Introduction to
Approximation Algorithms

Mong-Jen Kao (高孟駿)

Friday 13:20 – 15:10

Outline

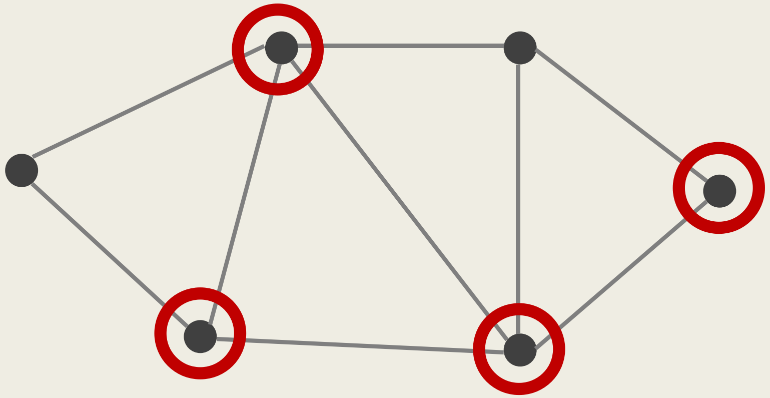
- The Vertex Cover Problem

- The (Weighted) Vertex Cover Problem
  - An 2-approximation by the “Layering” Technique

The Vertex Cover Problem

The (Cardinality) Vertex Cover Problem

- Given a graph $G = (V, E)$,
  compute a <u>*minimum-size*</u> *vertex subset* $U \subseteq V$ such that,
  for any edge $e \in E$, at least one endpoint of e is in U.



Intuitively, we are covering the edges using the vertices.

Status of the Vertex Cover Problem

- The vertex cover problem is a well-known ***NP-complete*** problem.
  - It is a <u>*benchmark problem*</u> used in many fields for testing the performance of all sorts of techniques.

- The vertex cover problem can be ***approximated*** to a ratio of 2.
  - For hypergraphs, f-approximation can be obtained, where f is the maximum size of the hyperedges.

Let’s see how this can be done!

Status of the Vertex Cover Problem

In terms of ***approximation hardness***,

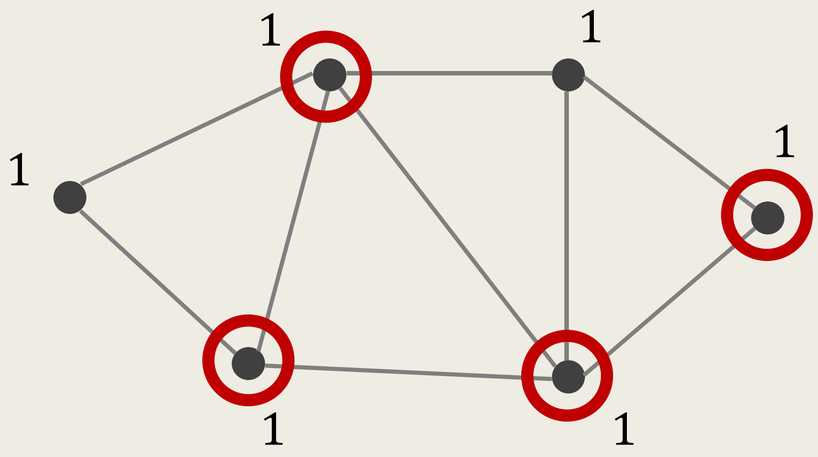
- It is NP-hard to obtain a $\left(\sqrt{2} - \epsilon\right)$-approximation, for any $\epsilon > 0$, unless P=NP.

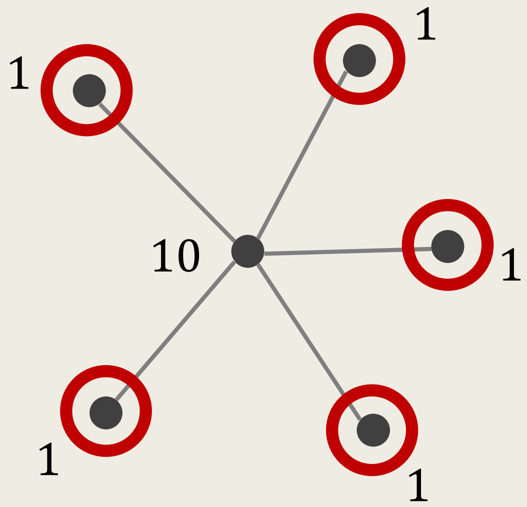
- If we assume the *Unique Game Conjecture (UGC)*, then $(2 - \epsilon)$-approximation is also NP-hard to obtain, for any $\epsilon > 0$.
  - The lower bound generalizes to $f - \epsilon$ for hypergraphs.

The (Weighted) Vertex Cover Problem

The Vertex Cover Problem

- Given a graph $G = (V, E)$ and ***a vertex weight function*** $w : V \to Q^+$, compute a minimum-weight vertex subset $U \subseteq V$ such that, for any edge $e \in E$, at least one endpoint of e is in U.





2-Approximation

by the “Layering” Technique

The Layering Technique for Vertex Cover

- We introduce a clever way to deal with the vertex cover problem.
  - The approximation ratio we obtain here is 2.
  - It can be generalized to **hypergraphs** to yield an f-approximation, where f is the maximum size of the hyperedges.

- The idea is to decompose the input instance, including the ***graph*** and the ***weight function***, in a way such that, the total weight in each layer is well-bounded.

Outline

- Degree-Weighted Functions
- Sketch of the Algorithm
- Algorithm Description & Analysis

Degree-Weighted Function

- We say that a weight function $w : V \to Q^+$ is ***degree-weighted***,
  if there is some $c > 0$ such that $w(v) = c \cdot \deg(v)$ holds for all $v \in V$.
  - i.e., the **vertex weights are proportional to their degrees**.

- The following lemma is intrinsic to the cover problems.

Lemma 2.

Let w be a degree-weighted function of the vertices.

Then, $\boldsymbol{w(V) \leq 2 \cdot w(U)}$ holds ***for any feasible vertex cover*** $\boldsymbol{U}$ for G.

For degree-weighted functions,
the total weight is not too large compared to any VC!

Lemma 2.

Let w' be a degree-weighted function of the vertices.

Then, $\boldsymbol{w'(V) \leq 2 \cdot w'(U)}$ holds ***for any feasible vertex cover*** $\boldsymbol{U}$ for G.

Proof.

Since U is a feasible vertex cover, it covers all the edges in E, and

$$w'(U) = \sum_{v \in U} w'(v) = \sum_{v \in U} c \cdot \deg(v) \geq c \cdot |E| .$$

Since U is a vertex cover, each edge is counted at least once.

On the other hand,

$$w'(V) = \sum_{v \in V} c \cdot \deg(v) = 2 \cdot c \cdot |E| \leq 2 \cdot w'(U) .$$

Each edge is counted exactly twice.

By the above inequality.

The Layering Algorithm

- By Lemma 2, when the vertices are degree-weighted, the weight of any feasible vertex cover cannot be too small.
  - Even taking all the vertices isn't too bad compared to OPT.

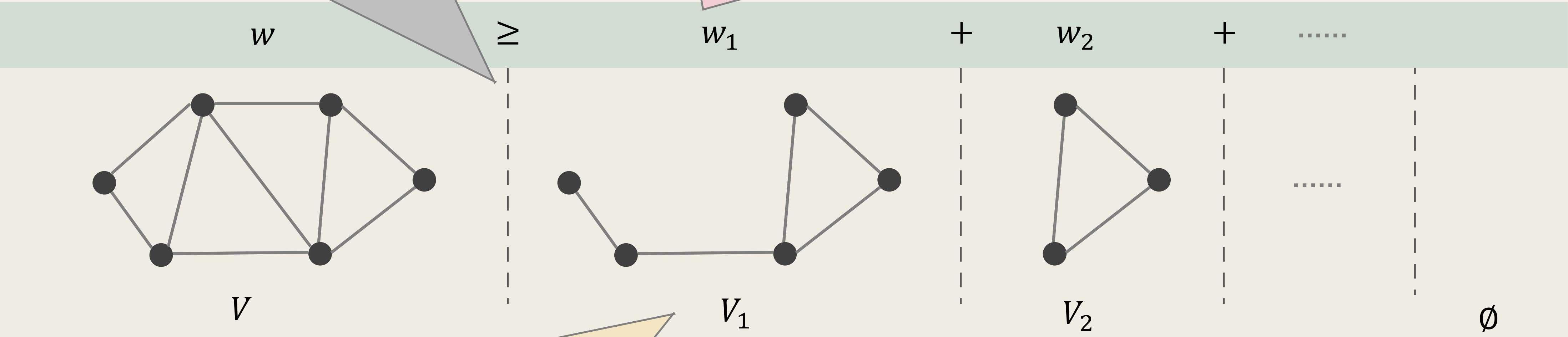
- The idea of the layering algorithm is to ***greedily decompose the weight function*** into ***a sequence of degree-weighted functions***.
  - In each iteration, a degree-weighted function is formed, and the weight of each vertex decreases correspondingly.

- In particular, we will
  - ***Decompose the weight function w* into $w_1, w_2, \ldots, w_k$**, and possibly some left-over weights, and
  - Form a nesting sequence of vertex subsets $V \supseteq V_1 \supseteq V_2 \supseteq \cdots \supseteq V_k$

Between layers, some vertices are removed from consideration.

We make w_i proportional to vertex degrees in V_i.

A **valid vertex cover** will be **formed during this process.**



For each layer,
the induced subgraph of V_i is considered.

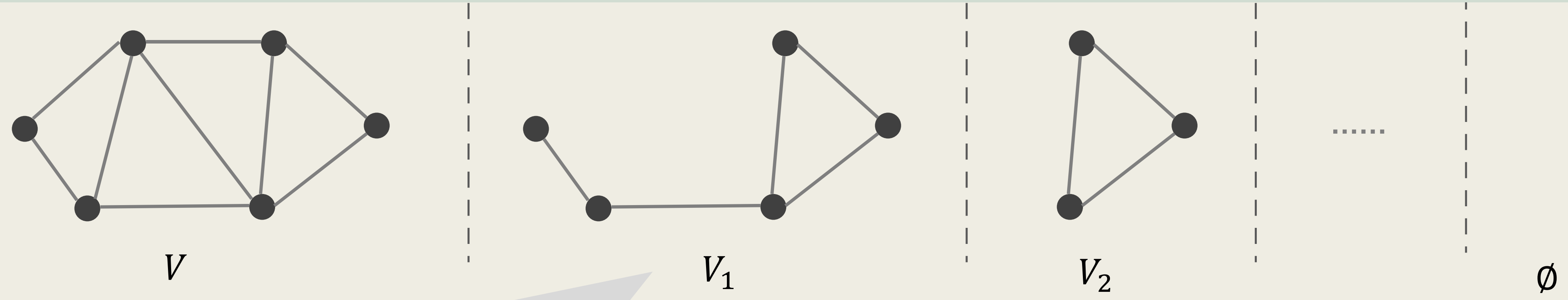
- The key in bounding the overall cost is to guarantee that, the weight function in each layer is degree-weighted.

Between layers, some vertices are removed from consideration.

We make w_i proportional to vertex degrees in V_i.

A **valid vertex cover** will be **formed during this process**.

$$w \geq w_1 + w_2 + \cdots\cdots$$



For each layer, **the induced subgraph of V_i** is considered.

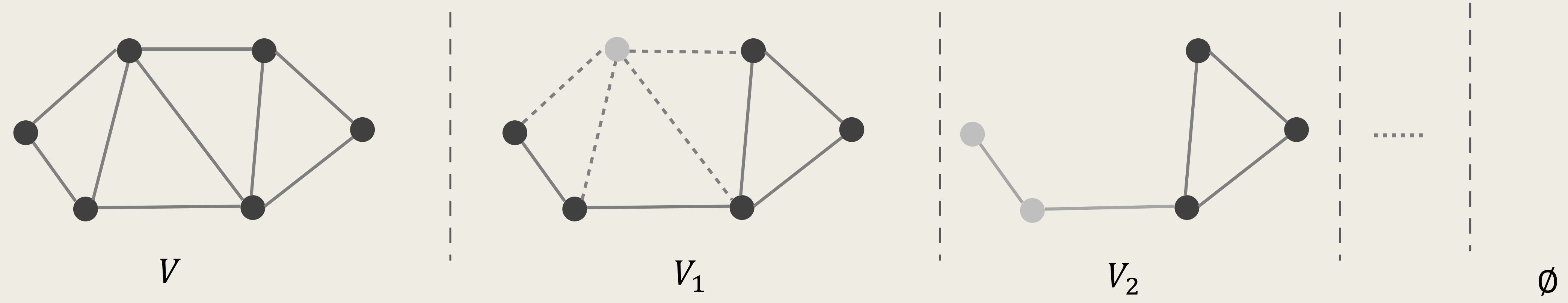
We will guarantee that, the total weight of V_i w.r.t. w_i is well-bounded!

The Layering Algorithm

- The idea of the layering algorithm is to ***greedily decompose the weight function*** into ***a sequence of degree-weighted functions***.
  - In each iteration, a degree-weighted function is formed, and the weight of each vertex decreases correspondingly.
  - When the weight of a vertex is fully-decomposed (becomes zero), the algorithm selects it into the solution set.
  - After each iteration, vertices with *zero weight* or *zero degree* are removed from consideration.

- In particular, we will
  - ***Decompose the weight function*** $\boldsymbol{w}$ **into** $\boldsymbol{w_1, w_2, \ldots, w_k}$, and possibly some left-over weights, and
  - Form a nesting sequence of vertex subsets $\boldsymbol{V \supseteq V_1 \supseteq V_2 \supseteq \cdots \supseteq V_k}$.

$$w \geq w_1 + w_2 + \cdots\cdots$$



Vertices whose weight are fully-decomposed are included in the solution and removed from consideration.

A valid vertex cover will be formed during this process.

The Algorithm Description

The Layering Algorithm

- Let $G = (V, E)$ and $w : V \to \mathbb{Q}^{+}$ be the input instance of the vertex cover problem.

- During the execution,
  the algorithm maintains the following information.
  - w' : The residual weight function left to be decomposed.
  - V' : The set of remaining vertices in G.

- Initially, $w' \coloneqq w$ and $V' \coloneqq V$.

The Layering Algorithm

- Let w' denote the residual weight function left to be decomposed and V' the set of remaining vertices.

- In each iteration,
  the algorithm does the following until $V' = \emptyset$.

  1. Let $c \coloneqq \min_{v \in V'} w'(v)/\deg(v)$.
  2. Decreases $w'(v)$ by $c \cdot \deg(v)$ for all $v \in V'$.
     - <u>**Selects**</u> vertices with ***zero residual weight*** <u>**into the solution**</u>, and
     - Remove zero-weight vertices and then zero-degree vertices from V'.

Intuitively, $c \cdot \deg(v)$ is the <u>***largest degree-weighted function that can be defined***</u> for vertices in V'.

The Layering Algorithm

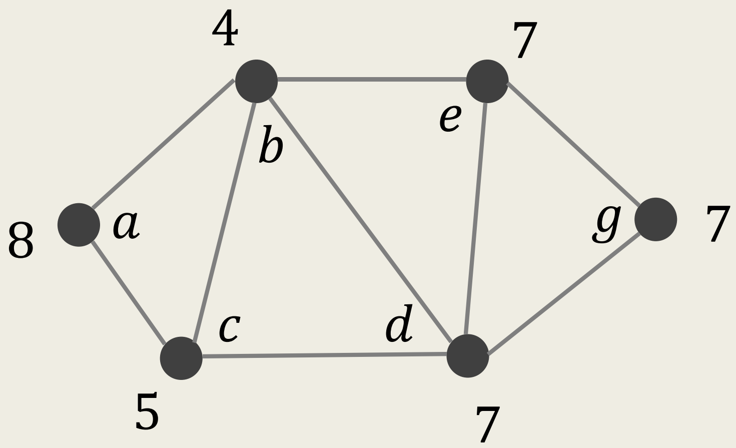
1. Let $w' \leftarrow w$, $V' \leftarrow V$, and $C \leftarrow \emptyset$.
2. For $i = 1,2, \dots,$ do until V' becomes empty
   - Let G_i be the subgraph induced by V' and $\deg_i$ be the degree function of G_i. Remove all v with $\deg_i(v) = 0$ from V'.
   - Let $c_i = \min_{v \in V'} w'(v) / \deg_i(v)$. Set $w'(v) \leftarrow w'(v) - c_i \cdot \deg_i(v)$ for all $v \in V'$.
   - Let W_i be the zero-weight vertices in V'. Set $C \leftarrow C \cup W_i$ and remove W_i from V'.
3. Output C as the approximation solution.

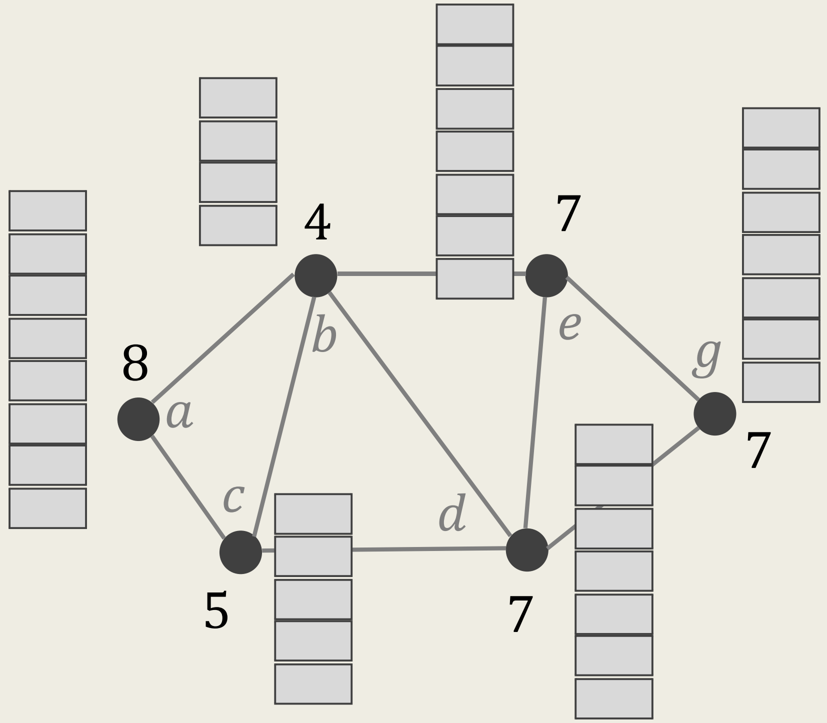
Define the function $w_i(v) \coloneqq c_i \cdot \deg_i(v)$.

Pick the fully-decomposed vertices.

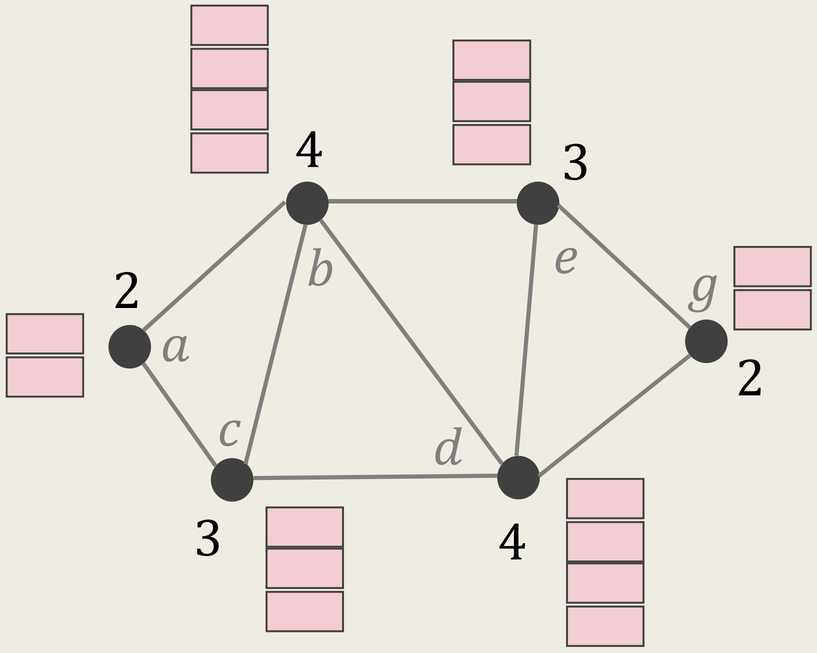
Example

- Consider the following example.

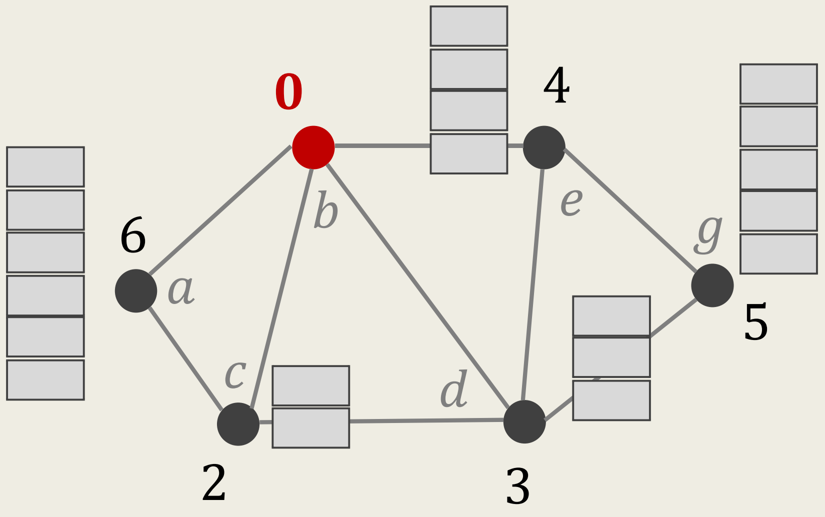


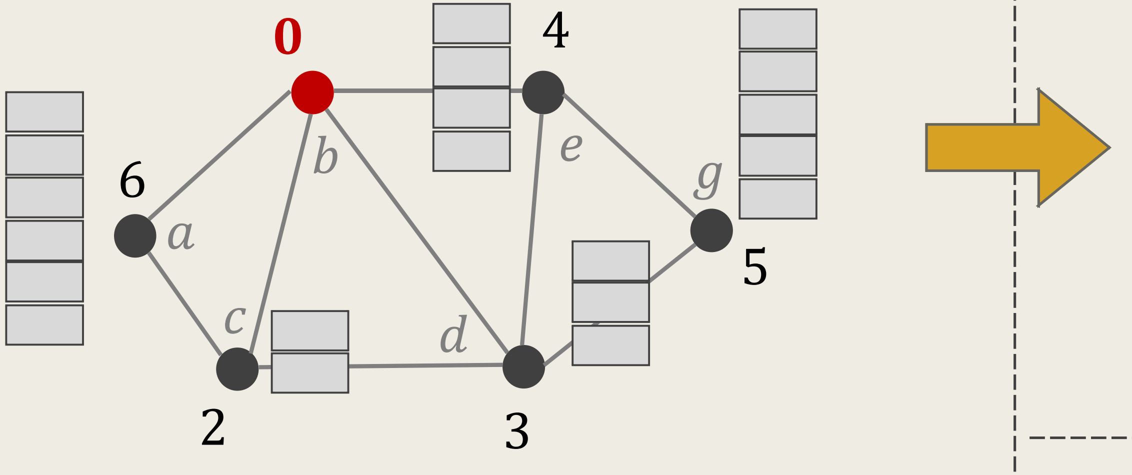


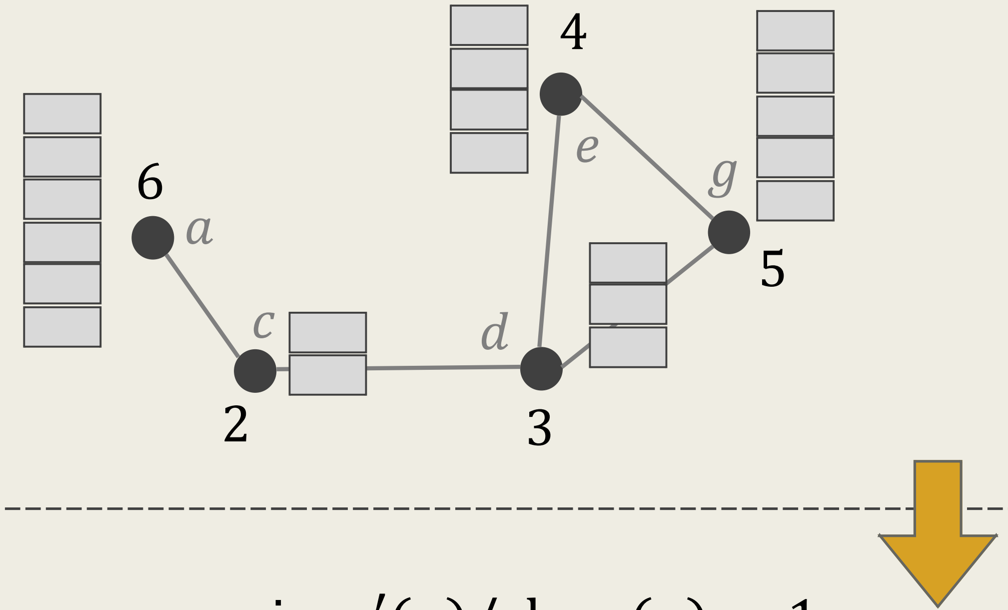
$$c_1 = \min_{v \in V'} w'(v) / \deg_1(v) = 1.$$



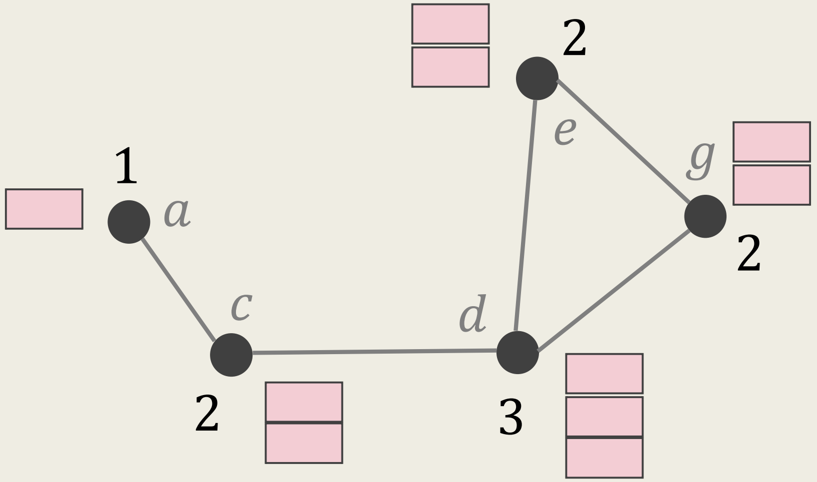
$$w_1 = c_1 \cdot \deg_1(v).$$



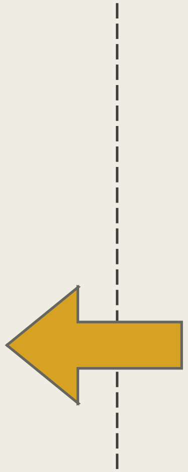


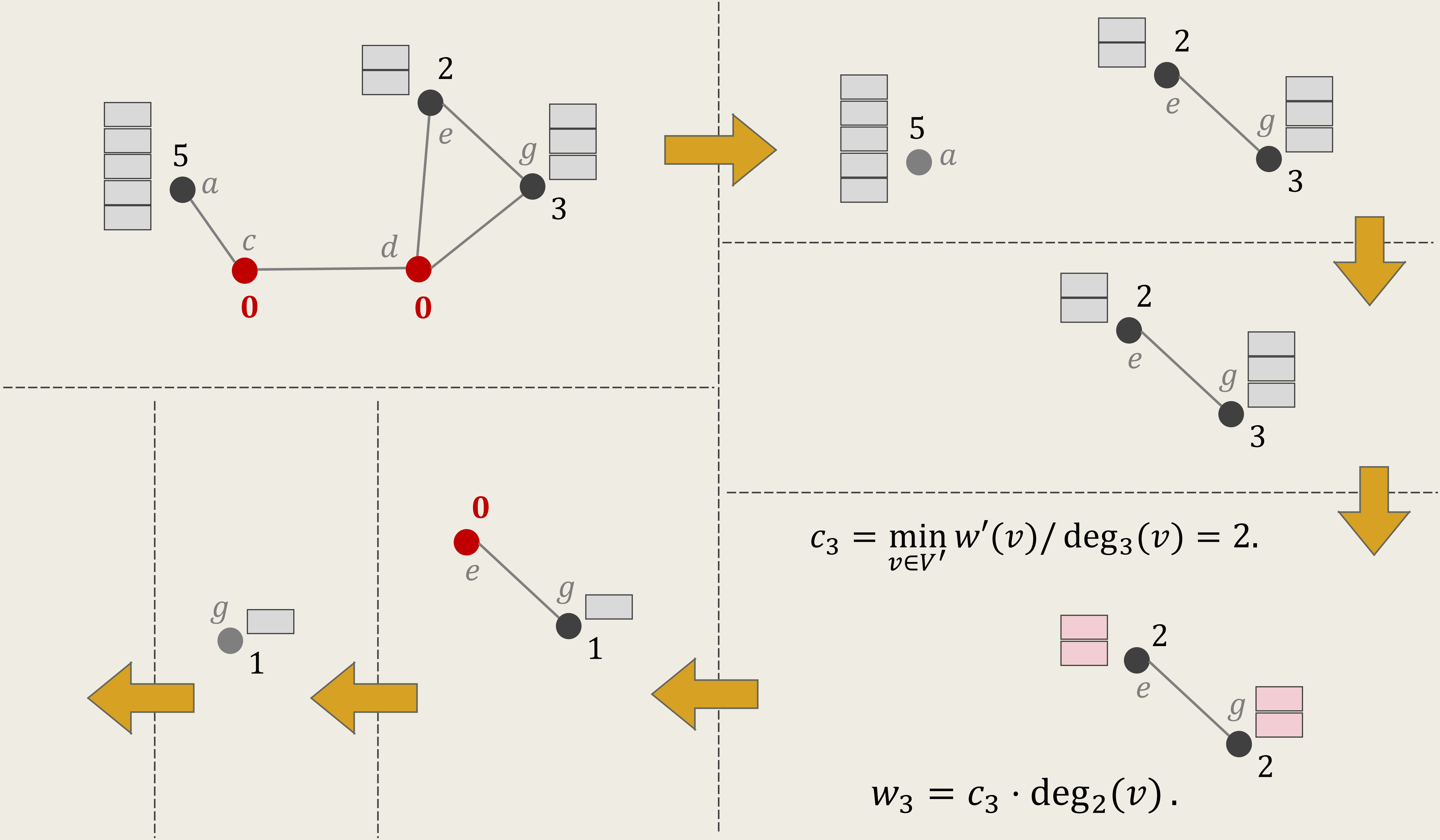


$$c_2 = \min_{v \in V'} w'(v) / \deg_2(v) = 1.$$



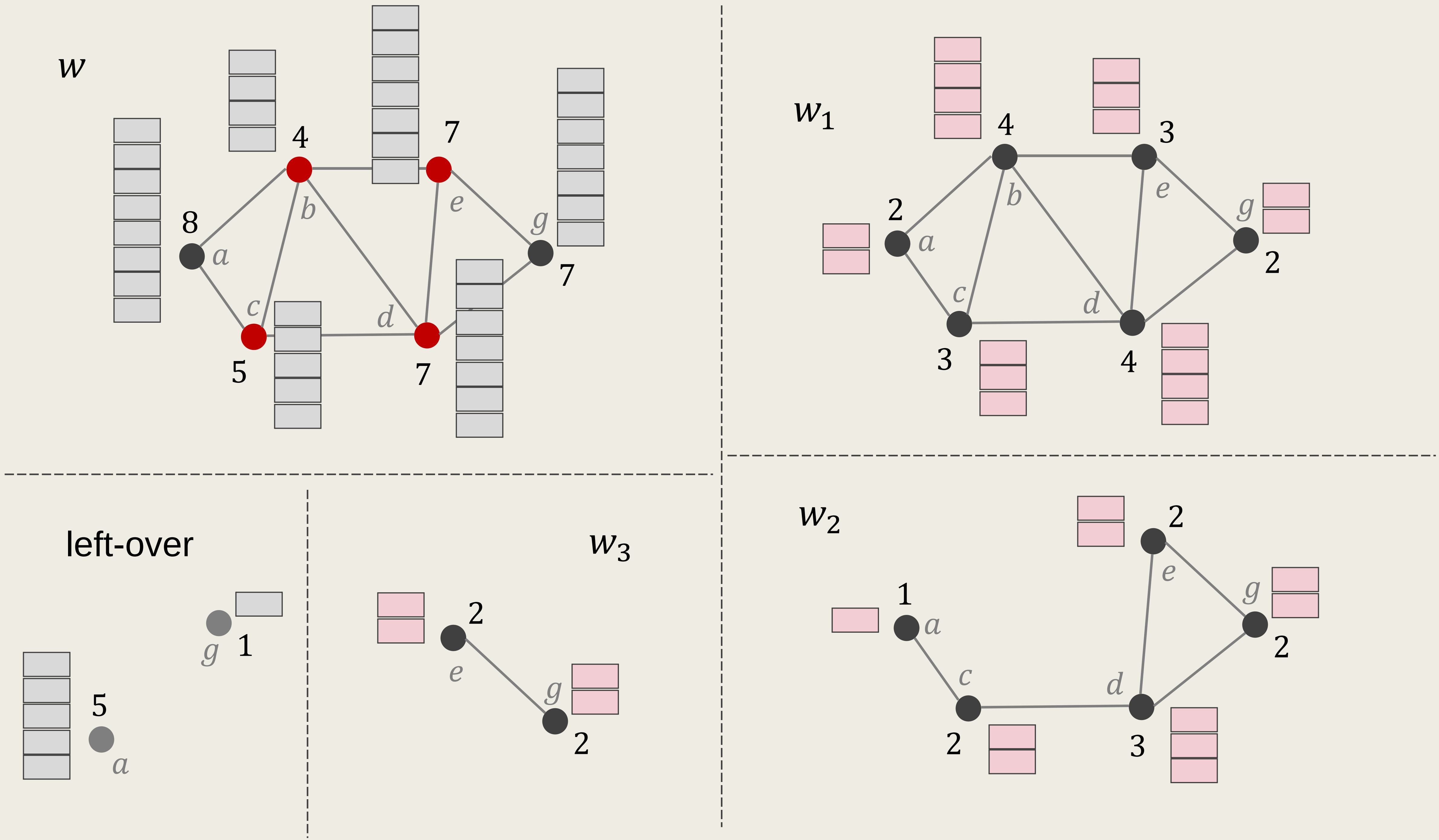
$$w_2 = c_2 \cdot \deg_2(v).$$





$$c_3 = \min_{v \in V'} w'(v) / \deg_3(v) = 2.$$

$$w_3 = c_3 \cdot \deg_2(v) \, .$$


w
8
4
7
7
5
7
a
b
c
d
e
g
w_1
2
4
3
2
3
4
a
b
c
d
e
g
left-over
5
a
1
g
w_3
2
e
2
g
w_2
1
a
2
c
3
d
2
e
2
g

The Analysis of the Algorithm

- We will prove the following theorem.

Theorem 3.

The layering algorithm computes a 2-approximation for the vertex cover problem.

- We need to prove the following two statements.
  - (Feasibility) The algorithm terminates in polynomial time, and $\mathcal{C} \coloneqq \bigcup_{i \geq 1} W_i$ is a feasible vertex cover for G.
  - (Approximation Guarantee) $w(\mathcal{C}) \leq 2 \cdot w(\mathcal{C}^*)$, where $\mathcal{C}^*$ is an optimal vertex cover for G.

The Feasibility

- To see that the algorithm terminates in polynomial time, observe that,
  - In each iteration, at least one vertex becomes zero-weight, and is removed from V'.
  - Hence, the algorithm terminates in $O(|V|)$ iterations.

The Feasibility

- Next, we prove that $\mathcal{C} \coloneqq \bigcup_{i\geq 1} W_i$ is a feasible vertex cover for G.

- Observe that,
  when a vertex is removed from V', either $w'(v) = 0$ or $\deg_i(v) = 0$.

  – If $w'(v) = 0$, then v is selected into $\mathcal{C}$, and
    all its incident edges are covered.

  – If $\deg_i(v) = 0$, then all its incident edges have already been covered.

  Since $V' = \emptyset$ when the algorithm terminates, all the edges are covered.

The Approximation Guarantee

- Since $\mathcal{C}^*$ is a feasible cover for G, it is feasible for G_i for all $i \geq 1$.
- By the decomposition scheme and Lemma 2, we have

The vertices in $\mathcal{C}$ are fully-decomposed.

$$w(\mathcal{C}) = \sum_{v \in \mathcal{C}} w(v) = \sum_{v \in \mathcal{C}} \sum_{i \geq 1} w_i(v)$$

$$\leq \sum_{i \geq 1} w_i(V) \leq \sum_{i \geq 1} 2 \cdot w_i(\mathcal{C}^*) \leq 2 \cdot w(\mathcal{C}^*) .$$

By Lemma 2.