Notes on the correctness of greedy algorithms

Simone Silvestri  
Department of Computer Science  
Missouri University of Science and Technology  
silvestris@mst.edu

I. INTRODUCTION

Greedy algorithms are a natural solution to many problems. These algorithms are generally efficient in terms of complexity and optimally solve a vast variety of problems. In this notes we discuss general guidelines to prove the correctness of a greedy algorithm, i.e. to prove that the algorithm actually outputs the optimal solution of the problem. In addition, we apply these guidelines to prove the correctness of the earliest termination greedy algorithm for the activity selection problem.

Note that proving the correctness of an algorithm always requires creativity, ingenuity, and also...luck. If the tentative proofing the correctness fails, we may get indications on a possible counter example, i.e. an input for which the algorithm does not output the optimal solution.

Algorithm 1: A general greedy algorithm

| Input: A set \( A = \{a_1, \ldots, a_n\} \) |
| Output: A solution \( S \subseteq A \) |

1. \( S = \emptyset \);  
2. while An element in \( A \setminus S \) can be added to \( S \) do  
3. \( a_k = \text{best element in } A \setminus S \);  
4. \( S = S \cup \{a_k\} \);  
5. return \( S \)

Algorithm 1 shows the structure of a general greedy algorithm. The algorithm takes as an input a set of elements \( A \) and returns a set \( S \subseteq A \) as a solution. \( S \) is initial empty and at each iteration the best element in \( A \setminus S \) is selected. The while loop terminates as soon as no more element can be added to \( S \) and \( S \) is returned. Depending of the specific problem under consideration, the general algorithm can be modified accordingly.

II. GENERAL GUIDELINES FOR THE CORRECTNESS OF GREEDY ALGORITHMS

The proof of the correctness of a greedy algorithm is based on three main steps:

1. The algorithm terminates, i.e. the while loop performs a finite number of iterations.
2. The partial solution produced at every iteration of the algorithm is a subset of an optimal solution, i.e. for each iteration there exists an optimal solution \( S^* \) s.t. \( S \subseteq S^* \).
3. The final solution output by the algorithm is indeed an optimal solution, i.e. it cannot be further extended.

III. PROOF OF THE EARLIEST TERMINATION ACTIVITY SELECTION ALGORITHM

Algorithm 2 shows the pseudo code of the earliest termination greedy algorithm for the activity selection problem.

Algorithm 2: Earliest termination

| Input: A set \( A = \{a_1, \ldots, a_n\} \) |
| Output: A solution \( S \subseteq A \) |

1. \( S = \emptyset \);  
2. while \( \exists a \in A \text{ s.t. } S \cup \{a\} \text{ does not create overlaps} \) do  
3. \( D = \text{subset of } A \setminus S \text{ of activities that do not overlap} \)  
4. \( a_k = \text{arg min}_{a_i \in D} f_i \);  
5. \( S = S \cup \{a_k\} \);  
6. return \( S \)

Theorem 3.1: The earliest termination greedy algorithm finds the optimal solution to the activity selection problem.

Proof: We structure the proof according to the general guidelines in Section II.

1: At each iteration of the while loop we add a new activity to \( S \). This can be repeated at most for all activities in \( A \), i.e. for at most \( n \) iterations.

2: Let \( S_h \) be the solution at the \( h \)-th iteration. Let \( m \) be the total number of iterations of the while loop, hence \( h = 0, \ldots, m \). We need to prove that \( \forall h = 0, \ldots, m \exists \text{ an optimal solution } S^* \text{ s.t. } S_h \subseteq S^* \). We prove this by induction.

Base case, \( h = 0 \): the statement is true because \( S_0 = \emptyset \), and \( \emptyset \) is a subset of any set, and thus \( S_0 \subseteq S^* \) for any optimal solution \( S^* \).

Inductive hypothesis: The statement is true at the \( h \)-th iteration, i.e. \( \exists \text{ an optimal solution } S^* \text{ s.t. } S_h \subseteq S^* \).

Inductive step: We prove that the statement is true at the iteration \( h + 1 \). We know that \( S_h \subseteq S^* \). Let \( a_k \) be the activity selected at the \( h + 1 \) iteration.

If \( a_k \in S^* \) the statement is true, since \( S^* \) is the optimal solution that includes \( S_{h+1} \).

If \( a_k \notin S^* \) we need to find another optimal solution that contains \( a_k \) and all the activities in \( S_h \). In particular, we look for an activity in \( S^* \) that can be substituted with \( a_k \). Such activity exists because \( |S_{h+1}| \leq |S^*| \). Let \( a_j \) be the activity in \( S^* \setminus S_h \) with the earliest termination and let \( S^\# = (S^* \setminus \{a_j\}) \cup \{a_k\} \)

We need to prove that \( S^\# \) is feasible (i.e. does not contain overlaps) and it is also optimal. Since \( a_j \in S^* \) and \( a_j \notin S_h \)
$a_j$ could have been selected by our algorithm at the $h+1$ iteration, but the algorithm selected $a_k$ instead. This implies that the termination time of $a_k$ is less than or equal to the termination time of $a_j$, i.e. $f_k \leq f_j$. As a result, $a_k$ does not overlap with the activities in $S^*$ and hence $S^\#$ is feasible.

To build $S^\#$ we removed the activity $a_j$ from $S^*$ and added the activity $a_k$. As a result, we have $|S^\#| = |S^*|$, hence $S^\#$ is optimal because it maximizes the number of selected activities.

3: We need to prove that $S_m$, i.e. the solution produced at the last iteration of the algorithm, is indeed optimal. We know that exists an optimal solution $S^*$ s.t. $S_m \subseteq S^*$. We proceed by contradiction. Let us assume that $|S_m| < |S^*$, i.e. $S_m$ is not optimal. Hence, there exists $a \in S^*$ s.t. $a \notin S_m$.

Since $a$ does not overlap with any activity in $S^*$, it does not neither overlap with the activities in $S_m$, because $S_m \subseteq S^*$. As a result, the condition of the while loop would have been true at the $m+1$ iteration, since there exist an activity in $A$ that does not overlap with the activities already selected in $S_m$. The algorithm would have then selected $a$ and added it to $S_m$, hence we get a contradiction since $S_m$ cannot be the final solution of the algorithm.

This implies that $|S_m| = |S^*|$, hence $S_m$ is optimal.