Lecture 2: Introduction (Continued)

This section covers the following topics

- Christofide's $\frac{3}{2}$-approximation algorithm for Metric TSP. (Vazirani, Alg 3.10)
- The Set-Covering problem. A $(\ln n)$-approximation algorithm. (CLR, Chapter 37)
- The subset-sum problem. A polynomial time approximation scheme. (CLR, Chapter 37)
- Minimum Makespan Scheduling. A 2-approximation algorithm. (Vazirani, Ch. 10)

Matchings

Let $G = (V, E)$ be a graph with cost function $c()$ on its edges.

- A *matching* of G is a set of edges $E' \subseteq E$ such that no two edges in E' share a vertex in common.
- A *perfect* matching is a matching in which $|E'| = \lfloor V/2 \rfloor$. Perfect matchings of complete graphs always exist.
- A *minimal-cost* perfect matching of a complete graph can be found in $O(|V|^3)$ time.

Metric TSP: Christofide's algorithm

A $\frac{3}{2}$-approximation algorithm for metric TSP.
(Vazirani, Alg 3.10)

<u>Christofide's algorithm</u>
1. Find a Minimum Spanning Tree T' of G.
2. Find a minimal cost perfect matching, M, on the vertices of odd-degree in T'.
3. Find an Eulerian Tour $\mathcal{T}'$ of G'
4. Output the vertices of G in the order in which they first appear in $\mathcal{T}'$. Let T be the Hamiltonian cycle thus created.

<u>Lemma:</u> The number of odd-degree vertices in T' is even.

<u>Lemma:</u> Let $V' \subseteq V$ such that $|V'|$ is even and let M be a minimum-cost perfect matching on V'. Then

$$c(M) \leq OPT(G)/2.$$

<u>Proof:</u> Let $\mathcal{T}$ be a TSP tour of G. Now let $\mathcal{T}'$ be the tour on V' that results by visiting the vertices in V' in the order defined by $\mathcal{T}$. By the triangle inequality

$$c(\mathcal{T}') \leq c(\mathcal{T}) = OPT(G).$$

Now note that taking every other edge in $\mathcal{T}'$ yields a perfect matching of V' so $\mathcal{T}'$ is the union of two perfect matchings of V', M' and M''. Since these matchings can't have cost less than the minimal one:

$$2c(M) \leq c(M') + c(M'') = c(\mathcal{T}') \leq OPT(G)$$

so

$$c(M) \leq OPT(G)/2.$$

Lemma: Christofide's algorithm is a $\frac{3}{2}$-approximation algorithm for the TSP problem on Metric Graphs. Furthermore, it runs in polynomial time in the graph size.

Proof: Recall that the algorithm builds a MST, T', and a minimal cost matching, M, of the odd-degree vertices in T'. It then builds an Eulerian path $\mathcal{T}'$ on $T' \cup M$ and cuts that down to a Hamiltonian tour T.

- $c(T') \le OPT(G)$ (same as previously).
- $c(M) \le OPT(G)/2$.
- $c(\mathcal{T}') = c(T') + C(M) \le \frac{3}{2} OPT(G)$.
- $c(T) \le c(\mathcal{T}')$ from the triangle inequality.

Combining these three facts gives

$$c(T) \le c(\mathcal{T}') \le \frac{3}{2} OPT(G).$$

The Set Covering Problem

Let X be a set and $\mathcal{F}$ a family of subsets of X such that $X = \cup_{F \in \mathcal{F}} F$.

For example $X = \{1,2,3,4,5,6\}$ and $\mathcal{F}$ contains the subsets

$$\begin{aligned} F_1 &= \{1,3,5\} \\ F_2 &= \{2,3,6\} \\ F_3 &= \{2,5,6\} \\ F_4 &= \{2,3,4,6\} \\ F_5 &= \{1,4\} \end{aligned}$$

A subset $F \in \mathcal{F}$ *covers* its elements.

The problem is to find a minimum-size subset $\mathcal{C} \subseteq \mathcal{F}$ that covers X, i.e., $X = \cup_{F \in \mathcal{C}} F$.

For example $\{F_1, F_2, F_4\}$ covers X but is not a minimal size solution.

$\mathcal{C} = \{F_1, F_4\}$ is a minimal size solution.

Finding a minimal size set cover is a NP-Hard problem.

A Greedy Set-Cover algorithm

$U \leftarrow X$
$\mathcal{C} \leftarrow \emptyset$
  select a $F \in \mathcal{F}$ that maximizes $|F \cap U|$
  $U \leftarrow U - F$
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{F\}$
return($\mathcal{C}$)

Lemma: Greedy set cover runs in time polynomial in $|X|$ and $\mathcal{F}$. Furthermore, its output $\mathcal{C}$ is a set-cover of X.

Lemma: Let OPT be the size of a minimum size set cover. Then

$$1 \le \frac{|\mathcal{C}|}{OPT(G)} \le H(\max\{|F| : F \in \mathcal{F}\})$$

where $H(d) = H_d = \Sigma_{i=1}^{d} \frac{1}{i}$.

Since $\forall F \in \mathcal{F}$, $|F| \le |X|$, and H_d the *Harmonic number*, grows like $H_d \sim \ln d$, this implies that Greedy set cover has an approximation ratio of about $\ln |X|$.

Let $\mathcal{C}^*$ be a minimal-size set cover and let F_i denote the ith subset added to $\mathcal{C}$. We now define a cost c_x for every $x \in X$. Let F_i be the *first* set in $\mathcal{C}$ that contains x. Then

$$c_x = \frac{1}{|F_i - (F_1 \cup F_2 \cup \cdots \cup F_{i-1})|}.$$

Note that this means that

$$\forall i, \quad \sum_{x \in F_i - (F_1 \cup F_2 \cup \cdots \cup F_{i-1})} c_x = 1.$$

Since both $\mathcal{C}$ and $\mathcal{C}^*$ cover X this means that

$$\begin{aligned} |\mathcal{C}| = \sum_{F_i \in \mathcal{C}} 1 &= \sum_{F_i \in \mathcal{C}} \left(\sum_{x \in F_i - (F_1 \cup F_2 \cup \cdots \cup F_{i-1})} c_x \right) \\ &= \sum_{x \in X} c_x \\ &\le \sum_{F \in \mathcal{C}^*} \sum_{x \in F} c_x. \end{aligned}$$

On the next slide we will prove that

$$\forall F \in \mathcal{F}, \ \sum_{x \in F} c_x \le H_{|F|}.$$

Combining this with the above yields

$$\begin{aligned} |\mathcal{C}| &\le \sum_{F \in \mathcal{C}^*} H_{|F|} \\ &\le |\mathcal{C}^*| \cdot H(\max\{|F| : F \in \mathcal{F}\}) \end{aligned}$$

which is what we wanted to prove.

We need to prove $\boxed{\forall F \in \mathcal{F},\ \Sigma_{x \in F}\, c_x \leq H_{|F|}}$.

For any set $F \in \mathcal{F}$ and i, define

$$u_i = |F - (F_1 \cup F_2 \cup \cdots \cup F_i)|$$

to be number of elements in F remaining uncovered after the first i sets in $\mathcal{C}$ were chosen. $u_0 = |F|$.

Let k be the smallest index such that $u_k = 0$, the first time at which all of F is covered by sets in $\mathcal{C}$.

Note (i) $u_{i-1} \geq u_i$ and (ii) $u_{i-1} - u_i$ elements of F are covered for the first time in $\mathcal{C}$ by F_i. Thus

$$\sum_{x \in F} c_x = \sum_{1 \leq i \leq k} (u_{i-1} - u_i) \cdot \frac{1}{|F_i - (F_1 \cup F_2 \cup \cdots \cup F_{i-1})|}$$

Now we use the algorithm definition to see that

$$|F_i - (F_1 \cup F_2 \cup \cdots \cup F_{i-1})| \geq |F - (F_1 \cup F_2 \cup \cdots \cup F_{i-1})| = u_{i-1}.$$

If this *was not true* then F would have been chosen by the greedy algorithm as the next set in $\mathcal{C}$ instead of F_i.
Thus

$$\sum_{x \in F} c_x \leq \sum_{1 \leq i \leq k} (u_{i-1} - u_i) \frac{1}{u_{i-1}}.$$

We need to prove $\boxed{\forall F \in \mathcal{F},\ \Sigma_{x \in F}\, c_x \leq H_{|F|}}$.

We know $\boxed{\Sigma_{x \in F}\, c_x \leq \Sigma_{1 \leq i \leq k} (u_{i-1} - u_i) \frac{1}{u_{i-1}}}$.

First note that $\forall a < b$, integers

$$H(b) - H(a) = \sum_{a < i \leq b} \frac{1}{i} \geq (b - a) \frac{1}{b}$$

Thus

$$\begin{aligned} \sum_{x \in F} c_x &\leq \sum_{1 \leq i \leq k} (H(u_{i-1}) - H(u_i)) \\ &= H(u_0) - H(u_k) \\ &= H(u_0) - H(0) \\ &= H(u_0) \\ &= H(|F|) \end{aligned}$$

and we are done.

A Better Definition

For any given optimization (minimization) problem and approximation algorithm $\mathcal{A}$ to solve it:

- Let Π = Set of all *instances* of the problem.
- For all instances $I \in \Pi$ define $size(I)$.
- For all instances $I \in \Pi$ define

$$\begin{aligned} OPT(I) &= \text{cost of optimal solution for } I \\ A(I) &= \text{cost of solution produced by } A \text{ on } I. \end{aligned}$$

Let $\rho(n)$ be a function such that

$$\forall I \in \Pi, \text{with } size(I) = n, \quad \frac{A(I)}{OPT(I)} \leq \rho(n).$$

Then A is a *factor $\rho(n)$ approximation algorithm ($\rho(n)$-approximation algorithm).*

(there is a similar definition for *maximization* problems)

The Subset-Sum Problem

Definition: An instance of the *subset-sum decision problem* is (S, t) where:
$S = \{x_1, x_2, \ldots, x_n\}$ a set of positive integers
t a positive integer.
The problem is whether some subset of S adds up exactly to t.

The subset-sum decision problem is NP-complete

The *subset-sum optimization problem* is to find a subset of S whose sum is as large as possible but no greater than t.

We will define a class of algorithms A_ϵ, such that, $\forall \epsilon > 0$,

- A_ϵ is an ϵ-approximation algorithm for subset-sum.
- A_ϵ runs in time polynomial in n, $\log t$ *and* $\frac{1}{\epsilon}$.

Such a class of algorithms is known as a

A fully polynomial-time approximation scheme.

An exponential time algorithm

If $S = \{x_1, x_2, \ldots, x_n\}$ is a set or list and x a real number then define

$$S + x = \{x_1, x_2, \ldots, x_n\} = \{x_1 + x, x_2 + x, \ldots, x_n + x\}.$$

If $L = \{x_1, x_2, \ldots, x_n\}$ is a sorted list and $L' = \{u_1, u_2, \ldots, u_m\}$ is a sorted list then we define **Merge-Lists**(L, L') to be the procedure that returns the sorted union of the two lists. This procedure runs in time $O(|L'| + |L|)$.

Exact-Subset Sums
$n \leftarrow |S|$
$L_0 \leftarrow < 0 >$
for $i = 1$ to n
    $L_i =$ Merge-Lists$(L_{i-1}, L_{i-1} + x_i)$
    remove from L_i all elements bigger than t.
return largest element in L_n.

Let P_i be the set of all values that can be obtained by selecting some subset of $\{x_1, x_2, \ldots, x_i\}$ and summing its members. Then L_i is a sorted list containing all elements in P_i of size no greater than t.
The algorithm therefore returns the correct answer.

Since L_i can have as many as 2^i items this algorithm can take $\Theta(2^n)$ time!

Trimming

Let $L\{x_1, x_2, \ldots, x_m\}$ be a list. To *trim* the list by parameter δ means to remove as many elements from L as possible in such a way that the list L' of remaining elements has the following property:

For every $y \in L$ there exists a $z \in L'$ such that
$$(1 - \delta)y \le z \le y.$$

Example: $L = < 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 >$ and $\delta = 0.1$.
A trimmed list would be $L' = < 10, 12, 15, 20, 23, 29 >$.

$Trim(L, \delta)$
$L' = < x_1 >$
$last = x_1$
for $i = 2$ to m
    if $last < (1 - \delta)x_i$
        then append x_i onto end of L'.
            $last = x_i$
return L'

This algorithm returns a trimmed list in $O(m)$ time.

The Actual Approximation Algorithm

Approx-Subset-Sum(S, t, ϵ)
$n \leftarrow |S|$.
$L_0 = < 0 >$.
for $i = 1$ to n
    $L_i \leftarrow$ Merge-Lists$(L_i, L_{i-1} + x_i)$
    $L_i \leftarrow Trim(L_i, \epsilon/n)$
    remove from L_i all elements bigger than t
return largest element in L_n

Note that when list L_i is trimmed we introduce a relative error of at most ϵ/n between the representative values remaining and the elements of the list. By induction we can show that, for every element $y \in P_i$ there exists some $z \in L_i$ such that
$$\left(1 - \frac{\epsilon}{n}\right)^i y \le z \le y.$$
Let $\overline{z}$ be the largest element in L_n. If y^* is a solution to the exact subset-sum problem then there exists a $z^* \in L_n$ such that
$$\left(1 - \frac{\epsilon}{n}\right)^n y^* \le z^* \le \overline{z} \le y^*.$$
But $\forall n > 1, \quad 1 - \epsilon \le \left(1 - \frac{\epsilon}{n}\right)^n$ so
$$(1 - \epsilon)y^* \le \overline{z}$$

and A_ϵ is an ϵ-approximation algorithm.

Running Time

The running time of the ith stage of the algorithm is $O(|L_i|)$.

After trimming, successive elements $z', z \in L_i$ have the property
$$z' < z\left(1 - \frac{\epsilon}{n}\right).$$
Therefore the total number of elements in L_i is at most

$$\begin{aligned} \log_{\frac{1}{1-\frac{\epsilon}{n}}} t &= \frac{\ln t}{-\ln\left(1 - \frac{\epsilon}{n}\right)} \\ &\le \Theta\left(\frac{n \ln t}{\epsilon}\right) \end{aligned}$$

The running time of A_ϵ is proportional to
$$\frac{n^2 \ln t}{\epsilon}$$

and the A_ϵ form a

Fully Polynomial Time Approximation Scheme

for subset-sum.

Minimum Makespan Scheduling (V. 79)

Given processing times for n jobs, $p_1, p_2, \ldots, p_n$, and an integer m, the *Minimum Makespan Scheduling* problem is to find an assignment of the jobs to m machines, $M_1, M_2, \ldots, M_m$ so that the final completion time (the makespan) is minimized. This is a NP-Hard problem.

In 1966 Graham analyzed the algorithm below to show that it is a 2-approximation algorithm. Possibly, first worst-case analysis of an approximation algorithm in the literature.

Minimum Makespan Scheduling
1. Order the n jobs arbitrarily
2. Schedule jobs on machines in this order, scheduling the next job on machine that has been assigned least work so far

Let M_i be the machine that completes last in the schedule produced by MMS and let j be the last job scheduled on this machine.
Let $start_j$ be the time at which job j started working. From the choice of M_i by the algorithm we know that *all of the other machines are totally busy until* $start_j$. Therefore

$$start_j \leq \frac{1}{m}\sum_i p_i \leq OPT.$$

Furthermore, $p_j \leq OPT$.
The makespan produced by the MMS is then

$$MMS = start_j + p_j \leq 2 \cdot OPT.$$