Set Cover

Defintion 5.4 Set Cover

Input:

- A universe U of n elements
- A collection of subsets of U
- $S=\{S_1,\ldots,S_k\}$
- A cost function c: S→**Q+**

Output: a minimum cost sub collection of S that covers all elements of U.

Frequency of an element: number of sets it is in. f: frequency of most frequent element.

Approximation algorithms for SC achieve either O(log n) or f.

Spezialfall VC:

U:=E

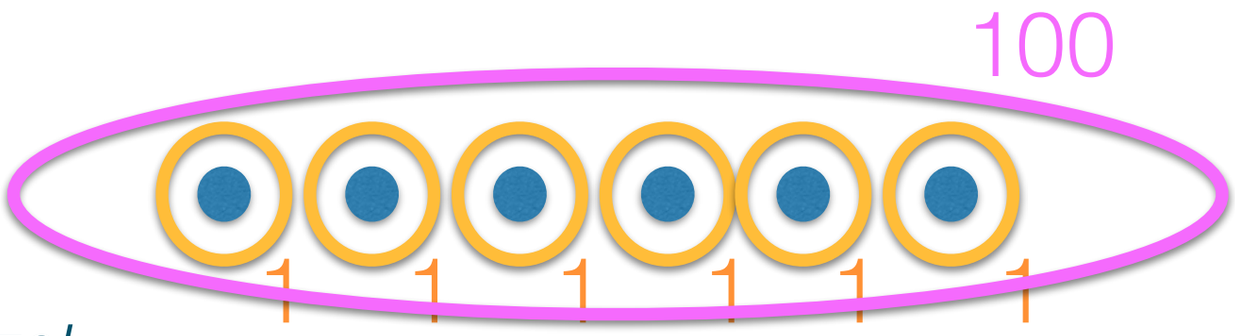
$S_i:=\{e\in E \mid e=\{v_i, w\} \in E, w\in V\}$

(f=2)

Approximation?

Idea: cover as much as possible at once

Possible problem: we do not consider the cost:



⟹ Consider the cost per covered element

⟹$|S_i|$ / $c(S_i)$ — covering per cost unit - *maximize!*

⟹$c(S_i)$ / $|S_i|$ — cost per element - *minimize!*

Note: if wer use this criteria repetedly, we should only consider the remaining un-covered elements.

Set Cover

Price of an element: the average cost at which it is covered
When a set S is picked, we can think of its cost being distributed equally among the new elements covered, to set their prices.

Algorithm 5.5 (Greedy Set Cover Algorithm)
Input: collection of subsets of U, costs $c(S_i)>0$ for each subset.
Output: a SC of U.

1. C=∅, S=∅ (C is the set of elements already covered at the beginning of an iteration)
2. WHILE (C≠U) DO
   Find the set whose cost-effectiveness is smallest, say S.
   α = c(S)/|S\C| (cost-effectiveness of S)
   Pick S, and for each element e∈S\C, set price(e)=α
   C=C∪S
3. Output the picked sets.

Set Cover

Algorithm 5.5 (Greedy Set Cover Algorithm)
Input: collection of subsets of U, costs $c(S_i)>0$ for each subset.
Output: a SC of U.
1. C=∅, S=∅ (C is the set of elements already covered at the beginning of an iteration)
2. WHILE (C≠U) DO
   Find the set whose cost-effectiveness is smallest, say S.
   α = c(S)/|S\C| (cost-effectiveness of S)
   Pick S, and for each element e∈S\C, set price(e)=α
   C=C∪S
3. Output the picked sets.

Lemma 5.6: For each k∈{1,…,n}, $price(e_k) \leq OPT/(n-k+1)$.
Proof:
In each iteration, when we choose a set S_i, we can cover the not yet covered elements C^ with cost at most OPT (C^=U-C).
$\Longrightarrow$ Among the elements in C^ must be an element with at most the average cost-effectiveness OPT/|C^| as price.
Let e_k be this element.
In the iteration in which we cover e_k at least the n-k+1 elements $e_k, \ldots, e_n$ were not covered.
$\Longrightarrow price(e_k) \leq OPT/|C^| \leq OPT/(n-k+1)$

Set Cover

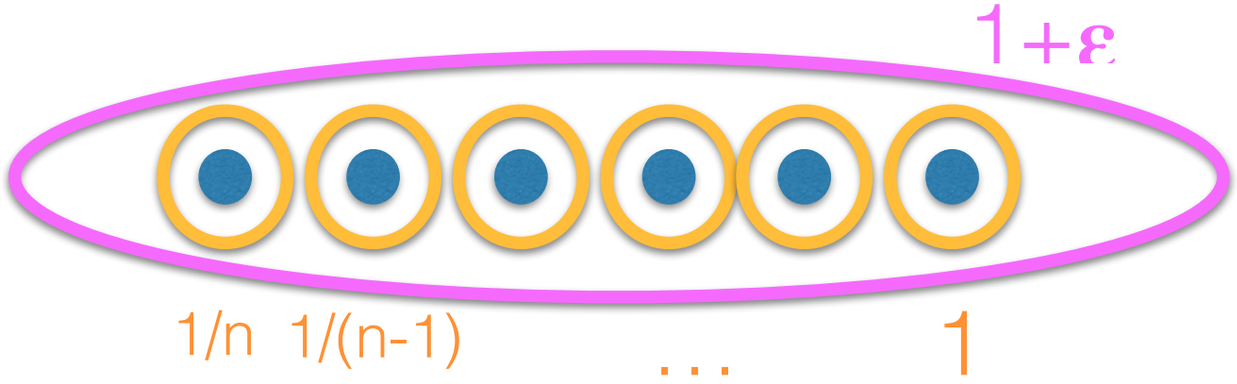
Algorithm 5.5 (Greedy Set Cover Algorithm)
Input: collection of subsets of U, costs $c(S_i)>0$ for each subset.
Output: a SC of U.
1. C=∅, S=∅ (C is the set of elements already covered at the beginning of an iteration)
2. WHILE (C≠U) DO
   Find the set whose cost-effectiveness is smallest, say S.
   α = c(S)/|S\C| (cost-effectiveness of S)
   Pick S, and for each element e∈S\C, set price(e)=α
   C=C∪S
3. Output the picked sets.

Theorem 5.7: The greedy algorithm is an H_n-approximation algorithm for the minimum set cover problem, where $H_n=1+1/2+\ldots+1/n$.
Proof:
For each i and $e_k \in S_i \backslash C$ $\text{price}(e_k)= c(S_i)/|S_i \backslash C|$ gives the cost fraction of e_k of the total cost of S_i.
$\Longrightarrow$ When we cover elements in the order $e_1,\ldots,e_k,\ldots e_n$:

$$\begin{aligned}\sum_{i\in S} c(S_i) &= \sum_{k=1}^{n} \text{price}(e_k) \\ &\leq \sum_{k=1}^{n} \frac{OPT}{n-k+1} \\ &= H_n \times OPT\end{aligned}$$

Set Cover

Tight?



Greedy outputs:
cover of the n singletons (in each iteration some singleton is the most cost-effective set)
$\Longrightarrow$Cost of algorithm: $1/n+1/(n-1)+\ldots+1 = H_n$

Optimal cover:
cost $1+\boldsymbol{\varepsilon}$

Set Cover

Set Cover is a very general problem: many optimization problems can be formulated as a Set Cover problem.

Set Cover cannot be approximated better than $\Omega(\log n)$ if not P=NP. For details see Chapter 29 of the Vazirani-book.

Application of Set Cover: Shortest Superstring

DNA analysis:
View: human DNA as very long string over four-letter alphabet
Scientists: try to decipher this string
Very long string $\Longrightarrow$ first decipher several overlapping short segments

Locations of these segments on the original DNA not known
Hypothesis: The shortest string which contains these segments as substrings is a good approximation to the original DNA string
More formal:

Problem 5.8: Shortest Superstring
Given: Finite alphabet Σ (for us: A,C,G,T), and a set of strings, $S=\{s_1,\ldots,s_n\}\in\Sigma^+$
Find: shortest string s that contains each s_i as a substring
Wlog: no string s_i is a substing of another string s_j, $j\neq i$

Example:
AC…..C , C….CG (each with k times C)
Can be combined to: AC^kG
If we add a third string C^{k+1}
$\Longrightarrow$Shortest superstring: $AC^{k+1}G$

The problem is NP-hard.

Application of Set Cover: Shortest Superstring

Greedy Algorithm

1: **Greedy Shortest Superstring**
2: **input:** A set of strings S.
3: **output:** A short superstring of S.
4: T $\leftarrow$ S
5: **while** $|T| > 1$ **do**
6: Let a and b be the most overlapping strings of T
7: Replace a and b with the string obtained by overlapping a and b
8: **end while**
9: T contains a superstring of S

source: http://fileadmin.cs.lth.se/cs/Personal/Andrzej_Lingas/superstring.pdf

Application of Set Cover: Shortest Superstring

Example

- S = T = {**CATGC**, CTAAGT, GCTA, TTCA, **ATGCATC**}
- T = {CATGCATC, **CTAAGT**, **GCTA**, TTCA}
- T = {**CATGCATC**, GCTAAGT, **TTCA**}
- T = {**TTCATGCATC**, **GCTAAGT**}
- T = {GCTAAGTTCATGCATC}

source: http://fileadmin.cs.lth.se/cs/Personal/Andrzej_Lingas/superstring.pdf

Application of Set Cover: Shortest Superstring

Approximation guarantee

- ALG $\leq 4 \cdot \mathrm{OPT}$ (proved by Blum et. al.)
- ALG $\leq 2 \cdot \mathrm{OPT}$ (conjectured)

Conjectured worst case

$S = \{ab^k, b^k c, b^{k+1}\}$

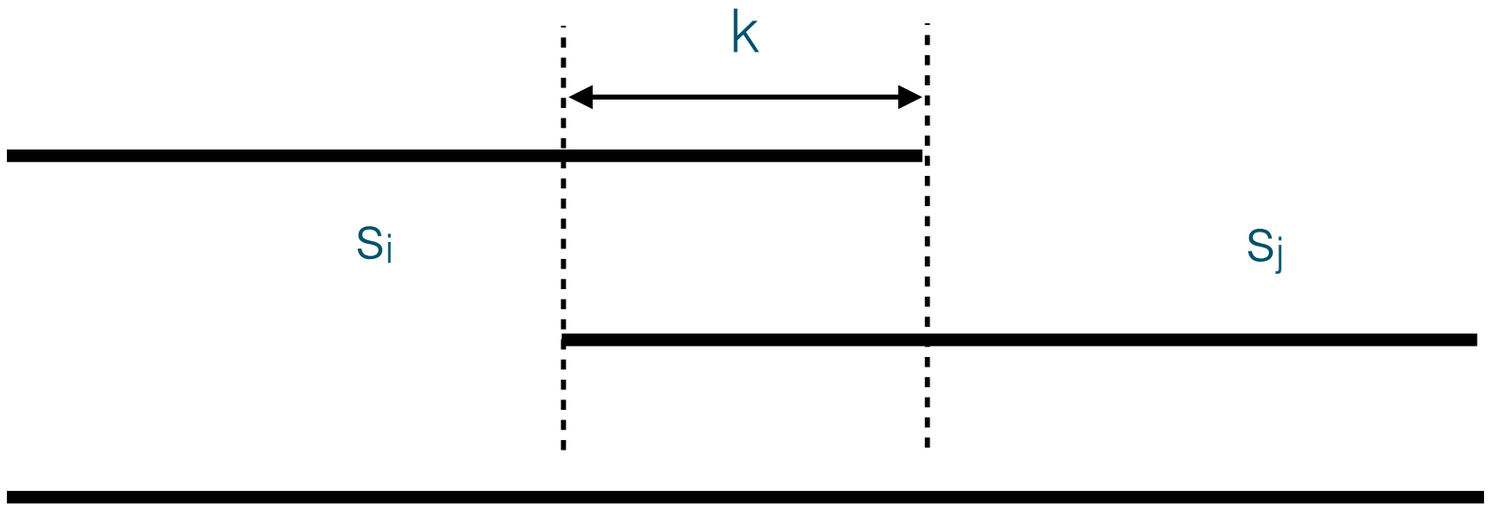
source: http://fileadmin.cs.lth.se/cs/Personal/Andrzej_Lingas/superstring.pdf

Application of Set Cover: Shortest Superstring

Using Set Cover:

We consider possibilities to concatenate pairs of strings.

For s_i, $s_j \in S$, $k>0$: if the last k symbols of s_i are the same as the first k symbols of s_j, let σ_{ijk} be the string obtained by overlapping these k positions of s_i and s_j:



Set Cover: Choose *sets* that cover all *elements* with *least* cost, we define instance ***S:***

Elements:

- The input strings $S=\{s_1,\ldots,s_n\}$

Subsets:

- M= set of strings σ_{ijk} for all valid choices of i,j,k
- $\beta=S \cup M$
- For a string $\pi \in \beta$: $set(\pi)=\{s \in S \mid s \text{ is a substring of } \pi\}$

Cost of a subset: $c(\pi)=|\pi|$

Application of Set Cover: Shortest Superstring

Algorithm 5.9 (Shortest Superstring via Set Cover)

1. Use the greedy set cover algorithm to find a cover for the instance $\mathcal{S}$. Let set(π_1), …,set(π_k) be the sets picked by this cover.
2. Concatenate the strings $\pi_1,\ldots,\pi_k$ in any order.
3. Output the resulting string, say s.

Example

S = {CATGC, CTAAGT, GCTA, TTCA, ATGCATC}

π	Set	Cost
CATGC..... ....CTAAGT CATGCTAAGT	CATGC, CTAAGT, GCTA	10
CATGC.. ...GCTA CATGCTA	CATGC, GCTA	7
......CATGC ATGCATC.... ATGCATCATGC	CATGC, ATGCATC	11
CTAAGT... .....TTCA CTAAGTTCA	CTAAGT, TTCA	9
ATGCATC..... ......CTAAGT ATGCATCTAAGT	CTAAGT, ATGCATC	12
GCTA...... ...ATGCATC GCTATGCATC	GCTA, ATGCATC	10
TTCA...... ...ATGCATC TTCATGCATC	TTCA, ATGCATC, CATGC	10
GCTA... .CTAAGT GCTAAGT	GCTA, CTAAGT	7
TTCA... ..CATGC TTCATGC	CATGC, TTCA	7
CATGC... .ATGCATC CATGCATC	CATGC, ATGCATC	8
CATGC	CATGC	5
CTAAGT	CTAAGT	6
GCTA	GCTA	4
TTCA	TTCA	4
ATGCATC	ATGCATC	7

source: http://fileadmin.cs.lth.se/cs/Personal/Andrzej_Lingas/superstring.pdf

Application of Set Cover: Shortest Superstring

Algorithm 5.9 (Shortest Superstring via Set Cover)

1. Use the greedy set cover algorithm to find a cover for the instance $\mathcal{S}$.
   Let set(π_1),…,set(π_k) be the sets picked by this cover.
2. Concatenate the strings $\pi_1,\ldots,\pi_k$ in any order.
3. Output the resulting string, say s.

Lemma 5.10: $OPT \leq OPT_{\mathcal{S}} \leq 2\ OPT$ (OPT= OPT for SSP, $OPT_{\mathcal{S}}$ =OPT for SCP)

Proof:

s is a feasible superstring, hence, we have $OPT \leq OPT_{\mathcal{S}}$

Let s be a shortest superstring of $s_1, \ldots, s_n$, $|s|=OPT$

Sufficient: produce *some* set cover of cost at most 2 OPT.

Consider the leftmost occurence of the strings $s_1, \ldots, s_n$ in string s

No string substring of another $\Longrightarrow$ these n leftmost occurrences start at distinct places in s

$\Longrightarrow$they also end at distinct places

Renumber the strings in the order in which their leftmost occurrences start

$\Longrightarrow$Also the order in which they end

Algorithm 5.9 (Shortest Superstring via Set Cover)

1. Use the greedy set cover algorithm to find a cover for the instance $\mathcal{S}$.
   Let set(π_1),…,set(π_k) be the sets picked by this cover.
2. Concatenate the strings $\pi_1,\ldots,\pi_k$ in any order.
3. Output the resulting string, say s.

Proof ctd:

We partition the ordered list of strings into groups:

- Each group: contiguous set of strings from this list
- Let b_i and e_i denote the index of the first and last string in the i-th group ($b_i=e_i$ is fine)

➡ $b_1=1$

- e_1=largest index of a string that overlaps with s_1 (there exists at least one such string: s_1)
- In general: if $e_i<n$ $b_{i+1}=e_i+1$
- e_{i+1} is largest index of string that overlaps with s_{bi+1}
- Eventually: $e_t=n$ for some $t\leq n$

  For each pair of strings (s_{bi}, s_{ei}), let $k_i>0$ be the length of the overlap between their leftmost occurrences in s (may be different from their max overlap)

  Let $\pi_i=\sigma_{bi.ei.ki}$

$\Longrightarrow$ {set(π_i) | $1\leq i\leq t$} is a solution for $\mathcal{S}$

Critical observation: π_i does not overlap π_{i+1}

Proof for i=1 (same argument for arbitrary i):

Assume π_1 overlaps π_3

$\Longrightarrow$ occurence of s_{b3} in s overlaps the occurrence of s_{e1}

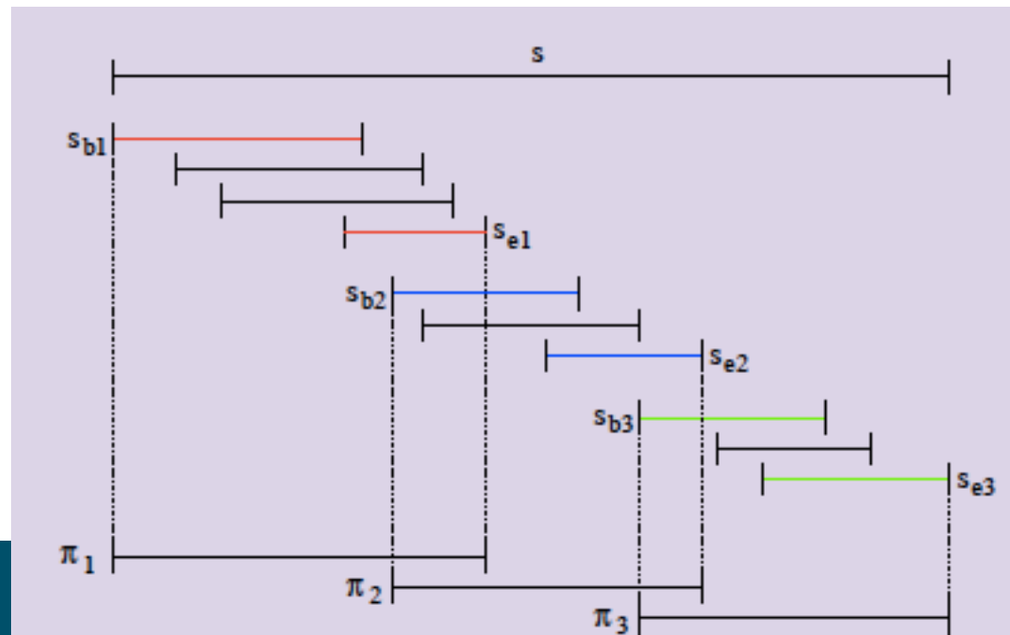
But: s_{b3} does not overlap s_{b2} (ow. s_{b3} would have been put in the 2nd group

$\Longrightarrow$ s_{e1} ends later than s_{b2}

contradiction (property of endings of strings)

$\Longrightarrow$ Each symbol of s is covered by at most two of the π_i's

$\Longrightarrow$ $OPT_{\mathcal{S}} \leq \Sigma_i\ |\pi_i| \leq 2\ OPT$



Application of Set Cover: Shortest Superstring

Algorithm 5.9 (Shortest Superstring via Set Cover)

1. Use the greedy set cover algorithm to find a cover for the instance $\mathcal{S}$.
   Let set(π_1),…,set(π_k) be the sets picked by this cover.
2. Concatenate the strings $\pi_1,\ldots,\pi_k$ in any order.
3. Output the resulting string, say s.

The size of the universal set in $\mathcal{S}$ is n (=number of strings in the given shortest superstring instance) + Lemma 5.10 + Theorem 5.7:

Theorem 5.11: Algorithm 5.9 is a $2H_n$-approximation for the shortest superstring problem, where n is the number of strings in the given instance.