CSE 3101 Design and Analysis of Algorithms
Meta Steps for Unit 4

Jeff Edmonds

This contains the **most important** concepts in this unit. You will not be able to pass this course without knowing and understanding these. The steps provided **must** be followed on all assignments and tests in this course. Do **not** believe that because you know the material, you can answer the questions in your own way. Though this material is necessary, it does not contain everything that you need. You must read the book, go to class, review the slides, and ask lots of question.

Chapter 16: Greedy Algorithms

When providing a greedy algorithm and its proof of correctness, the following are all the paragraphs that should be included, their headings, and what they should contain.

Carefully read the full description of the steps on pg 225 of HTA. Sorry, in the book A_t is not defined. $optS_{LI}$ is used instead of S_{t-1}, $optS_{ours}$ instead of S_t, and *consistent* instead of *extends*.

Specifications: This is likely part of the question and hence does not need to be written. However, before writing anything consider: What are the objects in the instance? What decisions need to be made about each such object in order to produce a solution. When does such a sequence of decisions make a valid/invalid solution? What is the measure according to which the solution is optimal? When writing your answer be sure to use the details of the question at hand.

The Greedy Choice: Each iteration the algorithm grabs the object which according to some simple criteria, seems to be the "best" (or "worst") from amongst the unconsidered objects in the instance. **Describe this criteria. Is it a fixed or an adaptive decision, i.e. does it depend on what objects have been seen already?** The algorithm then makes an irrevocable decision about this object. **How is this decision made?**

Hint: Usually the decision made is simply to commit to the next object unless it creates conflicts with the objects that the algorithm has already been committed to.

Example: With MST, the algorithm chooses the edge with the smallest weight. It decides to commit to this edge if it does not create a cycle with the previously committed to edges.

A_t: Let A_t denote the choices made by the algorithm during the first t time steps. See Figure 1.

Example: With MST, A_t specifies for each of the t edge with the smallest weight whether it was committed to or rejected.

S_t Extends A_t: For S_t to be solution for our instance, it must make a decision for every object in our instance. We say that S_t is a solution that *extends* A_t, if for the objects that algorithm has already made a decision about, S_t makes the same decision, i.e. $\forall$ object $Obj \in$ instance I, if $A_t(Obj) \neq$ undefined, then $S_t(Obj) = A_t(Obj)$.

Example: With MST, S_t specifies a full spanning tree of the graph.

The Loop Invariant: We have not gone wrong. There is at least one valid optimal solution S_t that extends the choices A_t made so far by the algorithm.
(If the problem is such that an instance may have no valid solutions, then the loop invariant needs to be modified to: " If our instance does have a valid solution, then there is at least one valid optimal solution S_t that extends the choices A_t made so far by the algorithm.)

Initially ($\langle pre \rangle \rightarrow \langle LI \rangle$): Initially no choices have been made (i.e. $A_0 = \emptyset$) and hence all optimal solutions S_0 extend these choices.

Maintaining the Loop Invariant ($\langle LI_{t-1} \rangle$ & not $\langle exit \rangle$ & $code_{loop} \rightarrow \langle LI_t \rangle$): Consider an arbitrary iteration.

Fly in From Mars: The algorithm has iterated $t-1$ times making decisions A_{t-1} about the first $t-1$ objects and is now at the top of the loop. **If useful state or at least think about the types of decisions the algorithm makes.** All that we know is that the loop invariant is true and the exit condition is not.

S_{t-1}: The loop invariant states that there is at least one optimal solution that extends the choices made by the algorithm before this iteration. Let S_{t-1} denote one such solution. (Note that this is a full solution and as such specifies a decision about each object in the instance.) If you like have a fairy god mother hold it.

Taking a Step: Let Obj_t denote the next "best" object chosen by the algorithm during the t^{th} iteration. Denote $A_t = A_{t-1}+$ decision made about Obj_t.

Example: With MST, there are two cases. First suppose that the algorithm commits to the next best edge $Obj_t = \langle u, v\rangle$ because it does not create a cycle with the previously committed to edges.

Instructions for Modifying S_{t-1}: Give the prover's detailed instructions on how the fairy godmother should modify S_{t-1}. We will use S_t to denote what she constructs.

Warning: Be sure you give a *full* description of how S_t is constructed *before* you try proving anything about it. When marking, this is a first thing I look at.

Perhaps S_t is Already Consistent: S_{t-1} is already consistent with the first $t-1$ decisions made by the algorithm. If S_{t-1} happens to be consistent with the decision A_t the algorithm made about Obj_t, then we are done. Sometimes because of the decisions already made by the algorithm, the algorithm is forced to make this decision about Obj_t. Usually, this occurs when the algorithm's instructions are to commit to taking Obj_t unless it can't because Obj_t "conflicts" with the previous decisions made by the algorithm. In such a case, if the algorithm rejected Obj_t, then we know it was forced to. Such forced decision are good for the prover, because we know that because the fairy godmother has made the same first $t-1$ decisions, that she too is forced to make this same decision about Obj_t.

Making S_t Consistent: If the decision the fairy godmother has made about Obj_t is different than A_t, then the first modification the prover must tell the fairy godmother is to change this decision, namely $S'_t = S_{t-1}+$ A_t's decision made about Obj_t. For example, if the algorithm committed to adding Obj_t and she did not, then get her to add it.

Check for and Fix Any Conflict: We know that S_{t-1} is a valid solution as it is. Changing the decision made about Obj_t most likely makes it no longer valid. Find and describe some object Obj' in S_{t-1} (but not in A_{t-1}) that "conflicts" with this changed decision about Obj_t and describe a way of changing the decision made by the fairy godmother about Obj' so that the solution S_t is made to be again valid. Define S_t to be the new solution $S_t = S_{t-1}+$ A_t's decision made about Obj_t+ the prover's new decision about Obj'. Maybe the decision about more than one object needs to be changed. The book uses the notation S_{ours} instead of S_t because it is the prover who creates the new solution. However, people got confused, thinking that this is the solution created by the algorithm.

The Fairy Godmother's Decisions are not Ordered: Do **not** say "Add object Obj_t that the algorithm took next and delete the object Obj'_t that the fairy godmother took next." Unlike the algorithm, the fairy godmother does not make decisions about the objects in any particular order. Hence, it is meaningless to say, "Let Obj'_t be the object that the fairy godmother considered at time t." Because the algorithm makes a decision about an object each time step, we can define Obj_t to be the one it makes a decision about at time t. Yes, the fairy godmother has made a decision the objects $Obj_1, Obj_2, \ldots, Obj_{t-1}$ and in fact made the same decision about these that that algorithm did. Yes, she made a decision about Obj_t and if fact about about every object in the input. This does not mean that there is an object she decided about "next". Note, we might use the notation Obj'_t instead of Obj', but this is because this is the object that **you** as the prover choose to change at time t.

Example: For MST, if S_{t-1} already contains the edge $Obj_t = \langle u, v\rangle$ added by the algorithm then we are done. Otherwise, because S_{t-1} spans the graph without cycles, it must contain some other unique path from u to v. The algorithm can't have already committed to all the edges in this

path or adding $\langle u, v\rangle$ would create a cycle. The algorithm can't have already rejected an edge in this path or else for consistency S_{t-1} would have rejected it too. Let $Obj'_t = \langle u', v'\rangle$ be some edge in this path from u to v in S_{t-1} for which no decision has been yet by the algorithm. Let $S_t = S_{t-1} + \langle u, v\rangle - \langle u', v'\rangle$.

ProvingS_t is a Valid Solution: By the loop invariant, S_{t-1} is a valid solution. (i.e. **repeat what it means to be valid**.) The prover made sure that his modifications did not make it invalid. The changes may temporarily have made it invalid, but all of these problem were fixed. **Add a few sentences.** Hence, S_t is a valid solution.

Warning: Do not use the word "valid" without defining it. Let me know that you have read the question. What does it mean for this problem for a solution to be valid? Careful. What needs to be proved to prove that *everything* about S_t is valid?

Example: With MST, we must prove that S_t is acyclic and spans the graph. Even if you can't get the full proof be sure to minimally state what it is that needs to be proved.

Adding $\langle u, v\rangle$ created one cycle with the unique path in S_{t-1} from u to v, but we broke this cycle by removing $Obj'_t = \langle u', v'\rangle$. Hence, S_t is acyclic as was S_{t-1}. Because S_{t-1} spans the graph, it has a path from node x to node y. The path from x to y in S_t may have to go the other way around the cycle that was changed. A picture would help. Hence, S_t is acyclic.

Proving S_t Extends A_t: By the loop invariant S_{t-1} extends A_{t-1}. The prover made sure that his modifications did not change any of these decisions and changed S_{t-1}'s decision about Obj_t. **Add a few sentences.** Hence, S_t extends both with earlier decisions made by algorithm and this most recent decision, i.e. extends A_t.

Warning: Do not use the word "extends" without defining it. Let me know that you have read the question. What decisions did the Algorithm make before, that we know S_{t-1} extends, and how are you going to prove that S_t is still consistent with these? What decision did the algorithm just make and how are you going to prove that S_t is now consistent with it?

Example: For MST, we were careful that the only edge $Obj'_t = \langle u', v'\rangle$ removed from S_{t-1} had not preciously been committed to by the algorithm and we were careful to add the edge $Obj_t = \langle u, v\rangle$ just committed to by the algorithm.

ProvingS_t is an Optimal Solution: By the loop invariant, S_{t-1} is one of the optimal solutions for this instance. The prover made sure that his modifications did not make S_t worse than S_{t-1}. **Add a few sentences.** Hence, S_t is an optimal solution.

Warning: Again let me know that you have read the question. What does it mean for this problem for a solution to be optimal? i.e. what measure of success is minimized or maximums. What do you need to prove that S_t is at least as good as S_{t-1}? No need to waist your efforts or insult the fairy god mother by mentioning that in some cases S_t is in fact better than S_{t-1}.

Example: For MST, $Obj'_t = \langle u', v'\rangle$ is defined to be some edge for which no decision has been yet by the algorithm. The algorithm sorted the edges by weight and is currently deciding about edge $Obj_t = \langle u, v\rangle$. Hence the weight of $\langle u, v\rangle$ is less than or equal to that of $\langle u', v'\rangle$ and hence the weight of S_t is less than or equal to that of S_{t-1}.

$\rightarrow$ **$\langle LI_t\rangle$:** Because S_t witnesses the fact that there is at least one optimal solution that extends A_t, we know that the loop invariant has been maintained.

Second Case: With MST, now suppose the the algorithm rejects the next best edge $Obj_t = \langle u, v\rangle$ because it creates a cycle with the previously committed to edges. In this case, the god mother must have the same the previously committed to edges and hence also can't take $\langle u, v\rangle$. Let $S_t = S_{t-1}$.

Exiting Loop ($\langle LI\rangle$ & $\langle exit\rangle \rightarrow \langle post\rangle$): By the exit condition the algorithm has made a decision about every object in the instance. Hence, the algorithm has a full solution. By the loop invariant, this extends an optimal valid solution. Hence, the algorithm must have an optimal valid solution.

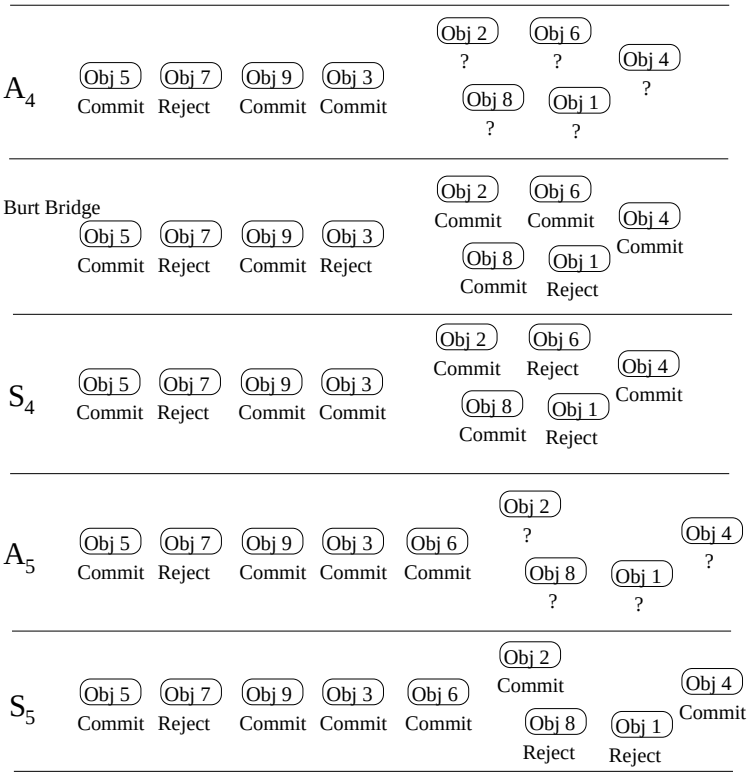

Figure 1: Here A_4 denotes the decisions made by the greedy algorithm during the first four time steps. Based on its greedy criteria, it chose to look at objects 5, 7, 9, and 3 in the first 4 time steps and to irrevocably commit to all of these except for object 7. (Note that unlike in the text above, these numbers index the objects, not the time step that they were chosen.) Irrevocably committing to object 3 means that it has "burnt the bridge" of having any optimal solutions in which object 3 is rejected. The loop invariant assures us that the algorithm has not burned in this way all of the optimal solutions. Here S_4, held by the fairy godmother is a valid optimal solution that extends the decisions A_4 that the algorithm has made so far. Being a full solution, it makes a decision about every object in the input instance, but being consistent with the algorithm it also commits to objects 4, 9, and 3 and rejects object 7. What other decisions S_4 makes is unknown to the algorithm and to the prover. In the 5^{th} time step, the algorithm commits to object 6. The prover gives instructions to the fairy godmother to commit to object 6 as well, if she has not already done so. Doing this alone makes the solution she is holding either in invalid or of less worth. Hence the prover also instructions to the fairy godmother to change her commitment to object 8 to a reject. Let S_5 denote what she holds after the changes. The prover then proceeds to prove that S_5 is a valid solution, that it is an extension of A_5 that the algorithm has done so far and is optimal.