Advanced Algorithms
Lecture 6

Lecture by Shay Mozes
Typeset by Steven Karas

2016-12-11
Last edited 18:19:29 2016-12-28

1 Covering a Network

How many facilities should be build such that each customer is within a given distance from the nearest facility? Each client can have different requirements, and facilities may be located only at vertices or anywhere along an edge.

This problem is NP-Hard with a reduction from the Dominating Set problem.

1.1 Trees

Attach a "string" to each vertex as long as its requirement. Select leaves in the tree, and stretch the string along their edge. If the string reaches the other vertex, reduce the string length there to the smaller of the current value and the excess left over from stretching. If it doesn't reach, create a facility along the edge. Remove the leaf and the edge from the network, and repeat until the tree is empty. Note that searching for facilities prior to "stretching" strings is necessary.

Optimality proof Claim: this algorithm produces an optimal solution.[1] Let k be the number of facilities output by the algorithm. We will show that a set A of k vertices such that for any two vertices $v_1, v_2 \in A$ it holds that $d(v_1, v_2) > S_{v_1} + S_{v_2}$.. In other words, there are no two vertices in A that can be served by the same facility. As such, in order to serve all the vertices in A, we need at least $|A|$ facilities, and therefore $OPT \geq k$.

$\Leftarrow$ d is the distance function, and S is the requirements

For each string, we will define the controlling vertex v. The string is S_v away from v. Per our algorithm, when a facility c is created at the end of the string, the distance from c to v is exactly S_v. The set A is composed of all the vertices that controlled a string when it created a facility. Because when we create a facility the string that created it is removed, it follows that $|A| = k$. Let v_1, v_2 be vertices in A which controlled strings that created facilities c_1, c_2 where c_1 was created prior to c_2. Because the algorithm processes only leaves, the path from v_1 to v_2 must pass by way of c_1 at distance S_{v_1}. Note that $d(v_1, v_2) = \underbrace{d(v_1, c_1)}_{=S_{v_1}} + \underbrace{d(c_1, v_2)}_{>S_{v_2}} \geq S_{v_1} + S_{v_2}$.

[1] We assume correctness. Correctness can be proved by induction, as explained in class.

2 Partition Problems

Given a set of numbers $A = \{a_1, ..., a_n\}$ such that $\sum_{a \in A} = 2B$. Is there a subset A' such that $\sum_{a \in A'} = B$?

For example, given $A = \{5, 5, 7, 3, 1, 9, 10\}$; $A' = \{10, 5, 5\}$, $A \backslash A' = \{7, 3, 1, 9\}$, and $B = 20$.

This problem is NP-Hard.

2.1 Powers of 2

For example, given $A = \{32, 16, 16, 8, 4, 2, 2\}$; $A' = \{32, 8\}$, $A \backslash A' = \{16, 16, 4, 2, 2\}$, and $B = 40$.

Algorithm Sort the items such that $a_1 \leq ... \leq a_n$. Let $S_1 = S_2 = \emptyset$ and $s_1 = s_2 = 0$. For each $s \in S$: add s to the smaller of S_1 or S_2, tracking the sums in s_1, s_2. If $s_1 = s_2$, then there is a partition. Otherwise, there is not.

Correctness Proof If the algorithm produces a partition, it exists. We need to prove that if the algorithm does not exist, then a partition does not exist.

Lemma Let A_1, A_2 be two sets of power-2 integers, such that each integer is $\geq 2^v$. Then $\sum_{a \in A_1} a - \sum_{a \in A_2} a$ is a multiple of 2^v.

Assume that the algorithm does not find a partition. Therefore at some point, one set was larger than B. Consider the time when a set is about to become larger than B. When this happened, some item of size 2^v is considered, and the remaining size in both A_1, A_2 is less than 2^v.

Assume the negative that a partition does exist. Then we can swap the subsets $A_1 \subseteq S_1$, $A_2 \subseteq S_2$ to fix the partition produced by the algorithm. Because all integers thus far are $\geq 2^v$, the difference $|A_1 - A_2|$ is at least 2^v (because it is a nonzero multiple of 2^v). Therefore at least one of the sets overflows, contradicting our assumption that a partition exists.

3 Interval Graphs

An interval graph is the intersection graph of intervals on the reals, where the vertices are the intervals, and the edges connect intersecting intervals.

3.1 Maximum Independent Set

Otherwise known as the activity selection problem. We want to maximize the number of activities we participate in. Lots of rides, each starting and ending at a different time.

Formally Given a set $S = \{a_1, ..., a_n\}$ of n activities where $a_i = (s_i, f_i)$ where s_i is the start time, and f_i is the finish time of a_i. We want to find a maximum subset of S of non-conflicting activities.

Dynamic Programming Solution Let $S_{ij} = \{k : f_i \leq s_k < f_k \leq s_j\}$ be the subset of activities that start after a_i finishes and finish before a_j starts. NOTE: Add $a_0 = (0, 0)$ and $a_{n+1} = (\infty, \infty)$.

Define $C[i, j]$ be the maximum number of activities from S_{ij} that can be selected.

$$C[i,j] = \begin{cases} \max\limits_{k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{if } S_{ij} \neq \emptyset \\ 0 & \text{if } S_{ij} = \emptyset \end{cases}$$

Greedy Solution Note that the activity selection problem exhibits the *greedy choice* property; A locally optimal choice gives us a globally optimal solution.

Theorem If S is an activity selection instance sorted by finish time, then there exists an optimal solution $A \subseteq S$ such that $\{a_1\} \in A$

Proof

Complexity $O(n \log n)$, but the intuition is simple, and easier to implement.

MIS is Activity Selection Any interval graph has an interval representation in which all interval endpoints are distinct integers and this representation can be done in poly-time[2]. Therefore, Activity Selection is MIS.

4 Graph families

1. Trees
2. Intersection graphs
3. Chordal graphs
4. Planar graphs, surface embedded graphs - navigation
5. Random graphs
6. Serial-Parallel
7. many many more

Explanation of various properties, interesting problems and solutions around 817pm.

5 Tree-like graphs

5.1 Path Decomposition

Paths are easier to deal with than trees (conceptually).

We can build a path with some simple operations, starting from an empty graph: Introduce a vertex, introduce an edge, forget a vertex (forever)[3].

Pathwidth One less than the size of the largest bag.

⇐ Bags, components, cells, etc. are all names for this

[2]Proof left as exercise

[3]Example of such a construction given in the lecture

k-Clique The pathwidth of K_n is $n-1$, and the decomposition always uses exactly one component/bag

Formally A path decomposition P of G is a path of bags such that ever vertex in G is in some bag, every edge in G is in some bag, and for every $v \in G$, the bags containing v are connected in P.

5.2 Tree decomposition

Similar to path decomposition, but we allow joining two bags together[4].

Treewidth The treewidth of G is the smallest width of a tree decomposition of G.

Maximum Weighted Independent Set on Trees $M_{\text{out}}[u]$ is the maximum weight of an IS that does not include u in the subtree T_u $M_{\text{in}}[u]$ is the maximum weight of an IS that does include u in the subtree T_u

MWIS on Tree Decompositions For every bag B and every subset $U \subseteq B$:

$M[B, U]$ is the size of the maximum IS in the subgraph induced by all vertices in all bags in T_B such that all vertices of U are in the IS, and all the vertices $B \setminus U$ are not in the IS.

If U is not an IS in the subgraph of G induced by the vertices of B, then $M[B, U] = -\infty$.

$$M[B, U] = -\infty \text{ if } U \text{ is not an IS in } B$$

$$M[B, U] = w(U) \text{ if } B \text{ is a leaf}$$

$$M[B, U] = w(U) + \sum_{B_i \text{ child of B}} \max_{Y} \{M[B_i, Y]\} - w(U \cap Y)$$

Complexity $O(n \cdot 4^k)$, where k is the treewidth.

[4]Example given in lecture and on slide 71