COMP4600

Approximation Algorithms

Weifa Liang

Research School of Computer Science

The Australian National University
Canberra, ACT 0200

COMP4600

Textbook: Introduction to Algorithms.
T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein. The MIT Press, 3rd Edition, 2009 or later.

Reference Books:

- Computers and Intractability: A Guide to the Theory of NP-Completeness.
  M. R. Garey and D. S. Johnson, 1979. New York: W.H. Freeman.
  ISBN 0-7167-1045-5.

- Approximation Algorithms for NP-Hard Problems.
  D. S. Hochbaum, 1995. Boston: PWS publishing Company.
  ISBN 0-534-94968-1.

- Approximation Algorithms.
  Vijay V. Vazirani, Springer, 2003, ISBN 3-540-65367-8.

- The Design of Approximation Algorithms.
  David P. Williamson and David B. Shmoys, Cambridge University Press, 2010

- Design and Analysis of Approximation Algorithms.
  Ding-Zhu Du, Ker-I Ko, and Xiaodong Hu, 2012, ISBN 978-4614-1701-9,
  Springer.

Contact: A/Prof Weifa Liang, Room N334, CSIT building.
Email: wliang@cs.anu.edu.au, Tel: (02)-6125-3019 (O).

Chapter 35. Approximation Algorithms

Many problems of practical significance are NP-complete, the cost of finding an exact solution to these problems is very expensive and sometimes impossible, instead, a fast and performance-guaranteed approximate solution is desperately needed.

We will introduce polynomial-time approximation algorithms for several well-known NP-complete problems. Intuitively, a problem is NP-complete if it is unlikely to be solved in polynomial time.

There are three methods to approach NP-complete problems.

- Develop exponential running time algorithms with input size being small
- Consider special cases of the problem that can be solved in polynomial time
- Find "**near-optimal**" solutions to the problems in polynomial time

35.1 Performance Ratio or Approximation Ratio

An algorithm for a problem has an approximation ratio of $\rho(n)$ if, for any input of problem size n, the cost C of the solution produced by the algorithm is within a factor of $\rho(n)$ of the cost C^* of the optimal solution:

$$\max\{\frac{C}{C^*}, \frac{C^*}{C}\} \leq \rho(n).$$

An algorithm with approximation ratio of $\rho(n)$ is referred to as a $\rho(n)$-approximation algorithm.

- For a maximization problem, $0 < C \leq C^*$.
- For a minimization problem, $0 < C^* \leq C$.

For example, an $O(\log n)$-approximation algorithm, or a 2-approximation algorithm, etc.

35.1 Approximation Scheme

An approximation scheme for an optimization problem is an approximation algorithm that takes an input not only **an instance of the problem**, but also **an accuracy parameter** $\varepsilon > 0$ such that for any fixed ε, the scheme is a $(1+\varepsilon)$-approximation algorithm. Note that ε may be in $0 < \varepsilon \leq 1$ or $\varepsilon > 1$, depending on with which types of problems dealt.

- For any fixed $\varepsilon > 0$, we say an approximation scheme is a **Polynomial-Time Approximation Scheme** (PTAS for short), if it runs in time polynomial in the size n of its input instance, e.g., $O(n^{2/\varepsilon})$.

- We say an approximation scheme is a **Fully Polynomial-Time Approximation Scheme** (FPTAS), if it runs in time polynomial both in $\frac{1}{\varepsilon}$ and in the size n of its input instance, e.g., $O((\frac{1}{\varepsilon})^{2.5} n^3)$.

35.1 The vertex cover problem

A vertex cover in an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ such that

- If (u, v) is an edge in E, then either $u \in V'$ or $v \in V'$ (or both).
- The size of a vertex cover is the number of vertices in it, $|V'|$.

The vertex cover problem is to find a vertex cover V' of minimum size, i.e.,

$|V'| = \min_{U' \subseteq V}\{|U'| \mid U' \text{ is a vertex cover in } G\}$.

35.1 The vertex cover problem (continued)

Approx_Vertex_Cover(G)

1 $C \leftarrow \emptyset$; /* the vertex cover set */

2 $E' \leftarrow E$;

3 **while** $(E' \neq \emptyset)$ **do**

/* let (u, v) be an arbitrary edge in E' */

4 $C \leftarrow C \cup \{u, v\}$;

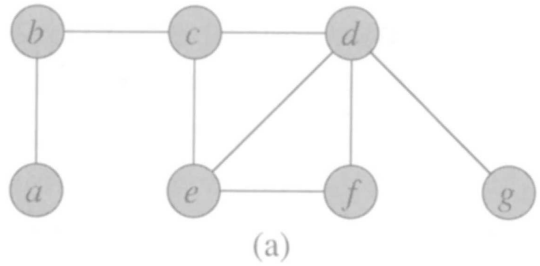
5 remove from E' every edge incident on either u or v;

i.e., $E' \leftarrow E' \setminus \{(x, u), (y, v) \mid (x, u) \in E', (y, v) \in E'\}$

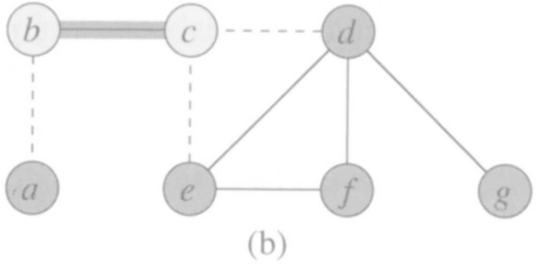
endwhile

6 **Return** C.

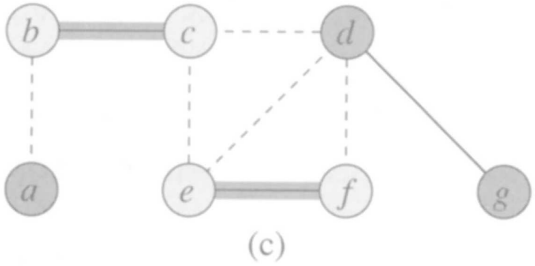
35.1 The vertex cover problem (cont)



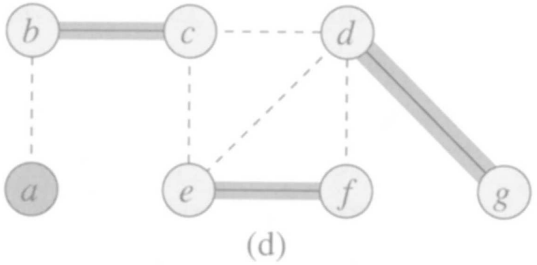
(a)



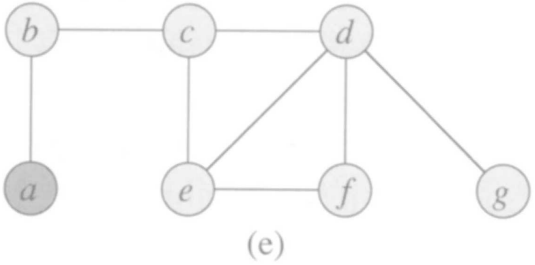
(b)



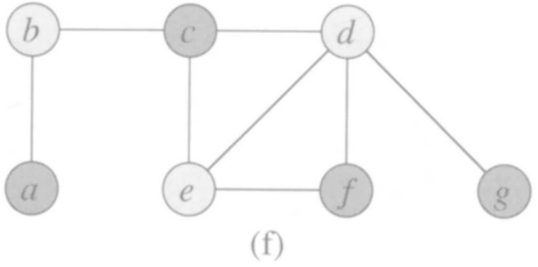
(c)



(d)



(e)



(f)

35.1 The vertex cover problem (continued)

Theorem: Approx_Vertex_Cover is a polynomial-time 2-approximation algorithm.

Proof: Let C be the set of vertices returned by the approximation algorithm. Let A be the set of edges picked at Step 4. In order to cover the edges in A, any vertex cover including **the optimal cover C^*** must include at least one endpoint of each edge in A. Notice that **no two edges in A share a common endpoint**. Thus, $|C^*| \geq |A|$. Therefore the size of C can be estimated as follows.

$$|C| = 2|A| \leq 2|C^*|.$$

35.2 The TSP problem

Traveling Salesman Problem (TSP): given a complete graph $G = (V, E)$, there is a nonnegative cost $c(u, v)$ associated with each edge $(u, v) \in E$, the problem is to find a Hamiltonian cycle in G with minimum cost, where a graph is complete if there is an edge between any pair of nodes in the graph.

A special case of TSP: TSP with the triangle inequality.

Triangle inequality: if for any three vertices u, v and w in G, we have

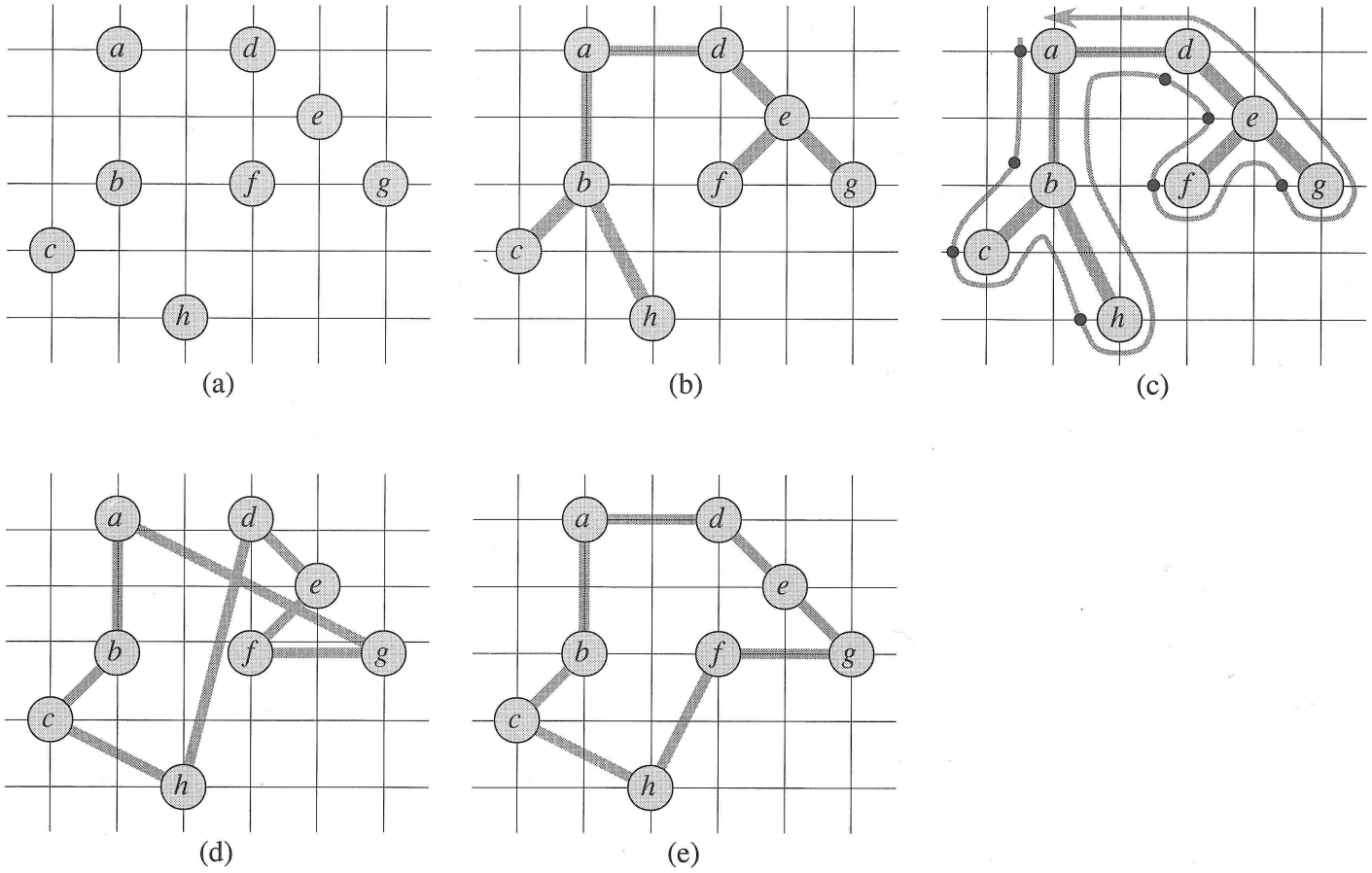
$$c(u, w) \leq c(u, v) + c(v, w).$$

Approx_TSP_Tour(G, c)

1 Select an arbitrary vertex $r \in V$ as the root vertex;
2 Find a Minimum Spanning Tree (MST) T of G rooted at r, using the Prim algorithm;
3 Let L be the list of vertices in T visited in a **preorder tree walk**;
4 **Return** the Hamiltonian cycle H that is derived by visiting the vertices in the order L

35.2 TSP with the triangle inequality (continued)

An example:



35.2 TSP with the triangle inequality (continued)

Theorem: Approx_TSP_Tour is a polynomial-time 2-approximation algorithm for TSP with the triangle inequality.

Proof: Let H^* be the optimal tour. Clearly, $c(T) \leq c(H^*)$ because T is an MST in G, where $c(T)$ is the sum of the weighted edges in T.

Meanwhile, $c(L) \leq 2c(T)$ due to the fact that each edge in T is visited at most twice in the preorder traversal of T. H is derived from L and $c(H) \leq c(L)$ following the triangle inequalities. Thus,

$$c(H) \leq c(L) \leq 2c(T) \leq 2c(H^*).$$

35.2 Improved approximation algorithm for TSP with the triangle inequality

Christophiste's algorithm (1976)

Chris_Approx_TSP_Tour(G, c)

1 Find a Minimum Spanning Tree (MST) T of G rooted at r;
2 Construct a weighted bipartite graph $G_B = (X, Y, E_{XY})$
where X and Y are the sets of **odd-degree tree nodes** in T,
there is an edge between $x \in X$ and $y \in Y$ if $x \neq y$
and its weight is the Euclidean distance between x and y;
3 Find a minimimum-weighted perfect matching M_B in G_B;
4 Find a Euler tour L in the Eulerian graph $M_B \cup T$;
5 **return** the Hamiltonian cycle H that is derived from L.

35.2 Improved approximation algorithm for TSP with the triangle inequality

Theorem: Chris_Approx_TSP_Tour is a polynomial-time 1.5-approximation algorithm for TSP with the triangle inequality.

Proof: Let O be the set of odd-degree nodes of the MST T in G (a node is odd if the number of edges incident to it is odd). Notice that the number of odd nodes in a graph must be even (can you prove this claim?). Let $|O| = 2k$ and $k \geq 1$ is an integer. Let H^* be the optimal Hamiltonian cycle, mark all nodes of O in H^*, a tour C induced by the marked nodes in H^* then is formed, which consists of $2k$ nodes and edges. Clearly,

$$c(C) \leq c(H^*),$$

by the triangle inequality.

As the number of odd-degree nodes are even, let $e_1, e_2, \ldots, e_{2k-1}, e_{2k}$ be the edge sequence in C, then $M_1 = \{e_1, e_3, \ldots, e_{2i-1}, \ldots, e_{2k-1}\}$ and $M_2 = \{e_2, e_4, \ldots e_{2i}, \ldots e_{2k}\}$

form two different maximum matchings in C consisting of the odd-degree nodes. Let $c(M_i) = \sum_{e \in M_i} w(e)$ with $i = 1, 2$.
Since $c(M_1) + c(M_2) = c(C)$, the smaller one, w.l.o.g, M_1, $c(M_1) \leq \frac{c(C)}{2} \leq \frac{c(H^*)}{2}$.

Since matching M_B is the minimum weight maximum matching in the subgraph $G[O]$ induced by the nodes in O,

$$c(M_B) \leq c(M_1) \leq \frac{c(M_1) + c(M_2)}{2} = \frac{c(C)}{2} \leq \frac{c(H^*)}{2}.$$

We thus have

$$c(H) \leq c(L) = c(T \cup M_B) = c(T) + c(M_B) \leq c(H^*) + \frac{c(H^*)}{2} \leq \frac{3c(H^*)}{2}.$$

35.3 Steiner tree problem

Steiner Tree problem: Given an undirected, weighted connected graph $G = (V, E, c)$ and a terminal set $D \subset V$ where $c: E \mapsto \mathbb{R}^+$, the problem is to find a tree spanning the nodes in $D = \{v_1, v_2, \ldots, v_{|D|}\}$ such that the weighted sum of the edges in the tree is minimized.

The decision version of the Steiner tree problem is NP-complete when $|D| \neq 2$ and $|D| \neq |V|$.

- When $|D| = 2$, the single source shortest path problem
- When $|D| = |V|$, the minimum spanning tree problem

35.3 Kou et al Algorithm

The Steiner tree problem is NP-hard, which means it is unlikely the problem can be solved in polynomial time unless P=NP. Instead, an approximation algorithm was devised by Kou et al [1981]. The basic steps of their algorithm are as follows:

Step 1. Compute all pairs shortest paths in G. An auxiliary complete graph K_D consisting of only the $|D|$ terminal nodes is constructed. The weight assigned to each edge in K_D is the length of the shortest path in G between the two endpoints of the edge.

Step 2. Find an MST in K_D. Let E_{MST} be the set of edges in the MST (which corresponds to a shortest path of G consisting of edges in G).

Step 3. A subgraph G_D of G is induced by the edges in E_{MST}. Note that each edge in E_{MST} corresponds to a shortest path in G.

Step 4. Find an MST in the subgraph G_D, and prune those branches of the tree that do not contain the nodes in D. The resulting tree is an approximate Steiner tree.

35.3 Analysis of the Steiner Tree Approximation Algorithm

Analysis of the Time Complexity:

- Step 1 takes $O(n^3)$ time
- Step 2 takes $O(|D|^2) = O(n^2)$ time
- Step 3 takes $O(|D|n) = O(n^2)$ time
- Step 4 takes $O(m \log n)$ time.

So, the algorithm takes $O(n^3)$ time.

3.3. Approximation ratio of Kou et al's algorithm

Theorem: Let OPT be the optimal cost of the Steiner tree and $Cost$ the cost of the approximate Steiner tree delivered by the approximation algorithm, then

$$\frac{Cost}{OPT} \leq 2. \tag{1}$$

Proof: Let T_{opt} be the Steiner tree in G spanning the nodes in D, we traverse T_{opt} by pre-order traversal and start from its root. Let $(v_1, \ldots, v_2)$,$(v_2, \ldots, v_3)$,..., $(v_{|D|-1}, \ldots, v_{|D|})$, $(v_{|D|}, \ldots, v_1)$ be the edge subsequences with the endpoints of each subsequence being terminals nodes, then, the total cost C of the edges in all subsequences is no more than twice of the cost of T_{opt} due to the fact that each of the tree edges is visited at most twice.

From the above subsequences, a subtree in G spanning the nodes $v_1, v_2, \ldots, v_{|D|}$, T' can be constructed, and the cost of the tree is no more than C. We also know that the cost of the MST in K_D, E_{MST} is a minimum spanning tree spanning the nodes in

D, then its cost $c(E_{MST})$ is no more than the cost of T', i.e., $c(E_{MST}) \leq c(T')$ while $c(T') \leq C$. Thus, $c(E_{MST}) \leq C$.

Let T be the MST in G_D delivered by the algorithm at Step 4, following the construction of T, the cost of $c(T)$ is no more than the cost of G_D while the cost of G_D is no more than the cost of the MST E_{MST} in K_D, $c(E_{MST})$, while the cost of the MST of K_D is no more than C. Thus, the cost of T is no more than than twice of the cost of T_{opt}.

35.3. Challenging questions related to Steiner Tree Problems

Node-weighted Steiner tree problem

- If each node instead of each edge is assigned a weight, find a Steiner tree such that the weighted sum of nodes in the tree is minimized. (There is an approximation algorithm with $2 \log D$ approximation ratio)
- Consider the node version, if the objective is to find a Steiner tree such that the weighted sum of non-leaf nodes is minimized
- If all terminal vertices in D must be tree leaves, it cannot be approximated better than $O(\log D)$ of the optimal.

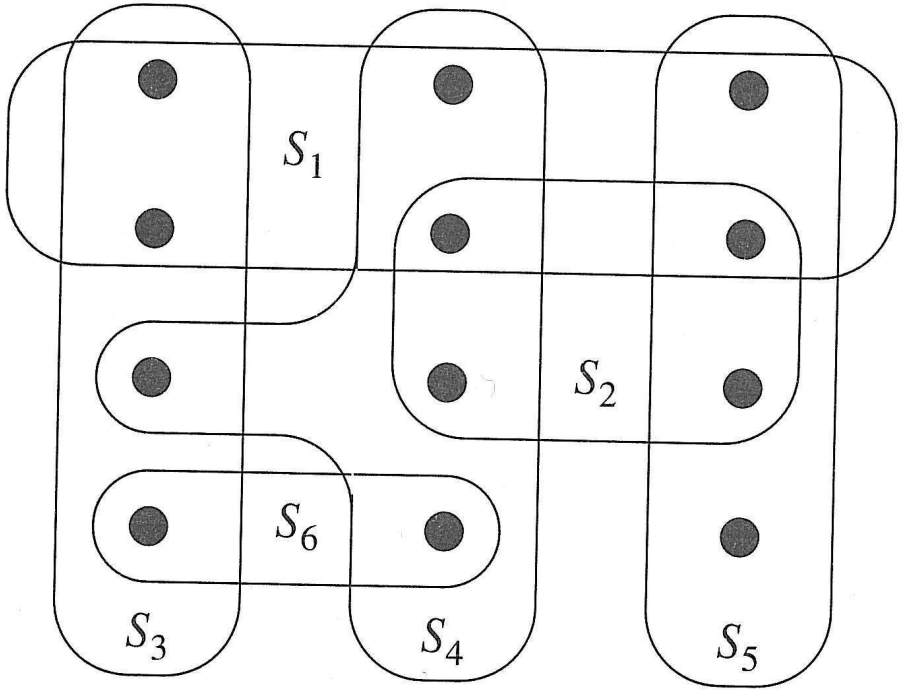
35.4 The set-covering problem

The set-covering problem is an optimization problem that models many resource allocation problems.

An instance $(X, \mathcal{F})$ of the set-covering problem consists of a finite set X and a family (or collection) $\mathcal{F}$ of subsets of X, such that every element of X belongs to at least one subset in $\mathcal{F}$: $X = \bigcup_{S \in \mathcal{F}} S$.

We say that a subset $S \in \mathcal{F}$ covers the elements in X. The problem is to find a minimum size subset $\mathcal{C} \subseteq \mathcal{F}$ whose members cover all of X: $X = \bigcup_{S \in \mathcal{C}} S$.

35.4 The set-covering problem (cont)



An example:

$\mathcal{F} = \{S_1, S_2, S_3, S_4, S_5, S_6\}$, X contains 12 black points. A minimum size set cover is $\mathcal{C}^* = \{S_3, S_4, S_5\}$. The greedy algorithm produces a cover $\mathcal{C} = \{S_1, S_4, S_5, S_3\}$ of size 4.

35.4 The set-covering problem(cont)

A greedy approximation algorithm

Greedy_Set_Cover$(X, \mathcal{F})$

```
1   U ← X; /* uncovered elements in X */
2   C ← ∅; /* the set of cover */
3   while U ≠ ∅ do
4       Select a S ∈ F that maximizes |S ∩ U|
5       U ← U − S; /* all elements in S are covered */
6       C ← C ∪ {S};
7   endwhile
8   return C
```

35.4 The set-covering problem(cont)

Time complexity analysis of **Greedy_Set_Cover**. Denote by $H(d)$ the dth harmonic number $H_d = \sum_{i=1}^{d} 1/i$, assuming that $H(0) = 0$.

Theorem: Greedy_Set_Cover is a polynomial time $\rho(n)$-approximation algorithm, where $\rho(n) = H(\max\{|S| \mid S \in \mathcal{F}\})$.

Proof. We assign a cost of 1 to each selected set by the algorithm, distribute the cost (1) over the elements that are covered **for the first time**, and then use these costs to derive the desired relationship between the size of the optimal set cover $\mathcal{C}^*$ and the size of the set cover $\mathcal{C}$ delivered by the algorithm.

Let S_i denote the i th subset selected by the algorithm. For an element $x \in X$, if x is covered for the first time, then,

$$c_x = \frac{1}{|S_i - (S_1 \cup S_2 \cup \ldots \cup S_{i-1})|},$$

35.4 The set-covering problem(cont)

At each step of the algorithm, one unit of cost is charged, thus, $|C| = \sum_{x \in X} c_x$. The cost assigned to the optimal cover is $\sum_{S \in C^*} \sum_{x \in S} c_x$, since each $x \in X$ is in at least one set $S \in C^*$, we have

$$\sum_{S \in C^*} \sum_{x \in S} c_x \geq \sum_{x \in X} c_x = |C|.$$

If we are able to show that

$$\sum_{x \in S} c_x \leq H(|S|),$$

then,

$$\begin{aligned} |C| &\leq \sum_{S \in C^*} \sum_{x \in S} c_x && (2) \\ &\leq \sum_{S \in C^*} H(|S|) && (3) \\ &\leq |C^*| H(\max\{|S| \mid S \in \mathcal{F}\}). && (4) \end{aligned}$$

35.4 The set-covering problem(cont)

The rest is to show that $\sum_{x \in S} c_x \leq H(|S|)$.

Consider any set $S \in \mathcal{F}$ and $i = 1, 2, \ldots, |\mathcal{C}|$ and let

$$u_i = |S - (S_1 \cup S_2 \cup \ldots \cup S_i)|$$

be the number of elements in S remaining uncovered after $S_1, S_2, \ldots, S_i$.

We have $u_0 = |S|$. Let k be the least index such that $u_k = 0$, so each element in S is covered by at least one of the sets $S_1, S_2, \ldots, S_k$. Then,

$$u_{i-1} \geq u_i.$$

35.4 The set-covering problem(cont)

At step i, S_i is chosen. Thus,

$$\sum_{x \in S} c_x = \sum_{i=1}^{k} (u_{i-1} - u_i) \cdot \frac{1}{|S_i - (S_1 \cup S_2 \cup \ldots \cup S_{i-1})|} \quad (5)$$

$$\leq \sum_{i=1}^{k} (u_{i-1} - u_i) \cdot \frac{1}{|S - (S_1 \cup S_2 \cup \ldots \cup S_{i-1})|} \quad (6)$$

$$= \sum_{i=1}^{k} (u_{i-1} - u_i) \frac{1}{u_{i-1}} \quad (7)$$

$$= \sum_{i=1}^{k} \sum_{j=u_i+1}^{u_{i-1}} \frac{1}{u_{i-1}} \quad (8)$$

(9)

The derivation correctness from Eq (5) to Eq. (7) is that
$|S_i - (S_1 \cup S_2 \cup \ldots \cup S_{i-1})| > |S - (S_1 \cup S_2 \cup \ldots \cup S_{i-1})|$ by the greedy strategy.

35.4 The set-covering problem(cont)

$$\leq \sum_{i=1}^{k} \sum_{j=u_i+1}^{u_{i-1}} \frac{1}{j} \tag{10}$$

$$= \sum_{i=1}^{k} \left(\sum_{j=1}^{u_{i-1}} \frac{1}{j} - \sum_{j=1}^{u_i} \frac{1}{j} \right) \tag{11}$$

$$= \sum_{i=1}^{k} (H(u_{i-1}) - H(u_i)) \tag{12}$$

$$= H(u_0) - H(u_k) \tag{13}$$

$$= H(u_0) - H(0) \tag{14}$$

$$= H(u_0) \tag{15}$$

$$= H(|S|) \tag{16}$$

35.5 The maximum coverage problem

Given a set system $\mathcal{F} = \{S_1, S_2, \ldots, S_m\}$ and a fixed parameter k, the **the maximum coverage** problem is to find k subsets in $\mathcal{F}$ such that the total weight of elements covered is maximized, $m > k$.

The maximum coverage problem is clearly NP-hard, as the set covering problem is reducible to it.

35.5 The maximum coverage problem(cont)

Greedy_Maximum_Coverage$(X, \mathcal{F})$

```
1    U ← F;
2    Greedy ← ∅;
3    for  l ← 1 to k do
4         select a S_l ∈ U that maximizes uncovered elements wt(S_l − Greedy ∩ S_l);
5         Greedy ← Greedy ∪ S_l;
6         U ← U \ {S_l};
7    endfor
8    return Greedy;
```

where $wt(Greedy)$ and $wt(OPT)$ are the total weight of elements covered by the greedy solution and the optimal solution respectively. Here S_l refers to the lth set selected by the greedy algorithm, $1 \leq l \leq m$.

35.5 The maximum coverage problem(cont)

Lemma: $wt(\bigcup_{i=1}^{l} S_i) - wt(\bigcup_{i=1}^{l-1} S_i) \geq \frac{wt(OPT) - wt(\bigcup_{i=1}^{l-1} S_i)}{k}$, for $l = 1, 2, \ldots, k$.

proof: $wt(OPT) - wt(\bigcup_{i=1}^{l-1} S_i)$ worth of elements are not covered by the first $(l-1)$ sets selected by the greedy heuristic but covered by the k sets of OPT. By the pigeonhole principle, one of the k sets in the optimal solution must cover at least $\frac{wt(OPT) - wt(\bigcup_{i=1}^{l-1} S_i)}{k}$ worth of these elements.

Since S_l is the set achieving maximum additional coverage,

$$wt(S_l) \geq \frac{wt(OPT) - wt(\bigcup_{i=1}^{l-1} S_i)}{k}.$$

35.5 The maximum coverage problem(cont)

Lemma: $wt(\bigcup_{i=1}^{l} S_i) \geq (1-(1-1/k)^l)wt(OPT)$, for $1 \leq l \leq k$.
proof: We proceed by induction on l. For $l = 1$, the result holds: $wt(S_1) \geq \frac{wt(OPT)}{k}$.

Following the above lemma, we have

$$
\begin{aligned}
wt(\bigcup_{i=1}^{l+1} S_i) &= wt(\bigcup_{i=1}^{l} S_i) + wt(\bigcup_{i=1}^{l+1} S_i) - wt(\bigcup_{i=1}^{l} S_i) \\
&\geq wt(\bigcup_{i=1}^{l} S_i) + \frac{wt(OPT) - wt(\bigcup_{i=1}^{l} S_i)}{k} \\
&= (1-1/k)wt(\bigcup_{i=1}^{l} S_i) + \frac{wt(OPT)}{k} \\
&\geq (1-1/k)(1-(1-1/k)^l)wt(OPT) + \frac{wt(OPT)}{k} \\
&= (1-(1-1/k)^{l+1})wt(OPT)
\end{aligned}
\tag{17}
$$

35.5 The maximum coverage problem(cont)

Theorem: $wt(Greedy) \geq (1-(1-1/k)^k)wt(OPT) > (1-1/e)wt(OPT)$.

Note that

$$lim_{k\to\infty} 1-(1-1/k)^k = 1 - lim_{k\to\infty}(1-1/k)^k = 1-1/e > 0.632$$

and $(1-(1-1/k)^k)$ is a decreasing function.

Load balancing problem

Given a set of m machines $M_1, M_2, \ldots, M_m$ and a set of n jobs, each job j has a processing time $t_j > 0$ with $1 \leq j \leq n$. We seek to assign each job to one of the m machines so that the loads placed on all machines are as "balanced" as possible.

Let $A(i)$ denote the set of jobs assigned to machine M_i. Under this assignment, machine M_i needs to work for a total time of $T_i = \sum_{j \in A(i)} t_j$, which is the load at machine M_i for all i, $1 \leq i \leq m$.

We seek to minimize a quantity known as the makespan, i.e., the maximum load among all machines $T = \max\{T_i \mid 1 \leq i \leq m\}$ is minimized.

Load balancing problem

Greedy_Balance

1 **for** $i \leftarrow 1$ to m **do**
2 $T_i \leftarrow 0$;
3 $A(i) \leftarrow \emptyset$;
4 **endfor**;
5 **for** $j \leftarrow 1$ to n **do**
6 Let M_i be a machine achieving the minimum load $\min\{T_{i'} \mid 1 \leq i' \leq m\}$;
7 Assign job j to machine M_i;
8 $A(i) \leftarrow A(i) \cup \{j\}$;
9 $T_i \leftarrow T_i + t_j$;
10 **endfor**

Load balancing problem

Lemma: Let T^* be the optimal makespan, then

(i) $T^* \geq \frac{1}{m} \sum_{j=1}^{n} t_j$;

(ii) $T^* \geq \max\{t_j \mid 1 \leq j \leq n\}$.

Load balancing problem

Theorem: Algorithm Greedy-Balance produces an assignment of jobs to machines with makespan $T \leq 2T^*$.

proof: We assume that machine M_i attains the maximum load T in our assignment. We assume that job j is the last job assigned to machine M_i, then the load of M_i is the smallest at that moment, which is $T_i - t_j$, and every other machine has a load at least $T_i - t_j$. Thus, we have $\sum_{k=1}^{m} T_k \geq m(T_i - t_j)$, or

$$T_i - t_j \leq \frac{1}{m}\sum_{k=1}^{m} T_k = \frac{1}{m}\sum_{j=1}^{n} t_j.$$

We thus have $T_i - t_j \leq \frac{1}{m}\sum_{j=1}^{n} t_j \leq T^*$ by the lemma.

As we assume that the makespan T is equal to T_i, we have

$$T = T_i = (T_i - t_j) + t_j \leq T^* + T^* = 2T^*.$$

Randomized Approximation Algorithms

In the following we study one very useful technique in designing approximation algorithms

- Randomization.

We say that **a randomized algorithm** for a problem has an **approximation ratio** of $\rho(n)$ if, for any input of size n, the *expected cost* C of the solution produced by the randomized algorithm is within a factor of $\rho(n)$ of the cost C^* of an optimal solution.

We also call a randomized algorithm that achieves an approximation ratio of $\rho(n)$ a **randomized $\rho(n)$-approximation algorithm**.

Randomized approximation algorithm for MAX-3-CNF satisfiability

Theorem: Given an instance of MAX-3-CNF satisfiability with n variables $x_1, x_2, \ldots, x_n$ and m clauses, the randomized algorithm that independently sets each variable to 1 with probability 1/2 and to 0 with probability 1/2 is a randomized 7/8-approximation algorithm.

proof: We define the indicator random variable $Y_i = I\{$ clause i is satisfied $\}$, so that $Y_i = 1$ as long as at least one of the literals in the ith clause has been set to 1. Since no literal appears in the same clause more than once, and we have assumed that no variable and its negation appear in the same clause, the settings of the three literals in each clause are independent. A clause is not satisfied only if all three of its literals are set to 0, and so $\Pr\{$ clause i is not satisfied$\} = (1/2)^3 = 1/8$. Thus, $\Pr\{$ clause i is satisfied$\} = 1 - 1/8 = 7/8$.

Let Y be the number of satisfied clauses overall, so that $Y = Y_1 + Y_2 + \ldots + Y_m$. Then we have

$$E[Y] = E[\sum_{i=1}^{m} Y_i] = \sum_{i=1}^{m} E[Y_i] = \sum_{i=1}^{m} 7/8 = 7m/8.$$

Clearly, there are at most m clauses satisfied. Thus, the approximation ratio is

$$\frac{7m/8}{m} \geq 7/8.$$