed/negated, and not originally present)

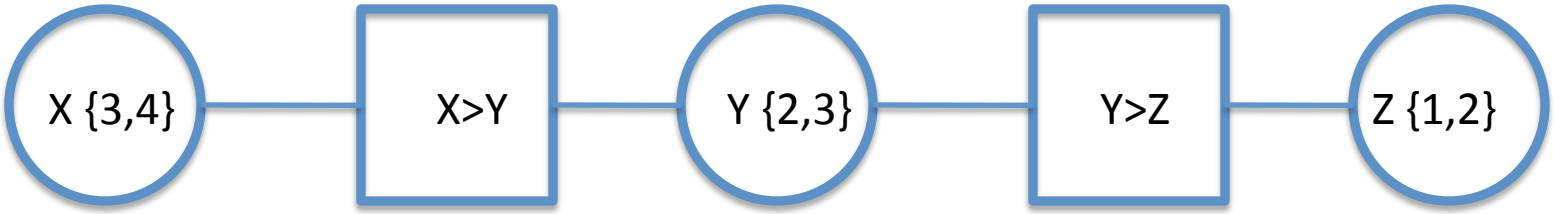
-0.5 pt for each missing answer, and extra answer
(but if they include extra negated answers, its ok)

5 minutes

6. Consider a CSP with the variables X, Y, Z, each with domain {1, 2, 3, 4}. Suppose the constraints are X > Y and Y > Z.

a) (3 pts) Draw the constraint network after applying the generalized arc consistency (GAC) algorithm to this CSP.

-0.5 pt for each missing answer answer from the domain of any variable. Be forgiving if the boxes and circles aren't explicitly shown, but the network structure should be clear



b) (3 pts) Eliminate variable Y in the network of part (a) -- that is, assuming the reduced domains obtained through the GAC algorithm. Show the new constraint on X and Z that results. We will try to grade this so that we minimize the cascading of errors (i.e., if you get part a wrong).

-0.5 for any missing element of either column

X > Y	
3	2
4	2
4	3

Y > Z	
2	1
3	1
3	2

X	Y	Z
3	2	1
4	2	1
4	3	1
4	3	2

or simply

X	Z
3	1
4	1
4	2

Intensional description: X > Z+1, but not answer I am looking for

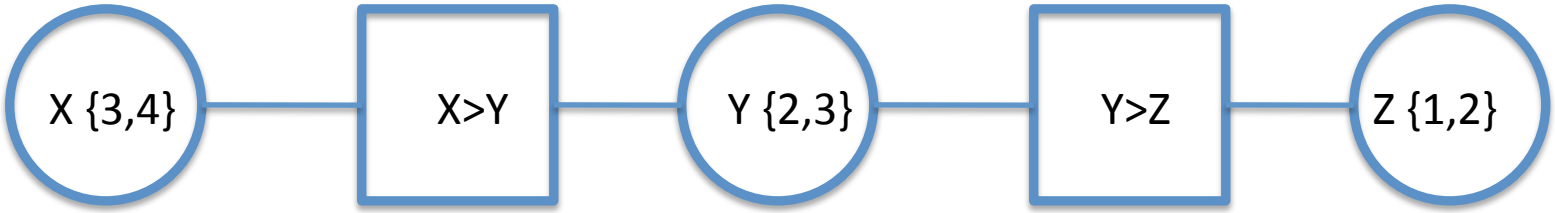
10 minutes

Showing work

6. Consider a CSP with the variables X, Y, Z, each with domain {1, 2, 3, 4}. Suppose the constraints are X > Y and Y > Z.

a) (3 pts) Draw the constraint network after applying the generalized arc consistency (GAC) algorithm to this CSP.

(X,X>Y) eliminate 1 from Domain(X) since 1 has no corresponding elements in Domain(Y) **X = {~~1,~~ 2, 3, 4}**
(Y,X>Y) eliminate 4 from Domain(Y) since 4 has no corresponding elements in Domain(X) **Y = {1, 2, 3, ~~4~~}**
(Y,Y>Z) eliminate 1 from Domain(Y) since 1 has no corresponding elements in Domain(Y) **Y = {~~1,~~ 2, 3, ~~4~~}**
(X,X>Y) revisit because of change to Domain of Y – eliminate 2 from Domain(X) **X = {~~1, 2,~~ 3, 4}**
(Z,Y>Z) eliminate 3 and 4 from Domain(Z) **Z = {1, ~~2, 3, 4~~}**



"All of the previously consistent arcs that could, as a result of pruning X have become inconsistent are placed back into the set to do. These are the arcs <Z, c'>, where c' is a constraint different from c that involves, and Z is a variable involved in c' other than X." (section 4.4 of Poole and Mackworth, 2nd Edition)

10 minutes

7.(a) (1 pt) Suppose that you have a search *tree* with a (maximum) branching factor of B. How many nodes will BOUNDED depth-first search generate and place on the frontier in the worst case if the depth bound is D. Assume that when a node is expanded, all of its children are generated and placed on the Frontier (aka fringe) at once. Give an exact answer (not an O-notation expression), but you need not "simplify" your answer. Write clearly!

All or nothing for each component

B^0 + B^1 + B^2 + . . . + B^D = $\sum_{i=0}^{D}$ B^i = (B^(D+1) – 1)/(B–1)

(b) (1 pt) Give the asymptotic TIME complexity (big-O expression) of a bounded depth-first search as a function of D and B assuming that time is proportional to number of states generated.

O(B^D)

(c) (1 pt) Give the asymptotic SPACE complexity (big-O expression) of a bounded DFS as a function of D and B assuming that space is proportional to the maximum number of states that must be retained in memory simultaneously.

O(BD) or O(B*D)

(d) (1 pt) Give the asymptotic TIME complexity (big-O expression) of a BREADTH-FIRST SEARCH (BFS) as outlined above (branching factor B to depth D) assuming that time is proportional to number of states generated.

O(B^D)

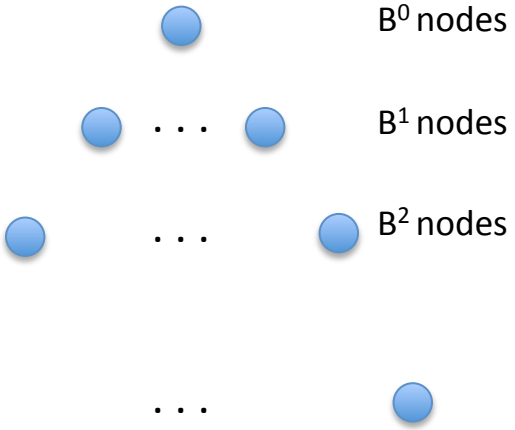
(e) (1 pt) Give the asymptotic SPACE complexity (big-O expression) of a BFS as outlined above (branching factor B, max depth D) assuming that space is proportional to the maximum number of states that must be retained in memory simultaneously, as a function of D and B.

O(B^D)

5 minutes

7.(a) (1 pt) Suppose that you have a search *tree* with a (maximum) branching factor of B. How many nodes will BOUNDED depth-first search generate and place on the frontier in the worst case if the depth bound is D. Assume that when a node is expanded, all of its children are generated and placed on the Frontier (aka fringe) at once. Give an exact answer (not an O-notation expression), but you need not "simplify" your answer. Write clearly!

B^0 + B^1 + B^2 + . . . + B^D = $\sum_{i=0}^{D}$ B^i = (B^(D+1) – 1)/(B–1)

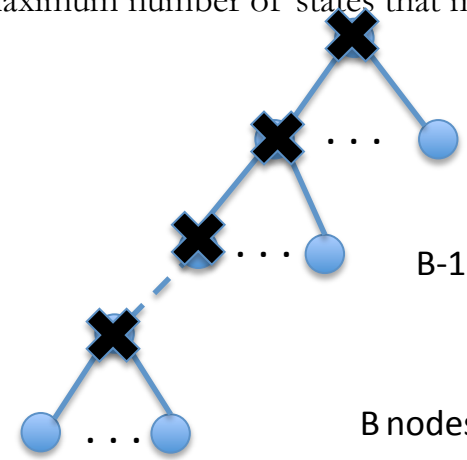


(b) (1 pt) Give the asymptotic TIME complexity (big-O expression) of a bounded depth-first search as a function of D and B assuming that time is proportional to number of states generated. B is considered a constant

(B^(D+1) – 1)/(B–1) (B^(D+1) – 1)/B) = B^D – 1/B = O(B^D)

(c) (1 pt) Give the asymptotic SPACE complexity (big-O expression) of a bounded DFS as a function of D and B assuming that space is proportional to the maximum number of states that must be retained in memory simultaneously.

O(BD) or O(B*D)



5 minutes

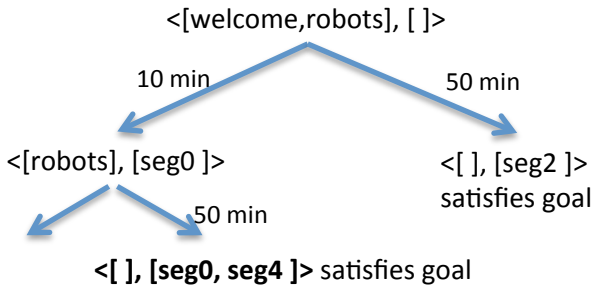
8. 4 pts) This question investigates using graph searching to design video presentations. Suppose there exists a database of video segments, together with their length in seconds and the topics covered, set up as follows:

Segment	Length	Topics covered
seg0	10	[welcome]
seg1	30	[skiing, views]
seg2	50	[welcome, artificial_intelligence, robots]
seg3	40	[graphics, dragons]
seg4	50	[skiing, robots]

Represent a node as a pair:

$$\langle To\_Cover, Segs \rangle ,$$

Taken from Poole and Mackworth, Artificial Intelligence



where $Segs$ is a list of segments that must be in the presentation, and To_Cover is a list of topics that also must be covered. Assume that none of the segments in $Segs$ cover any of the topics in To_Cover.

The neighbors of a node are obtained by first selecting a topic from To_Cover. There is a neighbor for each segment that covers the selected topic. [Part of this exercise is to think about the exact structure of these neighbors.]

For example, given the aforementioned database of segments, the neighbors of the node $\langle [welcome, robots] , [] \rangle$, assuming that $welcome$ was selected, are $\langle [] , [seg2] \rangle$ and $\langle [robots] , [seg0] \rangle$.

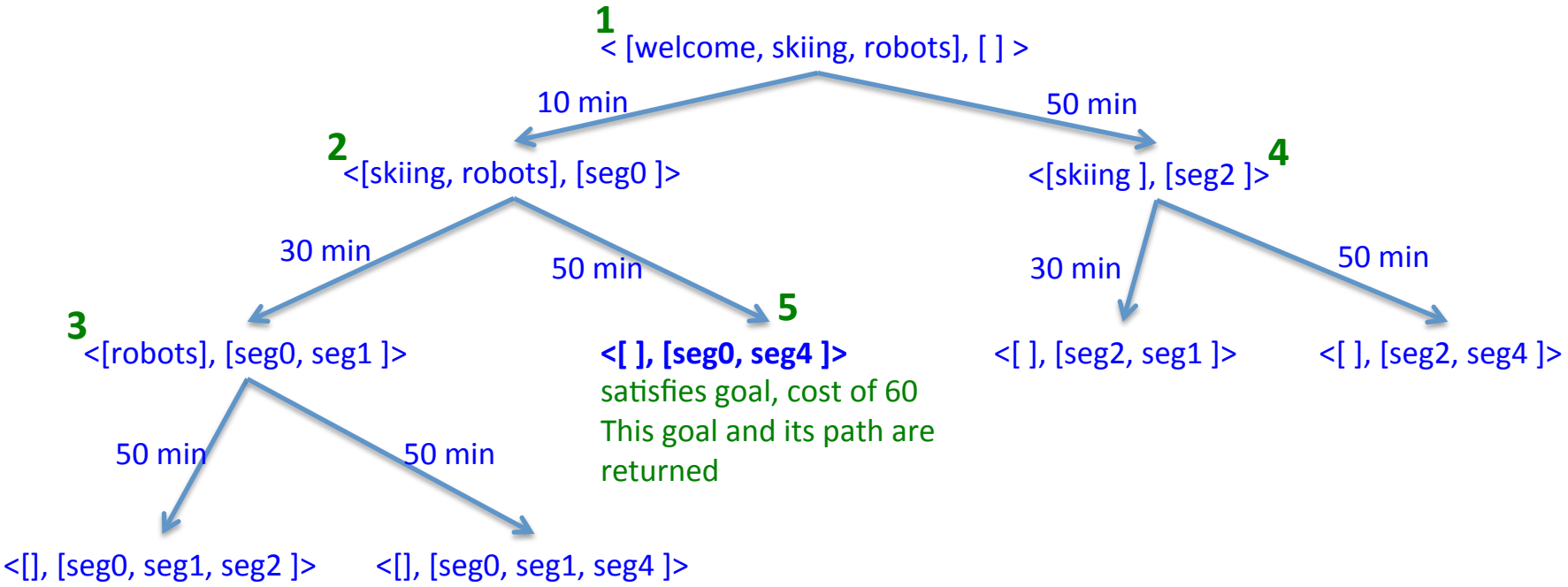
Thus, each arc adds exactly one segment but can cover one or more topics. Suppose that the cost of the arc is equal to the time of the segment added.

The goal is to design a presentation that covers all of the topics in $MustCover$. The starting node is $\langle MustCover, [] \rangle$, and the goal nodes are of the form $\langle [] , Presentation \rangle$. The cost of the path from a start node to a goal node is the time of the presentation. Thus, an optimal presentation is a shortest presentation that covers all of the topics in $MustCover$.

(a) Suppose that the goal is to cover the topics $[welcome, skiing, robots]$ and the algorithm always select the leftmost topic to find the neighbors for each node. Draw the search space expanded for a lowest-cost-first search until the first solution is found. This should show all nodes expanded, which node is a goal node, and the frontier when the goal was found.

15 minutes

Answer to question 8 here



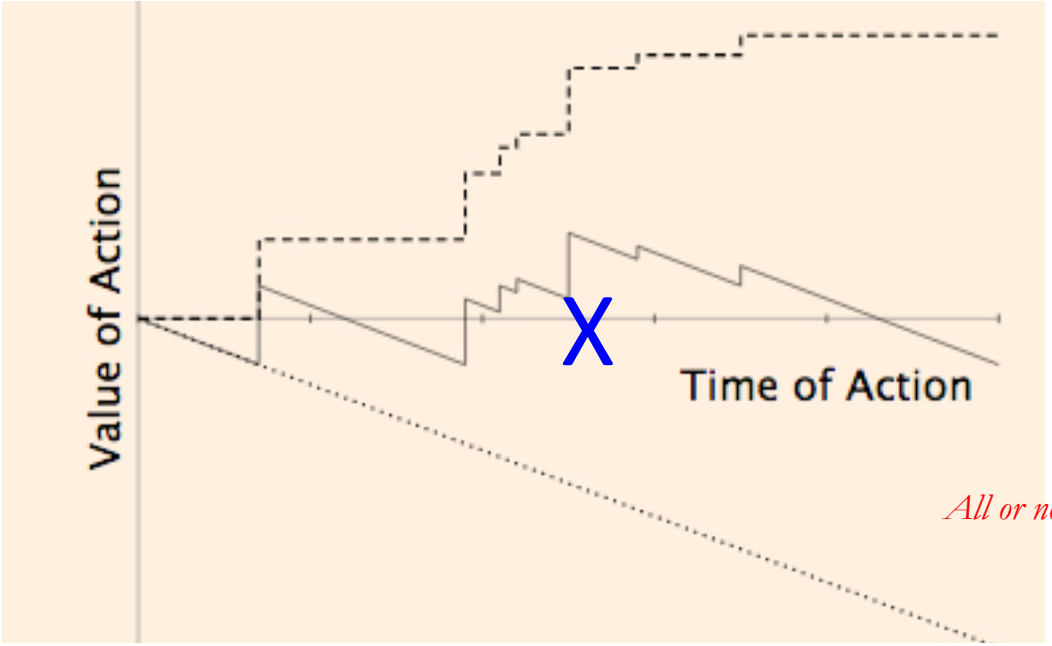
both seg0 and seg2 cover 'welcome', and this might be a constraint violation, but not relevant to this problem

Numbers 1-5 give the order in which nodes are removed from Frontier, checked for goalness, and expanded

The question asks for the "search space expanded", which implies all the nodes above, but if only those nodes "visited" are given (removed from Frontier), that would be acceptable this time.

-1 for any missing or extra node among 1-5, and their ancestors

9. (2 pts) An anytime algorithm continues to search for solutions even after the first solution, and subsequent solutions, are found. It is called "anytime" because it can output (or execute) the best solution found so far, any time a solution is required. Unfortunately, there is often a penalty associated with waiting to output/execute a solution, which the agent may not know about. This graph, taken from an Example by Poole and Mackworth (our textbook), illustrates that the quality of solutions found through reasoning increase with time by the top-most dashed line (we assume that the agent remembers and can output the best solution found so far, so this line will never decrease). But the quality of solutions implied by this line don't take into account the cost of waiting to act on the solution. The bottom-most dotted line shows the penalty associated with waiting to act on a solution.



All or nothing for each component

a) Put an 'X' along the timeline at the point where it is best for the agent to take action on the best solution found so far (even if the agent doesn't necessarily know that the time you indicate is best)

b) If the agent remembered solutions found on prior searches, how could these remembered solutions be used to increase the agent's performance when the same problems were encountered again?

The prior solutions could be offered up more quickly, and acted on, so that the penalty did not diminish their effects as much.

3 minutes

More on anytime algorithms

```
1:procedure Search(G, S, goal)
2:     Inputs
3:         G: graph with nodes N and arcs A
4:         s: start node
5:         goal: Boolean function of nodes
6:     Output
7:         path from s to a node for which goal is true
8:         or ⊥ if there are no solution paths
9:     Local
10:        Frontier: set of paths
11:    Frontier := {⟨s⟩}
12:    while Frontier ≠ {} do
13:        select and remove ⟨n_0, ..., n_k⟩ from Frontier
14:        if goal(n_k) then
15:            return ⟨n_0, ..., n_k⟩
16:    else Frontier := Frontier ∪ {⟨n_0, ..., n_k, n⟩ : ⟨n_k, n⟩ ∈ A}
17:    return ⊥
```

Figure 3.4: Search: generic graph searching algorithm

+ set of solutions implemented as priority queue organized by solution "quality"

Initialize set of solutions

Add <n0,…,nk> to set of solutions

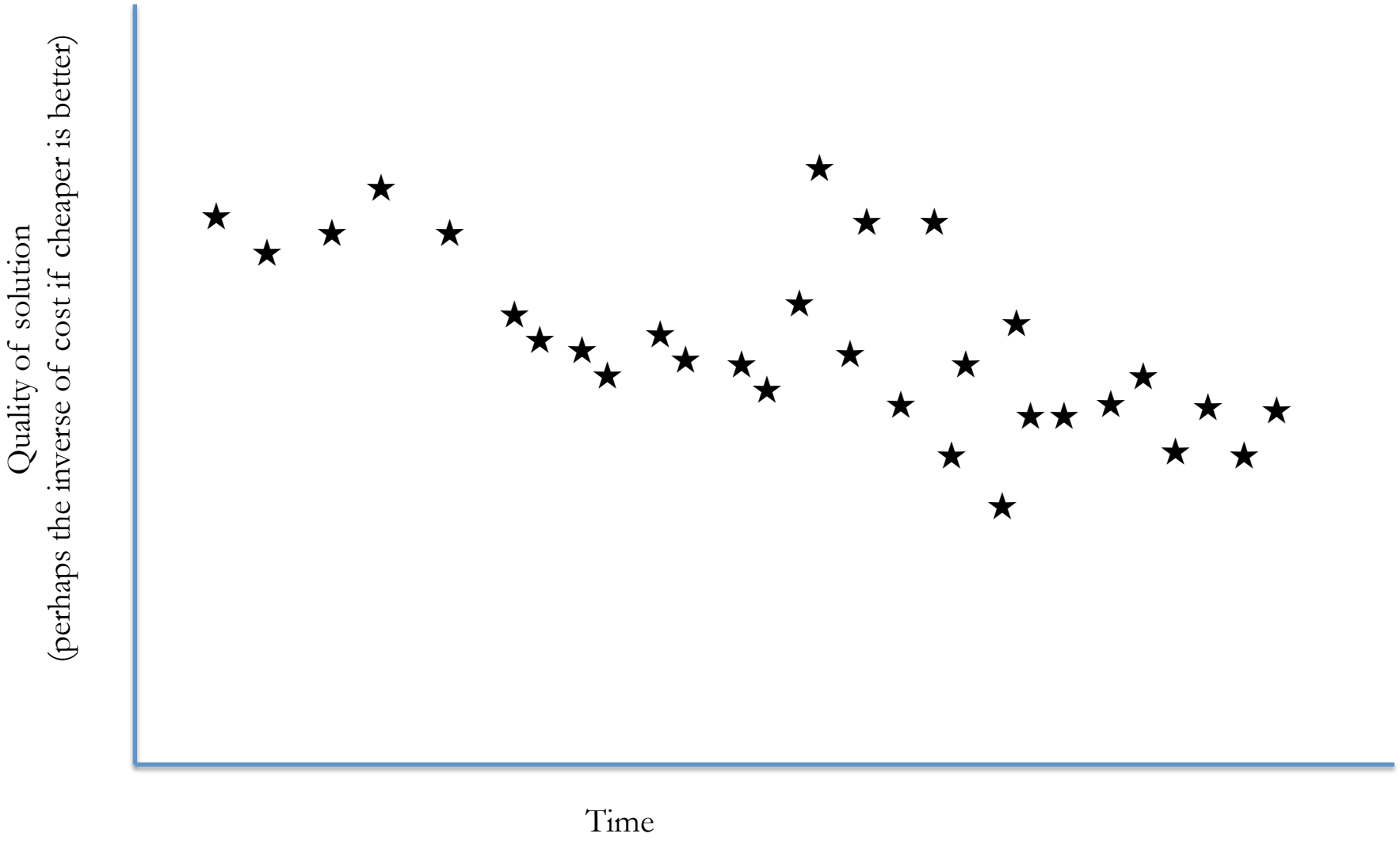
Suppose this is changed from a termination step, to a step that adds the solution to a set of solutions continues searching

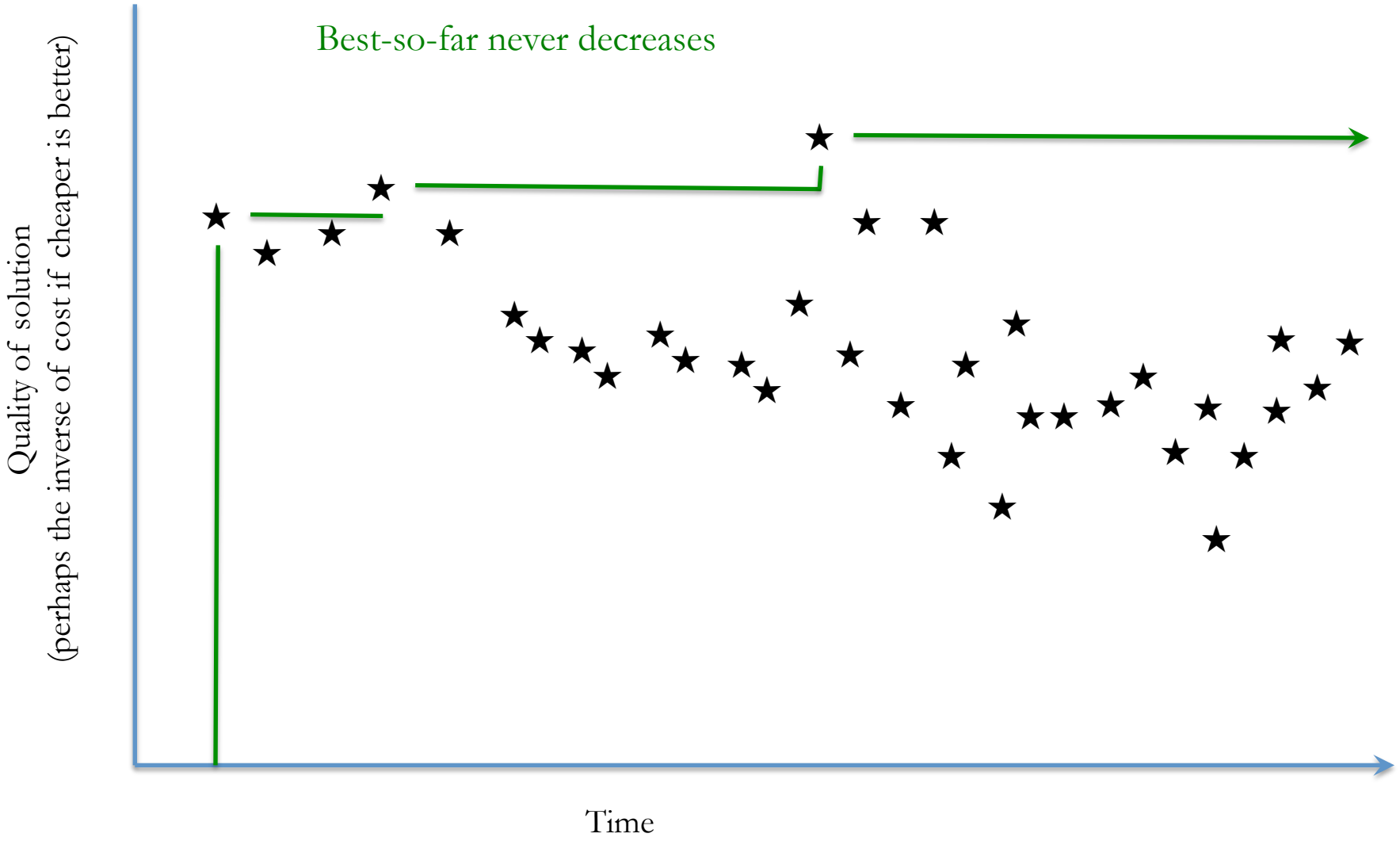
converted to anytime algorithm

This algorithm would run "forever" or whenever the state space was fully enumerated.

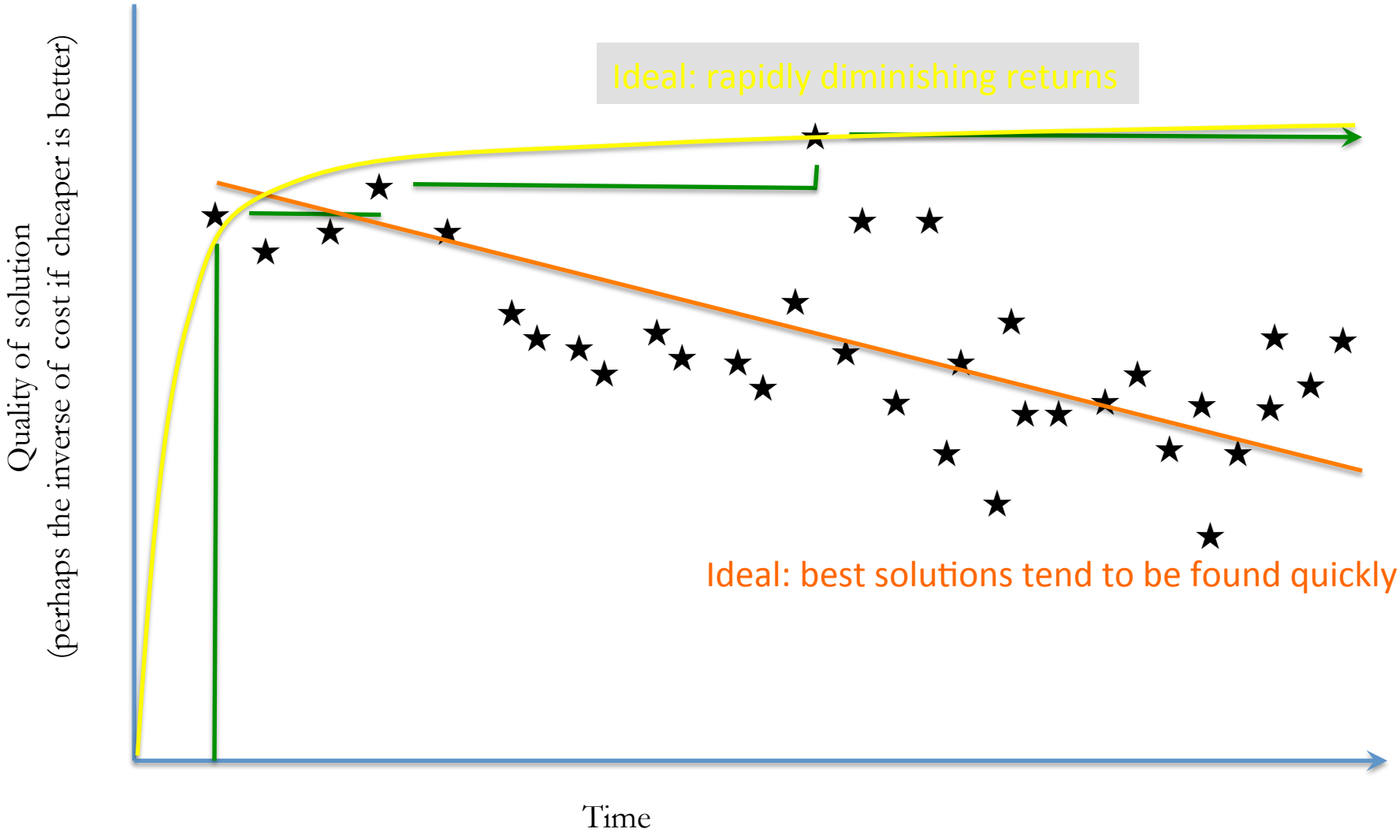
What characteristics are important in an anytime algorithm, since all solutions will be enumerated in any case?

We want high quality solutions found earlier in the search!


Quality of solution
(perhaps the inverse of cost if cheaper is better)
Time



We want something available that is very good, very fast, even if better solutions might trickle in, characterized by rapidly diminishing returns



We want something available that is very good, very fast, even if better solutions might trickle in, characterized by rapidly diminishing returns