Lecture 3 – Greedy algorithms and Matroids

Uriel Feige
Department of Computer Science and Applied Mathematics
The Weizman Institute
Rehovot 76100, Israel
uriel.feige@weizmann.ac.il

November 23, 2017

1 Greedy algorithms

When searching for the optimal solution to a problem that has many feasible solutions, search may progress in one of several ways, depending on the structure of the problem. As a rough and incomplete classification, these are three of the major approaches:

1. *Exhaustive search:* Searches through all feasible solutions, or through a large fraction of them (where "large fraction" is a term that may depend on the context). Used for problems that have very little structure, or for which the structure is poorly understood. For example, for finding maximum cliques in graphs.

2. *Dynamic programming:* Searches through a very small fraction of the feasible solution. Typically, dynamic programming is an iterative algorithm. At each irteration, a small (polynomial size) set of partial solutions is maintained, one of which is known to lead to the optimal solution. In the next iteration, members of the set are extended based on information from other members of the set. At the last step, the optimal solution is chosen from the set. To keep the sizes of the sets small, the problem needs to have sufficient structure so that all partial solutions can be grouped into a small number of equivalent classes. For example, subset sum with polynomial weights.

3. *Greedy:* This may be viewed as the ultimate form of dynamic programming, in which only one partial solution is maintained. The problem needs to have much more structure for this approach to work. For example, minimum spanning trees.

Remark: There are other commonly used search strategies, such as *branch and bound*. Also, sometimes the method of finding the optimal solution involves ingredients that do not have intuitive explanations as search procedures. For example, finding triangles in graphs using fast matrix multiplication.

We shall study several problems that can be solved to optimality in polynomial time using the greedy algorithm.

1.1 Independent sets in interval graphs

Let there be a collection of events, where event i is to start at time s_i and end at time t_i. Two events are *conflicting* if their time intervals overlap. We wish to schedule as many mutually nonconflicting events as possible in the same room.

In graph theoretic terms, we can think of each event as a vertex and two conflicting events have an edge between them. This is an *interval graph*, and we wish to find the maximum independent set in such a graph. The interval representation of this graph is given to us.

The following greedy algorithm works:

1. Sort all events in increasing order of their end time t_i.
2. Iteratively, pick the first event in the (remaining list), and remove from the list all events that overlap with it.

Remark: If the list does not contain two events such that the time interval of one completely contains the time interval of the other, then sorting by s_i gives the same order as sorting by t_i.

To see that the algorithm works, let $A = a_1, a_2, \ldots$ be the sequence of events scheduled by the algorithm, and let $O = o_1, o_2, \ldots$ be the sequence of events scheduled by the optimal solution. As there may be more than one optimal solution,we take the optimal solution which has the longest prefix in common with the greedy solution. We claim that necessarily, $A = O$. For the sake of contradiction, suppose otherwise. Let i be the first index in which $a_i \neq o_i$ (or for which o_i exists and a_i does not, if the A sequence is a prefix of the O sequence). Consider now a schedule O' identical to O, except that o_i is replaced by a_i. As a_i starts after $a_{i-1} = o_{i-1}$ ends, and as a_i ends before o_i does, this is still a legal schedule. As it schedules as many jobs as O but has a longer prefix in common with A, we have a contradiction.

1.2 Scheduling jobs so as to minimize sum of completion times

There is a collection of jobs. Job i has length l_i and weight w_i. We wish to schedule the execution of the jobs with no overlaps. Let t_i be the time at which job i finishes according to the schedule. Then the penalty associated with job i is $w_i t_i$, and we wish to minimize $\sum w_i t_i$.

We shall develop a greedy algorithm for this problem by first considering how the proof of optimality may look like, and then deriving the algorithm from the proof.

Consider two schedules, S and S', that are identical except that the order of two jobs i and j that are consecutive in S is flipped in S'. Then simple manipulations show that the total cost of both schedules is the same, except for one term: a term $w_j l_i$ in S is replaced by $w_i l_j$ in S'. If we assume that S is the optimal schedule, then we have $w_j l_i \leq w_i l_j$, which implies $w_i/l_i \geq w_j/l_j$.

This suggests the following algorithm.

1. Sort all jobs in order of decreasing ratios w_i/l_i (also called *density*).

2. Execute jobs in the sorted order.

To see that this algorithm generates an optimal schedule, consider any other schedule. It must have two consecutive jobs i and j with $w_i/l_i < w_j/l_j$. Then by flipping the order of i and j we get a less costly schedule, which is a contradiction.

1.3 Matroids

For certain classes of optimization problems, we have a characterization of the conditions under which the naive greedy algorithm produces the optimal solution. *Matroids* are a well known characterization of this form.

Let $S = \{e_1, \ldots, e_n\}$ be a finite set, and let $F \subset 2^S$ be a collection of subsets of S, called *independent* sets. We say that F is *hereditary* if it is closed under taking subsets. That is, if $X \in F$ and $Y \subset X$, then $Y \in F$. For example, if S is the set of vertices in a graph and F is the collection of all independent sets in the graph, then F is hereditary.

Let $c : S \to R^+$ be a cost function on the elements of S. We wish to find $X \in F$ that maximizes $\sum_{e \in X} c(e)$. A greedy algorithm may construct X as the final set in a sequence $X_0, X_1, \ldots$ as follows:

1. Initially, $X_0 = \phi$.

2. X_{i+1} is a set in F of cardinality $i + 1$ that contains X_i (if such a set exists). Equivalently, let e_j be an element in $S \setminus X_i$ of maximum value among of $c(e_j)$ those for which $X_i \bigcup \{e_j\} \in F$. Then $X_{i+1} = X_i \bigcup \{e_j\}$, if such an e_j exists.

(A more efficient implementation will first sort all elements e in order of decreasing weight $c(e)$.)

For the hereditary family of independent sets in graphs, this greedy algorithm will fail to find a maximum independent set. (For example, consider a star in which the middle vertex has slightly higher cost then any of the other vertices.) But there are hereditary families for which the greedy algorithm produces the optimal solution, regardless of the particular cost function c. These are known as *matroids*.

Definition 1 *A hereditary family F of sets is a* matroid *if for all $X, Y \in F$, if $|X| > |Y|$ then there is some $e \in X \setminus Y$ such that $Y \bigcup \{e\} \in F$.*

The maximal independent sets of a matroid are called *bases*.

Proposition 1 *All bases of a matroid have the same cardinality, called the* rank *of the matroid.*

Proof: By the matroid property, an indendent set in F is maximal if and only if there is no set of larger cardinality in F. □

Remark: the terminology for matroids comes from terminology for matrices. Consider all columns of a matrix. Then we can define the following matroid. Independent sets correspond to sets of linearly independent columns. Bases correspond to those sets of columns that form a basis for the column subspace of the matrix. The rank of the matrix is the rank of the associated matroid.

Theorem 2 *The naive greedy algorithm optimizes over a hereditary family F for every cost function c iff F is a matroid.*

Proof: Assume that F is not a matroid, and let $X, Y \in F$ be such that $|X| > |Y|$ and for every $e \in X \setminus Y$, $Y \bigcup \{e\} \notin F$. Define a cost function c that is negligible on $S \setminus (X \bigcup Y)$, equal to $1+1/|X|$ for each $e \in Y$, and equal to 1 on the remaining members of X. Then the greedy algorithm will choose Y, for a total cost of $c(Y)$. But choosing X is strictly better, by the inequalities below:

$$c(Y) = |Y|(1 + \frac{1}{|X|}) = |Y| + \frac{|Y|}{|X|} < |Y| + 1 \leq |X| \leq c(X).$$

Assume now that F is a matroid. Observe that the maximum X must be a basis (because element costs are positive), and that the greedy algorithm must also produce a basis. Let $e_1, e_2, \ldots e_k$ be the elements chosen by the greedy algorithm. Let $f_1, f_2, \ldots f_l$ be the elements of any arbitrary basis, and assume that they are sorted in descending order. Then necessarily $c(e_i) \geq c(f_i)$ for every i. For the sake of contradiction, suppose otherwise. Let j be the least index with $c(f_j) > c(e_j)$. Then also $c(f_i) > c(e_j)$ for all $i < j$. Let $X = \{f_1, \ldots, f_j\}$ and $Y = \{e_1, \ldots e_{j-1}\}$. Then $X, Y \in F$ and $|X| > |Y|$. Hence Y can be extended by some member of $X \setminus Y$, which is a more greedy choice than e_j. This contradicts the assumption that $e_1, e_2, \ldots e_k$ is the output of the greedy algorithm. □

A well known example of matroids is that of the forests of a graph, known as a *graphic matroid*. For a graph $G(V, E)$, the set S is E, and $X \in F$ if the edges of X do not close a cycle. It is easy to see that F is hereditary. To see that the matroid property holds, observe that the number of connected components in the graph $G(V, X)$ is exactly $|V| - |X|$. Hence if $|X| > |Y|$, there are less connected components in $G(V, X)$ then in (V, Y), and hence at least one edge in X connects different connected components of Y, and hence does not close a cycle in $G(V, Y)$.

If G is connected, then the bases of the forest matroid are the spanning trees of G, and the greedy algorithm above produces a maximum weight spanning tree. To find a minimum weight spanning tree, the similar algorithm can be run in which edges are sorted in order of increasing rather than decreasing cost. Analogous to the proof of theorem 2 and using its notation, we will have that $c(e_i) \leq c(f_i)$ for every i. This algorithm for finding minimum spanning trees is known as Kruskal's algorithm, though it was already described by Borouvka in the 1920's.