Combinatorial Algorithms

Set Cover Problem

Set Cover

- *Given* a universe U of n elements, a collection of subsets of U, $S = \{S_1, \ldots, S_k\}$, and a cost function $c: S \rightarrow \mathbf{Q}^+$.
- *Find* a minimum cost subcollection of S that covers all elements of U.

Greedy strategy

- Among the first strategies one tries when designing an algorithm for an optimization problem is some form of the greedy strategy.
- Greedy algorithms work by making a sequence of decisions; each decision is made to optimize that particular decision, even though this sequence of locally optimal decisions might not lead to a globally optimal solution.
- The advantage of greedy algorithms is that they are typically very easy to implement, and hence greedy algorithm are a commonly used heuristics, even when they have no performance guarantee.

The greedy algorithm

- Let C be the set of elements already covered at the beginning of an iteration. During this iteration, define the **cost-effectiveness** of a set S_i to be the average cost at which it covers new elements? i.e., $\alpha_i = c(S_i)/|S_i - C|$.
- Define the **price** of an element to the average cost at which it is covered.
- Equivalently, when a set S_i is picked, we can think of its cost being distributed equally among the new elements covered, to set their prices.

Chvatal's Algorithm

0) **Input** $(U, S, c: S \to \mathbf{Q}^{+})$
1) $C \leftarrow \emptyset$**,** $Sol \leftarrow \emptyset$
2) **While** $C \neq U$ **do:**
   Find $S_i \in S - Sol$ such that
   $\alpha_i = c(S_i)/|S_i - C|$ is minimal.
   $Sol \leftarrow Sol \cup \{S_i\}$
   $C \leftarrow C \cup S_i$ (S_i is most cost-effective)
   Set $price(e) = \alpha_i$ for all $e \in S_i - C$
3) **Output** (Sol)

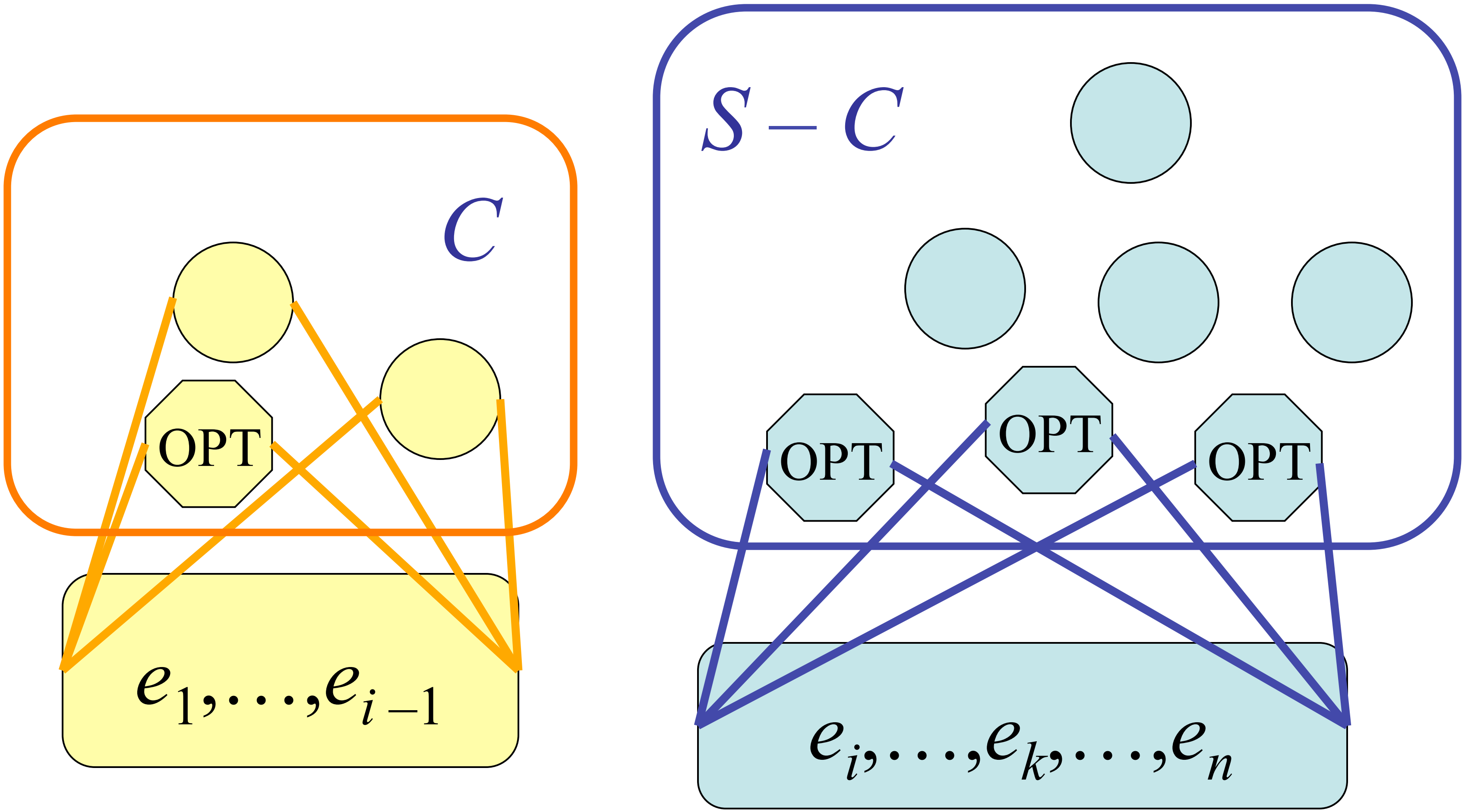
Analysis of Chvatal's Algorithm

- Number the elements of U in the order, in which were covered by the algorithm, resolving ties arbitrarily.
- Let $e_1,\ldots,e_n$ be this numbering.

- **Lemma 2.1**

For each $k \in \{1,\ldots,n\}$, $price(e_k) \leq \mathrm{OPT}/(n-k+1)$.

Proof of Lemma 2.1



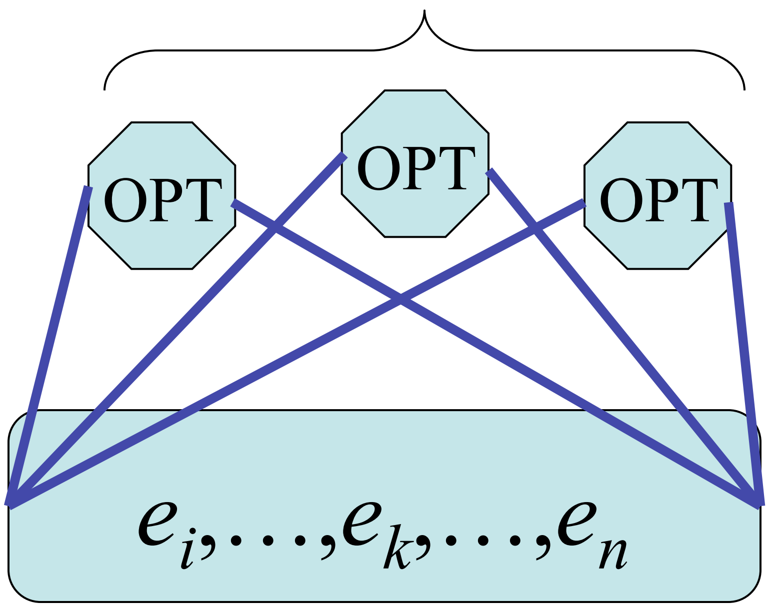
In any iteration, the leftover sets of the optimal solution can cover the remaining elements at a cost of at most OPT.

Proof of Lemma 2.1

$$\frac{a_1}{b_1} \geq \frac{a_2}{b_2} \geq \cdots \geq \frac{a_n}{b_n} \Rightarrow \frac{a_1 + \cdots + a_n}{b_1 + \cdots + b_n} \geq \frac{a_n}{b_n}$$

The total cost-effectiveness is at most OPT/($n - i + 1$) ≤ OPT/($n - k + 1$)

There must be one subset $S_j \in S - C$ with $\alpha_j \leq$ OPT/($n - k + 1$).



price(e_k) ≤ OPT/(n–k+1).

Performance of the Chvatal Algorithm

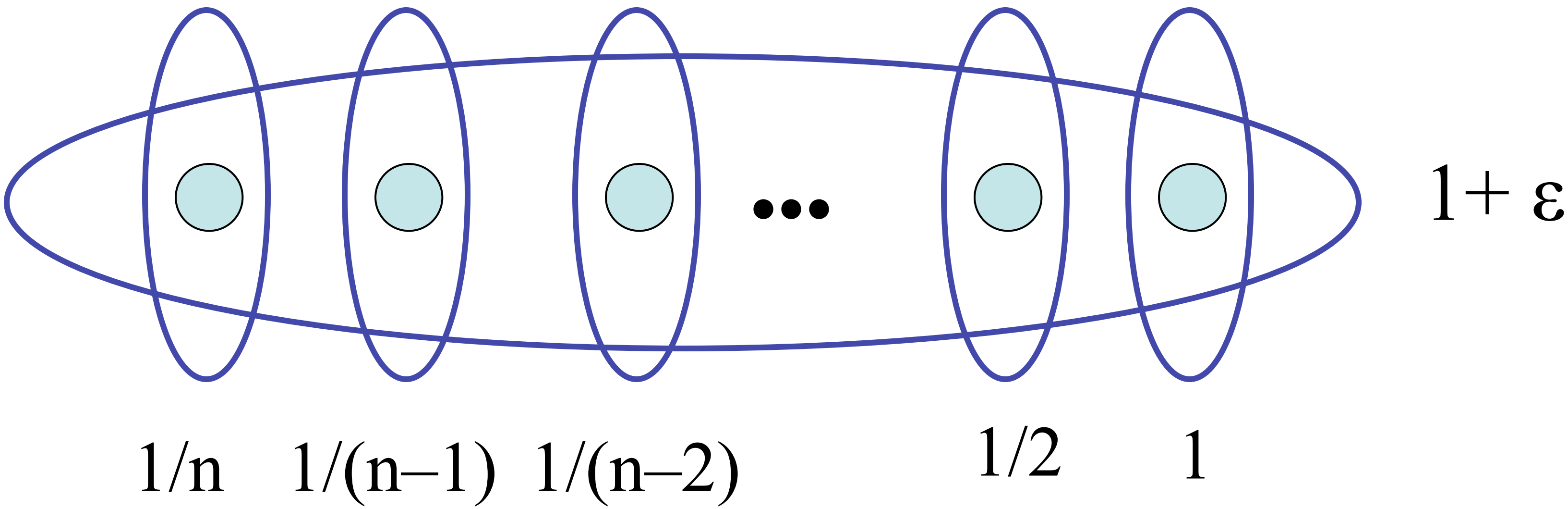
Theorem 2.2

Chvatal's Algorithm is an H_n factor approximation algorithm for the minimum set cover problem, where H_n=1+1/2+1/3+…+1/n.

Proof.

$$\sum_{S_i \in C} c(S_i) = \sum_{k=1}^{n} price(e_k) \le \left(1 + \frac{1}{2} + \cdots + \frac{1}{n}\right) OPT$$

Tight example



Vertex cover

- *Given* an undirected graph $G = (V, E)$, and a cost function on vertices $c: V \to \mathbf{Q}^+$.
- *Find* a minimum cost vertex cover.

Layering

- We introduce a technique of layering.
- The idea in layering is to decompose the given weight function on vertices into convenient functions, called degree-weighted, on a nested sequence of subgraphs G. For degree-weighted functions, we will show that we will be within twice the optimal even if we pick all vertices in the cover.

Degree-weighted function

- Let w: $V \rightarrow \mathbf{Q}^{+}$ be the function assigning weights to the vertices of the given graph $G = (V,E)$.
- We will say that a function assigning vertex weight is **degree-weighted**, if there is a constant $c > 0$, such that the weight of each vertex $v \in V$ is $c \cdot \deg(v)$.

Lower Bound

- **Lemma 2.3**

  Let w: $V \longrightarrow \mathbf{Q}^+$ be a degree-weighted function. Then $w(V) \leq 2$ OPT.

Proof

- Let c be the constant such that $w(c) = c \cdot \deg(v)$, and let U be an optimal vertex cover in G. Since U covers all the edges,

$$\sum_{v \in U} \deg(v) \geq |E|.$$

- Therefore, $w(U) \geq c|E|$. Now, since

$$\sum_{v \in V} \deg(v) = 2|E|, \quad w(V) = 2c|E|.$$

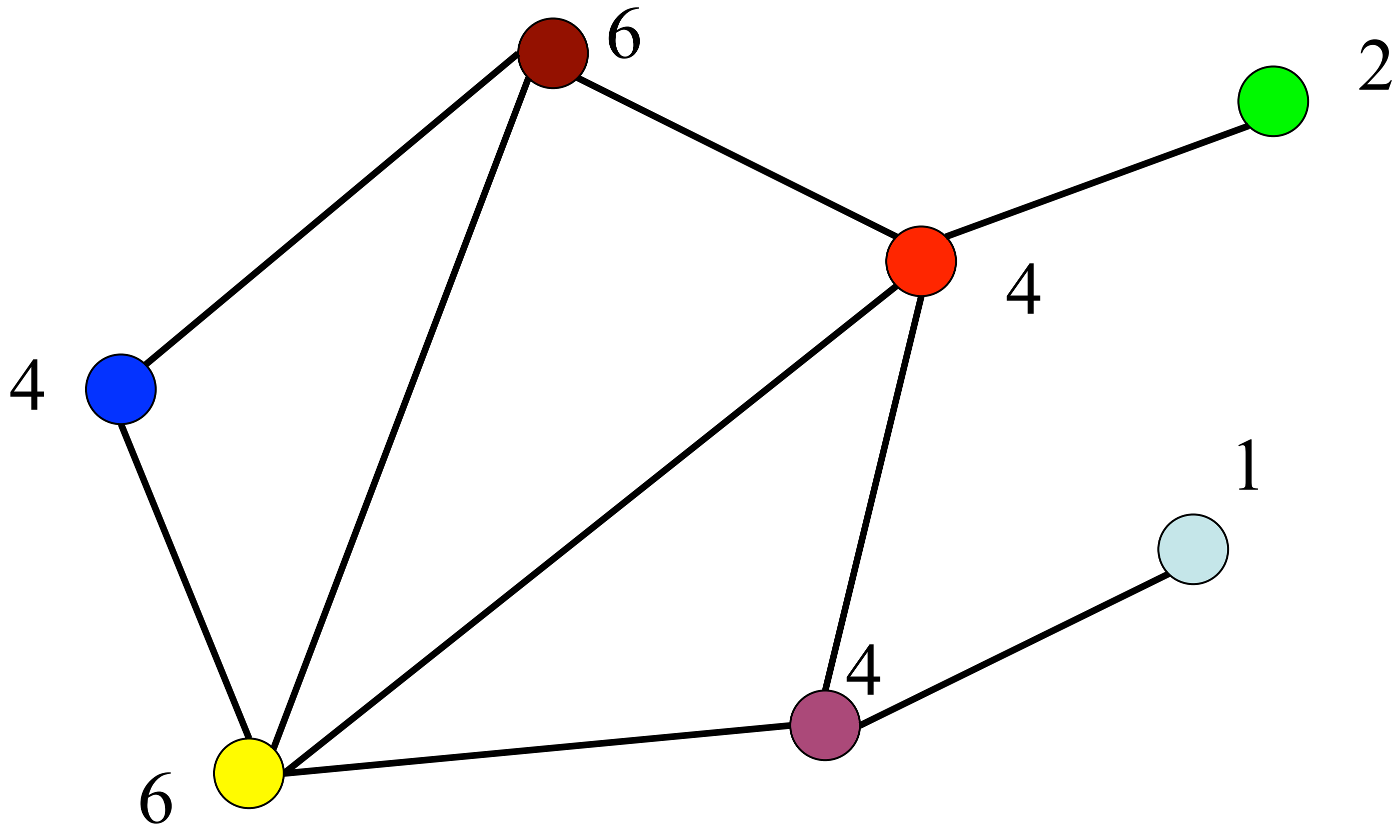
Largest degree-weighted function

- Let $w: V \rightarrow \mathbf{Q}^{+}$ be an arbitrary function.
- Let us define the **largest degree-weighted function in** w as follows:
  - Remove all degree zero vertices from the graph, and over the remaining vertices, compute $c = \min\{w(v)/\deg(v)\}$.
  - Then $t(v) = c \cdot \deg(v)$ is the desired function.
- Define $w'(v) = w(v) - t(v)$ to be the **residual weight function**.

The Layer Algorithm

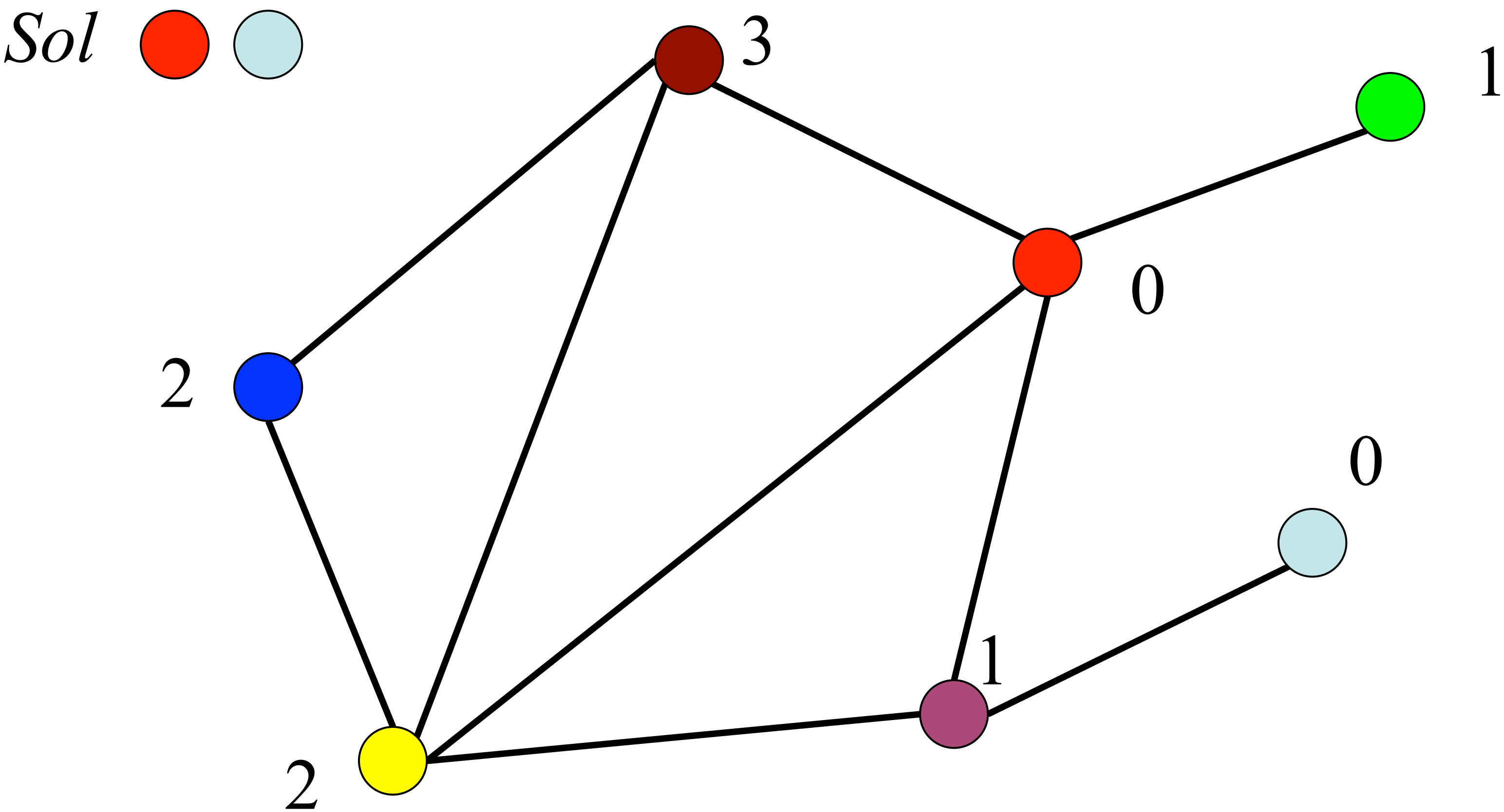
0) **Input** $(G = (V, E), w: V \to \mathbf{Q}^+)$
1) $Sol \leftarrow \varnothing$**,** $i \leftarrow 0$**,** $w'(v) \leftarrow w(v)$**,**
   $V_0 \leftarrow V - D_0$ $(D_0 = \{v \in V \mid \deg(v)=0\})$
2) **While** $V_i \neq \varnothing$ **do:**
   $w'(v) \leftarrow w'(v) - t_i(v)$
   $Sol \leftarrow Sol \cup W_i$ $(W_i = \{v \in V_i \mid w'(v)=0\})$
   $V_{i+1} \leftarrow V_i - W_i$
   $i \leftarrow i+1$
   $V_i \leftarrow V_i - D_i$ $(D_i = \{v \in G_i \mid \deg(v)=0\})$
3) **Output** (Sol)

Example

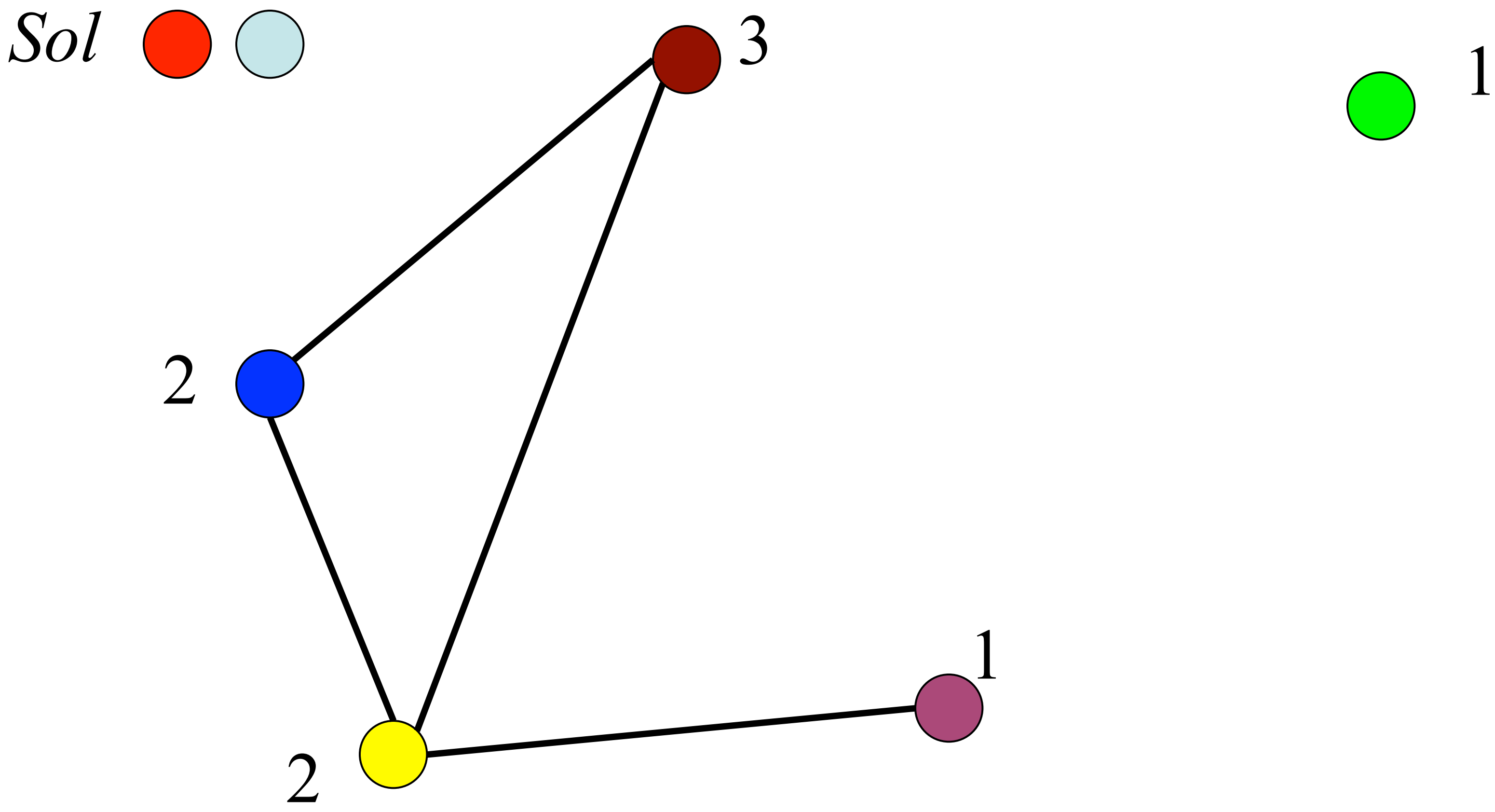


$$t(v) = 1 \cdot \deg(v)$$

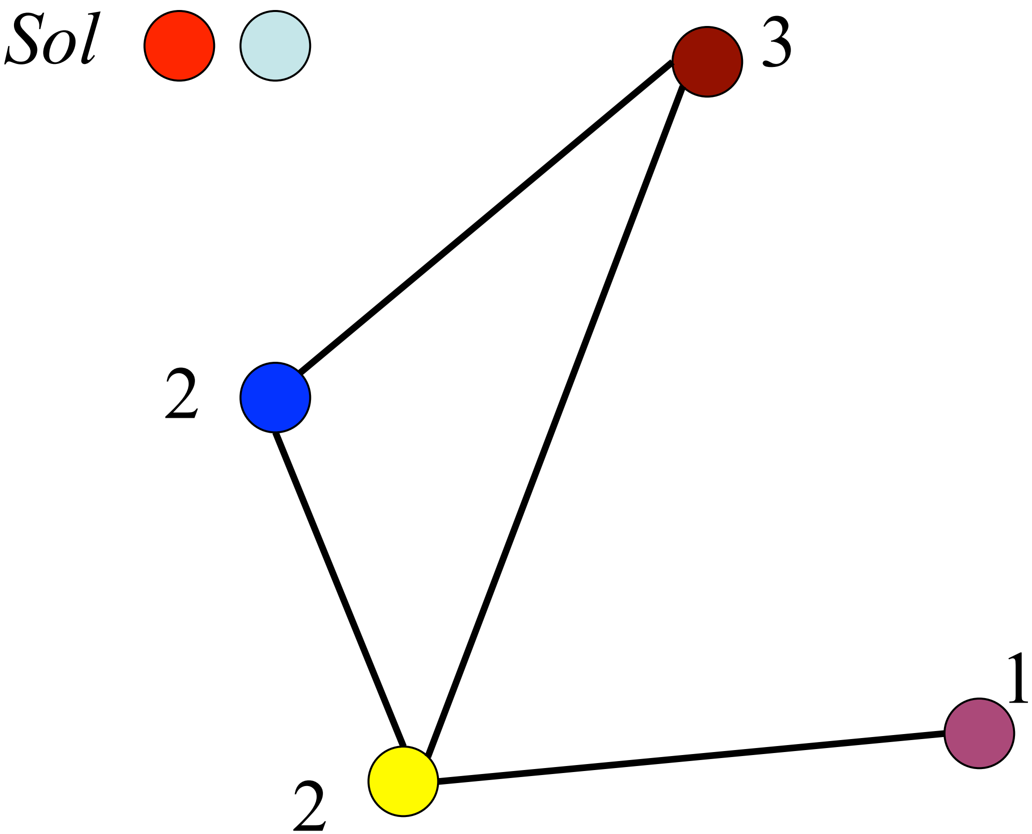
Example



Example

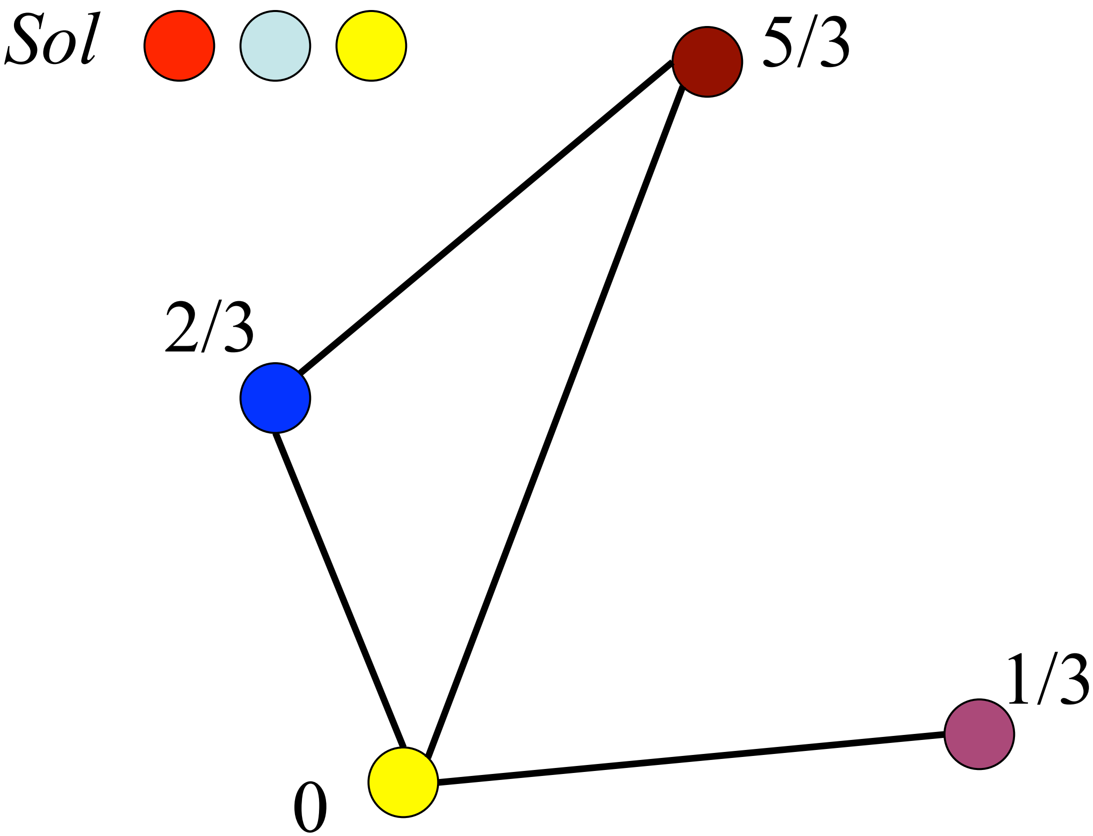


Example

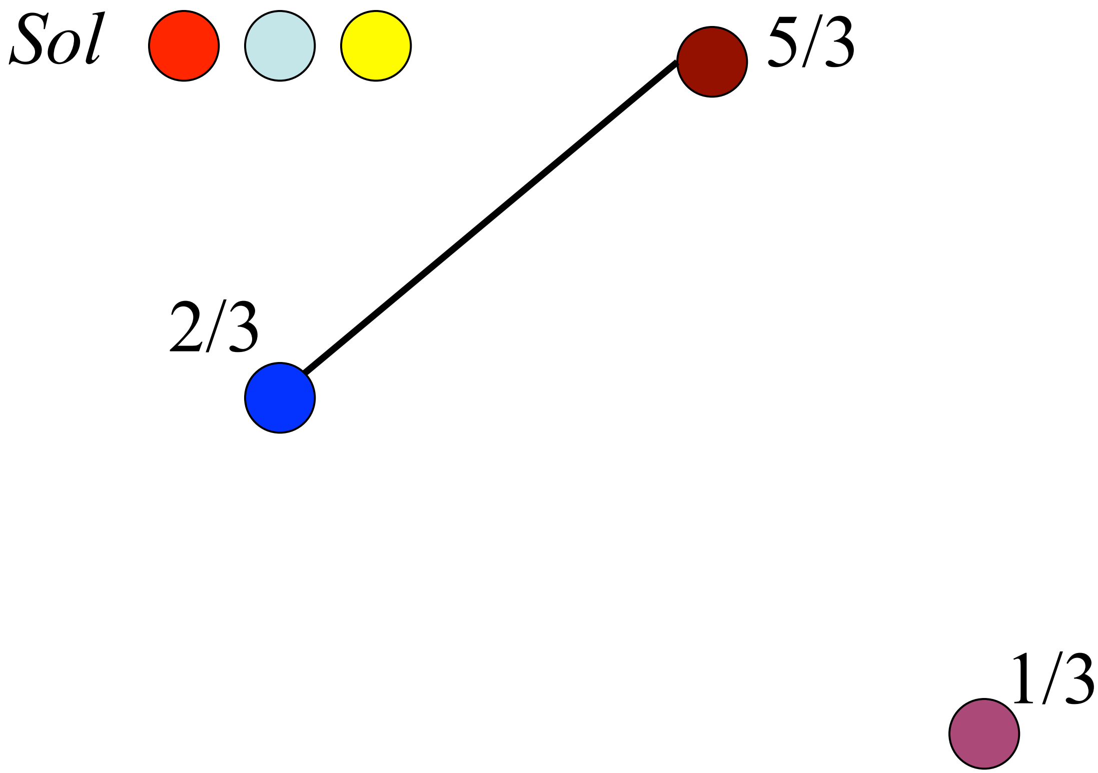


$$t(v) = (2/3) \cdot \deg(v)$$

Example

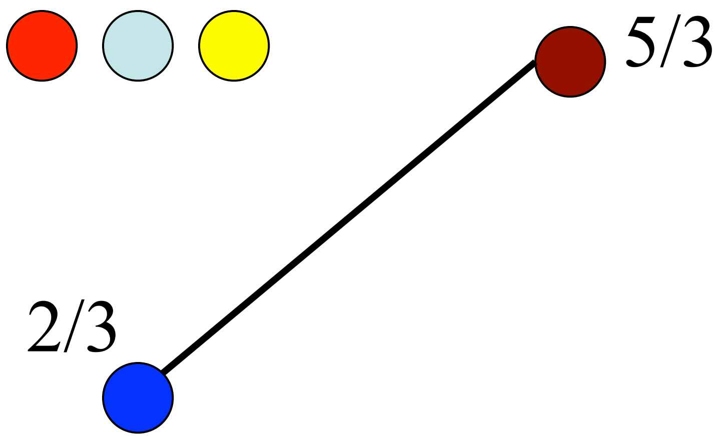


Example



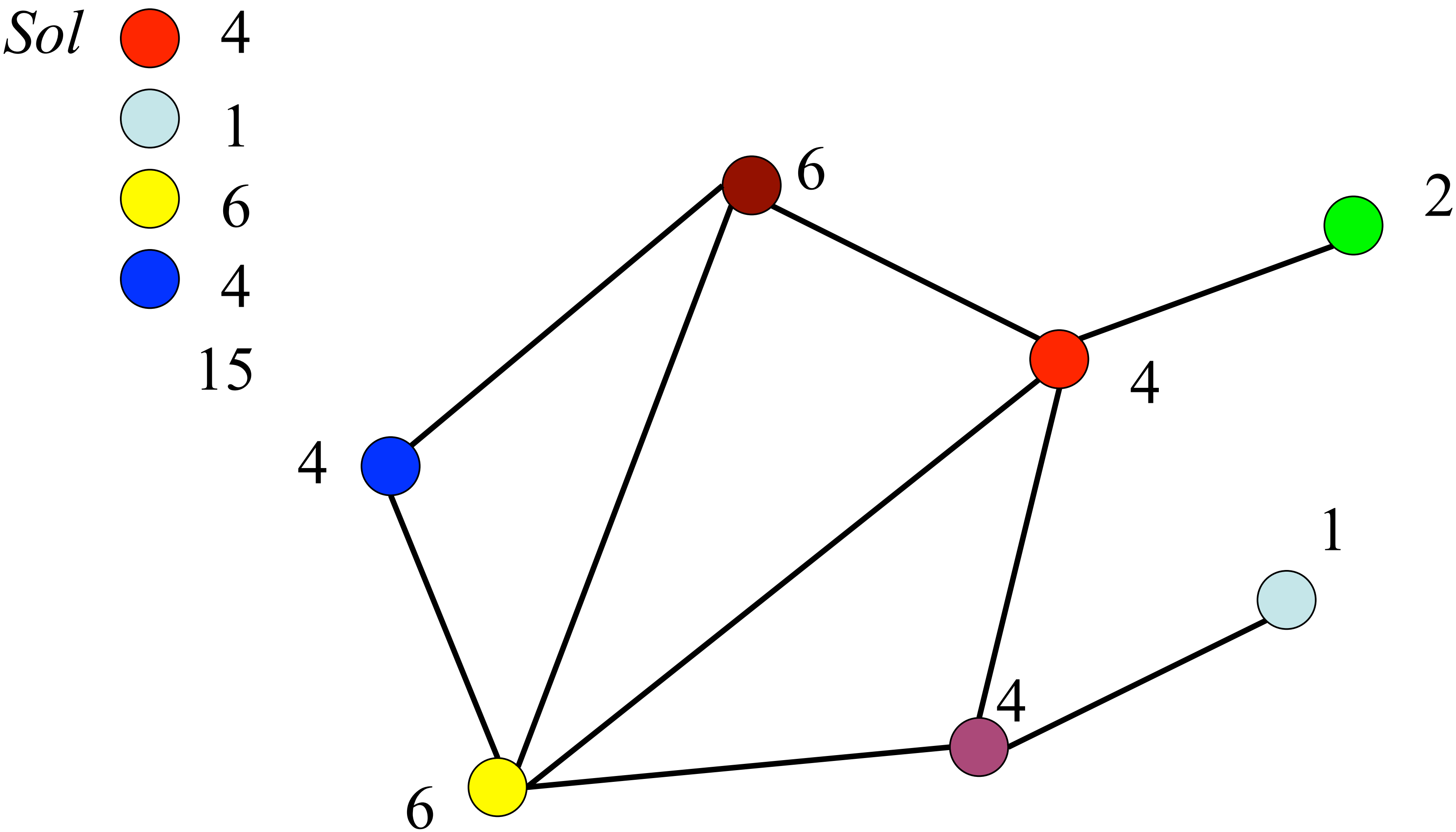
Example

Sol



$$t(v) = (2/3) \cdot \deg(v)$$

Example

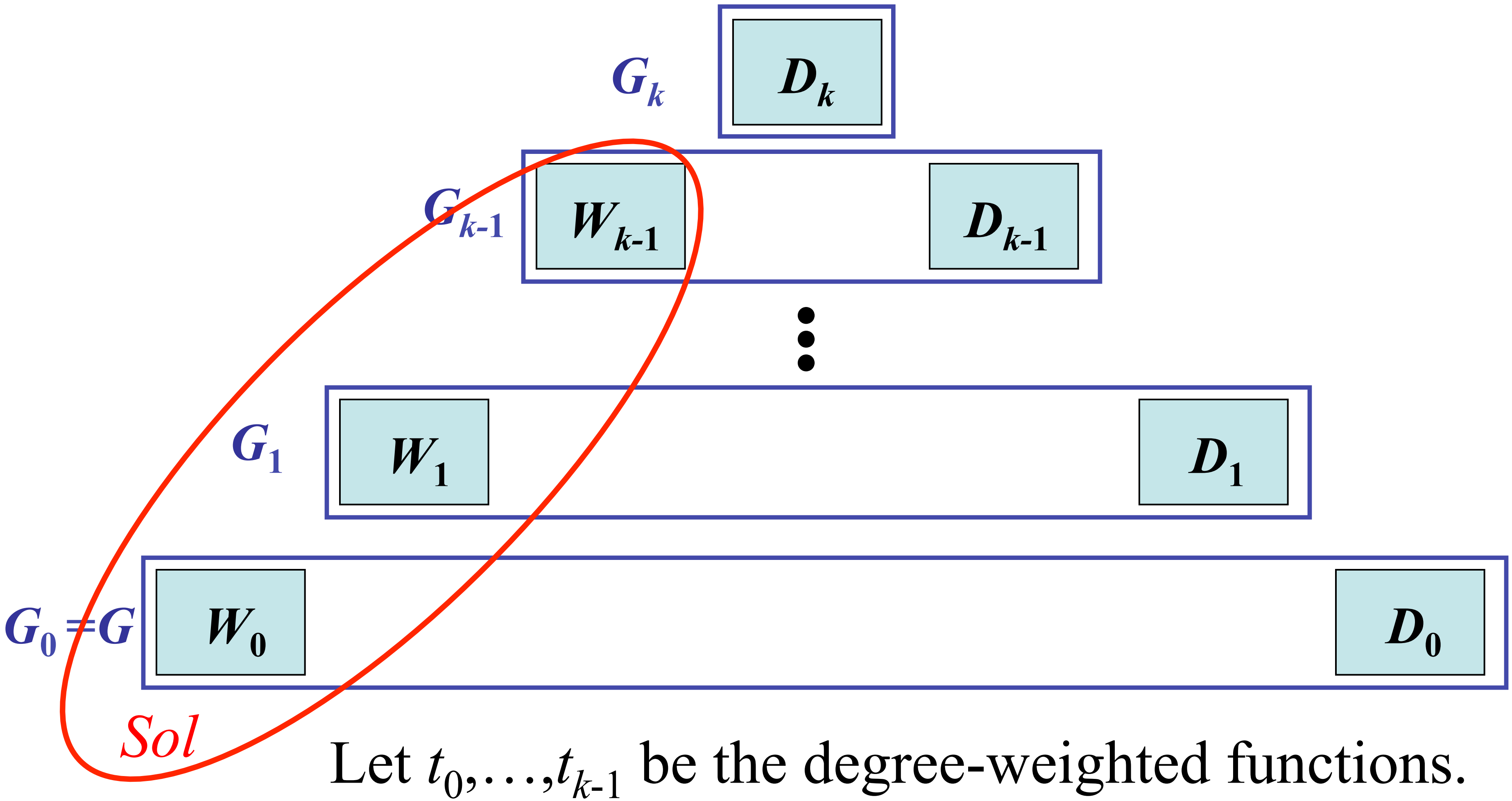


Approximation ratio of the Layer Algorithm

Theorem 2.4

The Layer Algorithm achieves an approximation guarantee of factor 2 for the vertex cover problem assuming arbitrary vertex weights.

Scheme of the Algorithm



Let $t_0,\ldots,t_{k\text{-}1}$ be the degree-weighted functions.

$C^* \cap G_i$ is a vertex cover for G_i

Proof of Theorem 2.4 (1)

- We need to show that set *Sol* is a vertex cover for G and $w(Sol) \leq 2$ OPT.
- Assume, for contradiction, that *Sol* is not a vertex cover for G. Then there must be an edge (u,v) with $u \in D_i$ and $v \in D_j$, for some i, j. Assume $i \leq j$. Therefore, (u,v) is present in G_i, contradicting the fact that u is a degree zero vertex.

Proof of Theorem 2.4 (2)

- Let C^* be an optimal vertex cover.
- Consider a vertex $v \in Sol$. If $v \in W_j$, its weight can be decomposed as $$w(v) = \sum_{i \le j} t_i(v).$$
- Consider a vertex $v \in V - Sol$. If $v \in D_j$, its weight can be decomposed as

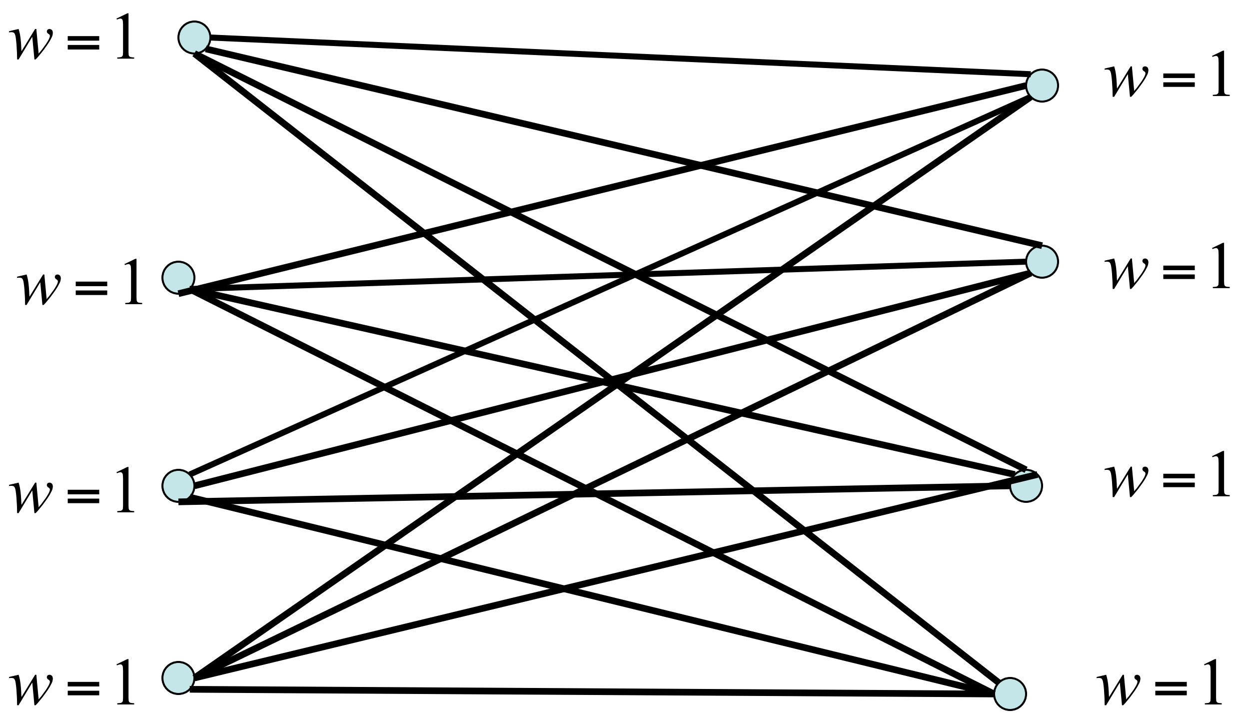
$$w(v) \ge \sum_{i < j} t_i(v).$$

Proof of Theorem 2.4 (3)

- $C^* \cap G_i$ is a vertex cover for G_i.
- Lemma 2.3 $\Rightarrow t_i(Sol \cap G_i) \leq 2\ t_i(C^* \cap G_i)$.
- By the decomposition of weights, we get

$$w(Sol) = \sum_{t=0}^{k-1} t_i\left(Sol \cap G_i\right) \leq 2\sum_{t=0}^{k-1} t_i\left(C^* \cap G_i\right) \leq 2w(C^*).$$

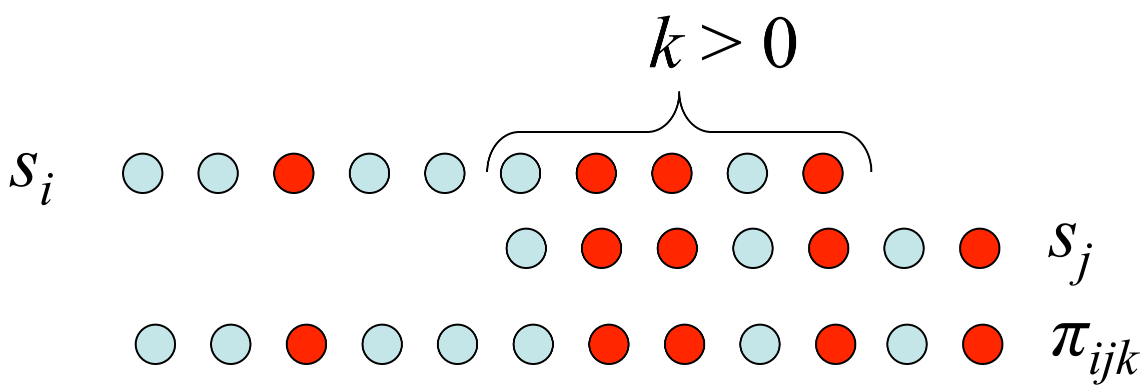
Tight example



Shortest Superstring

- *Given* a finite alphabet Σ, and a set of n strings $S = \{s_1, \ldots, s_n\} \subseteq \Sigma^+$.
- *Find* a shortest string s that contains each s_i as a superstring.
- Without lost of generality, we may assume that no string s_i is a substring of another string s_j, $i \neq j$.

Shortest Superstring as Set Cover



$M = \{\pi_{ijk} \mid \pi_{ijk}$ is a valid choice of $i, j, k\}$

$\pi \in M:\quad \text{set}(\pi) = \{s \in S \mid s$ is a substring of $\pi\}$

$U_{\text{cover}} \equiv S_{\text{string}}$

$S_{\text{cover}} \equiv \{\text{set}(\pi_{ijk}) \mid \pi_{ijk}$ is a valid choice of $i, j, k\}$

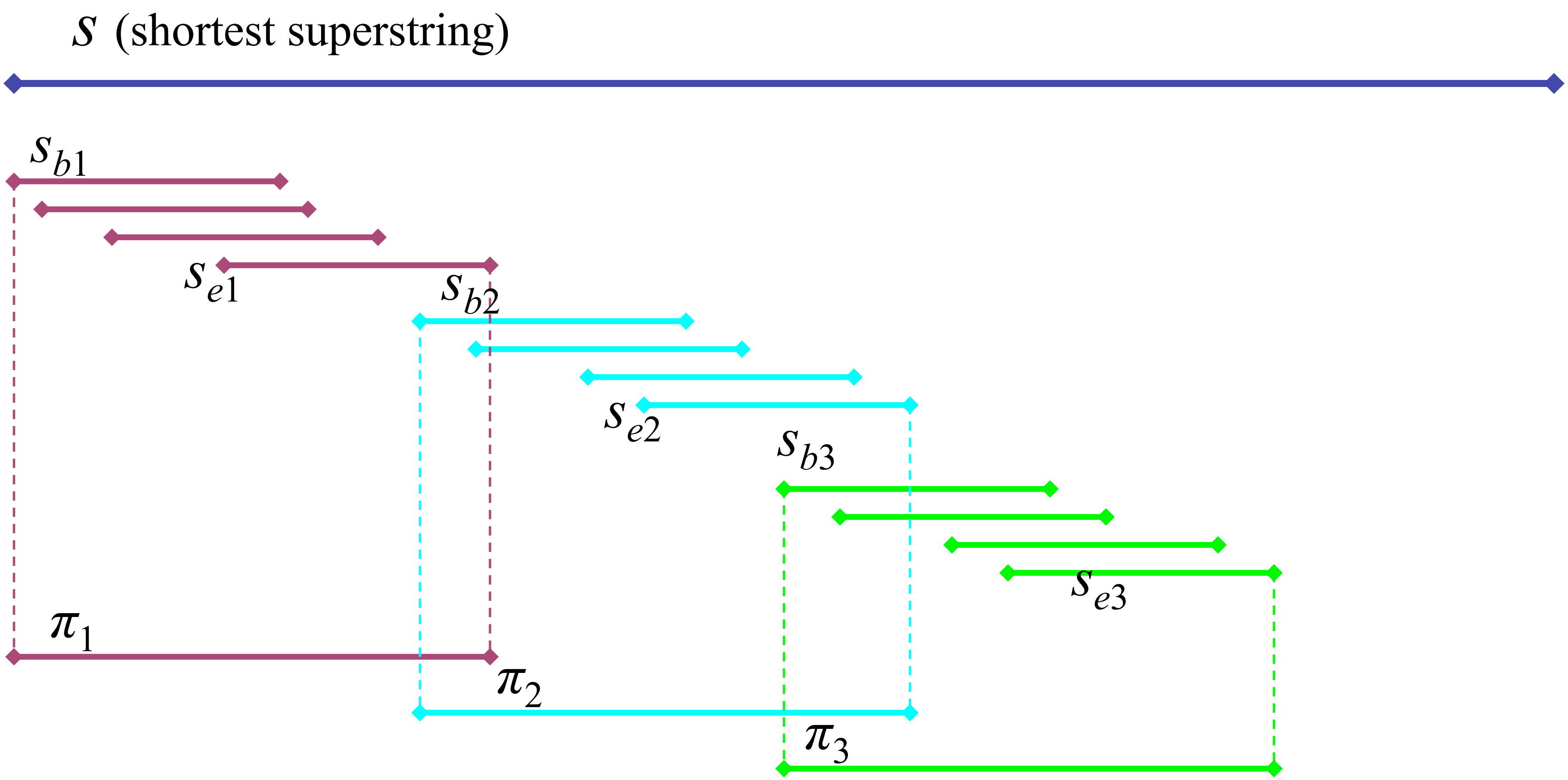
$c(\text{set}(\pi)) = |\,\pi\,|$

Lower bound

- **Lemma 2.5**

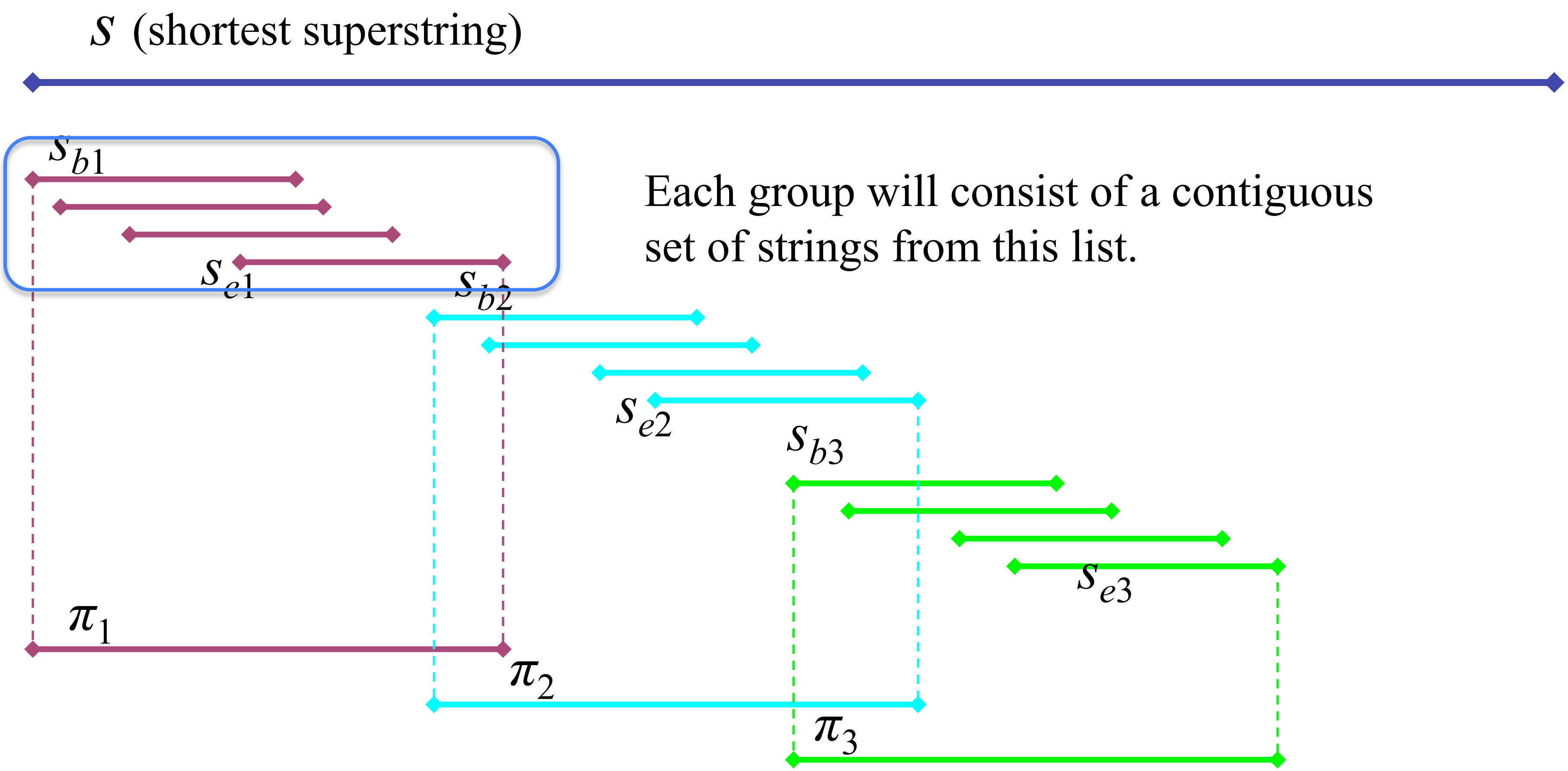
$$OPT_{string} \leq OPT_{cover} \leq 2\ OPT_{string}$$

$\text{OPT}_{\text{cover}} \leq 2\ \text{OPT}_{\text{string}}$



Consider the leftmost occurrence of the strings $s_1, \ldots, s_n$ in string s.
We will partition the ordered list of strings $s_1, \ldots, s_n$ in groups.

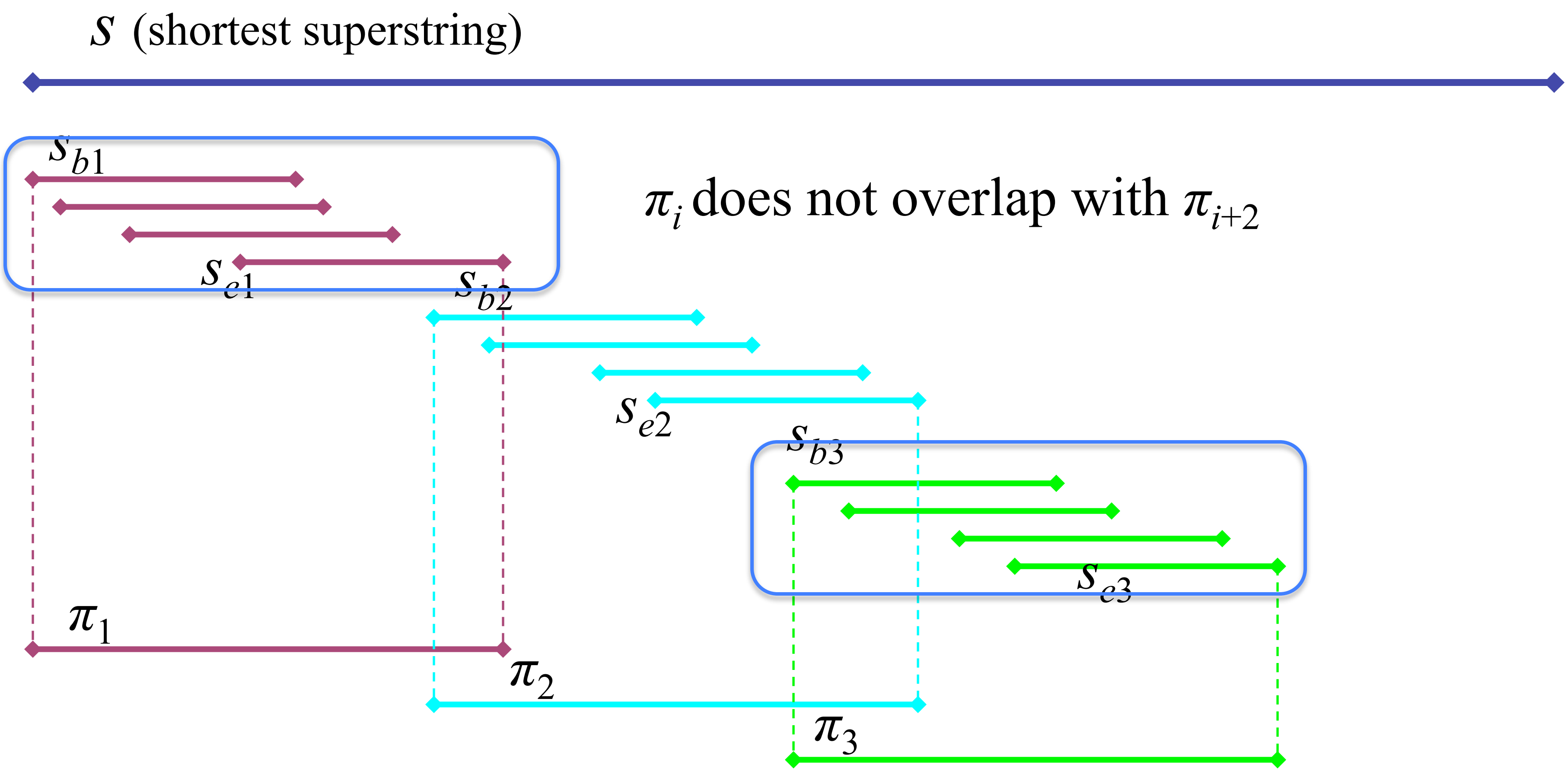
$OPT_{cover} \leq 2\ OPT_{string}$



Each group will consist of a 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_1=1, and e_1 is the largest index of a string that overlaps with s_1.

$\mathrm{OPT}_{\mathrm{cover}} \leq 2\ \mathrm{OPT}_{\mathrm{string}}$



π_i does not overlap with π_{i+2}

$\{\mathrm{set}(\pi_i) | i=1,\ldots,t\}$ is a solution for S, with cost $\sum_i \pi_i$.

Li's Algorithm

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

Approximation ratio of

Theorem 2.6

Li's algorithm is a $2H_n$ factor algorithm for the shortest superstring problem, where n is the number of strings in the given instance.

Exercises

The bin packing problem with bounded number of items per a bin.

- *Given* n items and their sizes $a_1,\ldots,a_n \in (0,1]$.
- *Find* a packing in unit-sized bins that minimizes the number of bin used under condition that each bin contains at most five items.

1. Reduce the above bin packing problem to the set cover problem.
2. Does your reduction polynomially depend on n?