CSC 373 H 1 Y — Summer 2007

University of Toronto — St. George Campus

Lecture Summary for Week 9

This summary is not a replacement for the lecture. If you miss a class, please arrange with a friend to take note for you.

4.4 Set Cover As a 0-1 IP

Given n and a collection $G_1, G_2, ..., G_m$ of subsets of $\{1, 2, ..., n\}$. The Set Cover problem is to find a minimum size subset C of $\{1, 2, ..., n\}$ that intersects with all G_i.

This problem is known to be NP-complete. Here we will consider a 0-1 IP formulation of this problem.

Introduce a variable x_i for each $i \in \{1, 2, ..., n\}$. The intended meaning of x_i is that x_i is 1 iff $i \in C$. We want to minimize the following function:

$$x_1 + x_2 + ... + x_n$$

The requirement that C intersects with all G_i is expressed by the following constraints:

$$\sum_{j \in G_i} x_j \geq 1 \qquad \text{for } 1 \leq i \leq m$$

Then, given the values of x_i the minimize the objective function, we can find the set C as follows:

$$i \in C \qquad \text{iff} \qquad x_i = 1 \tag{1}$$

Justification: for each set C that intersects with all G_i, the values of x_i defined by (1) satisfy the above constraints, and vice versa.

5 GREEDY ALGORITHM [CHAPTER 4]

The idea of greedy algorithms is to build up solutions in steps. At each step a (local) decision is made based on some criterion.

5.1 Interval Scheduling [Section 4.1]

Suppose that there is a processor that can process only one request at a time, and there are n requests, the i-th request has starting time $s(i)$ and finishing time $f(i)$ $(0 < s(i) < f(i))$. The Interval Scheduling problem is to find a *compatible* subset S of $\{1, ..., n\}$ of maximal cardinality. Here a subset S is said to be *compatible* if for all $i \neq j \in S$, the i-th and j-th requests do not overlap. (So a compatible subset S is really a schedule to process a subset of the requests. Here we want to process as many requests as possible.)

The brute force approach where all possible subsets of $\{1, ..., n\}$ are tried takes time at least 2^n, because there are 2^n subsets of $\{1, ..., n\}$.

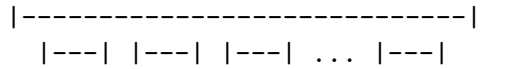
Before giving a algorithm that works correctly, let us first try some approaches.

First, suppose that we always take the earliest possible request:

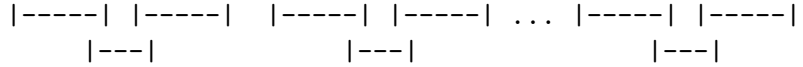
1. sort requests so that $s(1) \leq s(2) \leq \ldots \leq s(n)$

2. $S \leftarrow \varnothing$ % partial schedule
3. $t \leftarrow 0$ % last finish time of activities in S
4. for $i \leftarrow 1 \ldots n$ do
5. if $t \leq s(i)$ then do % request i is compatible with S
6. $S \leftarrow S \cup \{i\}$
7. $t \leftarrow f(i)$
8. return S

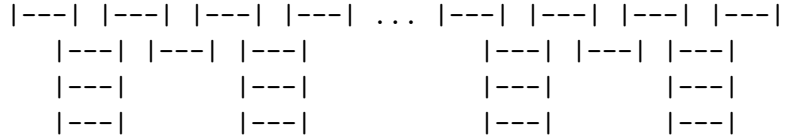
This approahc does not work. Here is an example where it fails to produce a compatible subset of maximum cardinality:



Now, suppose that we always takes the shortest possible request; this fixes the counter example above, but still not correct, because it fails for the following example:



We might also try to count the "overlap" of the request, and always take the request with the least number of overlaps. This works for the second example above, but fails for the following example:



The following algorithm tries to make the resource free as soon as possible. We will show that it actually works correctly.

1. sort requests so that $f(1) \leq f(2) \leq \ldots \leq f(n)$
2. $S \leftarrow \varnothing$ % partial schedule
3. $t \leftarrow 0$ % last finish time of activities in S
4. for $i \leftarrow 1 \ldots n$ do
5. if $t \leq s(i)$ then do % request i is compatible with S
6. $S \leftarrow S \cup \{i\}$
7. $t \leftarrow f(i)$
8. return S

To prove the correctness of this algorithm, we will define the notion of being "promising" for partial solutions and show that all partial solutions are "promising". The fact that the last partial solution (i.e., the output of the algorithm) is promising implies that the output is indeed optimal. This approach for proving the correctness of greedy algorithm turns out to be common for all greedy algorithms that we will consider in this course.

Let $S_0, S_1, \ldots, S_n$ be the partial solutions constructed by algo. at the end of each iteration.

Definition: S_i is called "promissing" if there is an optimal solution which extends using the requests from $\{i+1, \ldots, n\}$. i.e., there is an optimal solution OPT so that

$$S_i \subseteq OPT \subseteq S_i \cup \{i+1, \ldots, n\}$$

Note: OPT may not be unique (there may be more than one way to achieve optimal).

Prove by induction on i (# iterations) that S_i is "promising".

Base case: $S_0 = \varnothing$ is promising because any optimal solution extends S_0 using only requests from $\{1, \ldots, n\}$.

Ind. Hyp.: For some $i \geq 0$, assume that S_i is promising, i.e., there is an optimal OPT_i that extends S_i using only requests from $\{i+1, \ldots, n\}$.

Ind. Step: Prove that S_{i+1} is promising by showing that there exists an optimal solution OPT_{i+1} so that

$$S_{i+1} \subseteq OPT_{i+1} \subseteq S_{i+1} \cup \{i+2, \ldots, n\} \tag{2}$$

Consider the following cases:

Case 1: $S_{i+1} = S_i$ This means the request $i+1$ is not compatible with S_i. Take $OPT_{i+1} = OPT_i$, then the first $\subseteq$ in (2) holds by the assumption, and the second $\subseteq$ in (2) holds because OPT_i does not contain $i+1$ (since $i+1$ is not compatible with S).

Case 2: $S_{i+1} = S_i \cup \{i+1\}$ Here OPT_i may or may not include $i+1$. Consider both possibilities.

Subcase 2a: $i+1 \in OPT_i$ Take $OPT_{i+1} = OPT_i$, then the first $\subseteq$ in (2) holds by the I. H., and the second $\subseteq$ in (2) holds because both S_{i+1} and OPT_{i+1} contains $i+1$.

Subcase 2b: $i+1 \notin OPT_i$ Since OPT_i is optimal, it there must be some request j in OPT_i that overlaps with $i+1$. Let

$$OPT_{i+1} = OPT_i \setminus \{j\} \cup \{i+1\}$$

then (2) holds.

We have to argue that OPT_{i+1} is an optimal solution. First, OPT_{i+1} has the same cardinality as OPT_i. So we just have to argue that (the new request) $i+1$ is compatible with all other requests in OPT_{i+1}. This amounts to showing that j is the only request in OPT_i that overlaps with $i+1$. In fact, request j cannot be in S_i (since $S_{i+1} = S_i \cup \{i+1\}$ is compatible), so $j \geq i+2$. If there is another request $j' \in OPT_i$ that overlaps with $i+1$, then we also have $j' \geq i+2$. Since we sorted the requests in increasing order of finishing time, we have

$$f(i+1) \leq f(j), f(j')$$

So j and j' overlap, contradict the fact that OPT_i is compatible. □