Kernelization Complexity of Solution Discovery Problems

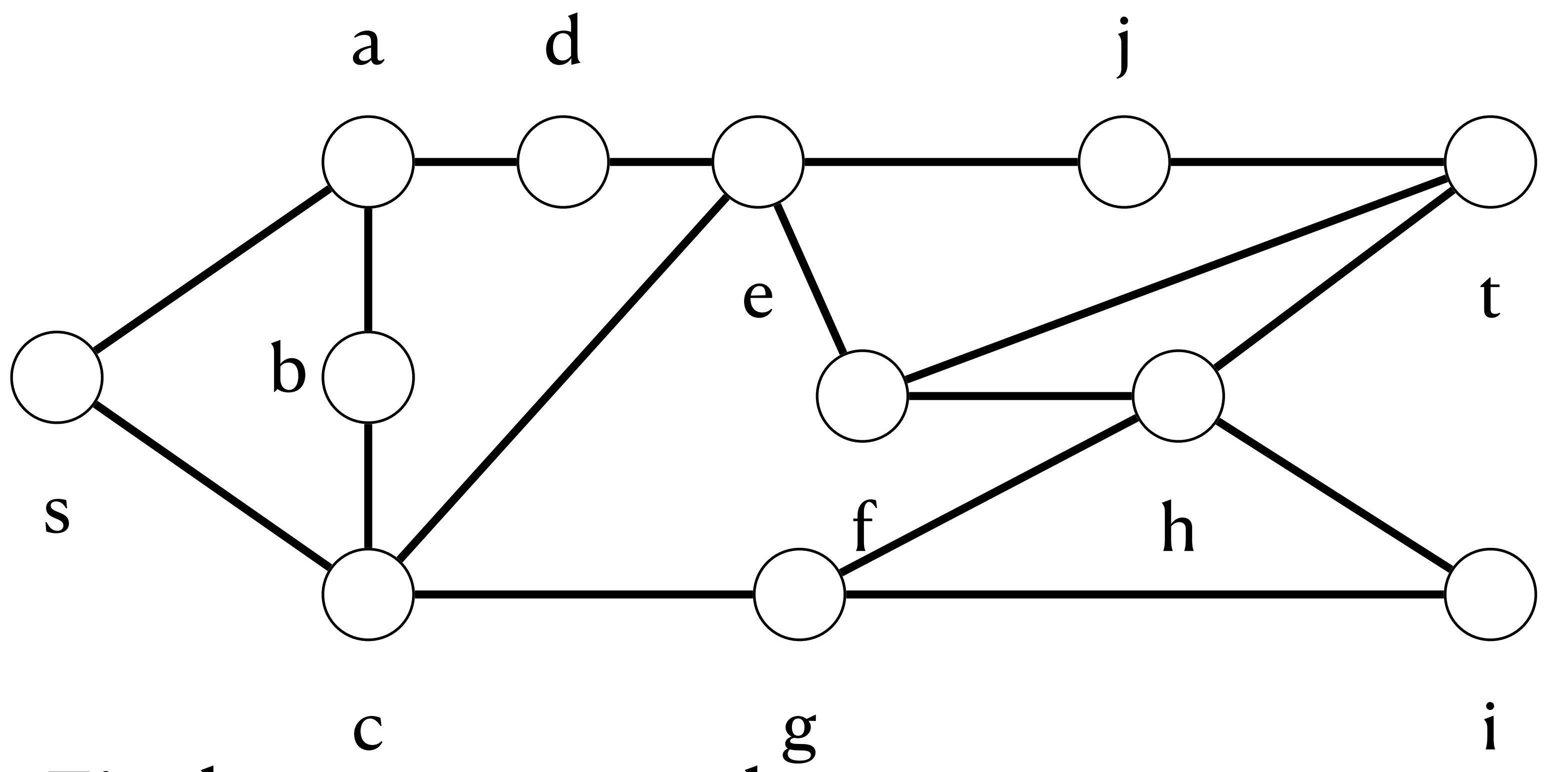
Mario Grobler[1], **Stephanie Maaz**[2], Amer E. Mouawad[3], Naomi Nishimura[2], Vijayaragunathan Ramamoorthi[1], Sebastian Siebertz[1]

[1]University of Bremen, Germany
[2]University of Waterloo, Canada
[3]American University of Beirut, Lebanon

December 11, 2024
ISAAC 2024
The 35th International Symposium on Algorithms and Computation

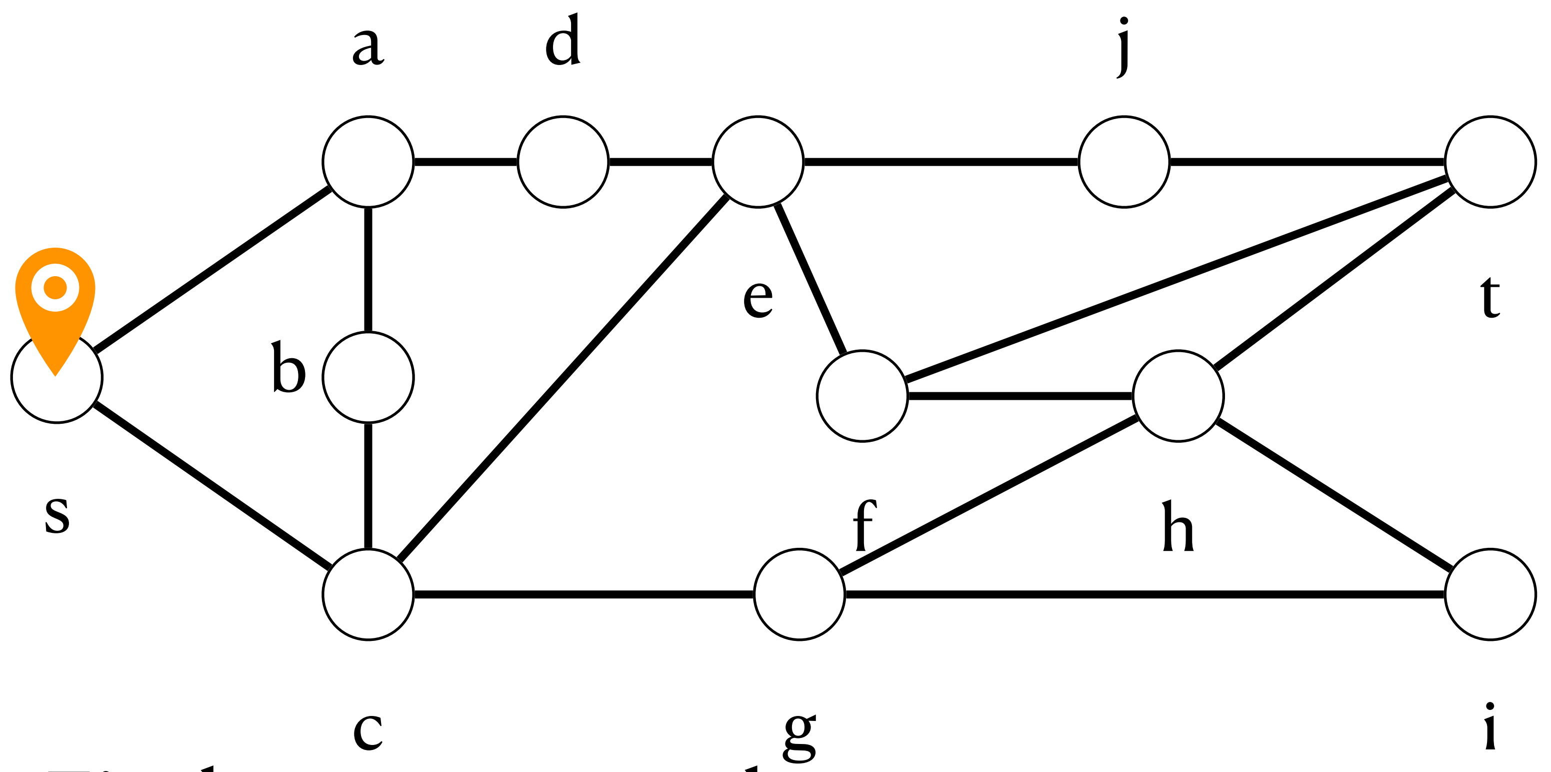
st-Vertex Cut Discovery Problem

st-Vertex Cut Discovery Problem



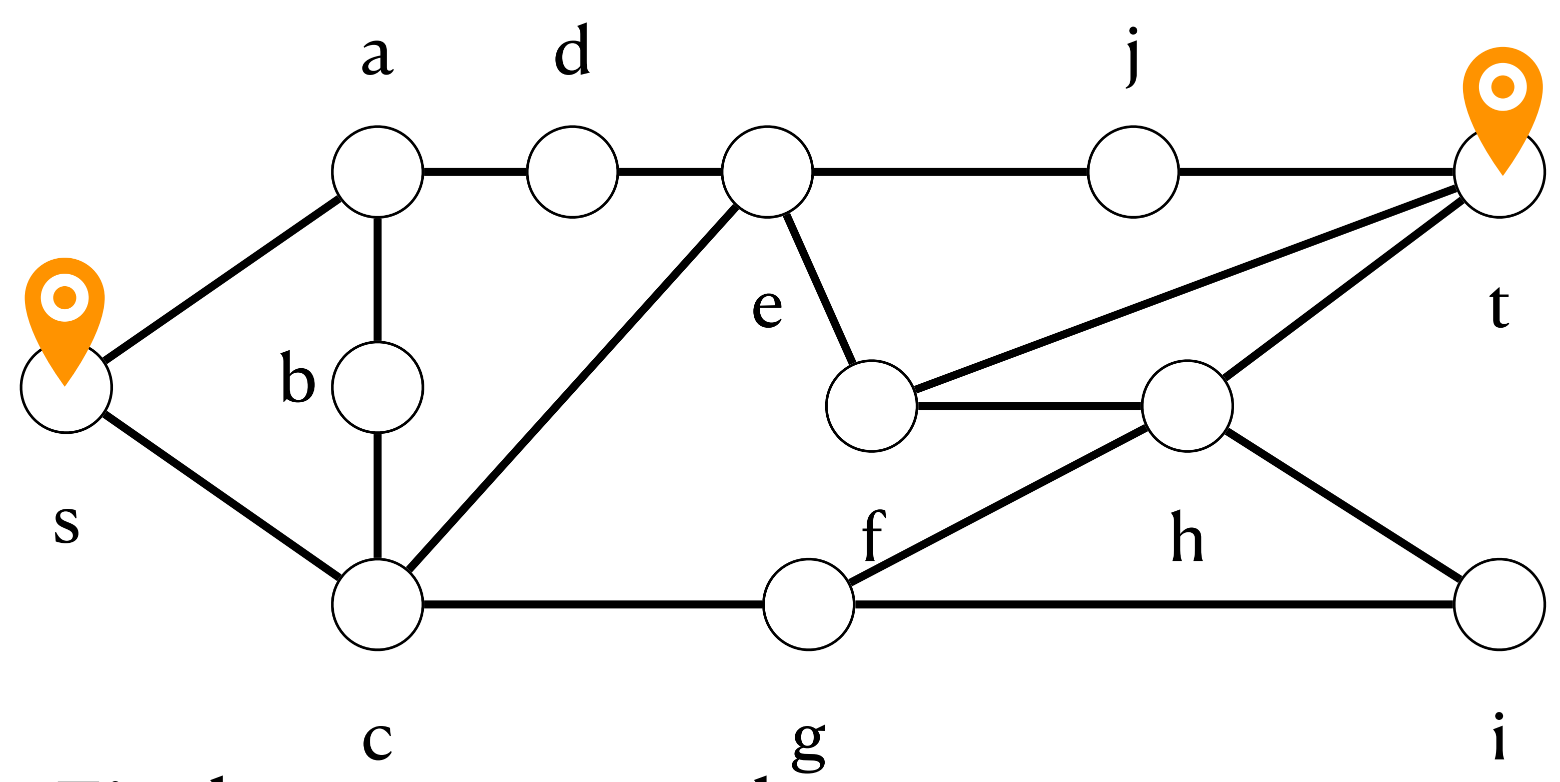
- **Classical st-Vertex Cut Problem**: Find a vertex cut that separates two vertices s and t.

st-Vertex Cut Discovery Problem



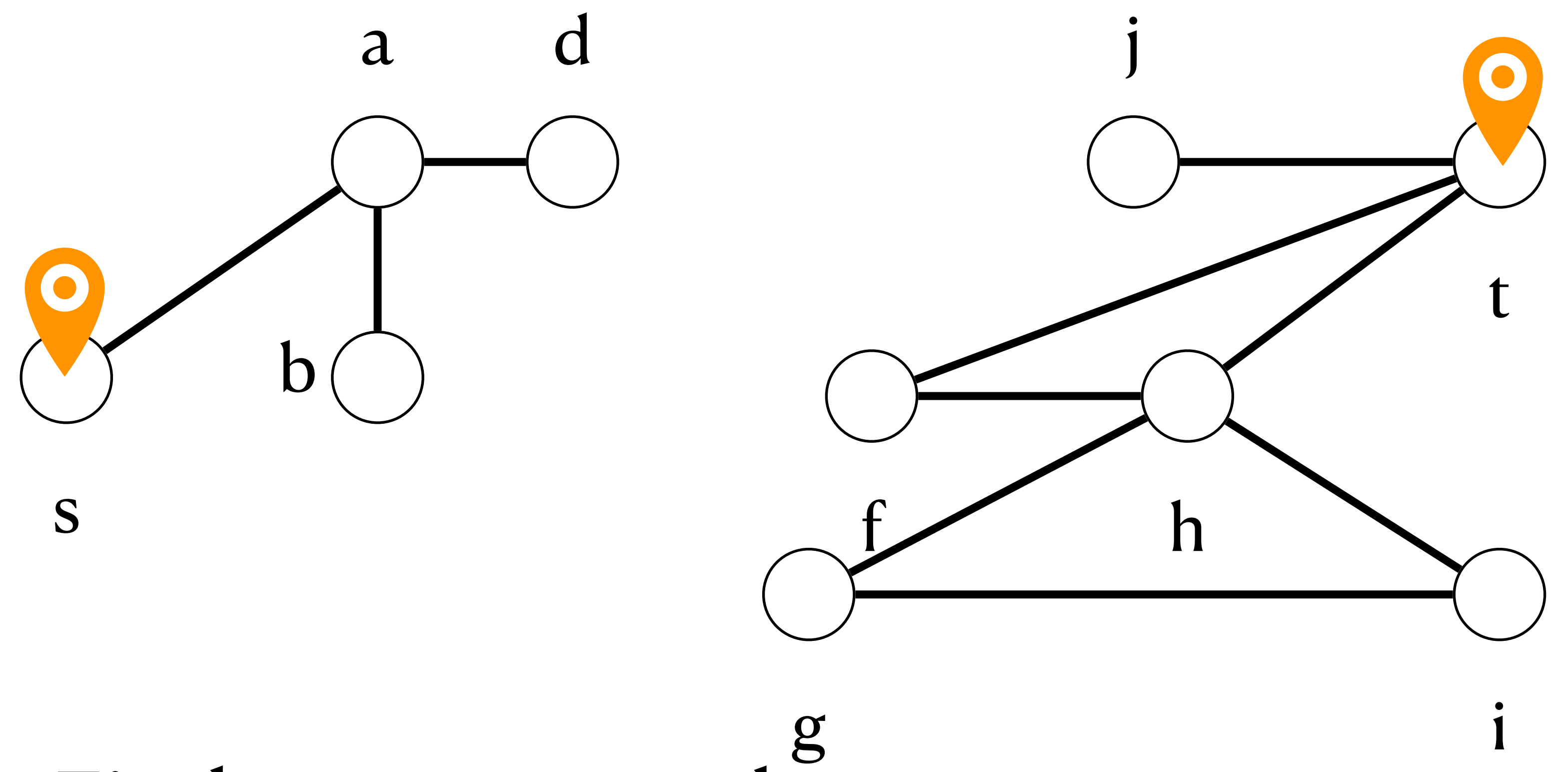
- **Classical st-Vertex Cut Problem**: Find a vertex cut that separates two vertices s and t.

st-Vertex Cut Discovery Problem



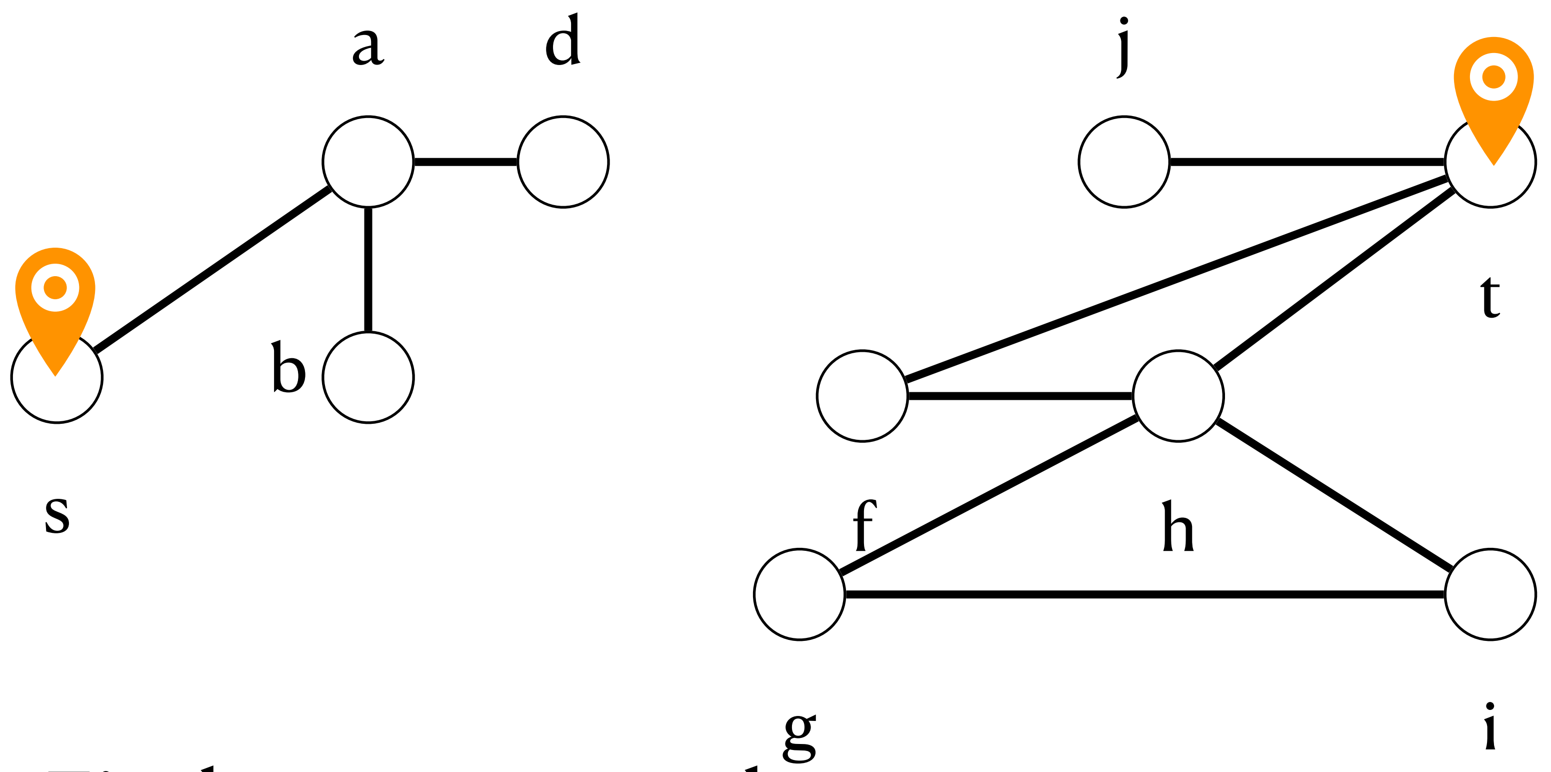
- **Classical st-Vertex Cut Problem**: Find a vertex cut that separates two vertices s and t.

st-Vertex Cut Discovery Problem



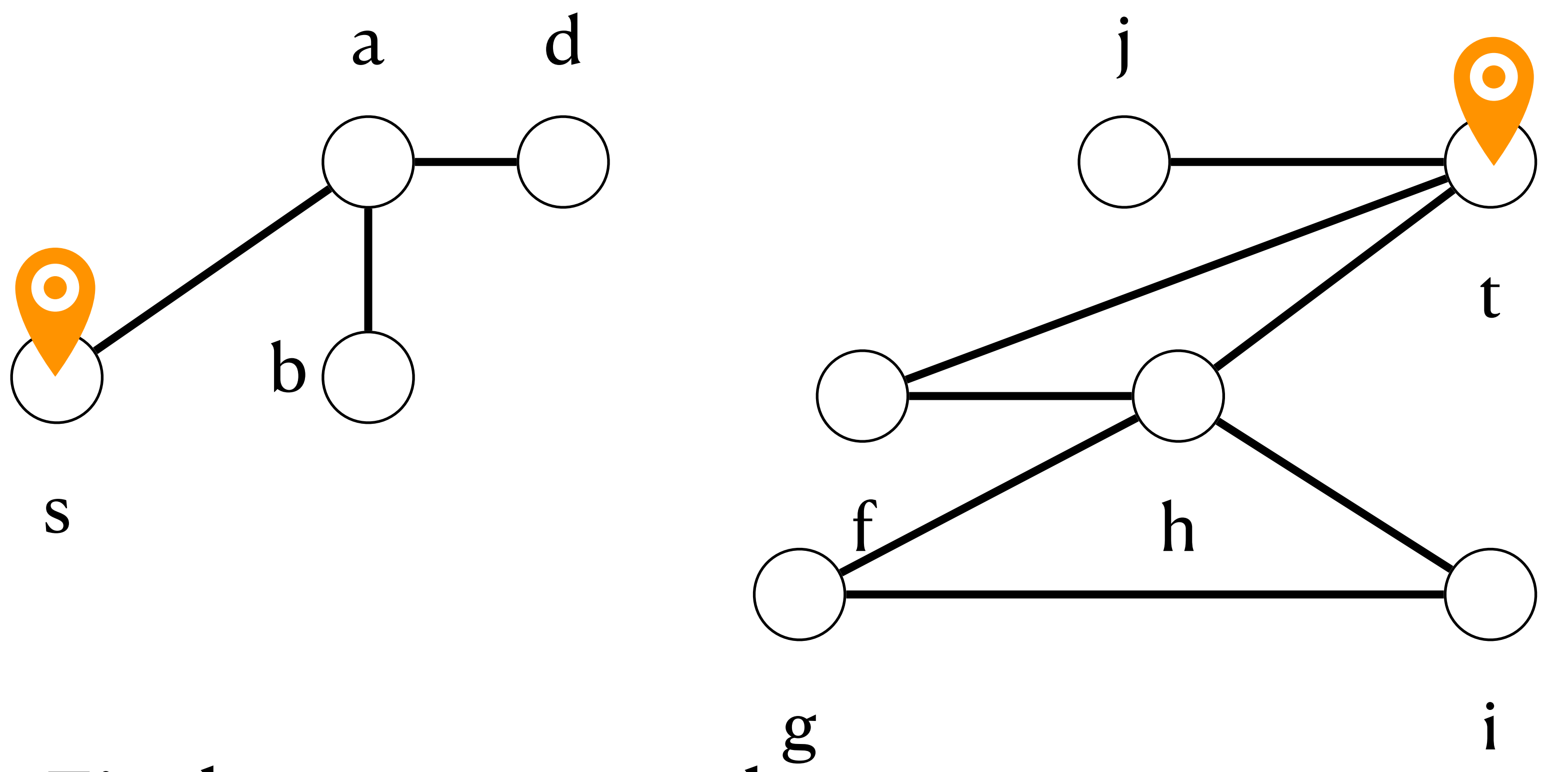
- **Classical st-Vertex Cut Problem**: Find a vertex cut that separates two vertices s and t.

st-Vertex Cut Discovery Problem

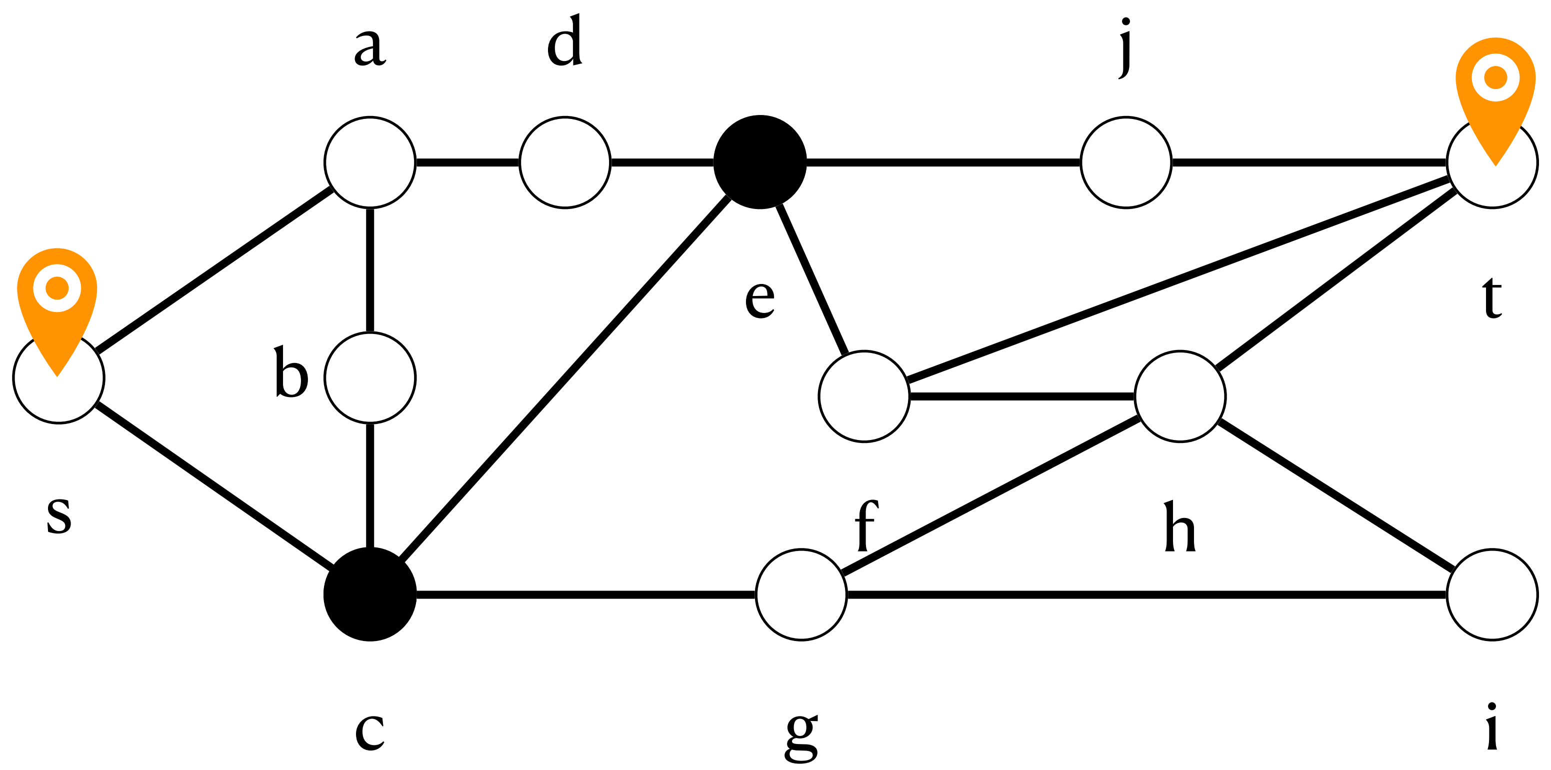


- **Classical st-Vertex Cut Problem**: Find a vertex cut that separates two vertices s and t.
- **Classical st-Vertex Cut Algorithm**: Start with a clean slate and achieve an optimal or feasible solution efficiently.

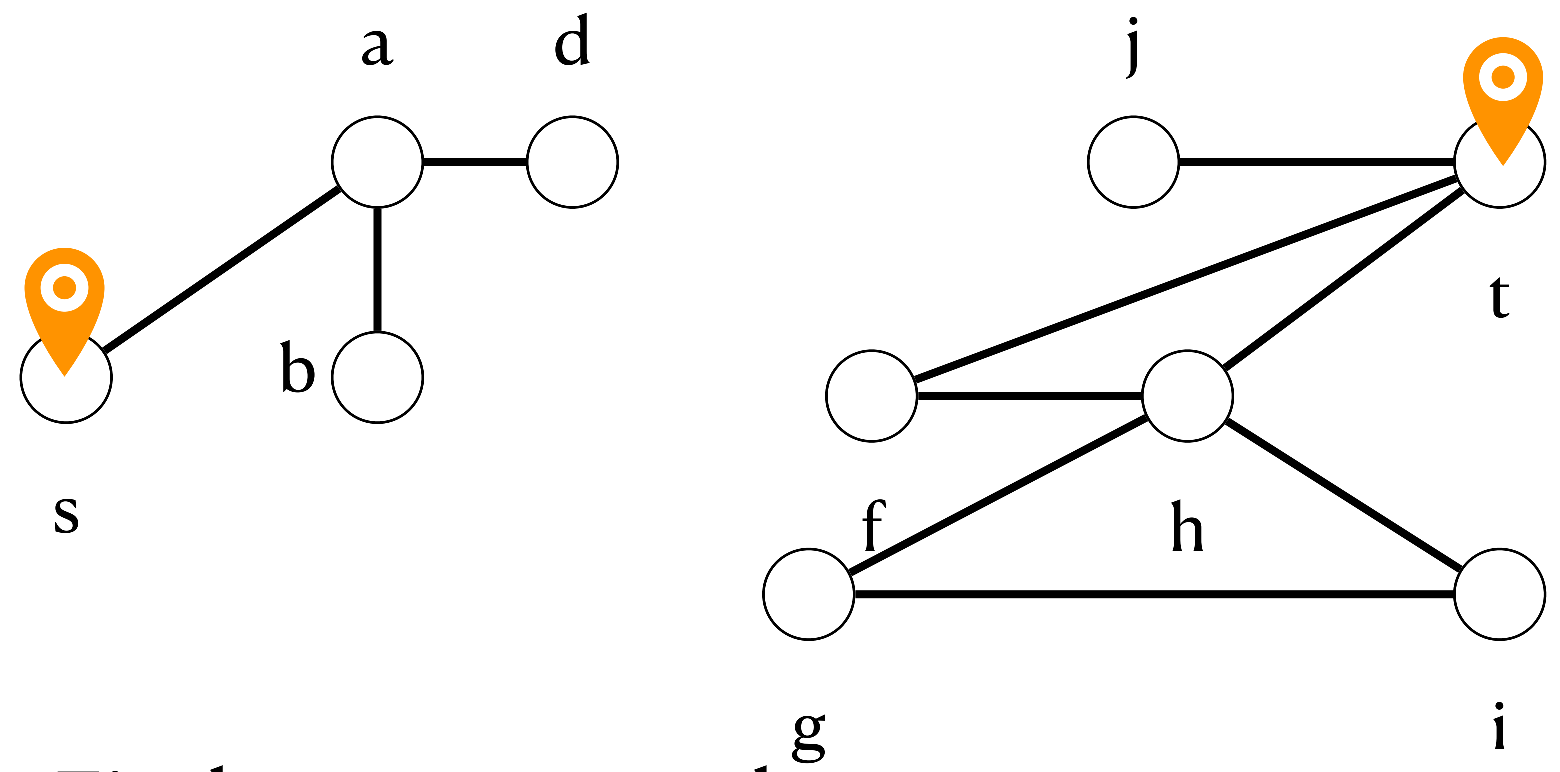
st-Vertex Cut Discovery Problem



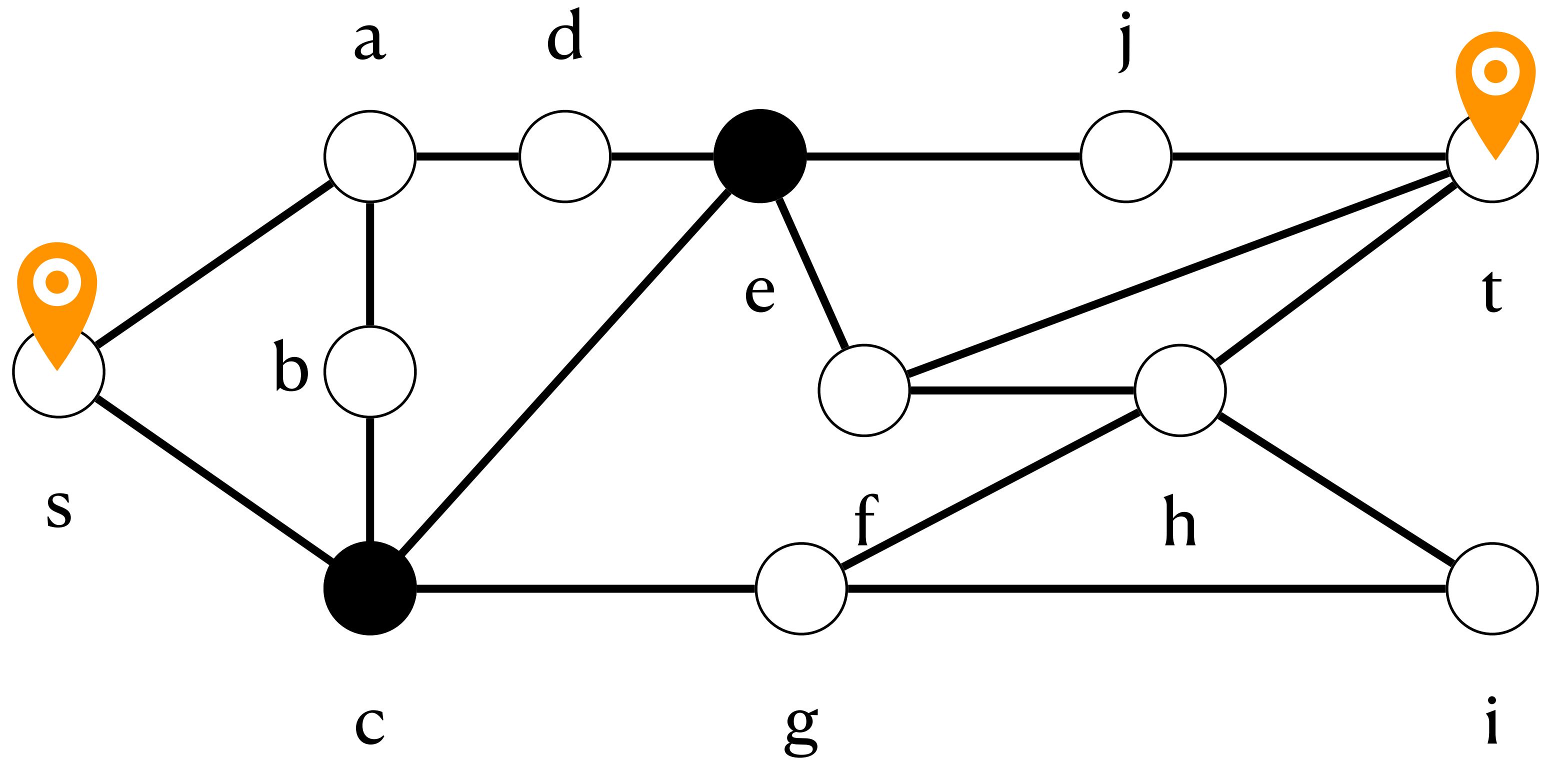
- **Classical st-Vertex Cut Problem**: Find a vertex cut that separates two vertices s and t.
- **Classical st-Vertex Cut Algorithm**: Start with a clean slate and achieve an optimal or feasible solution efficiently.



st-Vertex Cut Discovery Problem

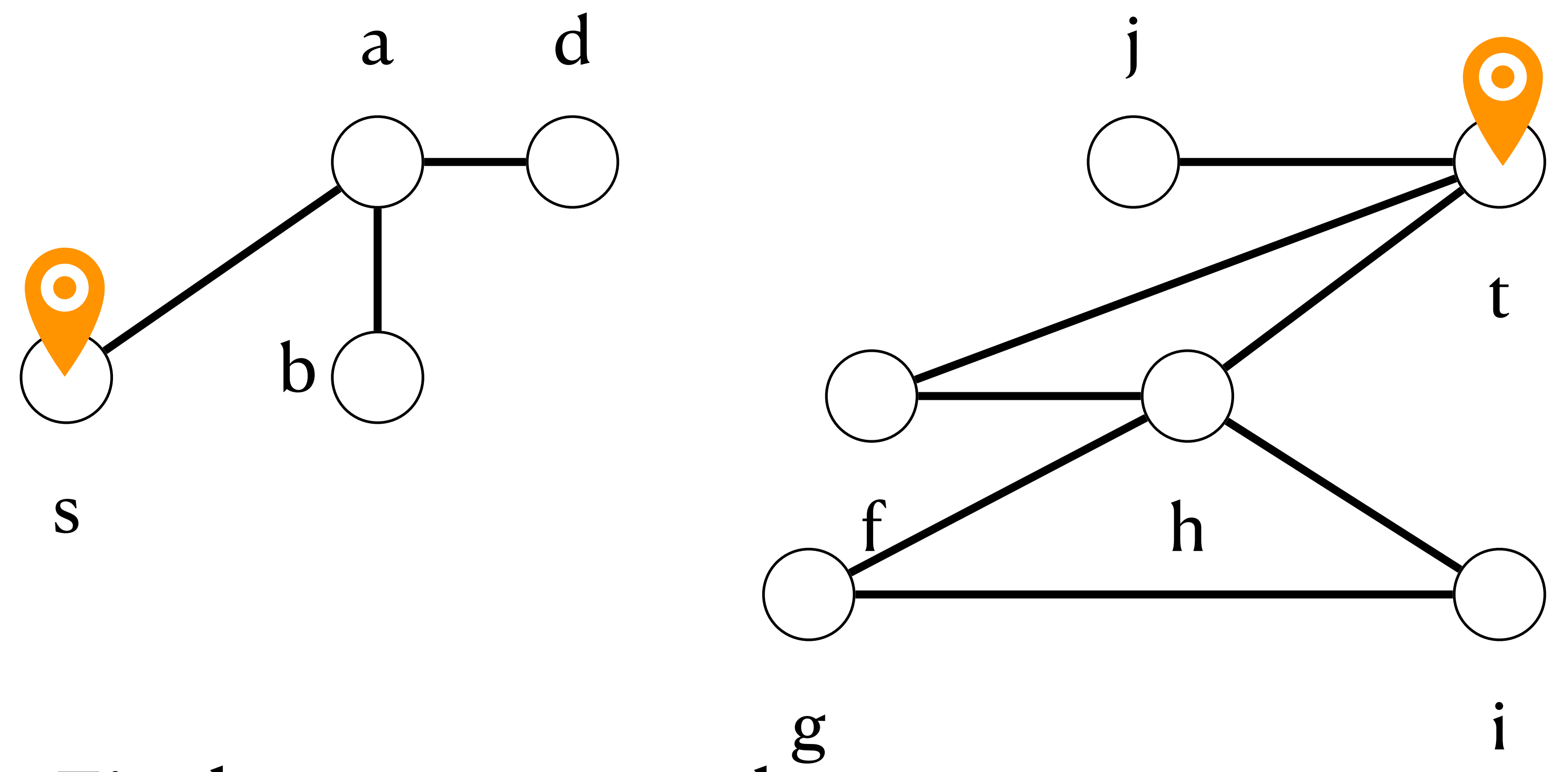


- **Classical st-Vertex Cut Problem**: Find a vertex cut that separates two vertices s and t.
- **Classical st-Vertex Cut Algorithm**: Start with a clean slate and achieve an optimal or feasible solution efficiently.

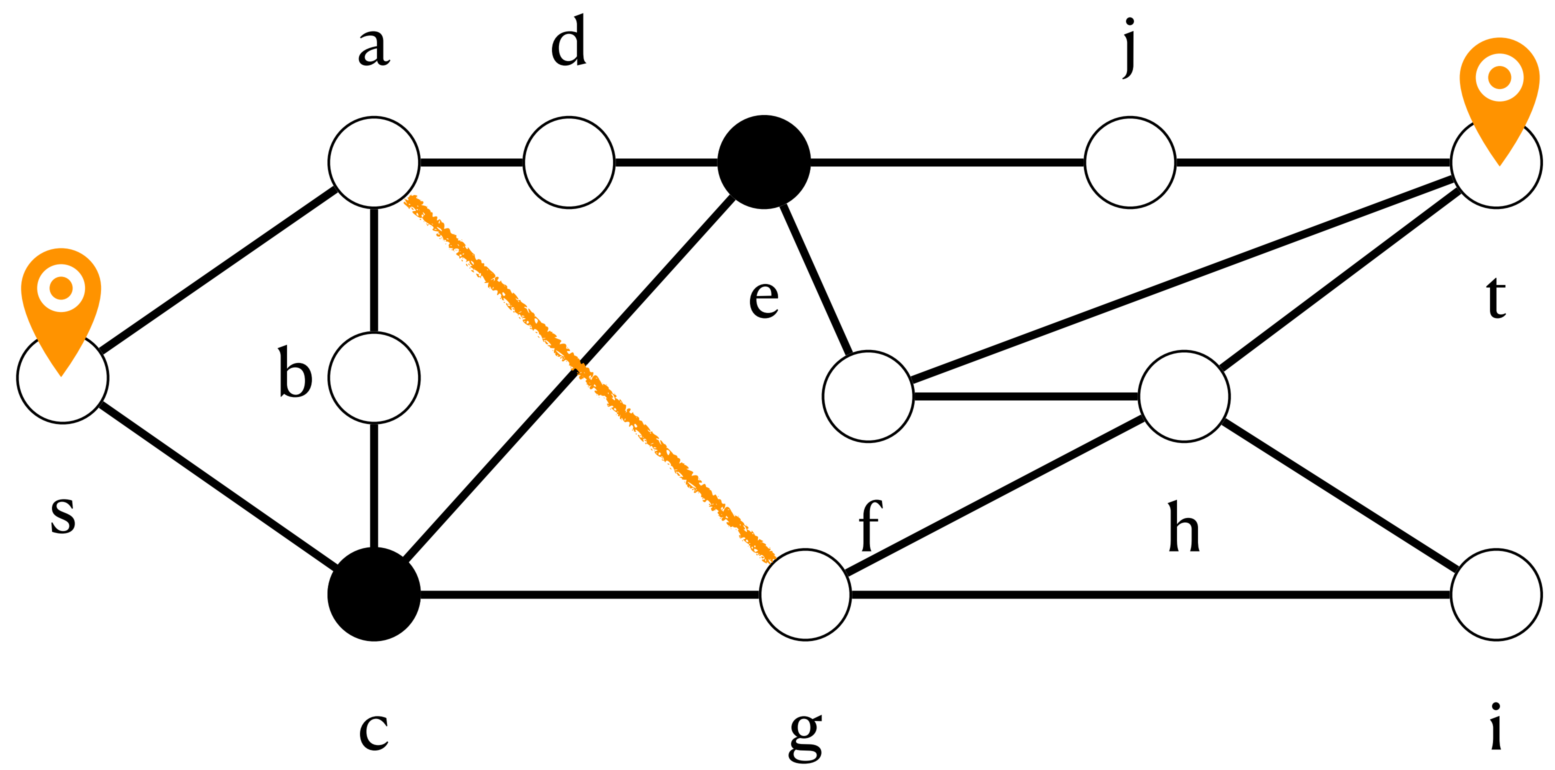


● a resource, a token

st-Vertex Cut Discovery Problem



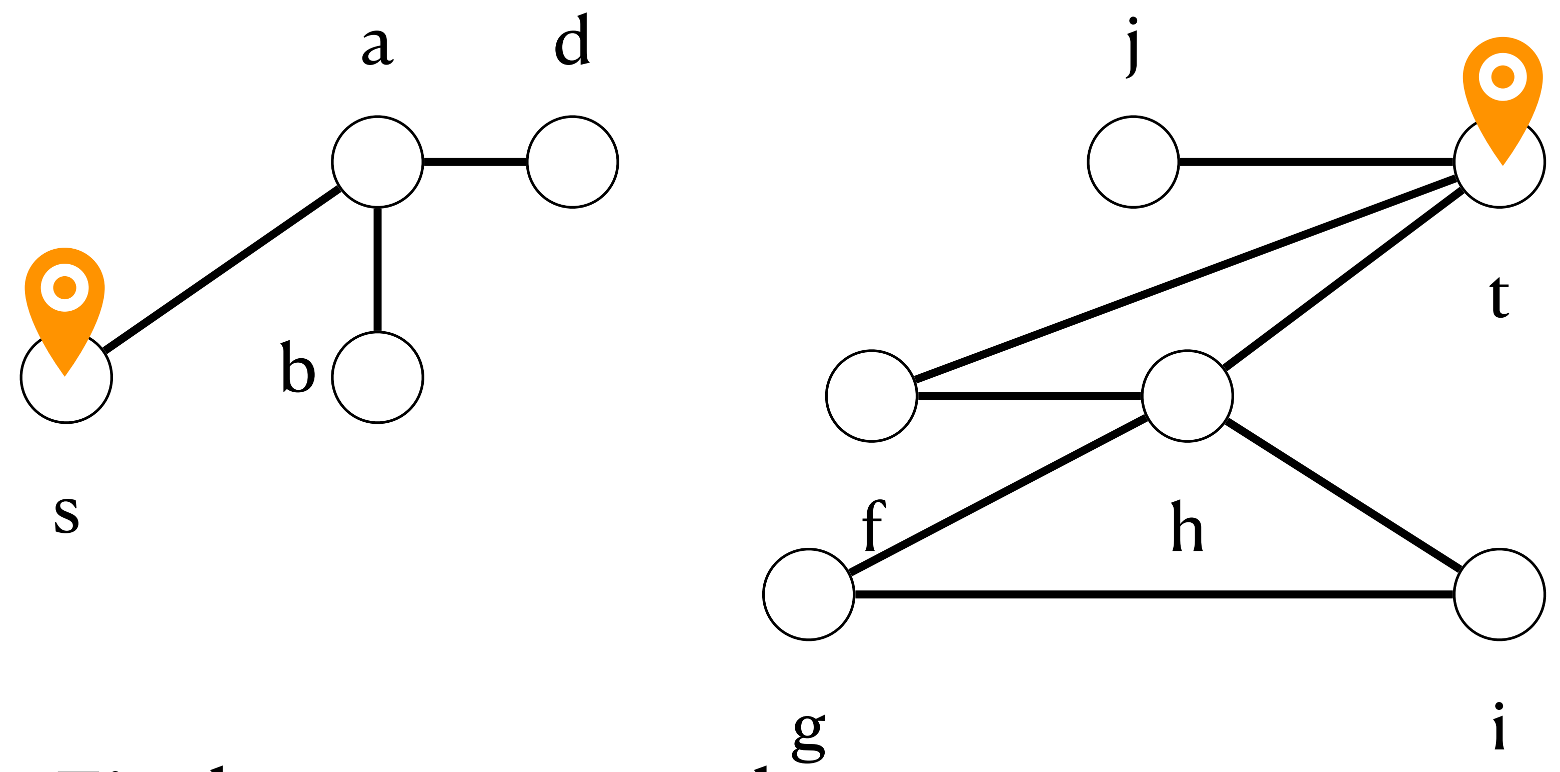
- **Classical st-Vertex Cut Problem**: Find a vertex cut that separates two vertices s and t.
- **Classical st-Vertex Cut Algorithm**: Start with a clean slate and achieve an optimal or feasible solution efficiently.



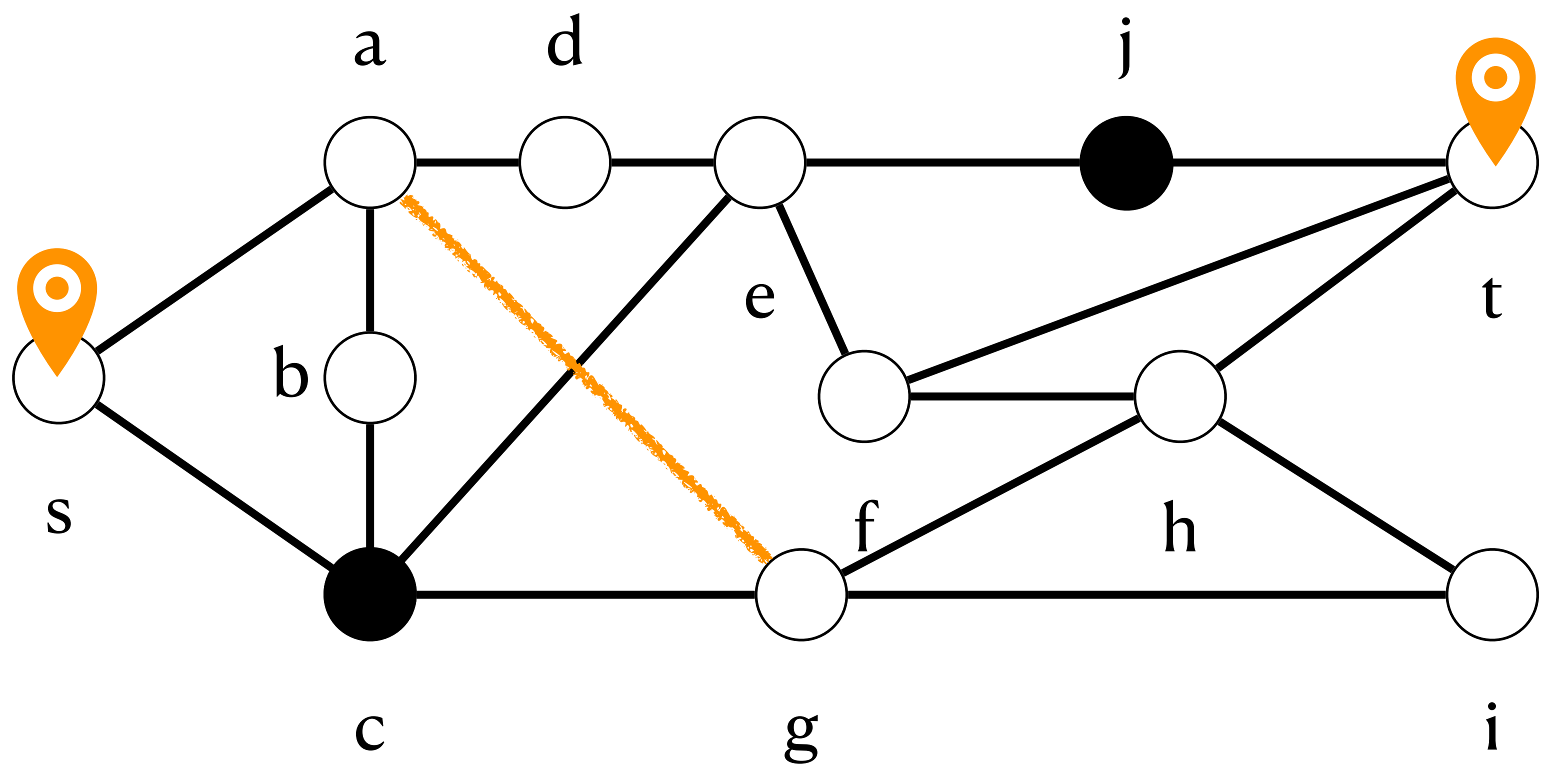
● a resource, a token

1. Infeasible solution viewed as tokens on vertices.

st-Vertex Cut Discovery Problem



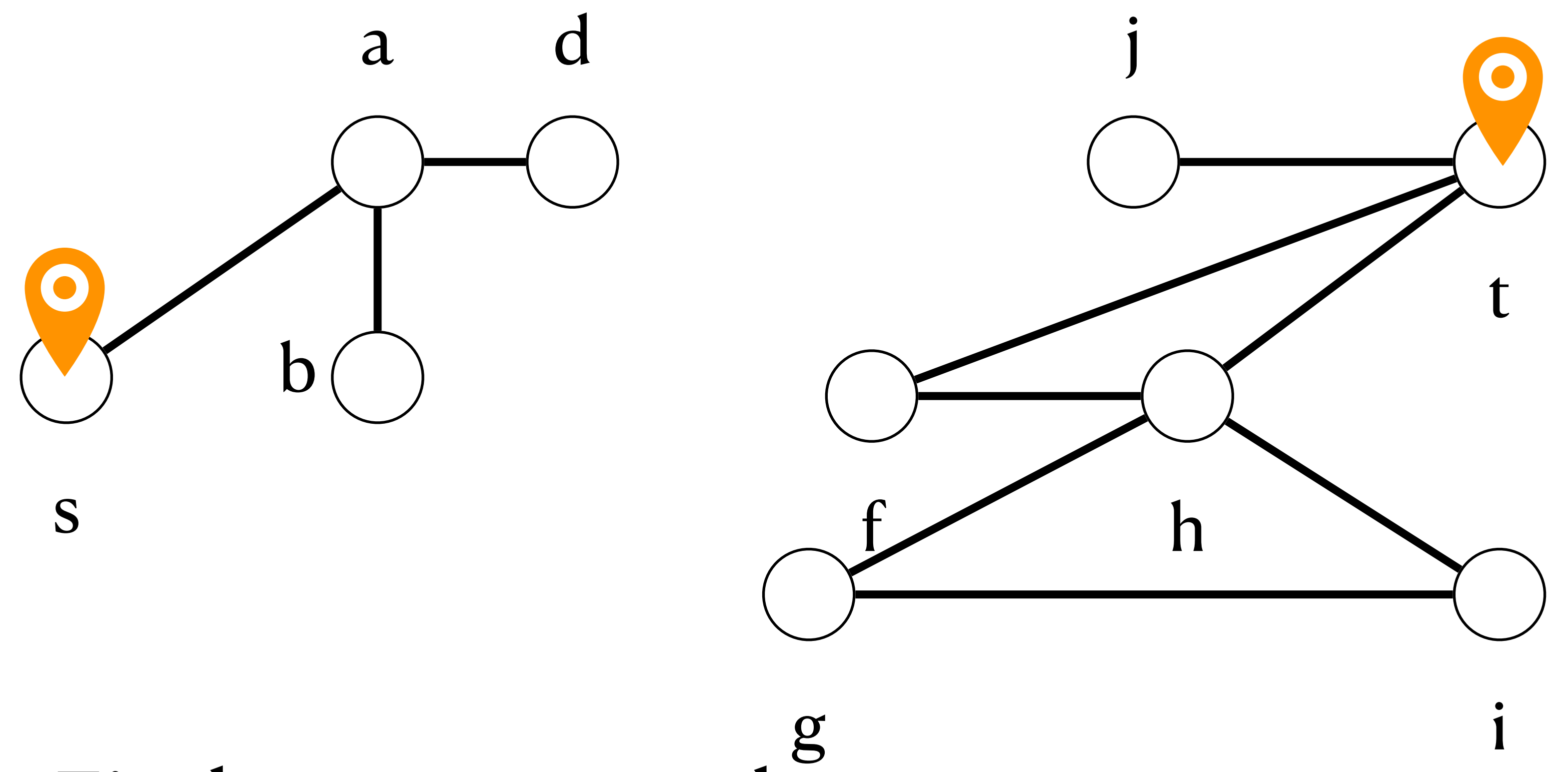
- **Classical st-Vertex Cut Problem**: Find a vertex cut that separates two vertices s and t.
- **Classical st-Vertex Cut Algorithm**: Start with a clean slate and achieve an optimal or feasible solution efficiently.



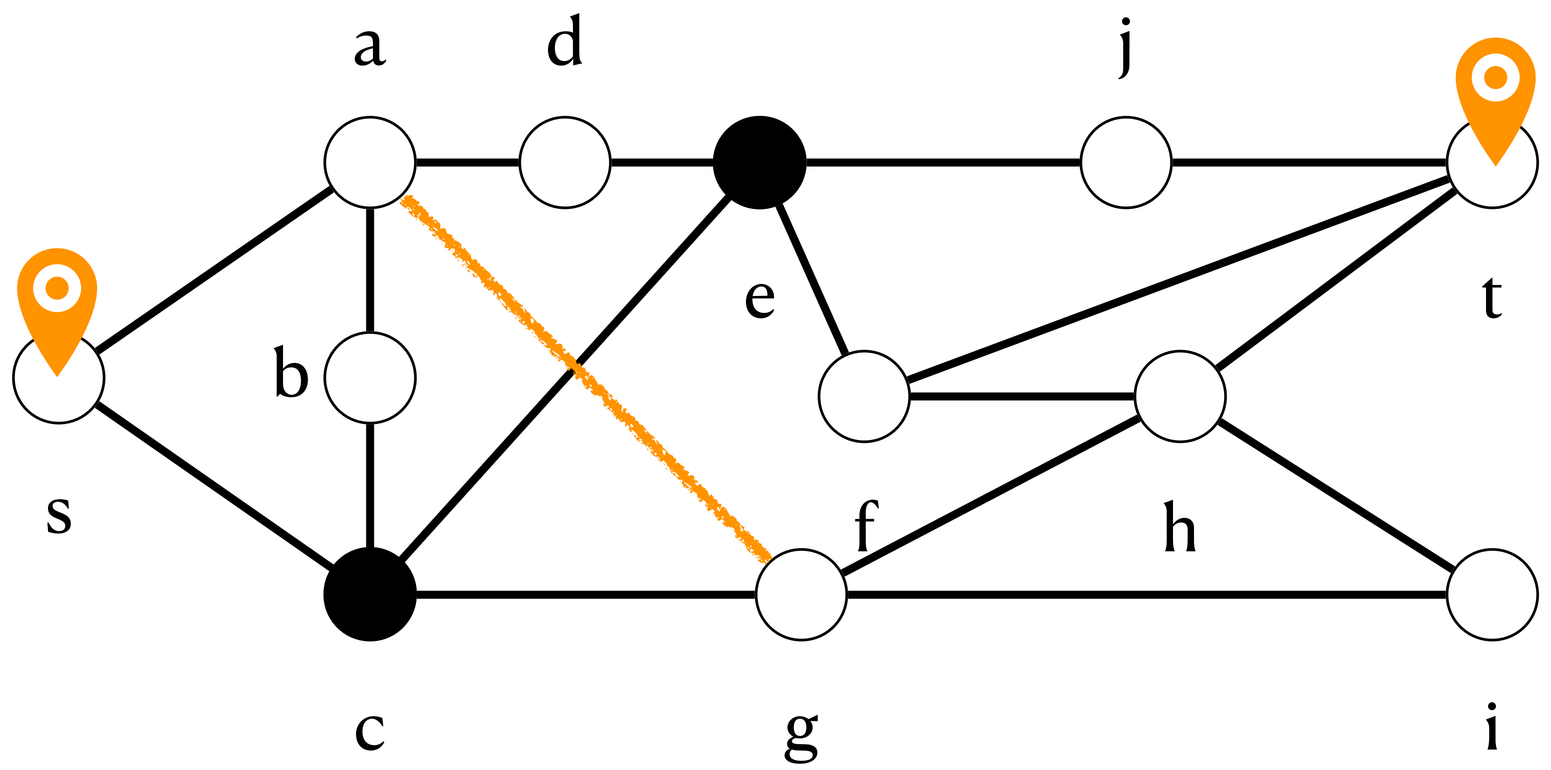
a resource, a token

1. Infeasible solution viewed as tokens on vertices.

st-Vertex Cut Discovery Problem



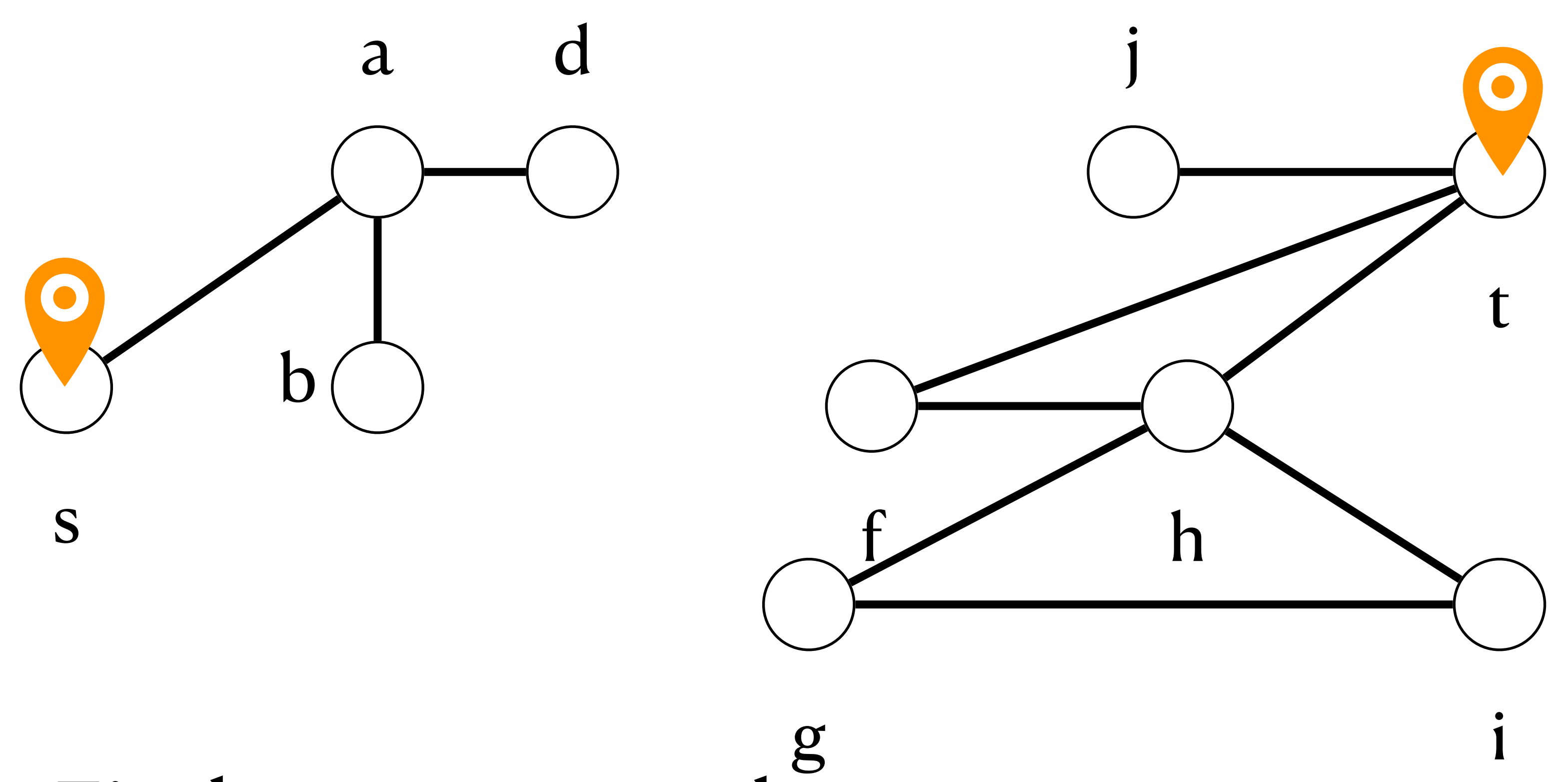
- **Classical st-Vertex Cut Problem**: Find a vertex cut that separates two vertices s and t.
- **Classical st-Vertex Cut Algorithm**: Start with a clean slate and achieve an optimal or feasible solution efficiently.



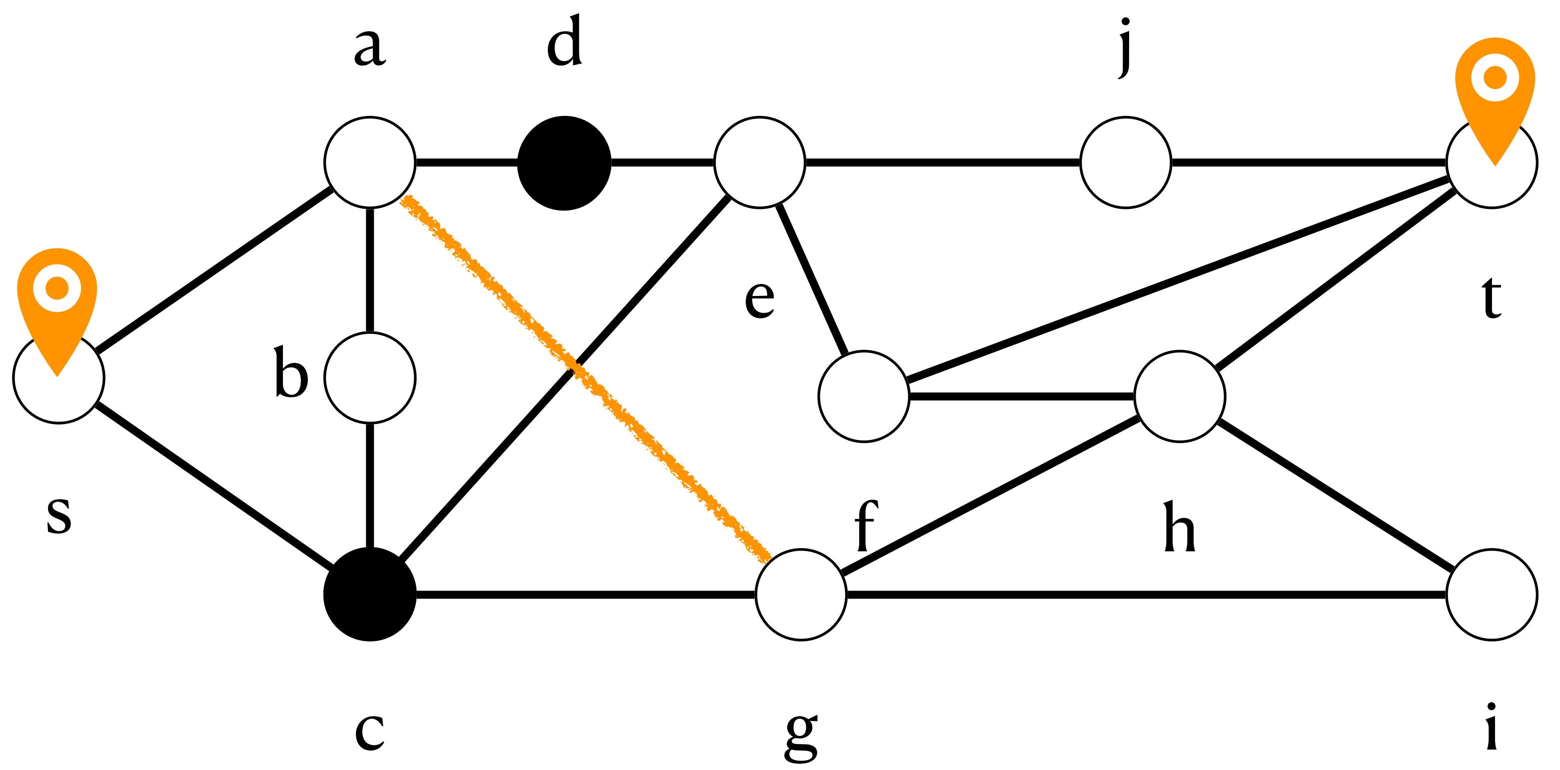
a resource, a token

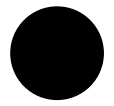
1. Infeasible solution viewed as tokens on vertices.
2. Modification Cost: total no. of *token slides* (no. of edges traversed).

st-Vertex Cut Discovery Problem



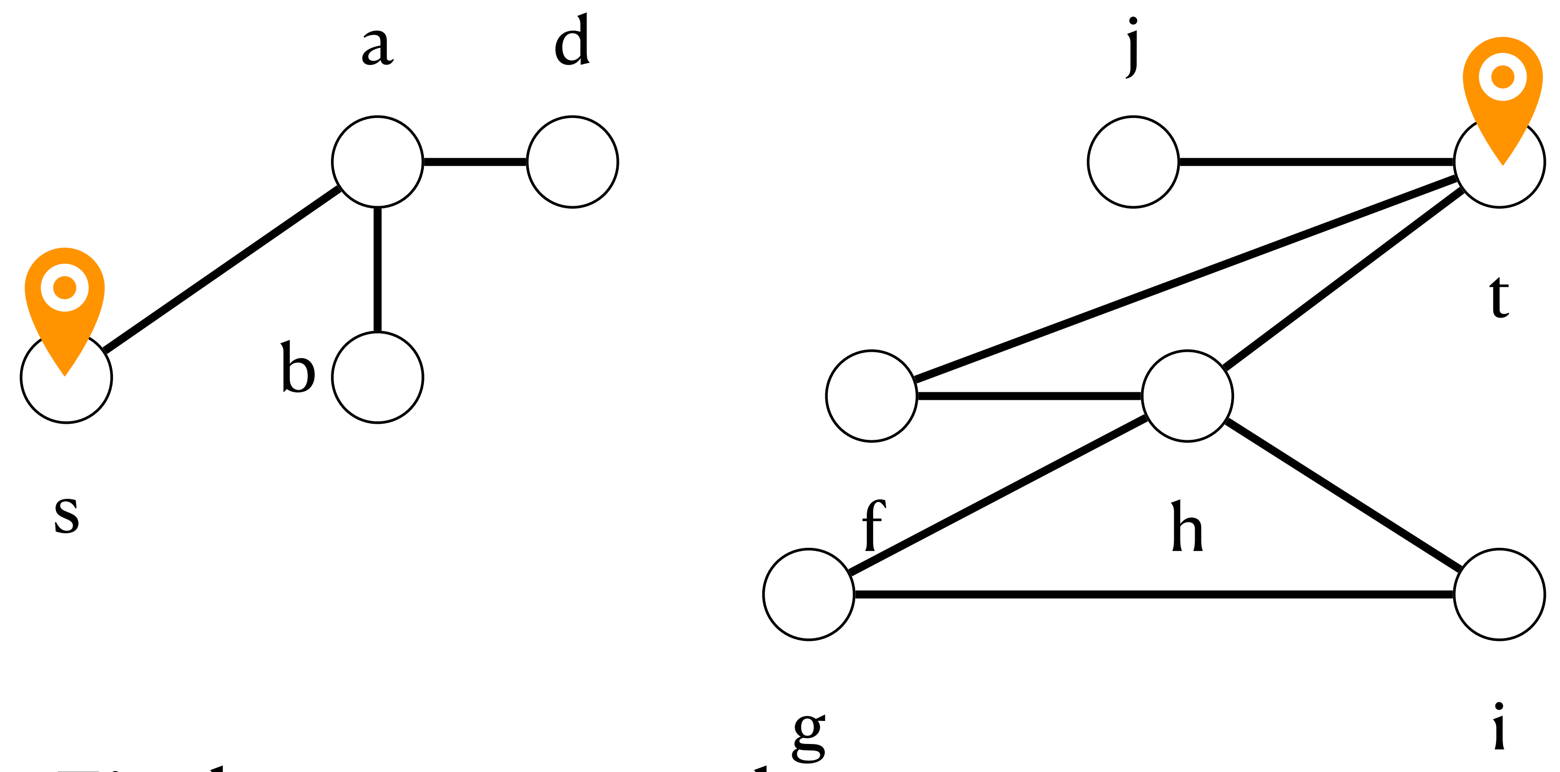
- **Classical st-Vertex Cut Problem**: Find a vertex cut that separates two vertices s and t.
- **Classical st-Vertex Cut Algorithm**: Start with a clean slate and achieve an optimal or feasible solution efficiently.



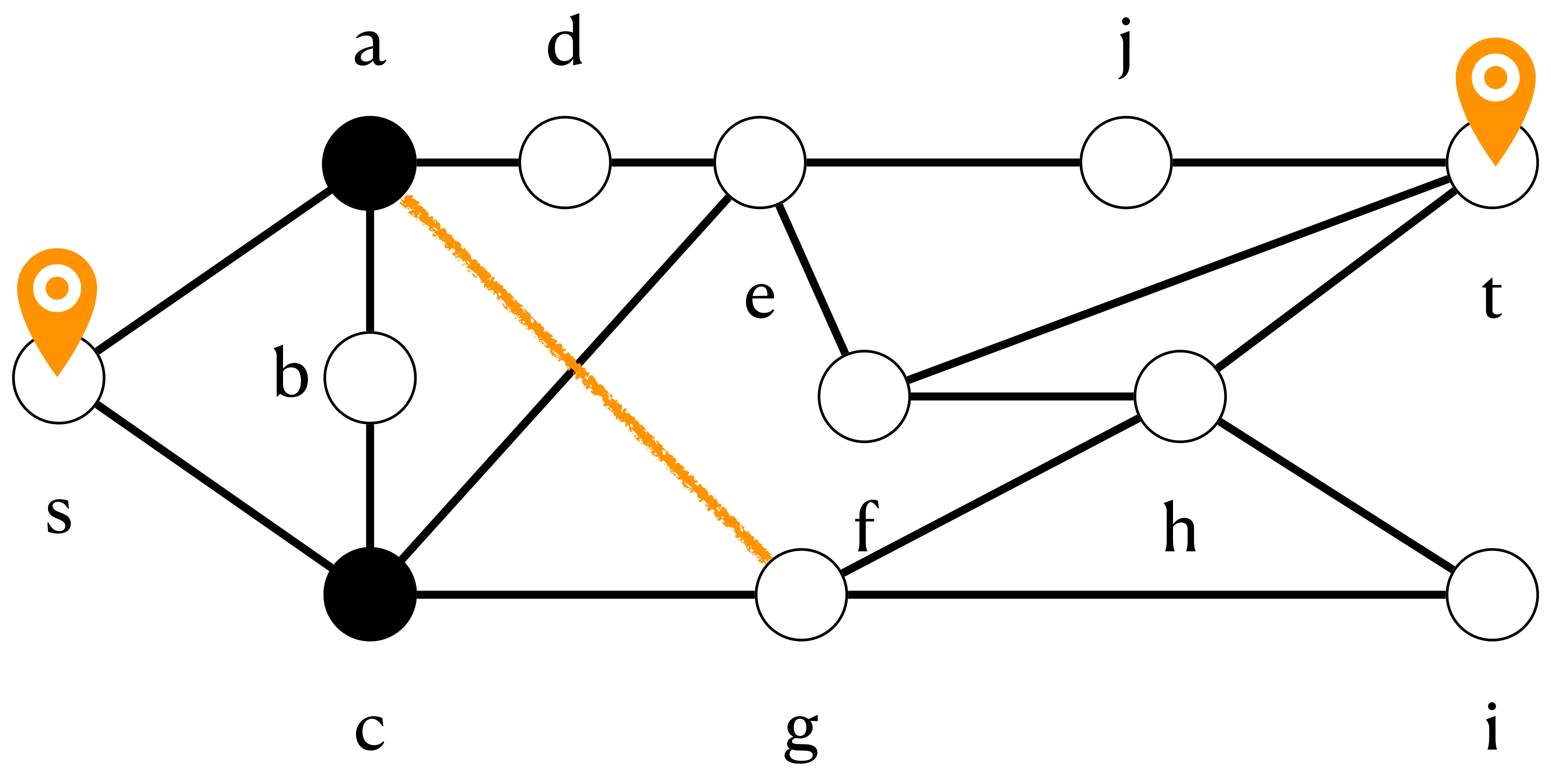
 a resource, a token

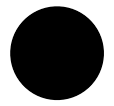
1. Infeasible solution viewed as tokens on vertices.
2. Modification Cost: total no. of *token slides* (no. of edges traversed).

st-Vertex Cut Discovery Problem



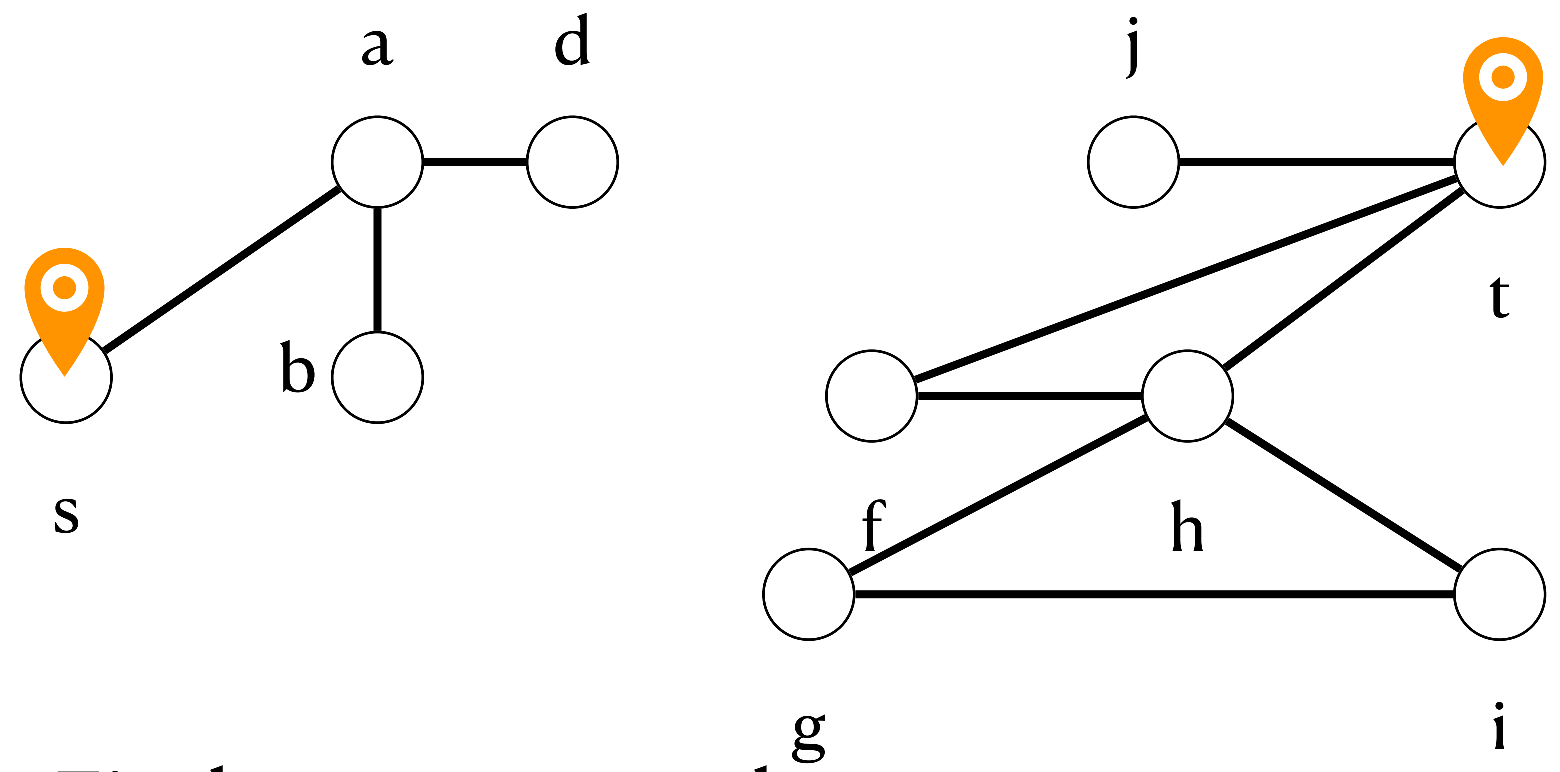
- **Classical st-Vertex Cut Problem**: Find a vertex cut that separates two vertices s and t.
- **Classical st-Vertex Cut Algorithm**: Start with a clean slate and achieve an optimal or feasible solution efficiently.



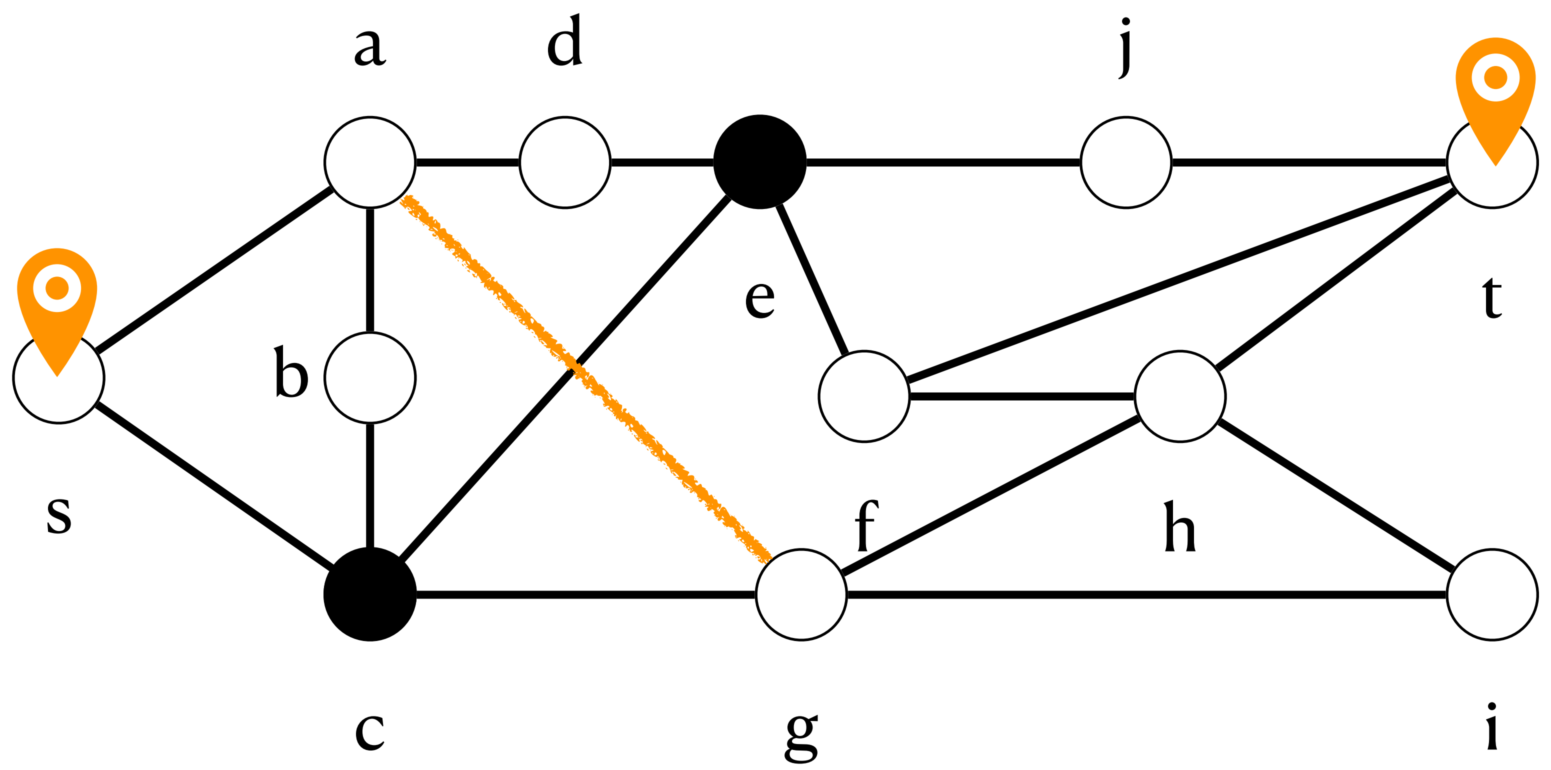
 a resource, a token

1. Infeasible solution viewed as tokens on vertices.
2. Modification Cost: total no. of *token slides* (no. of edges traversed).

st-Vertex Cut Discovery Problem



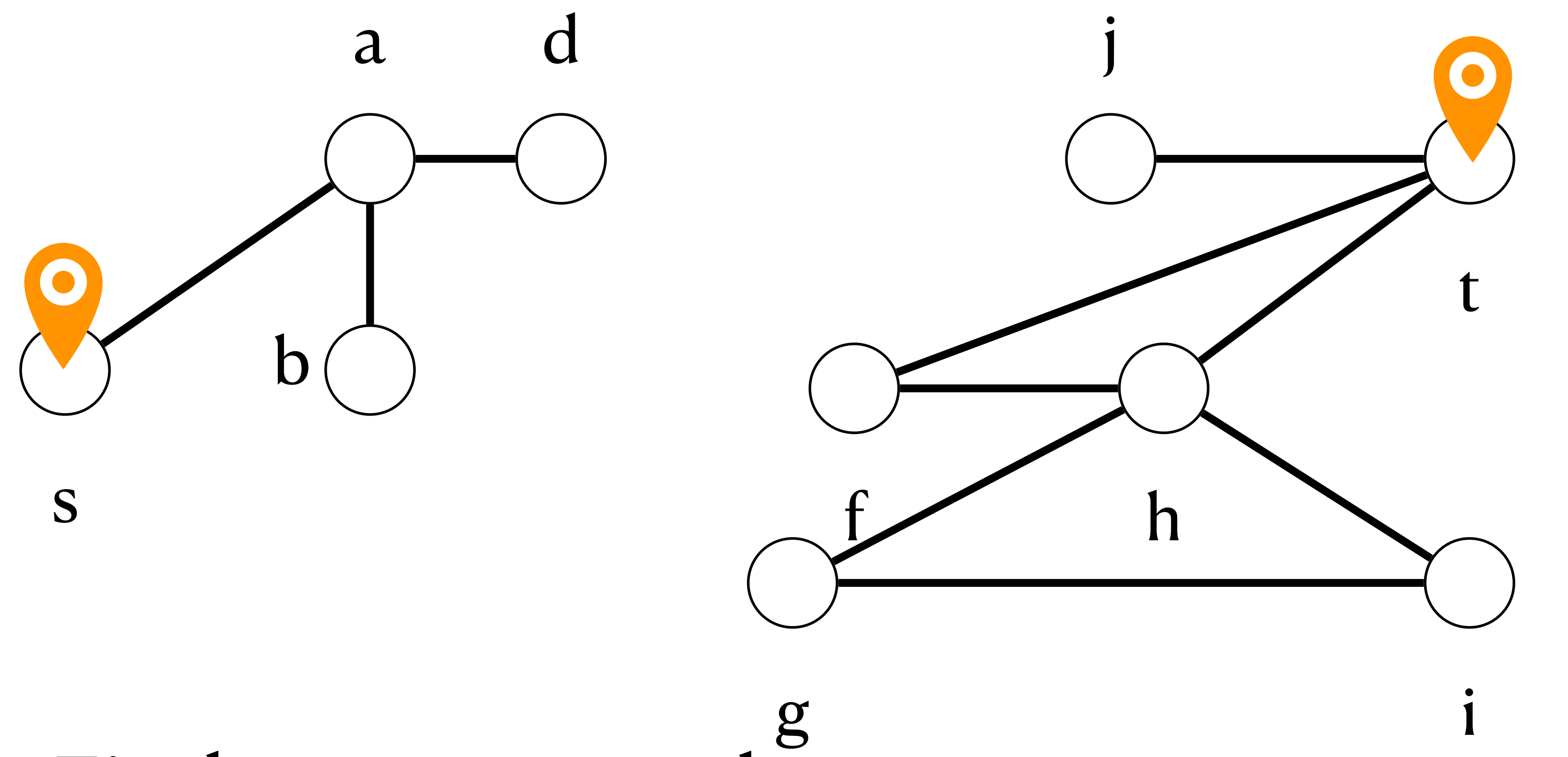
- **Classical st-Vertex Cut Problem**: Find a vertex cut that separates two vertices s and t.
- **Classical st-Vertex Cut Algorithm**: Start with a clean slate and achieve an optimal or feasible solution efficiently.



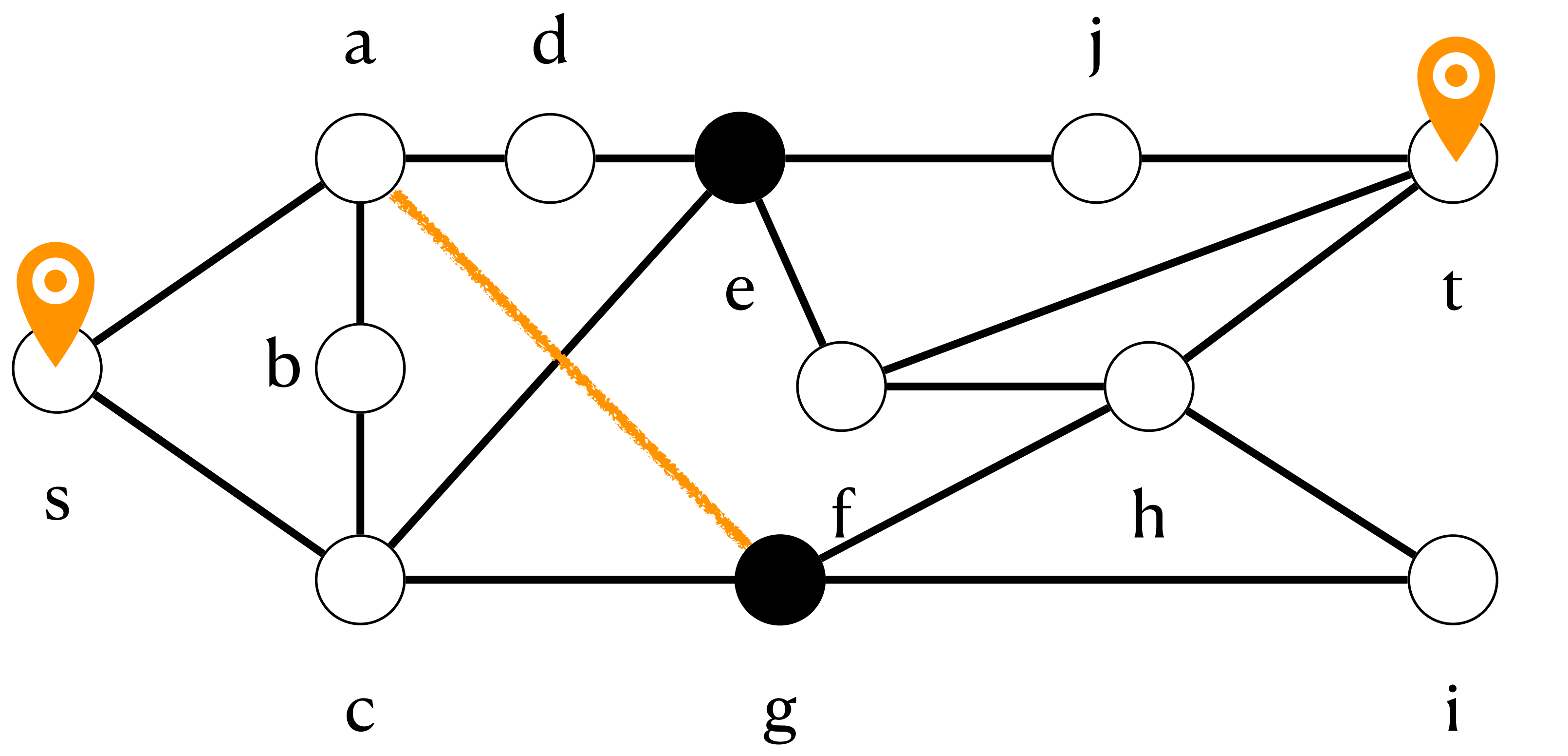
a resource, a token

1. Infeasible solution viewed as tokens on vertices.
2. Modification Cost: total no. of *token slides* (no. of edges traversed).

st-Vertex Cut Discovery Problem



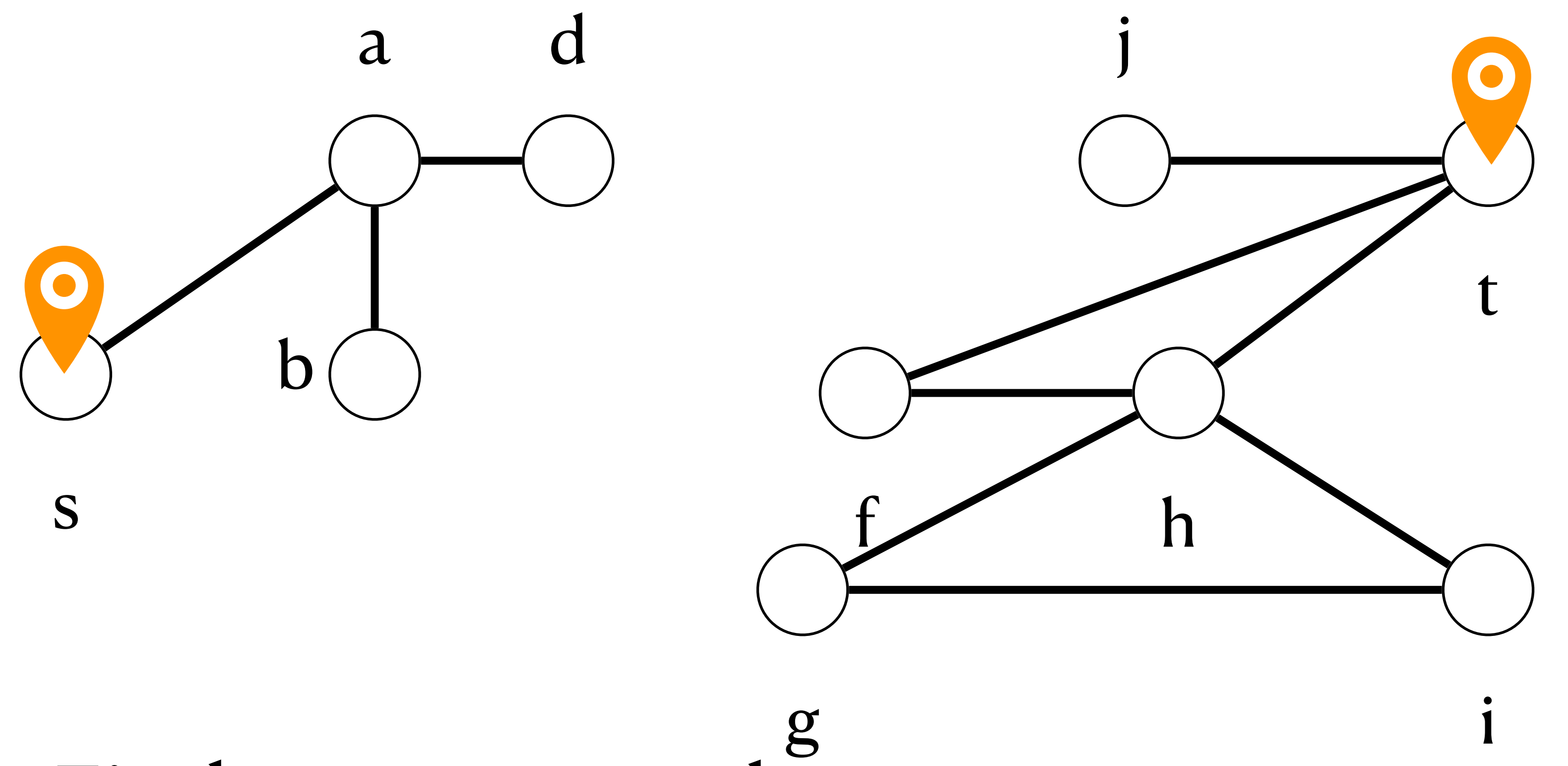
- **Classical st-Vertex Cut Problem**: Find a vertex cut that separates two vertices s and t.
- **Classical st-Vertex Cut Algorithm**: Start with a clean slate and achieve an optimal or feasible solution efficiently.



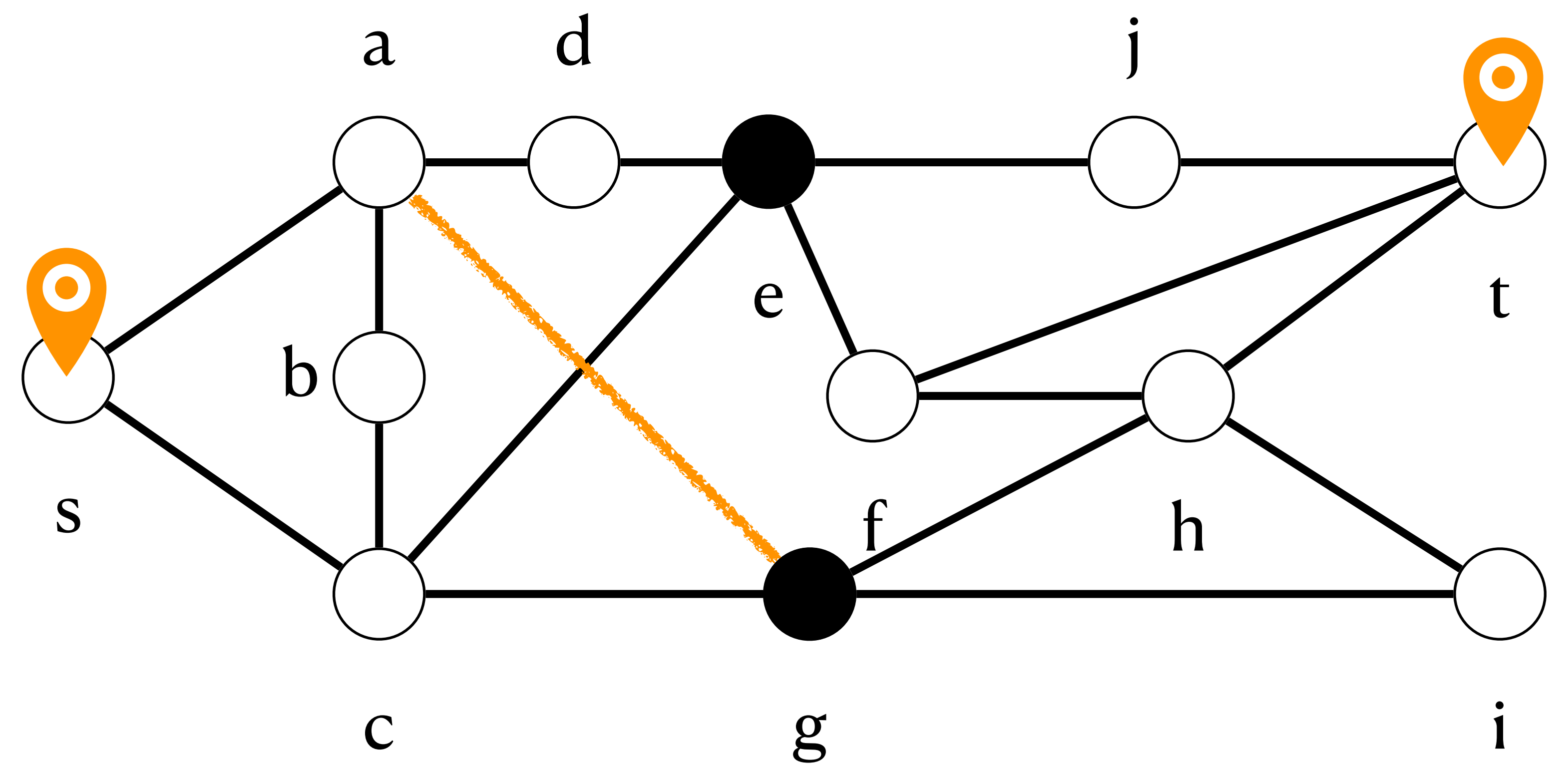
● a resource, a token

1. Infeasible solution viewed as tokens on vertices.
2. Modification Cost: total no. of *token slides* (no. of edges traversed).
3. Here, we have: *Modification Cost* = 1.

st-Vertex Cut Discovery Problem



- **Classical st-Vertex Cut Problem**: Find a vertex cut that separates two vertices s and t.
- **Classical st-Vertex Cut Algorithm**: Start with a clean slate and achieve an optimal or feasible solution efficiently.



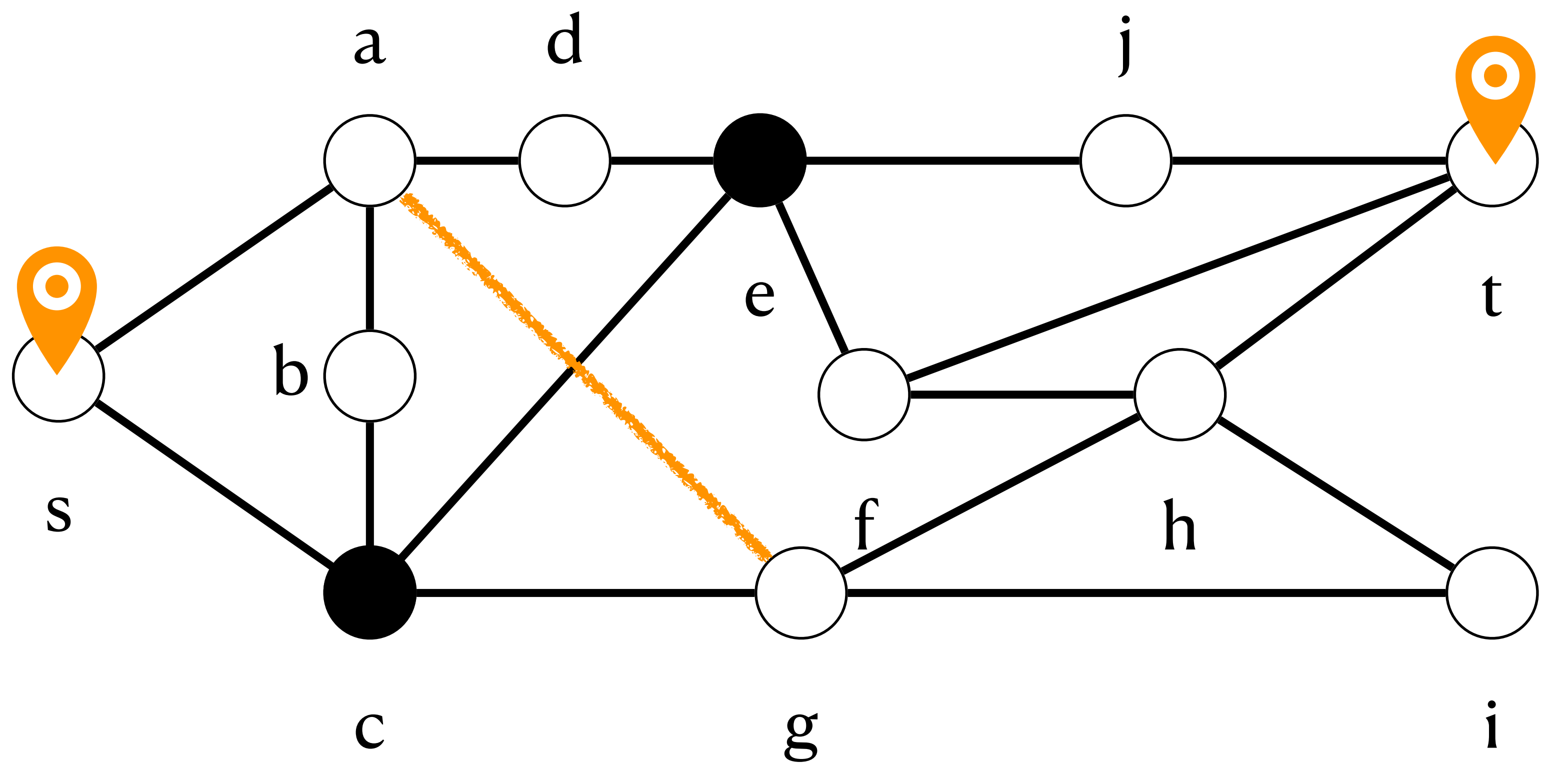
● a resource, a token

1. Infeasible solution viewed as tokens on vertices.
2. Modification Cost: total no. of *token slides* (no. of edges traversed).
3. Here, we have: *Modification Cost* = 1.

- **st-Vertex Cut Discovery Problem**: Find a vertex cut between s and t with <u>small</u> modification cost.

Solution Discovery Problem

- **st-Vertex Cut Discovery Problem**: Find a vertex cut between s and t with <u>small</u> *modification cost*.



a resource, a token

1. Infeasible solution viewed as tokens on vertices.
2. Modification Cost: total no. of *token slides* (no. of edges traversed).
3. Here, we have: *Modification Cost* = 1.

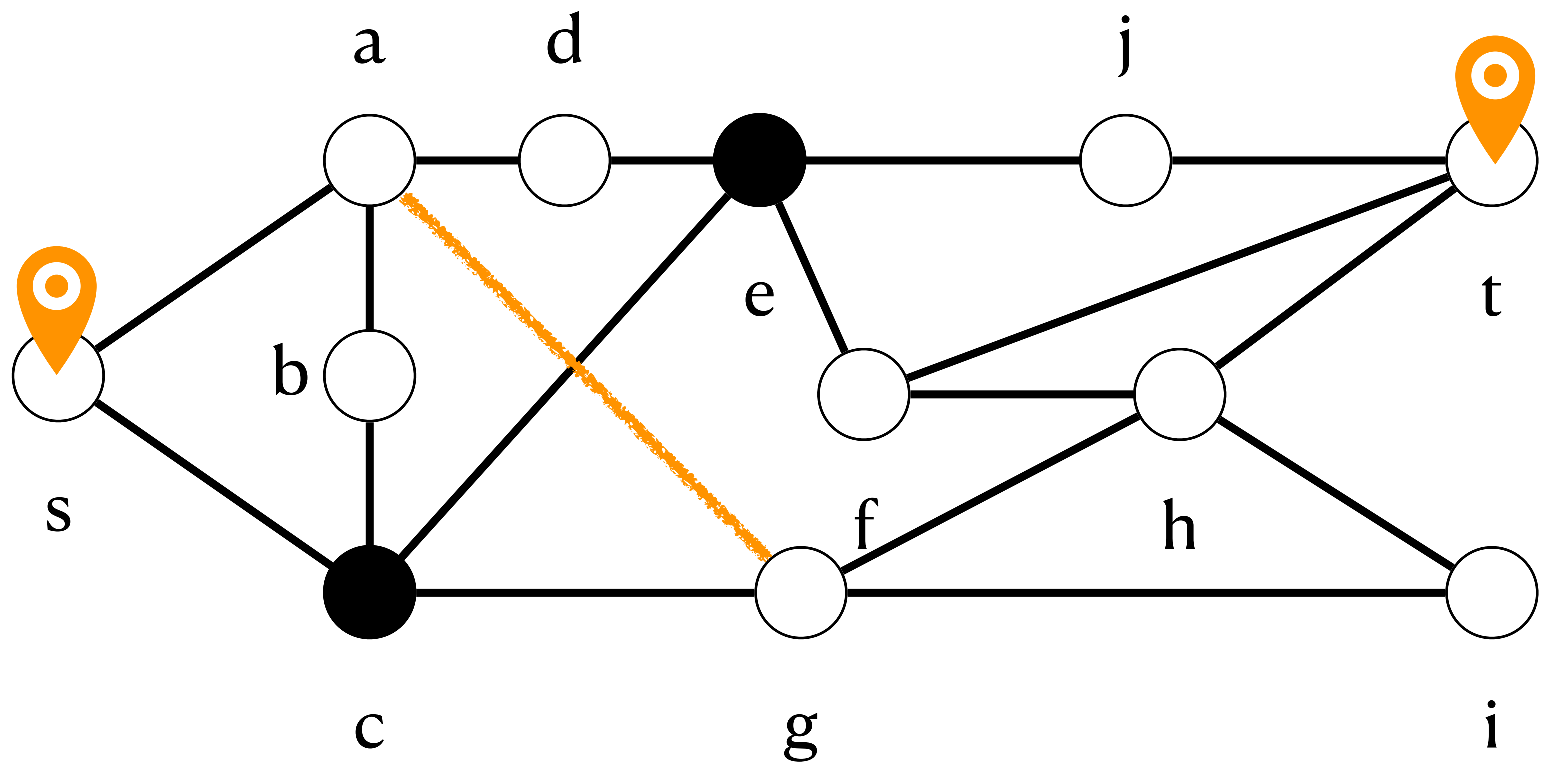
st-Vertex Cut-*discovery:*

Input: A graph G, two vertices $s, t \in V(G)$, an infeasible st-vertex cut NF, and an integer b.

Output: Compute a feasible st-vertex cut F such that F can be obtained from NF in at most b token slides.

Solution Discovery Problem

- **st-Vertex Cut Discovery Problem**: Find a vertex cut between s and t with small *modification cost*.



a resource, a token

1. Infeasible solution viewed as tokens on vertices.
2. Modification Cost: total no. of *token slides* (no. of edges traversed).
3. Here, we have: *Modification Cost* = 1.

st-Vertex Cut-*discovery:*

Input: A graph G, two vertices $s, t \in V(G)$, an infeasible st-vertex cut NF, and an integer b.

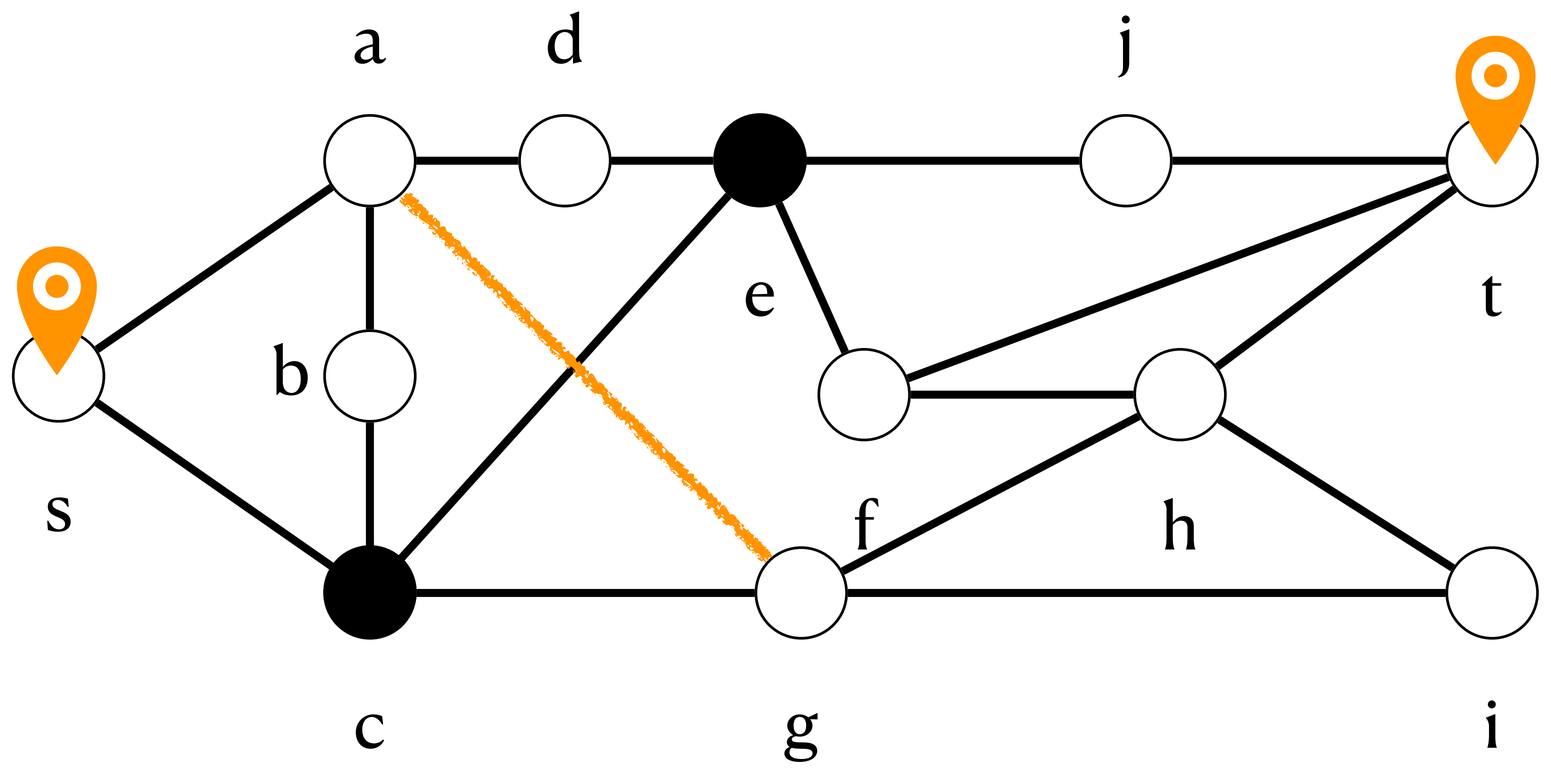
Output: Compute a feasible st-vertex cut F such that F can be obtained from NF in at most b token slides.

Π-*discovery:*

Input: A graph G, an infeasible Π NF, and an integer b.

Solution Discovery Problem

- **st-Vertex Cut Discovery Problem**: Find a vertex cut between s and t with <u>small</u> *modification cost*.



● a resource, a token

1. Infeasible solution viewed as tokens on vertices.
2. Modification Cost: total no. of *token slides* (no. of edges traversed).
3. Here, we have: *Modification Cost* = 1.

st-Vertex Cut-*discovery:*

Input: A graph G, two vertices $s, t \in V(G)$, an infeasible st-vertex cut NF, and an integer b.

Output: Compute a feasible st-vertex cut F such that F can be obtained from NF in at most b token slides.

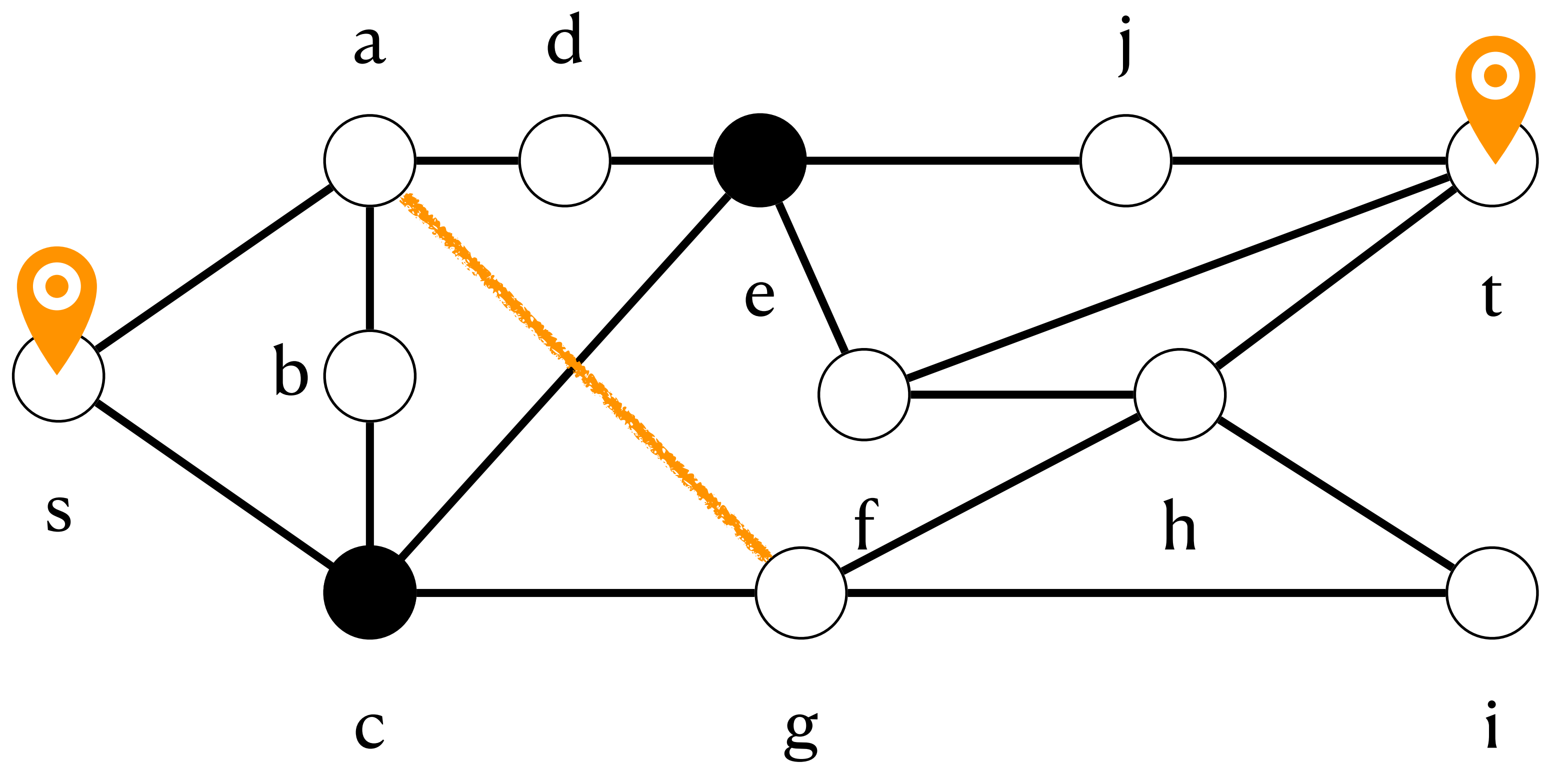
Π-*discovery:*

Input: A graph G, an infeasible Π NF, and an integer b.

Output: Compute a feasible Π F such that F can be obtained from NF in at most b token slides.

Solution Discovery Problem

- **st-Vertex Cut Discovery Problem**: Find a vertex cut between s and t with <u>small</u> *modification cost*.



a resource, a token

1. Infeasible solution viewed as tokens on vertices.
2. Modification Cost: total no. of *token slides* (no. of edges traversed).
3. Here, we have: *Modification Cost* = 1.

st-Vertex Cut-*discovery:*

Input: A graph G, two vertices $s, t \in V(G)$, an infeasible st-vertex cut NF, and an integer b.

Output: Compute a feasible st-vertex cut F such that F can be obtained from NF in at most b token slides.

Π-*discovery:*

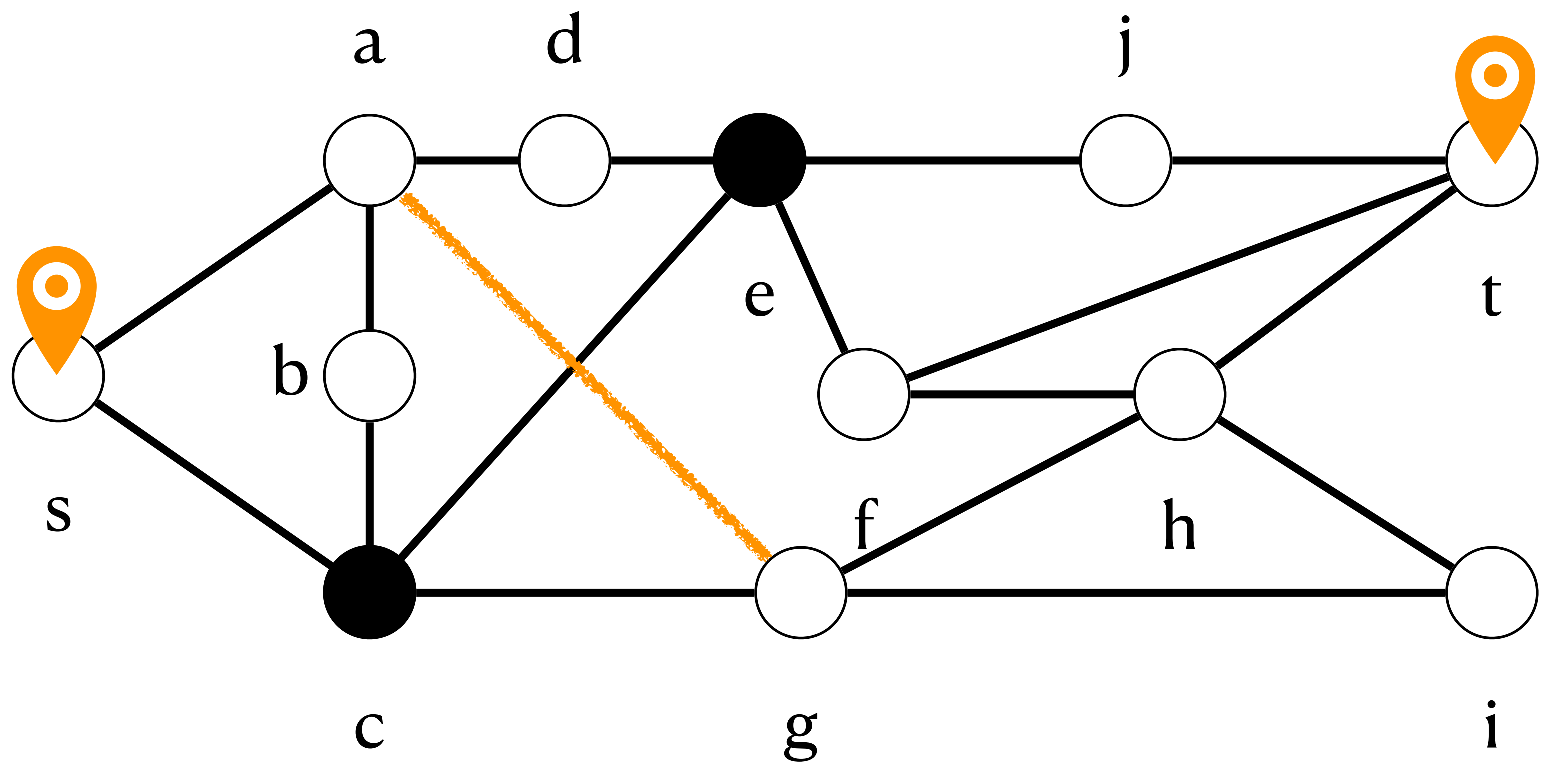
Input: A graph G, an infeasible Π NF, and an integer b.

Output: Compute a feasible Π F such that F can be obtained from NF in at most b token slides.

We consider for Π: st-Vertex Cut, Independent Set, Vertex Cover, Dominating Set, Matching, Shortest Path

Solution Discovery Problem

- **st-Vertex Cut Discovery Problem**: Find a vertex cut between s and t with <u>small</u> *modification cost*.



● a resource, a token

1. Infeasible solution viewed as tokens on vertices.
2. Modification Cost: total no. of *token slides* (no. of edges traversed).
3. Here, we have: *Modification Cost* = 1.

st-Vertex Cut-*discovery:*

Input: A graph G, two vertices $s, t \in V(G)$, an infeasible st-vertex cut NF, and an integer b.

Output: Compute a feasible st-vertex cut F such that F can be obtained from NF in at most b token slides.

Π-*discovery:*

Input: A graph G, an infeasible Π NF, and an integer b.

Output: Compute a feasible Π F such that F can be obtained from NF in at most b token slides.

We consider for Π: st-Vertex Cut, Independent Set, Vertex Cover, Dominating Set, Matching, Shortest Path

Solution Discovery Problem

st-Vertex Cut-*discovery:*

Input: A graph G, two vertices $s, t \in V(G)$, an infeasible st-vertex cut NF, and an integer b.

Output: Compute a feasible st-vertex cut F such that F can be obtained from NF in at most b token slides.

Π-*discovery:*

Input: A graph G, an infeasible Π NF, and an integer b.

Output: Compute a feasible Π F such that F can be obtained from NF in at most b token slides.

Solution Discovery Problem

st-Vertex Cut-*discovery:*

Input: A graph G, two vertices $s, t \in V(G)$, an infeasible st-vertex cut NF, and an integer b.

Output: Compute a feasible st-vertex cut F such that F can be obtained from NF in at most b token slides.

Π-*discovery:*

Input: A graph G, an infeasible Π NF, and an integer b.

Output: Compute a feasible Π F such that F can be obtained from NF in at most b token slides.

Solution discovery problem:

Input: An instance I of a source problem Π, an infeasible solution NF to I under Π, and an integer b.

Solution Discovery Problem

st-Vertex Cut-*discovery:*

Input: A graph G, two vertices $s, t \in V(G)$, an infeasible st-vertex cut NF, and an integer b.
Output: Compute a feasible st-vertex cut F such that F can be obtained from NF in at most b token slides.

Π-*discovery:*

Input: A graph G, an infeasible Π NF, and an integer b.
Output: Compute a feasible Π F such that F can be obtained from NF in at most b token slides.

Solution discovery problem:

Input: An instance I of a source problem Π, an infeasible solution NF to I under Π, and an integer b.
Output: Compute a feasible solution F to I under Π such that F can be obtained from NF in at most b modification steps.

Solution Discovery Problem

st-Vertex Cut-*discovery:*

Input: A graph G, two vertices $s, t \in V(G)$, an infeasible st-vertex cut NF, and an integer b.
Output: Compute a feasible st-vertex cut F such that F can be obtained from NF in at most b token slides.

Π-*discovery:*

Input: A graph G, an infeasible Π NF, and an integer b.
Output: Compute a feasible Π F such that F can be obtained from NF in at most b token slides.

Solution discovery problem:

Input: An instance I of a source problem Π, an infeasible solution NF to I under Π, and an integer b.
Output: Compute a feasible solution F to I under Π such that F can be obtained from NF in at most b modification steps.

Solution Discovery problems occur frequently in the real world, for example [Siraichi et al. 2018].

Solution Discovery Problem

st-Vertex Cut-*discovery:*

Input: A graph G, two vertices $s, t \in V(G)$, an infeasible st-vertex cut NF, and an integer b.
Output: Compute a feasible st-vertex cut F such that F can be obtained from NF in at most b token slides.

Π-*discovery:*

Input: A graph G, an infeasible Π NF, and an integer b.
Output: Compute a feasible Π F such that F can be obtained from NF in at most b token slides.

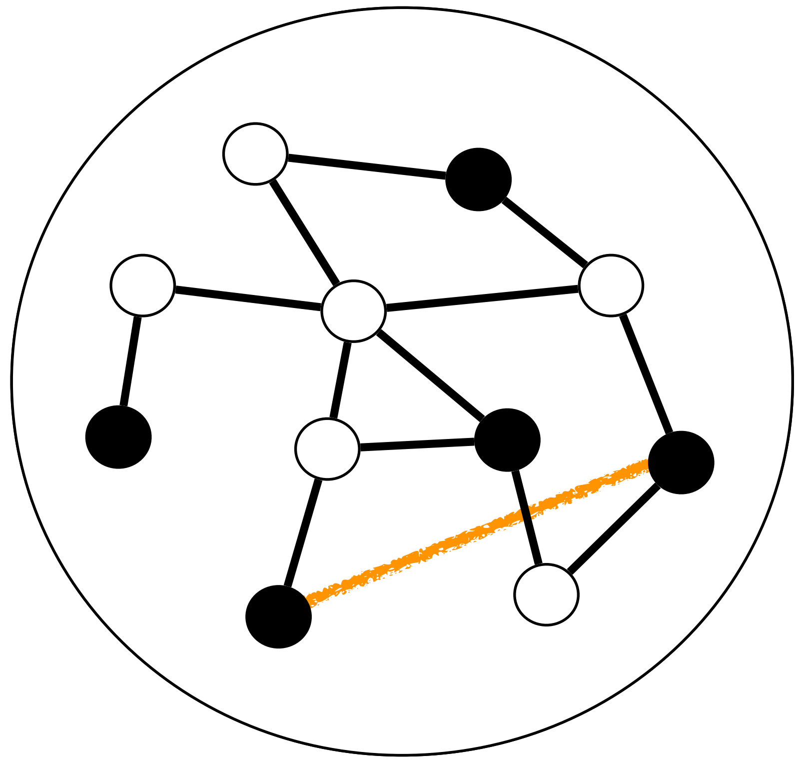
Solution Discovery problems occur frequently in the real world, for example [Siraichi et al. 2018].

Solution Discovery Algorithm is not...

Π-*discovery:*

Input: A graph G, an infeasible Π NF, and an integer b.

Output: Compute a feasible Π F such that F can be obtained from NF in at most b token slides.



● a resource, a token

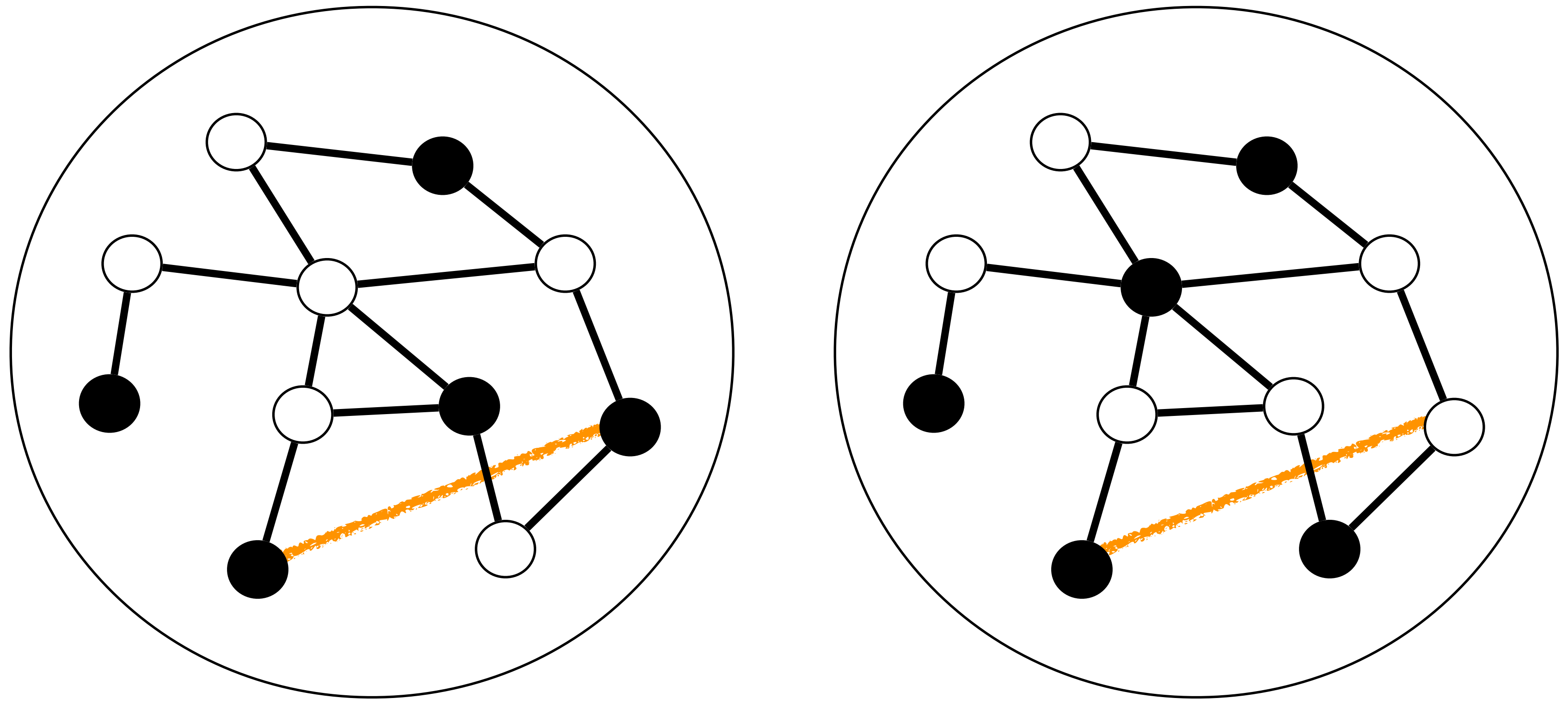
Reoptimization algorithm: given an Independent Set instance with a feasible solution and an orange edge to add to the instance compute a feasible solution efficiently.

Solution Discovery Algorithm is not...

Π-*discovery:*

Input: A graph G, an infeasible Π NF, and an integer b.

Output: Compute a feasible Π F such that F can be obtained from NF in at most b token slides.



● a resource, a token

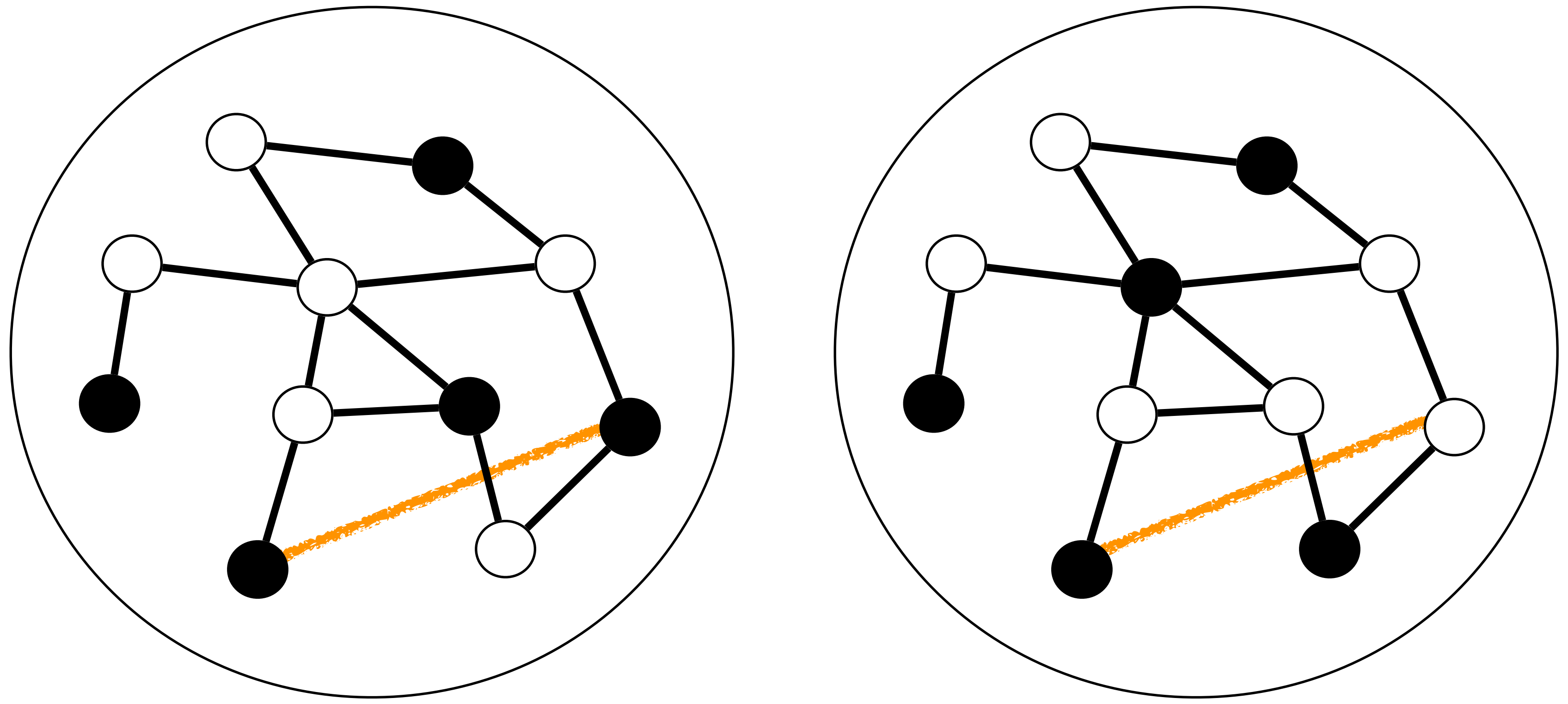
Reoptimization algorithm: given an Independent Set instance with a feasible solution and an orange edge to add to the instance compute a feasible solution efficiently.

Solution Discovery Algorithm is not...

Π-*discovery:*

Input: A graph G, an infeasible Π NF, and an integer b.

Output: Compute a feasible Π F such that F can be obtained from NF in at most b token slides.



● a resource, a token

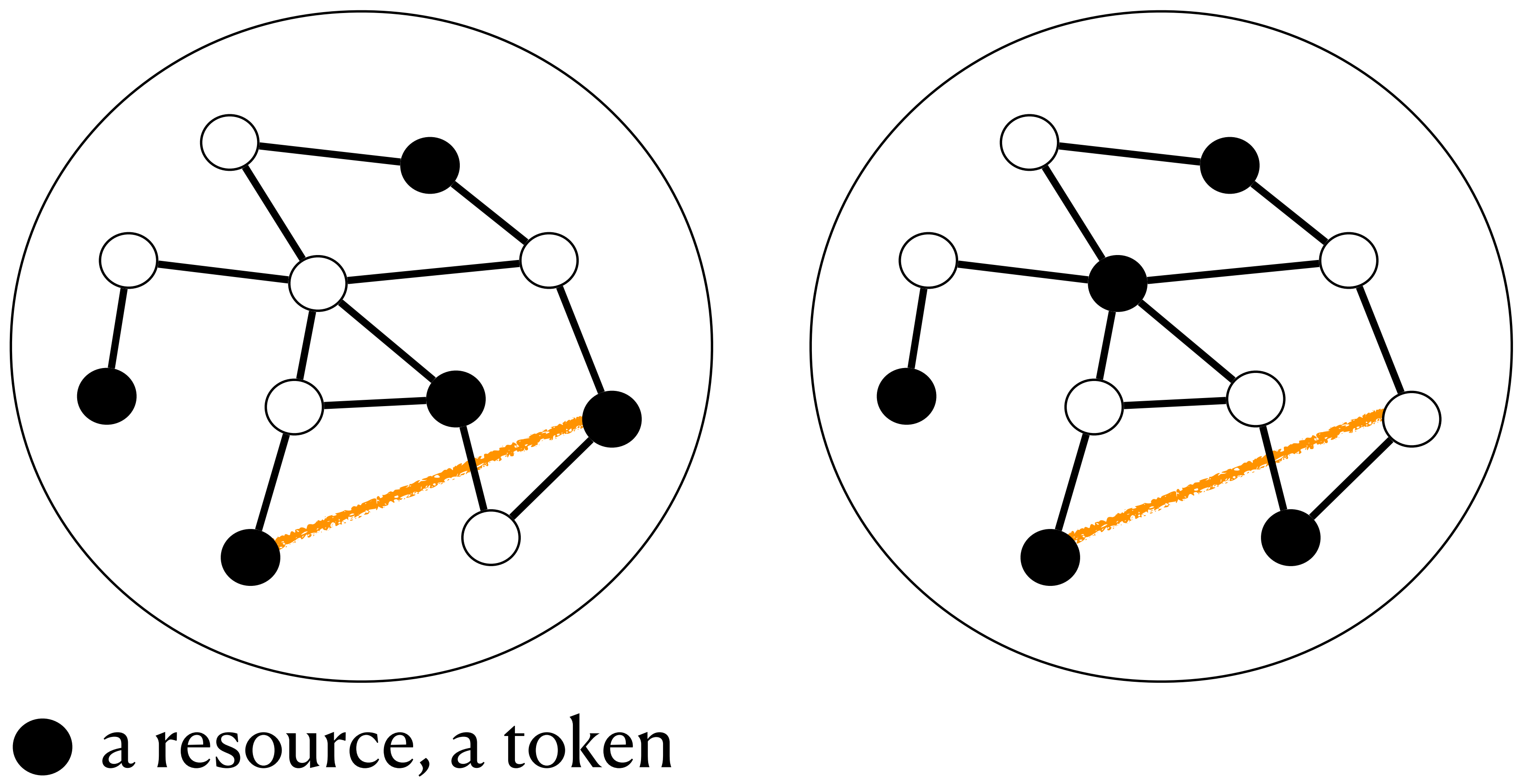
Reoptimization algorithm: given an Independent Set instance with a feasible solution and an orange edge to add to the instance compute a feasible solution efficiently.

No consideration
for modification costs

Solution Discovery Algorithm is not...

Π-*discovery:*

Input: A graph G, an infeasible Π NF, and an integer b.
Output: Compute a feasible Π F such that F can be obtained from NF in at most b token slides.



Reoptimization algorithm: given an Independent Set instance with a feasible solution and an orange edge to add to the instance compute a feasible solution efficiently.

No consideration
for modification costs

Solution Discovery Algorithms **are also different** from..

I. Dynamic Graph Algorithms

II. Combinatorial Reconfiguration Algorithms

III. Local Search Algorithms

Related Work

Π-*discovery:*

Input: A graph G, an infeasible Π NF, and an integer b.

Output: Compute a feasible Π F such that F can be obtained from NF in at most b token slides.

Related Work

Π-*discovery:*

Input: A graph G, an infeasible Π NF, and an integer b.

Output: Compute a feasible Π F such that F can be obtained from NF in at most b token slides.

We consider for Π: st-Vertex Cut, Independent Set, Vertex Cover, Dominating Set, Matching, Shortest Path

Related Work

Π-*discovery:*

Input: A graph G, an infeasible Π NF, and an integer b.

Output: Compute a feasible Π F such that F can be obtained from NF in at most b token slides.

We consider for Π: st-Vertex Cut, Independent Set, Vertex Cover, Dominating Set, Matching, Shortest Path

- [Fellows et al, 2023], [Grobler et al. 2024] show **NP-hardness** for all these solution discovery problems.

Parameterized and Kernelization Complexities

p: parameter x: instance for classical problem

Parameterized and Kernelization Complexities

- A *parameterized problem* with an algorithm that runs in $g(p) \cdot |x|^c$ time, where g is a function, is in FPT.

p: parameter x: instance for classical problem

Parameterized and Kernelization Complexities

- A *parameterized problem* with an algorithm that runs in $g(p) \cdot |x|^c$ time, where g is a function, is in FPT.
- FPT $\subseteq$ W[1] $\subseteq$ W[2] $\subseteq$ ... $\subseteq$ XNLP

p: parameter x: instance for classical problem

Parameterized and Kernelization Complexities

- A *parameterized problem* with an algorithm that runs in $g(p) \cdot |x|^c$ time, where g is a function, is in FPT.
- FPT ⊆ W[1] ⊆ W[2] ⊆ ... ⊆ XNLP
- XNLP-hard: W[t]-hard for every t.

p: parameter x: instance for classical problem

Parameterized and Kernelization Complexities

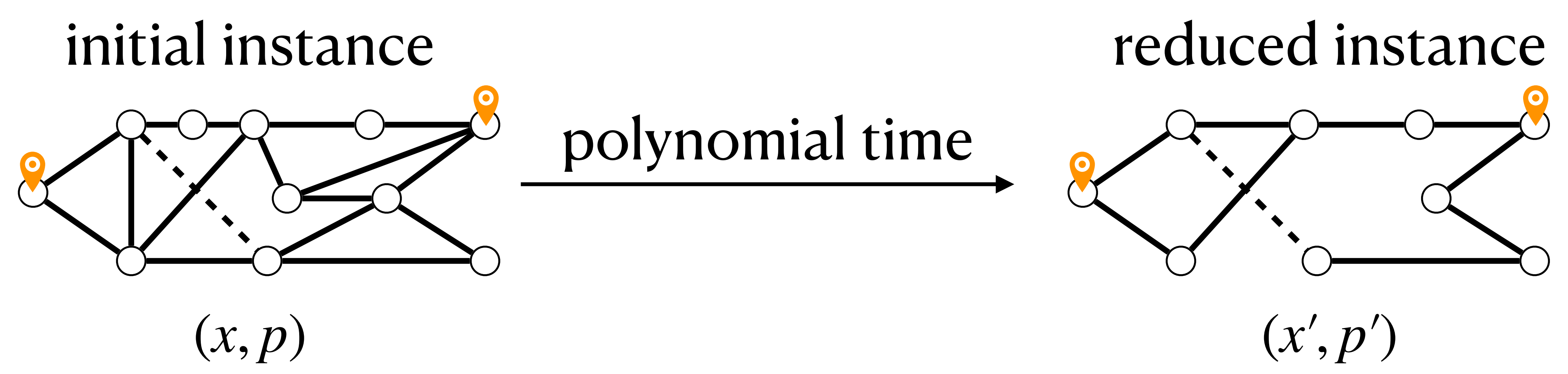
- A *parameterized problem* with an algorithm that runs in $g(p) \cdot |x|^c$ time, where g is a function, is in FPT.
- FPT ⊆ W[1] ⊆ W[2] ⊆ ... ⊆ XNLP
- XNLP-hard: W[t]-hard for every t.
- Every parameterized problem in FPT has a *kernel* [Cai et al. 1997].

p: parameter x: instance for classical problem

Parameterized and Kernelization Complexities

- A *parameterized problem* with an algorithm that runs in $g(p) \cdot |x|^c$ time, where g is a function, is in FPT.
- FPT $\subseteq$ W[1] $\subseteq$ W[2] $\subseteq$ ... $\subseteq$ XNLP
- XNLP-hard: W[t]-hard for every t.
- Every parameterized problem in FPT has a *kernel* [Cai et al. 1997].

Algorithm A



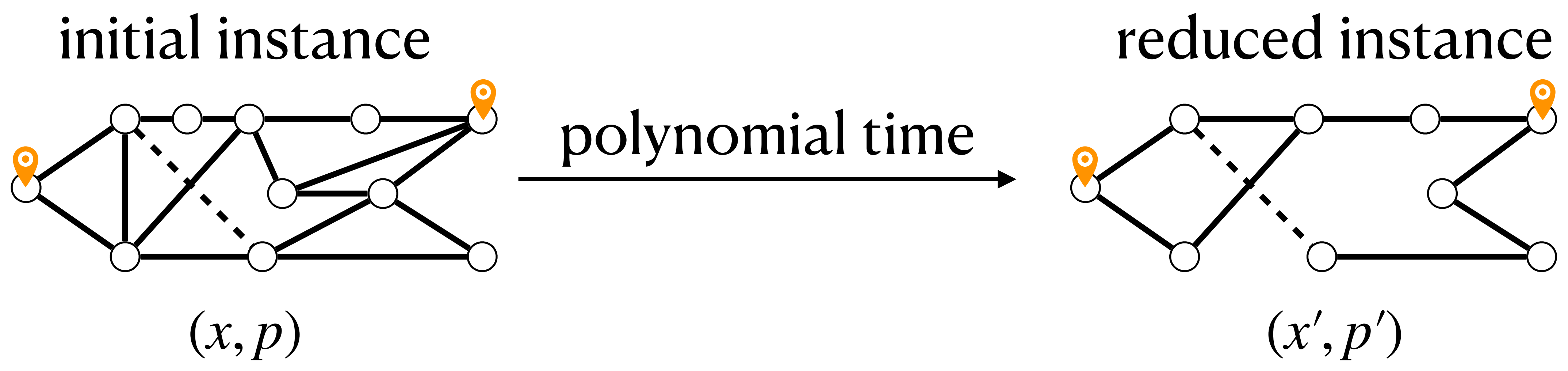
p: parameter

x: instance for classical problem

Parameterized and Kernelization Complexities

- A *parameterized problem* with an algorithm that runs in $g(p) \cdot |x|^c$ time, where g is a function, is in FPT.
- FPT ⊆ W[1] ⊆ W[2] ⊆ ... ⊆ XNLP
- XNLP-hard: W[t]-hard for every t.
- Every parameterized problem in FPT has a *kernel* [Cai et al. 1997].

Algorithm A



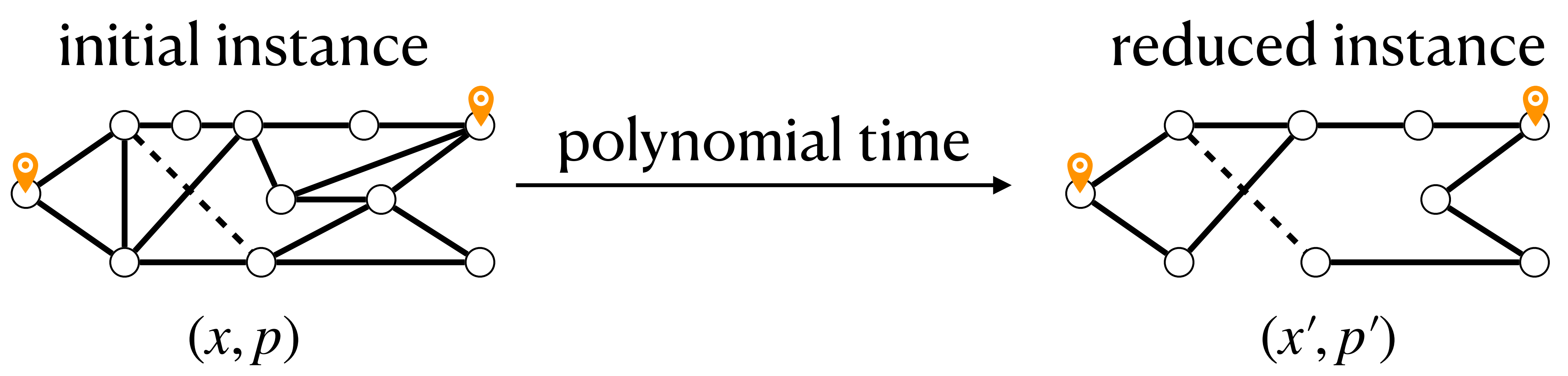
(x, p) is a yes-instance $\iff$ (x', p') is a yes-instance

p: parameter x: instance for classical problem

Parameterized and Kernelization Complexities

- A *parameterized problem* with an algorithm that runs in $g(p) \cdot |x|^c$ time, where g is a function, is in FPT.
- FPT ⊆ W[1] ⊆ W[2] ⊆ ... ⊆ XNLP
- XNLP-hard: W[t]-hard for every t.
- Every parameterized problem in FPT has a *kernel* [Cai et al. 1997].

Algorithm A



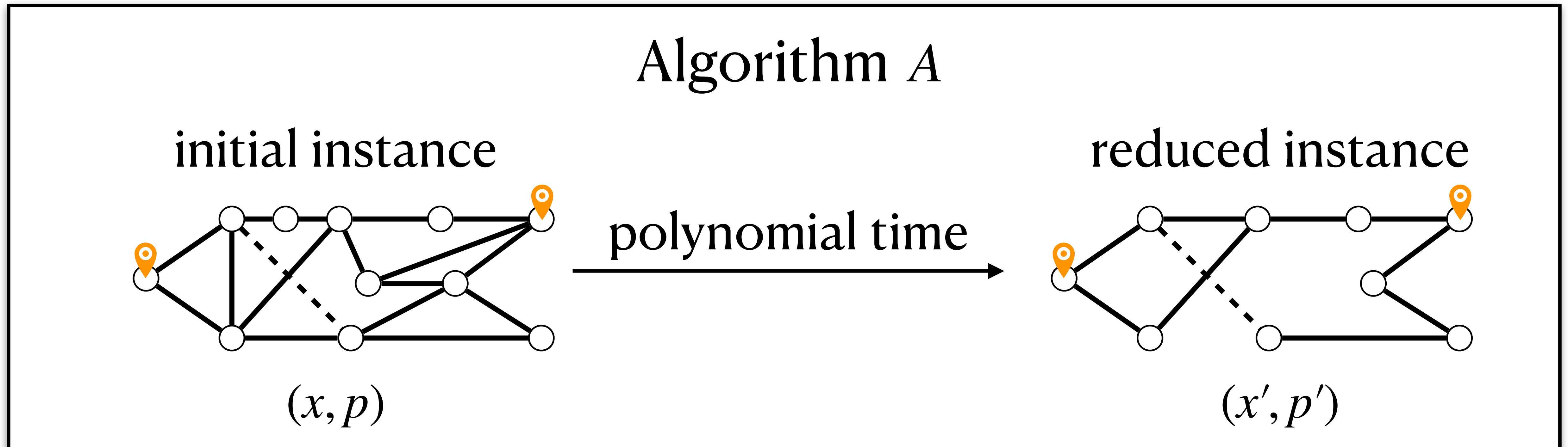
(x, p) is a yes-instance $\iff$ (x', p') is a yes-instance

$\exists f : \forall_{(x,p)}$ if $(x', p') \leftarrow A((x,p))$ then $|x'| + p' \leq f(p)$

p: parameter x: instance for classical problem

Parameterized and Kernelization Complexities

- A *parameterized problem* with an algorithm that runs in $g(p) \cdot |x|^c$ time, where g is a function, is in FPT.
- FPT ⊆ W[1] ⊆ W[2] ⊆ ... ⊆ XNLP
- XNLP-hard: W[t]-hard for every t.
- Every parameterized problem in FPT has a *kernel* [Cai et al. 1997].



(x, p) is a yes-instance $\iff$ (x', p') is a yes-instance
$\exists f : \forall_{(x,p)}$ if $(x', p') \leftarrow A((x, p))$ then $|x'| + p' \leq f(p)$

- Every parameterized problem in FPT has a kernel; not all are polynomial.

p: parameter x: instance for classical problem

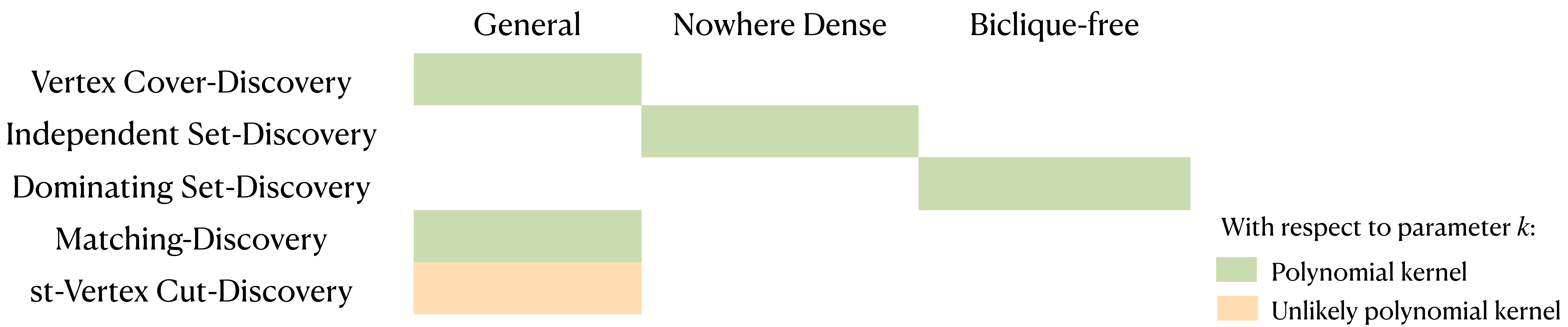
Our Results

- We studied the kernelization of the considered problems, unlike [Fellows et al, 2023] and [Grobler et al. 2024].

p: parameter x: instance for classical problem k: number of tokens b: number of token slides

Our Results

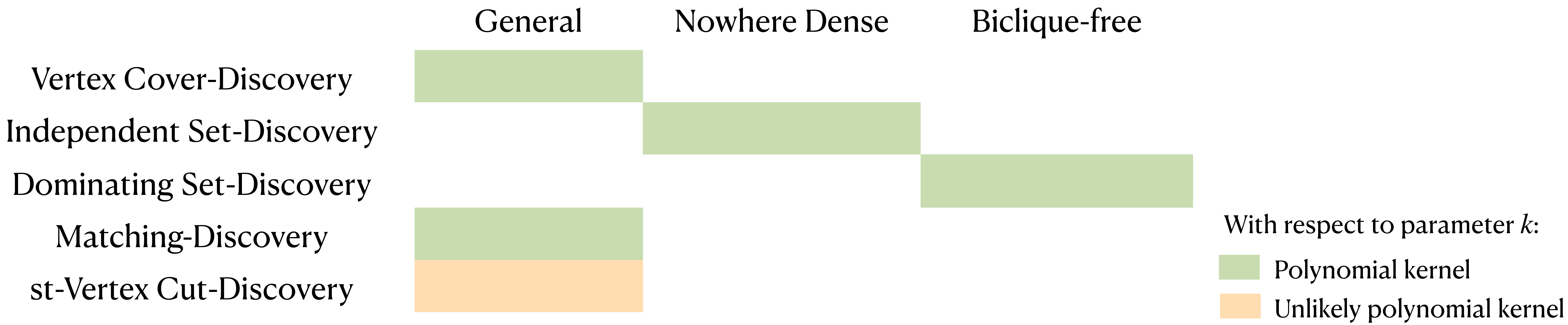
- We studied the kernelization of the considered problems, unlike [Fellows et al, 2023] and [Grobler et al. 2024].



p: parameter x: instance for classical problem k: number of tokens b: number of token slides

Our Results

- We studied the kernelization of the considered problems, unlike [Fellows et al, 2023] and [Grobler et al. 2024].

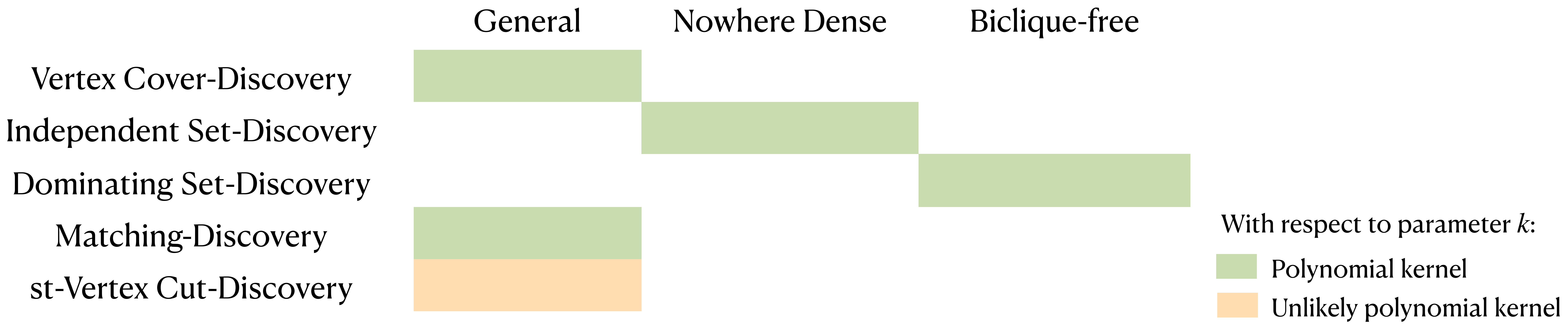


- Shortest Path-Discovery is unlikely to have a polynomial kernel with respect to parameter $k + b$ on general graphs.

p: parameter x: instance for classical problem k: number of tokens b: number of token slides

Our Results

- We studied the kernelization of the considered problems, unlike [Fellows et al, 2023] and [Grobler et al. 2024].



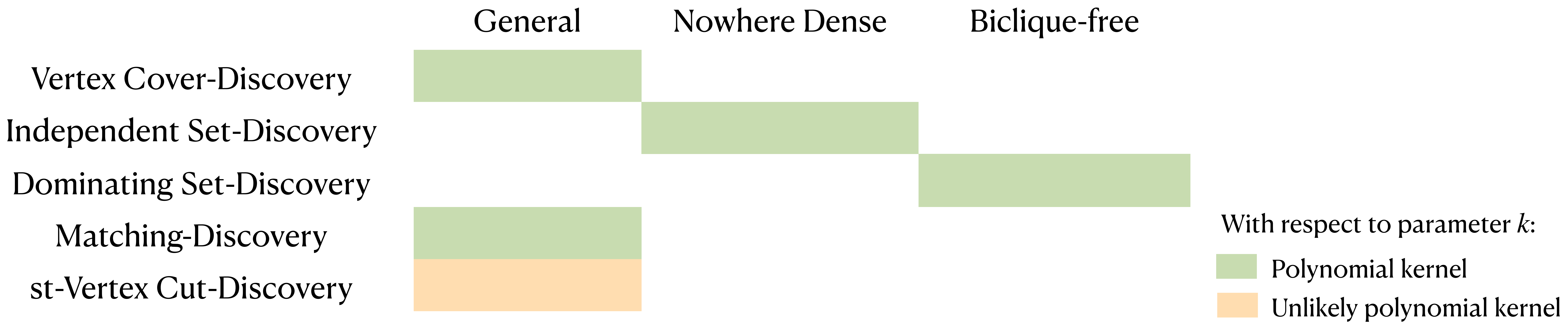
- Shortest Path-Discovery is unlikely to have a polynomial kernel with respect to parameter $k + b$ on general graphs.

- Vertex Cover-Discovery, Independent Set-Discovery, and Dominating Set-Discovery are unlikely to have a polynomial kernel with respect to parameter b+pathwidth, XNLP-hard with respect to parameter pathwidth, and $W[1]$-hard with respect to parameter feedback vertex set.

p: parameter x: instance for classical problem k: number of tokens b: number of token slides

Our Results

- We studied the kernelization of the considered problems, unlike [Fellows et al, 2023] and [Grobler et al. 2024].



- Shortest Path-Discovery is unlikely to have a polynomial kernel with respect to parameter $k + b$ on general graphs.

- Vertex Cover-Discovery, Independent Set-Discovery, and Dominating Set-Discovery are unlikely to have a polynomial kernel with respect to parameter b+pathwidth, XNLP-hard with respect to parameter pathwidth, and $W[1]$-hard with respect to parameter feedback vertex set.

p: parameter x: instance for classical problem k: number of tokens b: number of token slides

st-Vertex Cut Discovery with Respect to Parameter k

p: parameter x: instance for classical problem k: number of tokens b: number of token slides

st-Vertex Cut Discovery with Respect to Parameter k

- If an *or-cross-composition* from a classical NP-hard problem A into a parameterized problem B exists $\Rightarrow$ no polynomial kernel for B unless $NP \subseteq coNP \backslash poly$ [Bodlaender et al. 2014].

p: parameter x: instance for classical problem k: number of tokens b: number of token slides

st-Vertex Cut Discovery with Respect to Parameter k

- If an *or-cross-composition* from a classical NP-hard problem A into a parameterized problem B exists $\Rightarrow$ no polynomial kernel for B unless $NP \subseteq coNP \backslash poly$ [Bodlaender et al. 2014].
- An or-cross-composition includes a poly-time reduction from A to B.

p: parameter x: instance for classical problem k: number of tokens b: number of token slides

st-Vertex Cut Discovery with Respect to Parameter k

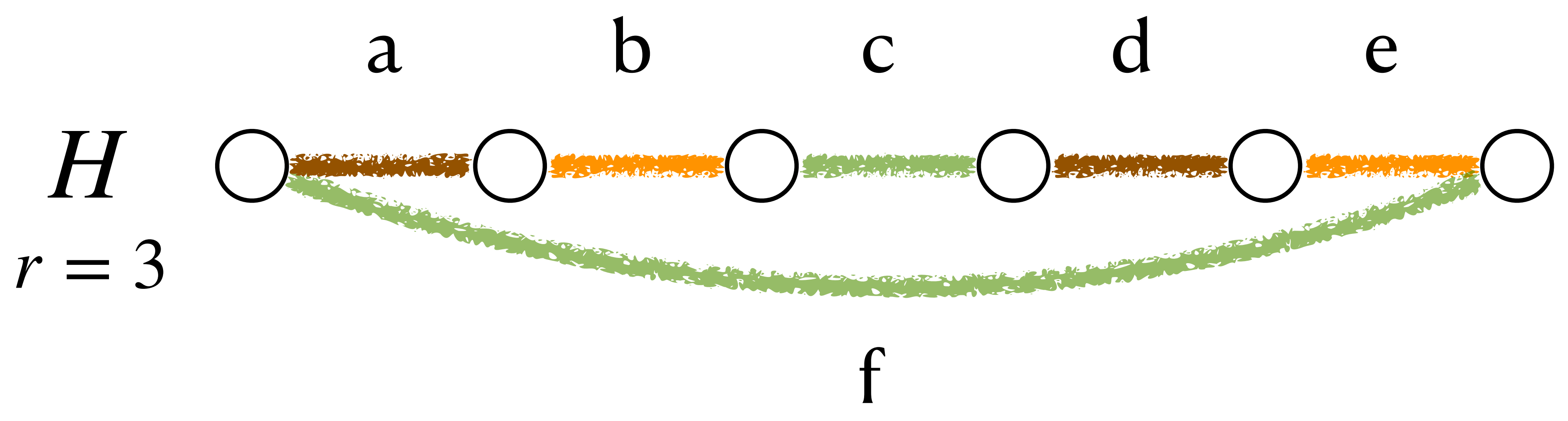
- If an *or-cross-composition* from a classical NP-hard problem A into a parameterized problem B exists $\Rightarrow$ no polynomial kernel for B unless $NP \subseteq coNP \backslash poly$ [Bodlaender et al. 2014].
- An or-cross-composition includes a poly-time reduction from A to B.

Rainbow Matching: NP-hard [Le and Pfender, 2012].

p: parameter x: instance for classical problem k: number of tokens b: number of token slides

st-Vertex Cut Discovery with Respect to Parameter k

- If an *or-cross-composition* from a classical NP-hard problem A into a parameterized problem B exists ⇒ no polynomial kernel for B unless $NP \subseteq coNP \backslash poly$ [Bodlaender et al. 2014].
- An or-cross-composition includes a poly-time reduction from A to B.



Rainbow Matching: NP-hard [Le and Pfender, 2012].

Input:

(i) 2-regular graph H,

(ii) edge coloring s.t. adjacent edges have different colors AND each color is used exactly twice,

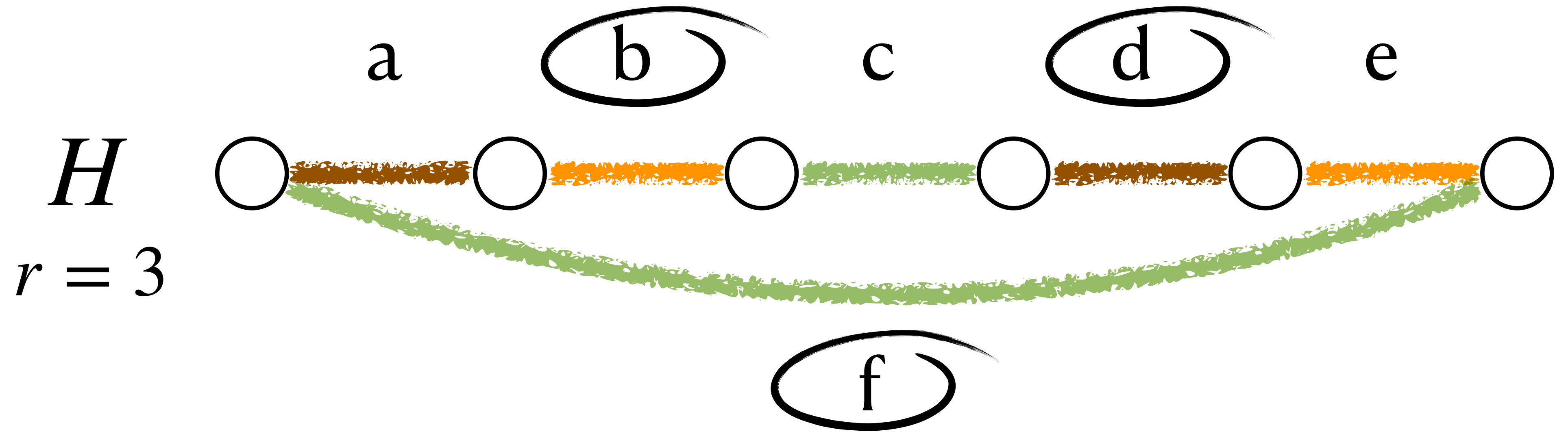
(iii) an integer r.

Output: If it exists, a *rainbow matching* (i.e. a matching whose edges have distinct colors) of size r.

p: parameter x: instance for classical problem k: number of tokens b: number of token slides

st-Vertex Cut Discovery with Respect to Parameter k

- If an *or-cross-composition* from a classical NP-hard problem A into a parameterized problem B exists ⇒ no polynomial kernel for B unless $NP \subseteq coNP\backslash poly$ [Bodlaender et al. 2014].
- An or-cross-composition includes a poly-time reduction from A to B.



Rainbow Matching: NP-hard [Le and Pfender, 2012].

Input:

(i) 2-regular graph H,

(ii) edge coloring s.t. adjacent edges have different colors AND each color is used exactly twice,

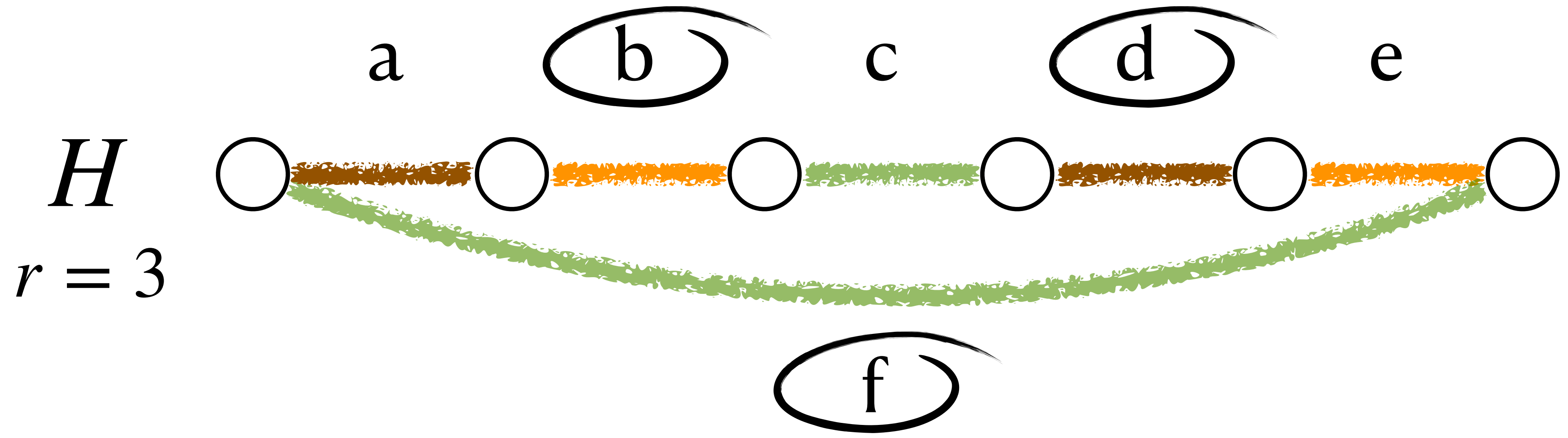
(iii) an integer r.

Output: If it exists, a *rainbow matching* (i.e. a matching whose edges have distinct colors) of size r.

p: parameter x: instance for classical problem k: number of tokens b: number of token slides

st-Vertex Cut Discovery with Respect to Parameter k

- If an *or-cross-composition* from a classical NP-hard problem A into a parameterized problem B exists ⇒ no polynomial kernel for B unless $NP \subseteq coNP\backslash poly$ [Bodlaender et al. 2014].
- An or-cross-composition includes a poly-time reduction from A to B.



Rainbow Matching: NP-hard [Le and Pfender, 2012].

Input:

(i) 2-regular graph H,

(ii) edge coloring s.t. adjacent edges have different colors AND each color is used exactly twice,

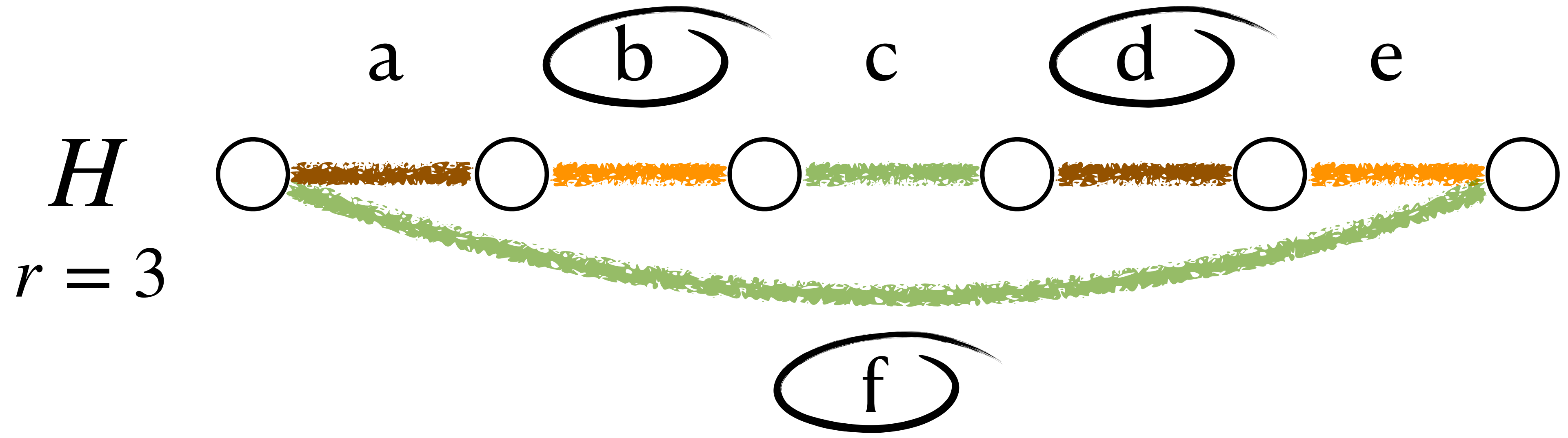
(iii) an integer r.

Output: If it exists, a *rainbow matching* (i.e. a matching whose edges have distinct colors) of size r.

p: parameter x: instance for classical problem k: number of tokens b: number of token slides

st-Vertex Cut Discovery with Respect to Parameter k

- If an *or-cross-composition* from a classical NP-hard problem A into a parameterized problem B exists ⇒ no polynomial kernel for B unless $NP \subseteq coNP \backslash poly$ [Bodlaender et al. 2014].
- An or-cross-composition includes a poly-time reduction from A to B.



Rainbow Matching: NP-hard [Le and Pfender, 2012].

Input:

(i) 2-regular graph H,

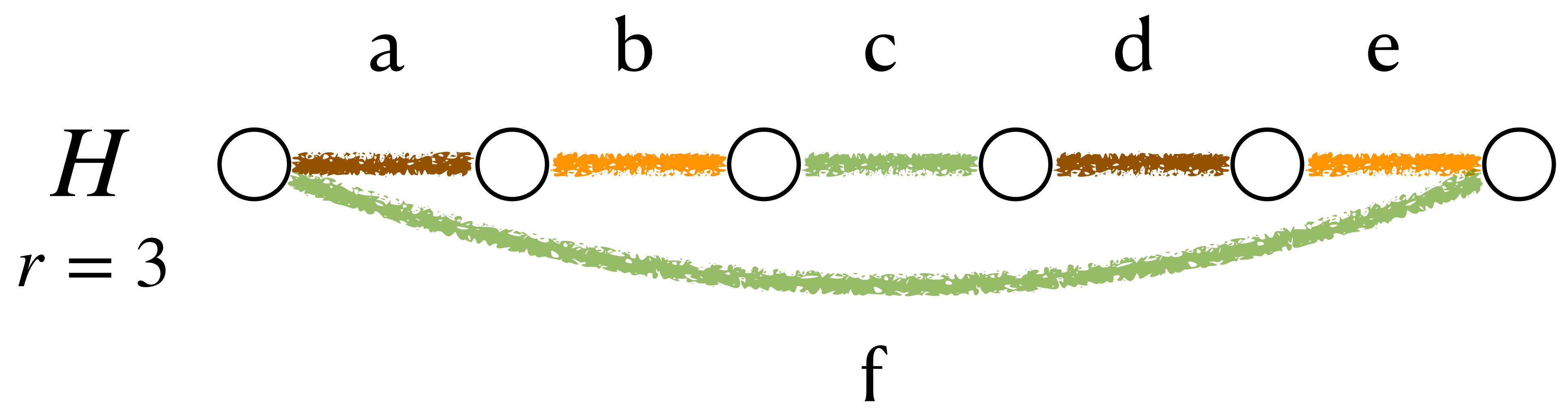
(ii) edge coloring s.t. adjacent edges have different colors AND each color is used exactly twice,

(iii) an integer r.

Output: If it exists, a *rainbow matching* (i.e. a matching whose edges have distinct colors) of size r.

p: parameter x: instance for classical problem k: number of tokens b: number of token slides s, t: vertices to cut r: size of matching

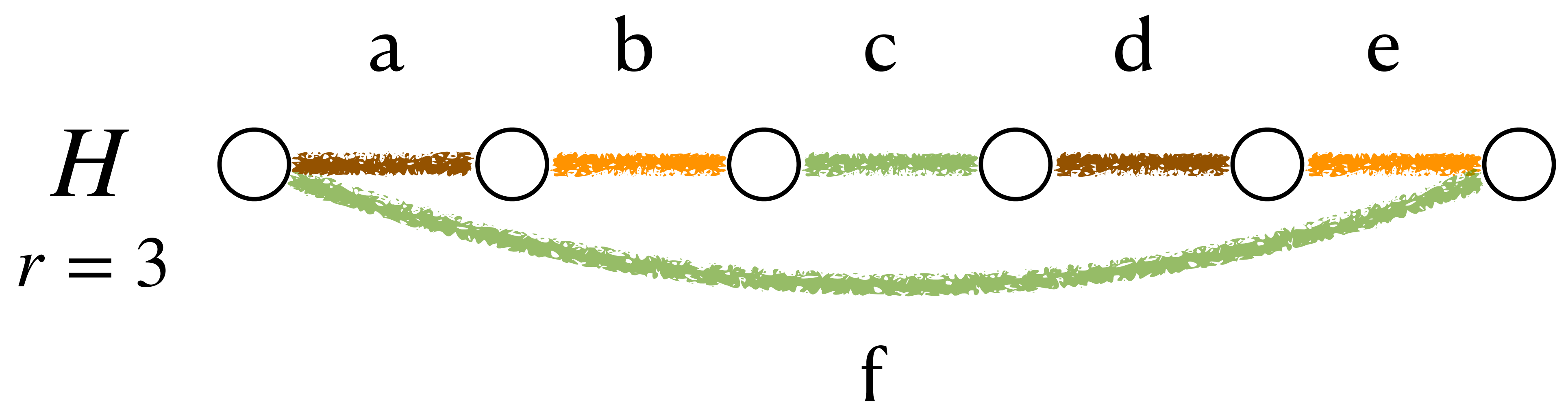
Reduction from Rainbow Matching to st-Vertex Cut Discovery



p: parameter x: instance for classical problem k: number of tokens b: number of token slides s, t: vertices to cut r: size of matching

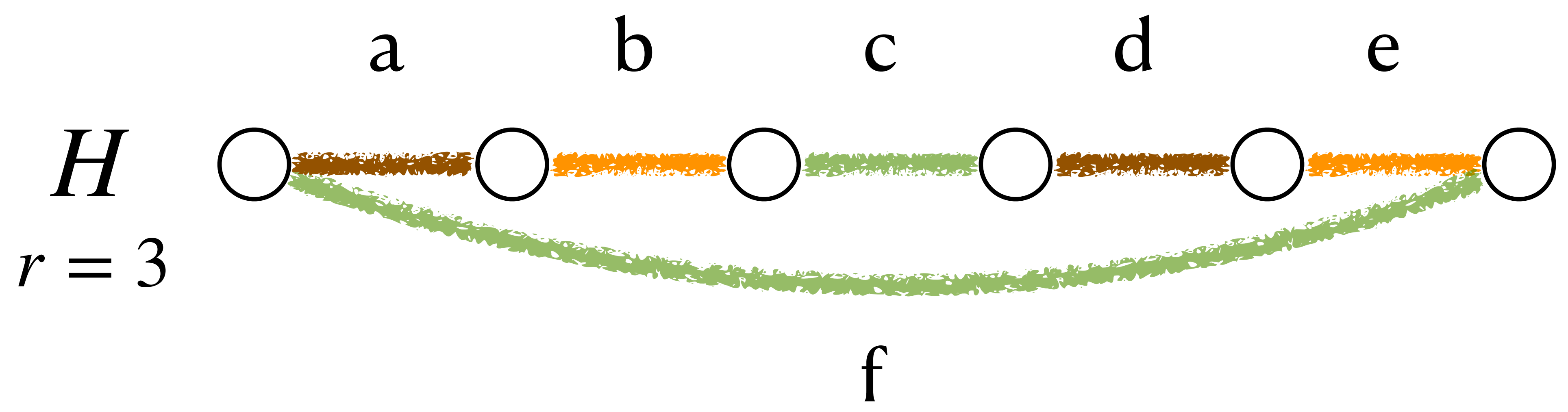
Reduction from Rainbow Matching to st-Vertex Cut Discovery

G



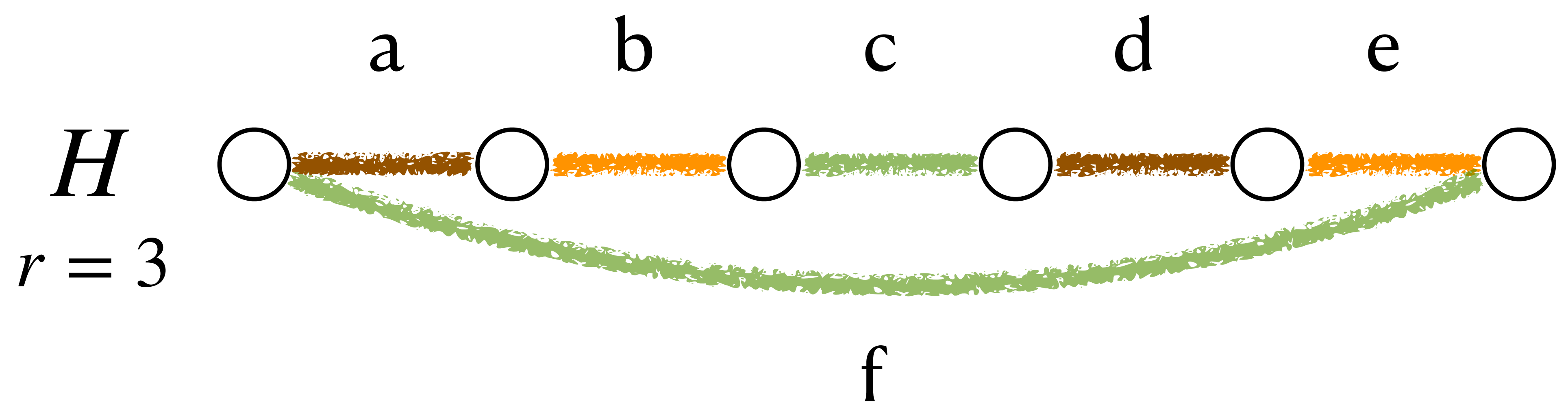
p: parameter x: instance for classical problem k: number of tokens b: number of token slides s, t: vertices to cut r: size of matching

Reduction from Rainbow Matching to st-Vertex Cut Discovery



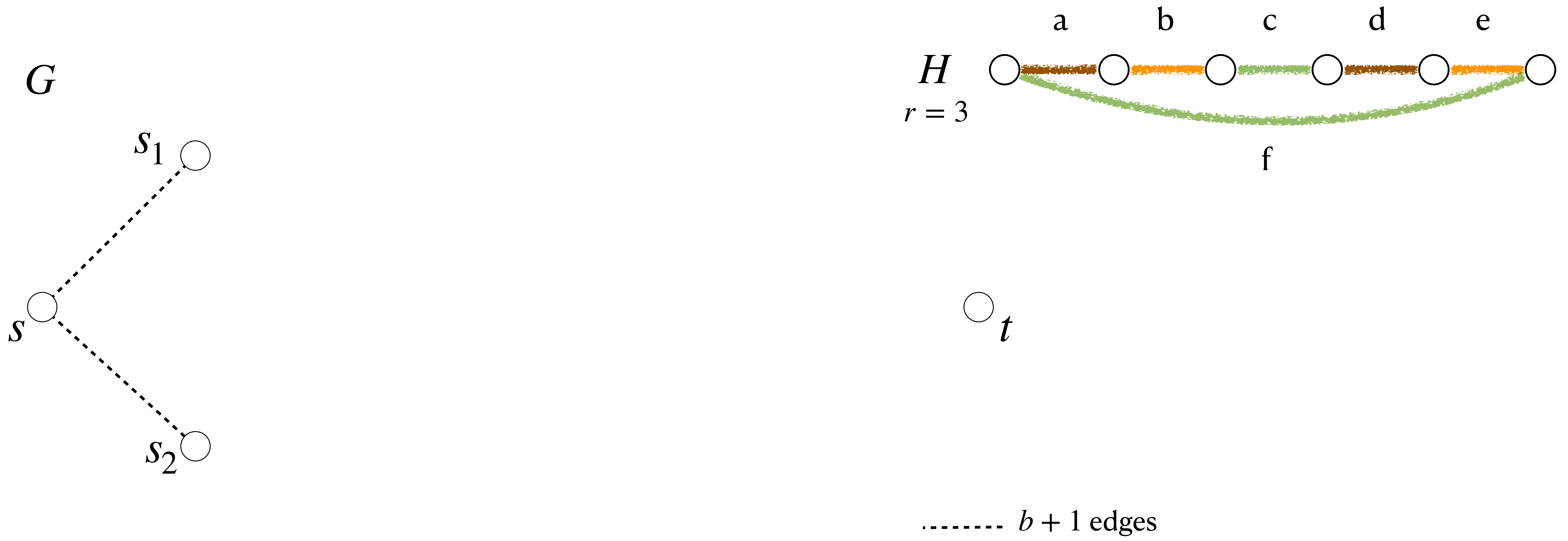
p: parameter x: instance for classical problem k: number of tokens b: number of token slides s, t: vertices to cut r: size of matching

Reduction from Rainbow Matching to st-Vertex Cut Discovery



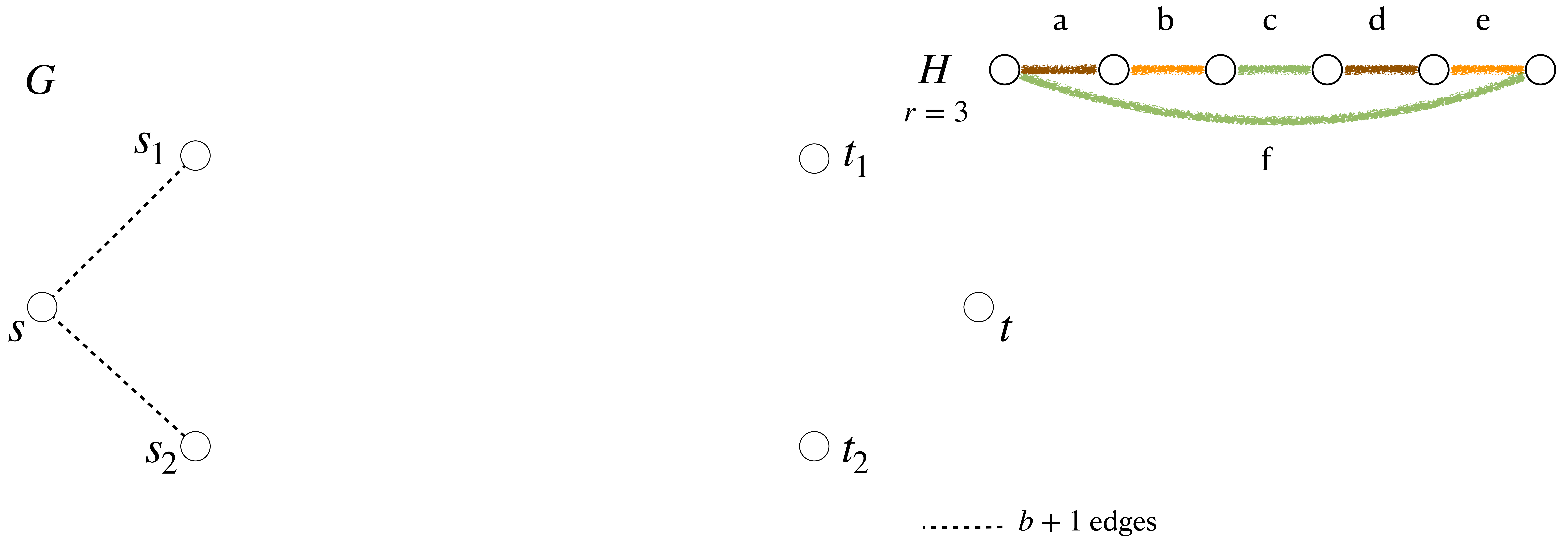
p: parameter x: instance for classical problem k: number of tokens b: number of token slides s, t: vertices to cut r: size of matching

Reduction from Rainbow Matching to st-Vertex Cut Discovery



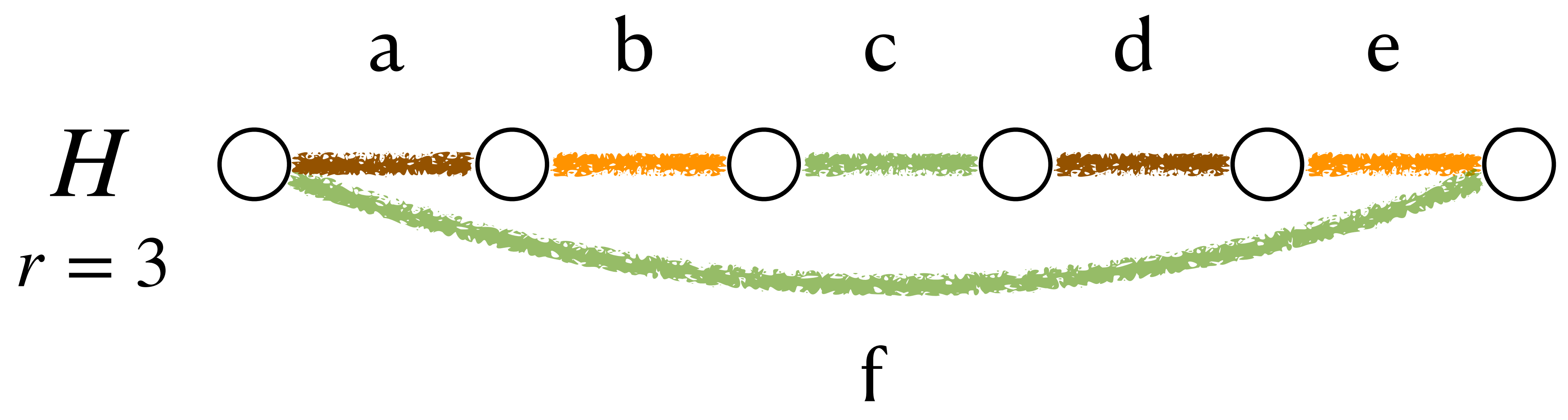
p: parameter x: instance for classical problem k: number of tokens b: number of token slides s, t: vertices to cut r: size of matching

Reduction from Rainbow Matching to st-Vertex Cut Discovery



p: parameter x: instance for classical problem k: number of tokens b: number of token slides s, t: vertices to cut r: size of matching

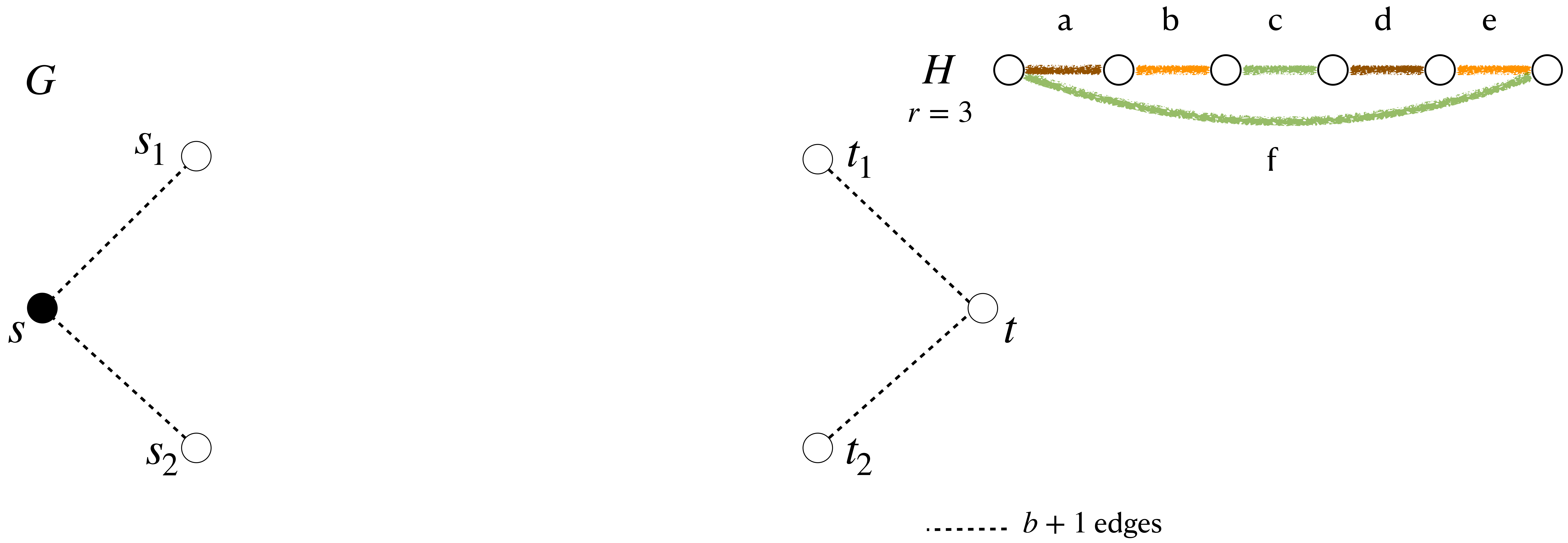
Reduction from Rainbow Matching to st-Vertex Cut Discovery



$b + 1$ edges

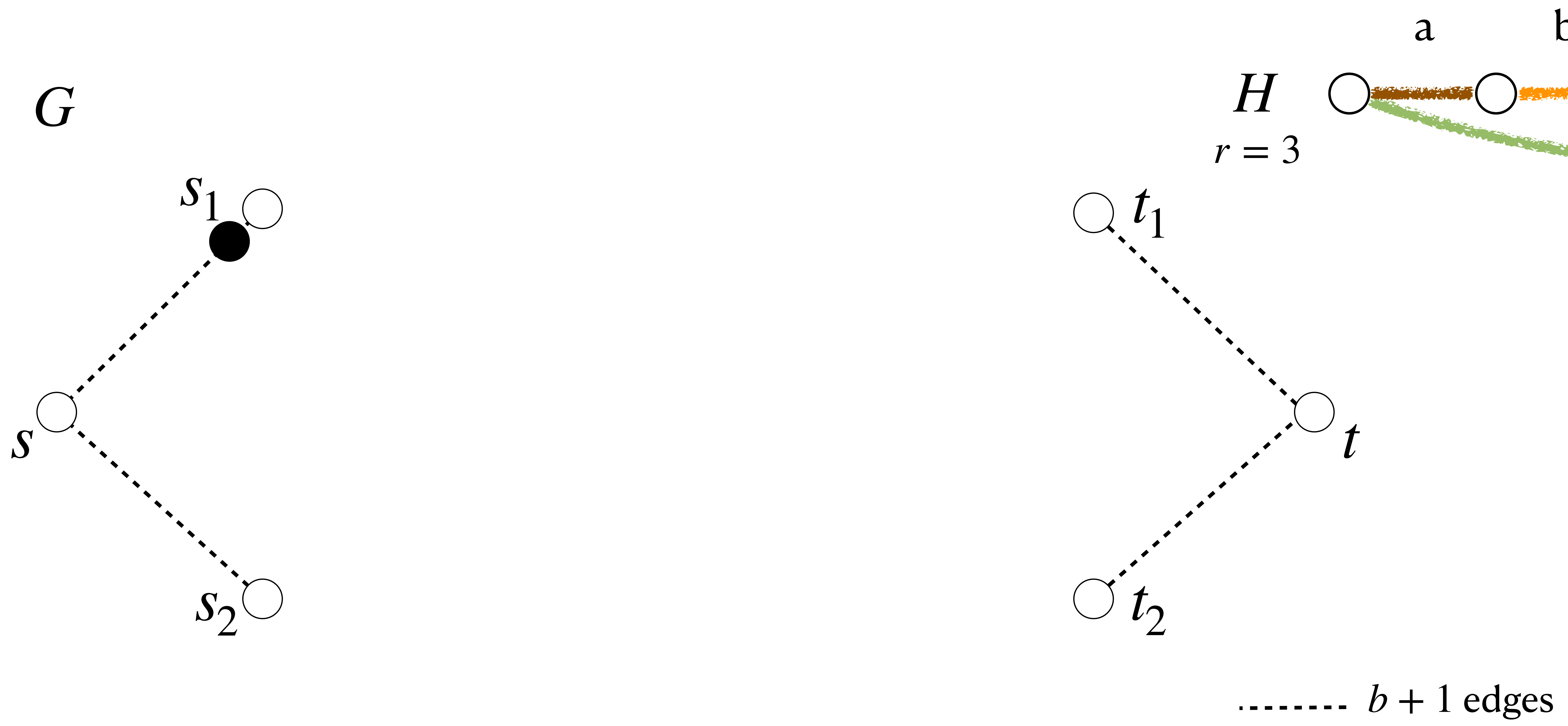
p: parameter x: instance for classical problem k: number of tokens b: number of token slides s, t: vertices to cut r: size of matching

Reduction from Rainbow Matching to st-Vertex Cut Discovery



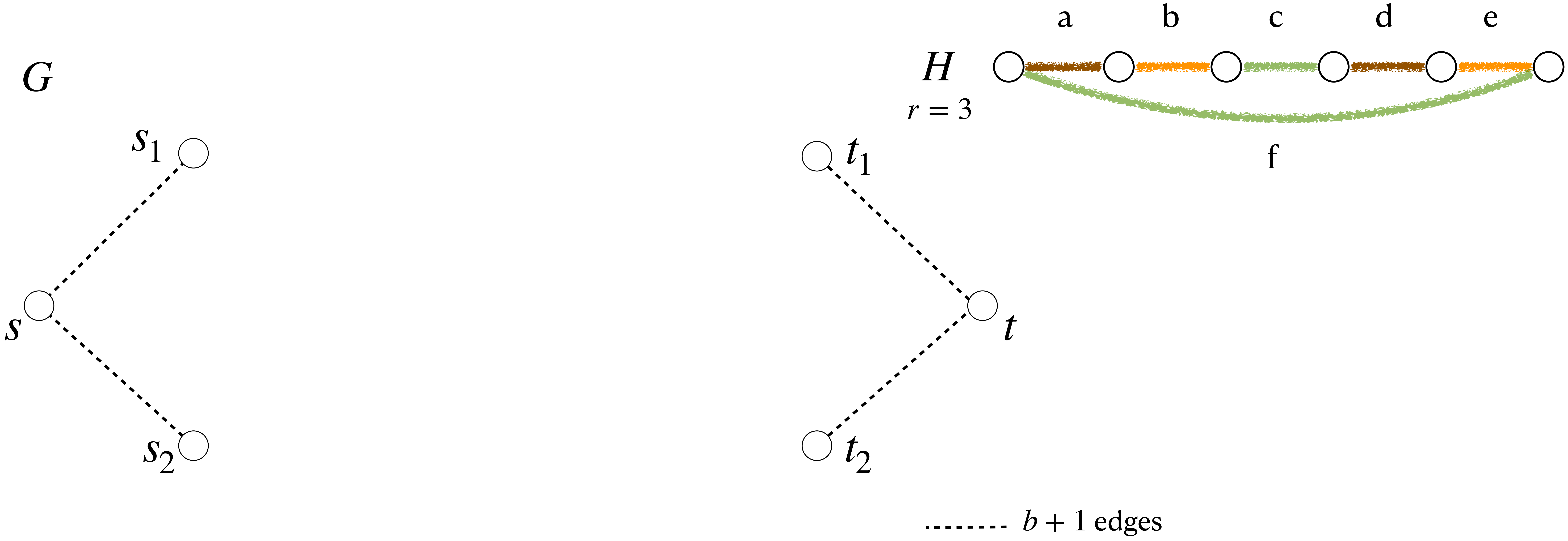
p: parameter x: instance for classical problem k: number of tokens b: number of token slides s, t: vertices to cut r: size of matching

Reduction from Rainbow Matching to st-Vertex Cut Discovery



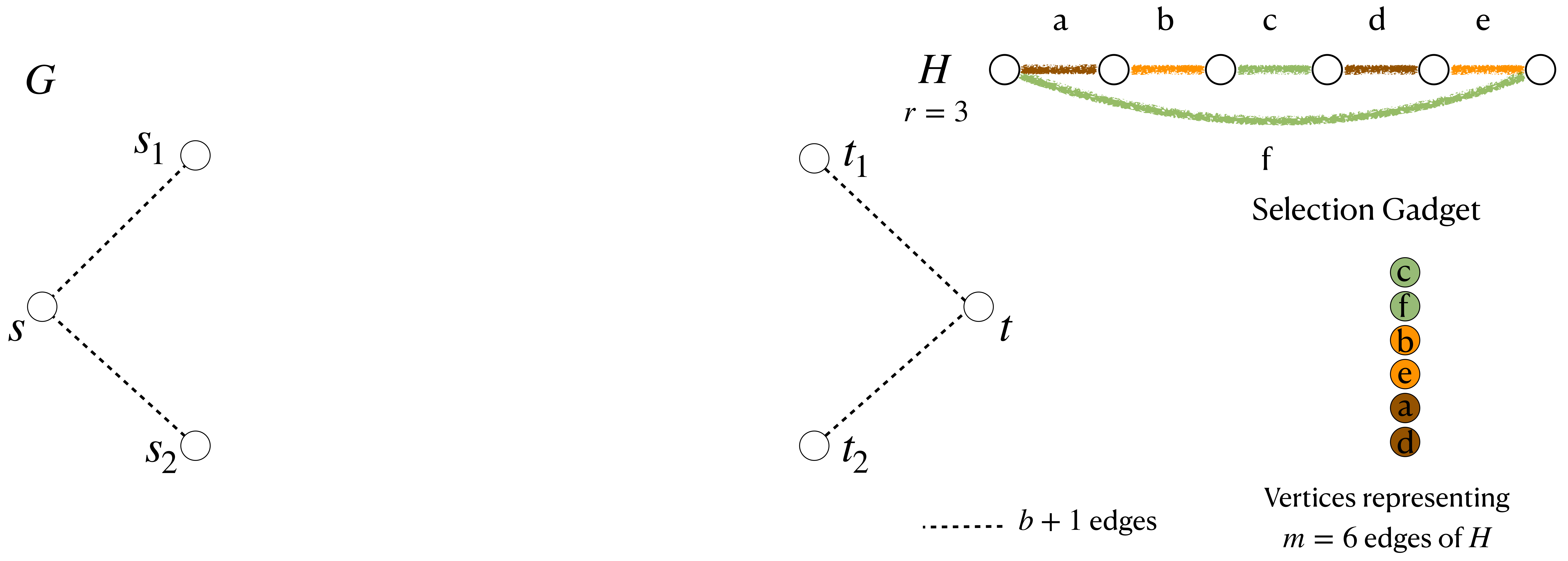
p: parameter x: instance for classical problem k: number of tokens b: number of token slides s, t: vertices to cut r: size of matching

Reduction from Rainbow Matching to st-Vertex Cut Discovery



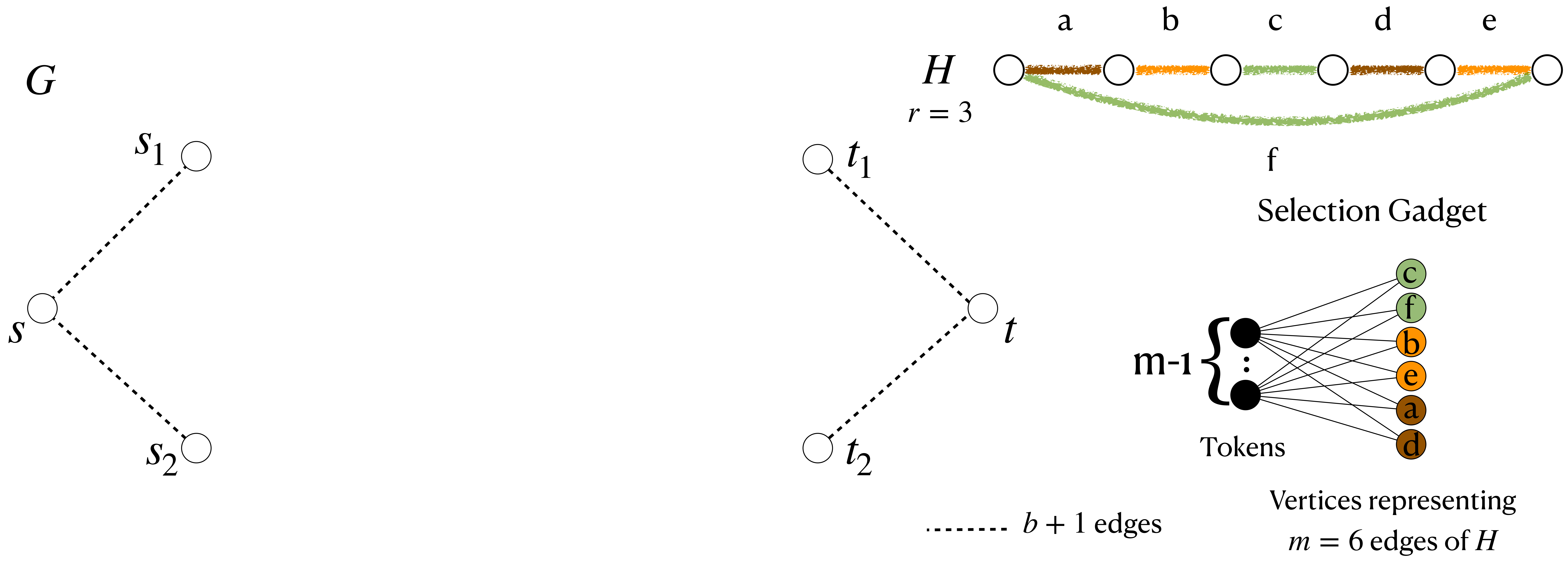
p: parameter x: instance for classical problem k: number of tokens b: number of token slides s, t: vertices to cut r: size of matching

Reduction from Rainbow Matching to st-Vertex Cut Discovery



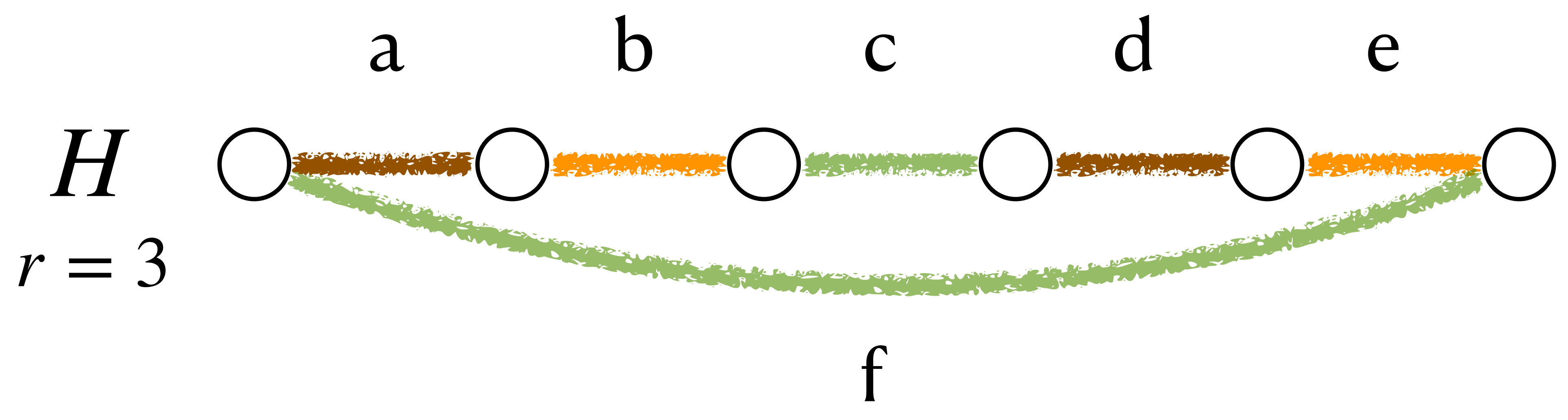
p: parameter x: instance for classical problem k: number of tokens b: number of token slides s, t: vertices to cut r: size of matching

Reduction from Rainbow Matching to st-Vertex Cut Discovery

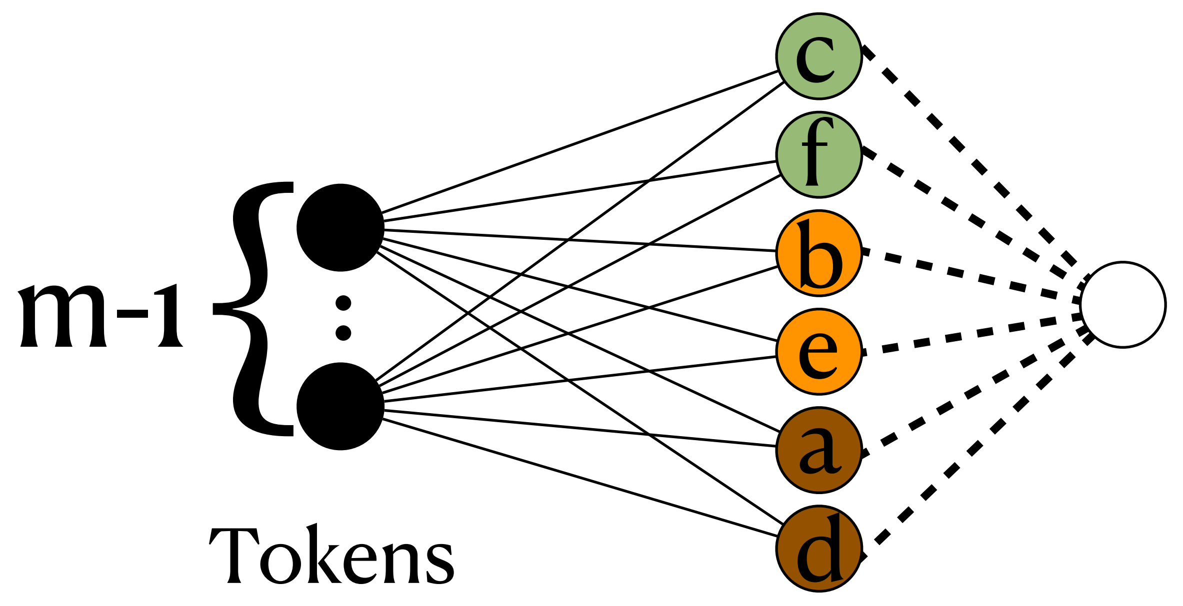


p: parameter x: instance for classical problem k: number of tokens b: number of token slides s, t: vertices to cut r: size of matching

Reduction from Rainbow Matching to st-Vertex Cut Discovery



Selection Gadget

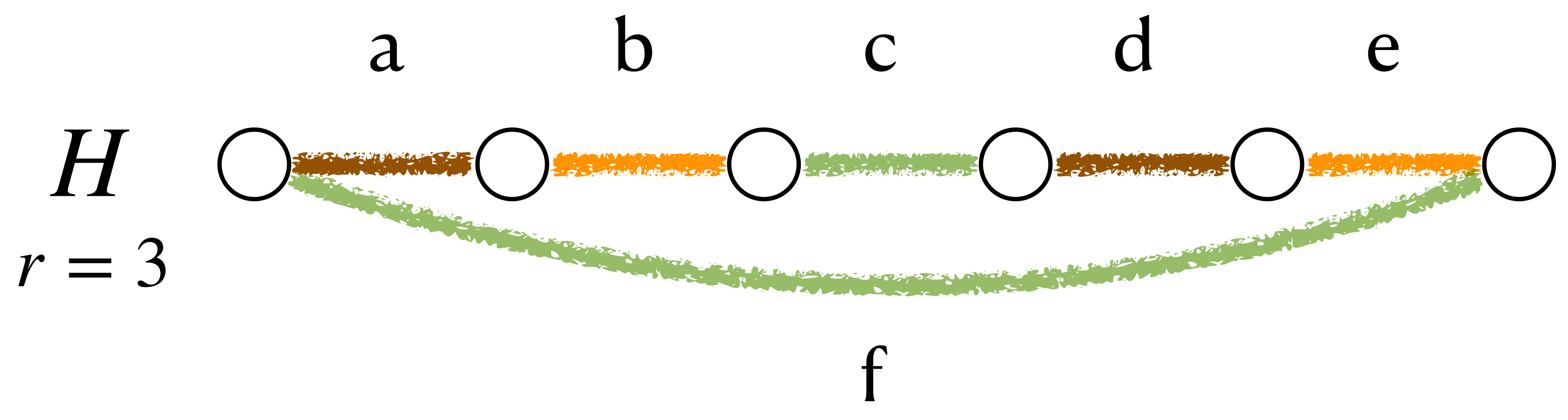


$b + 1$ edges

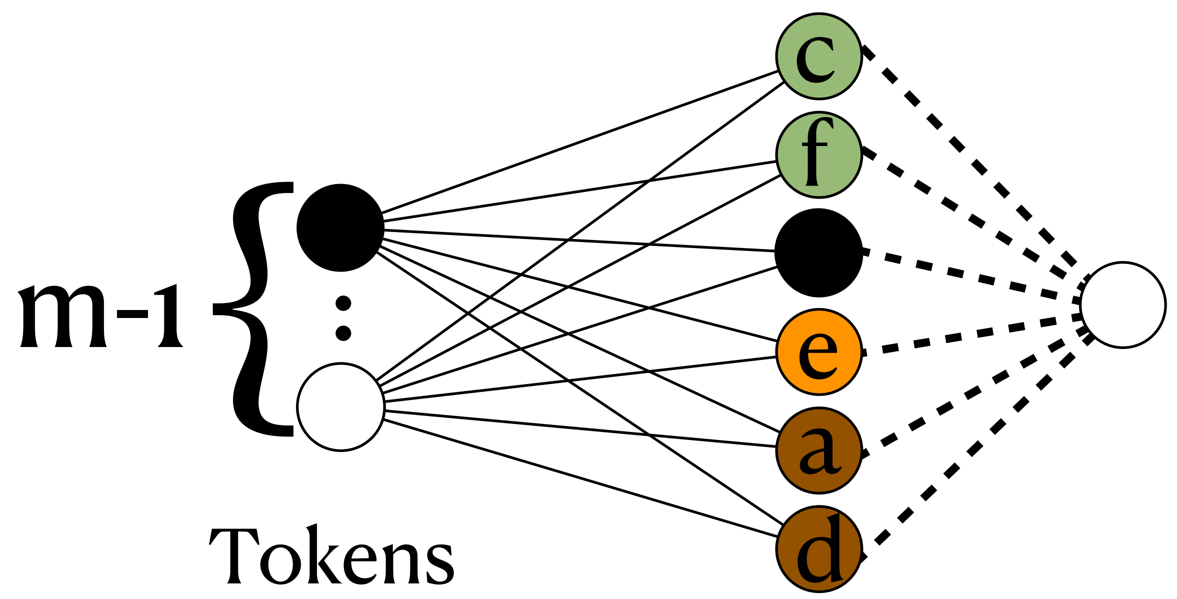
Vertices representing $m = 6$ edges of H

p: parameter x: instance for classical problem k: number of tokens b: number of token slides s, t: vertices to cut r: size of matching

Reduction from Rainbow Matching to st-Vertex Cut Discovery



Selection Gadget



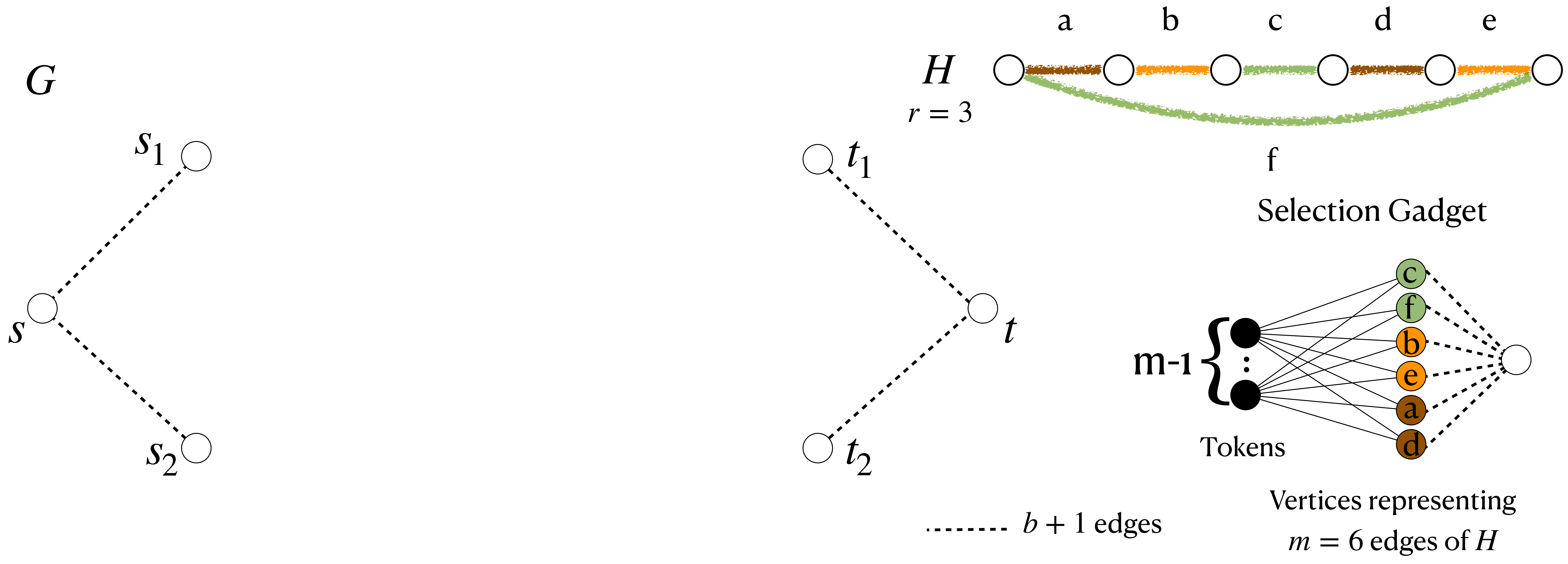
Tokens

Vertices representing $m = 6$ edges of H

.... $b + 1$ edges

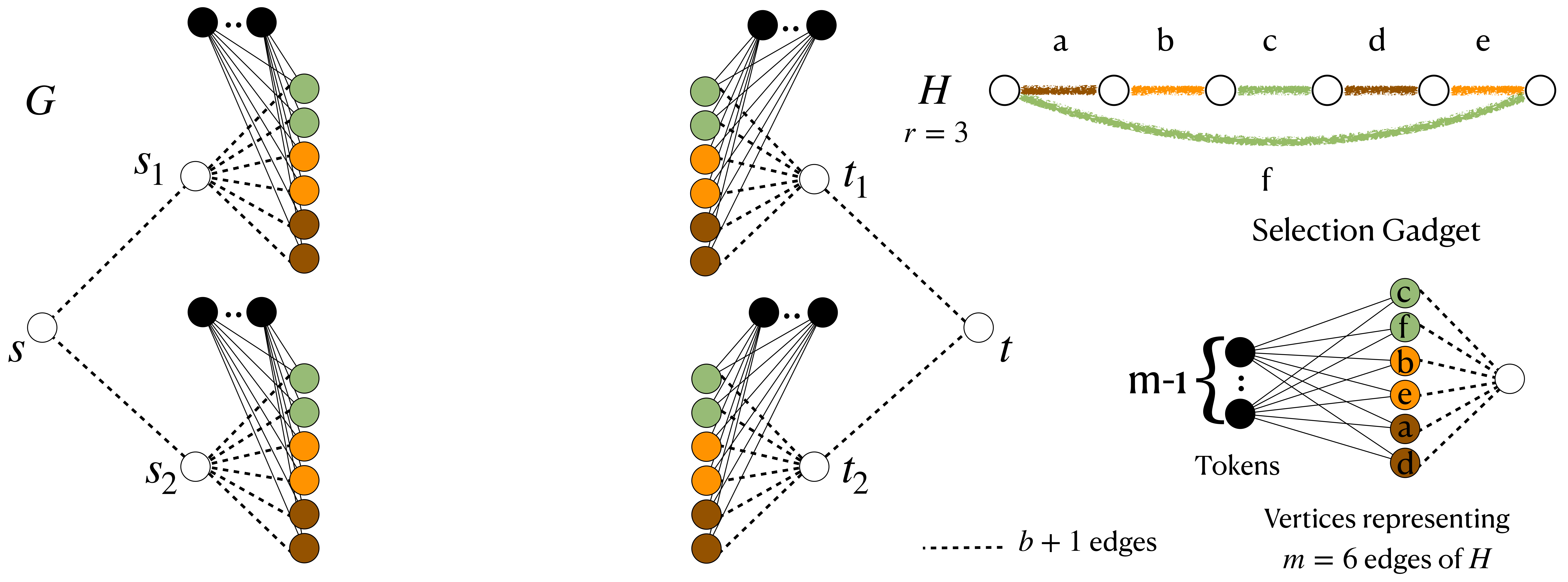
p: parameter x: instance for classical problem k: number of tokens b: number of token slides s, t: vertices to cut r: size of matching

Reduction from Rainbow Matching to st-Vertex Cut Discovery



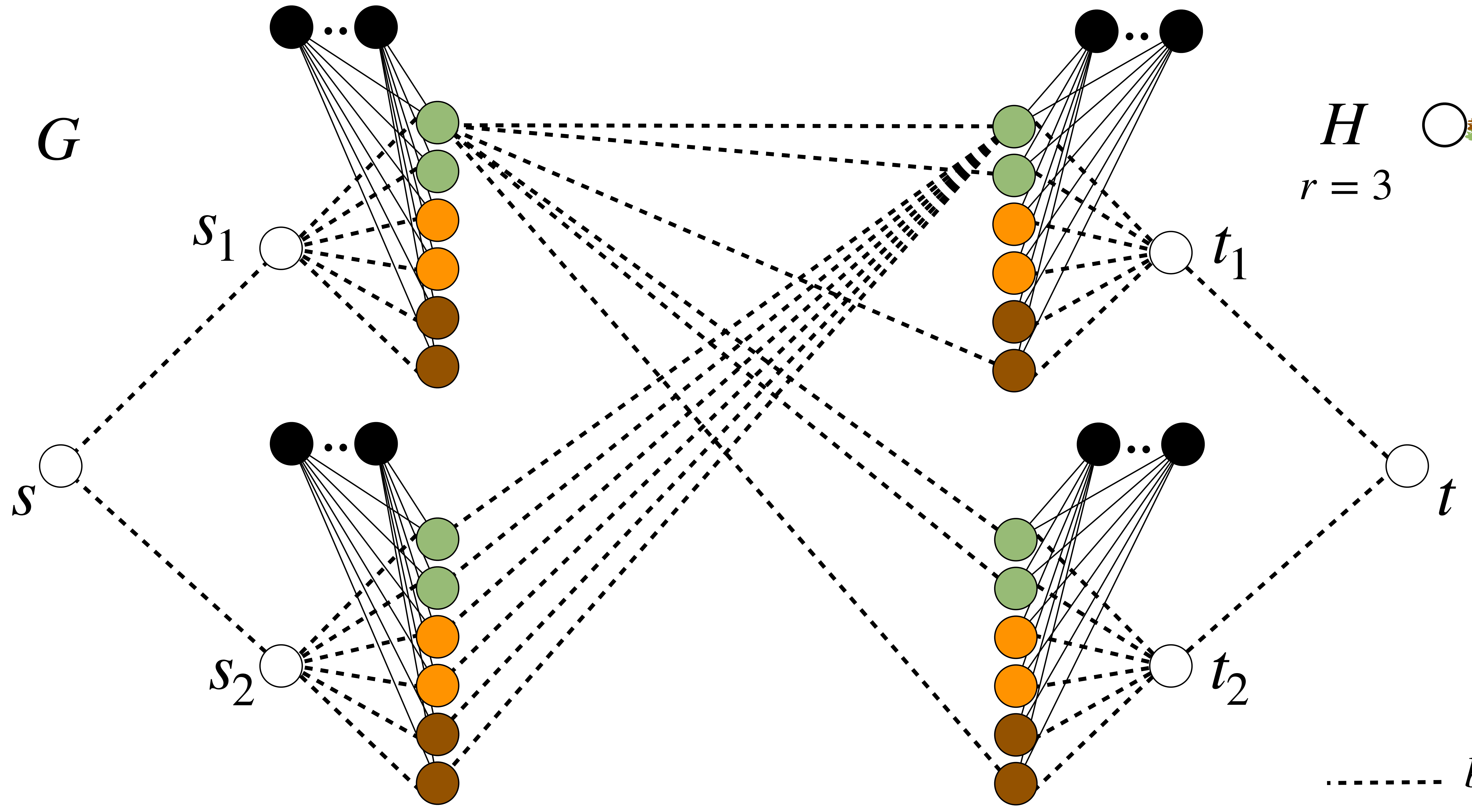
p: parameter x: instance for classical problem k: number of tokens b: number of token slides s, t: vertices to cut r: size of matching

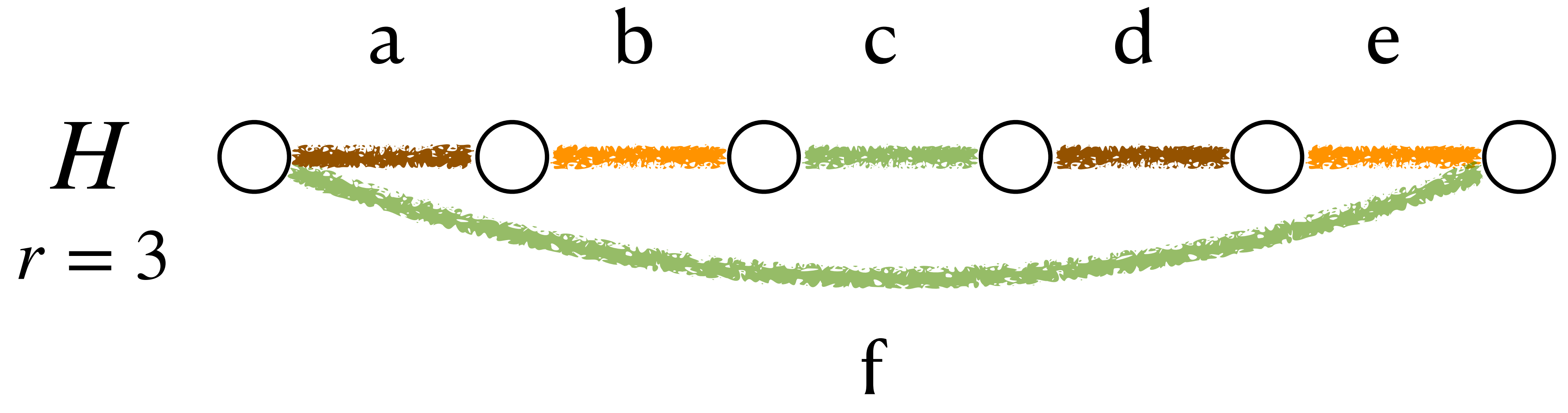
Reduction from Rainbow Matching to st-Vertex Cut Discovery



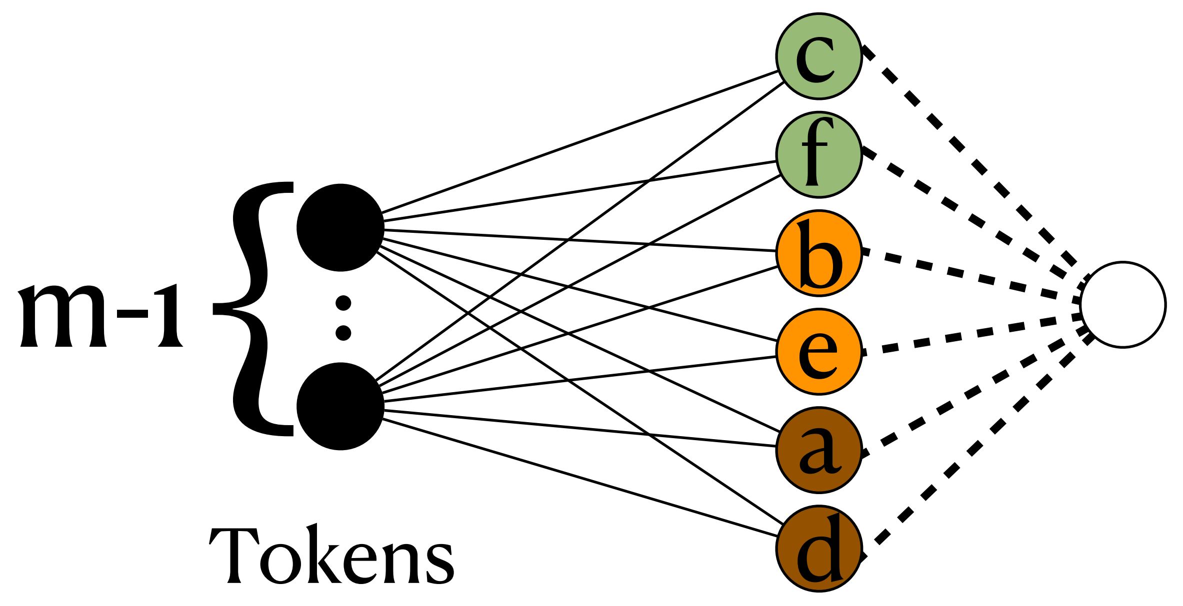
p: parameter x: instance for classical problem k: number of tokens b: number of token slides s, t: vertices to cut r: size of matching

Reduction from Rainbow Matching to st-Vertex Cut Discovery

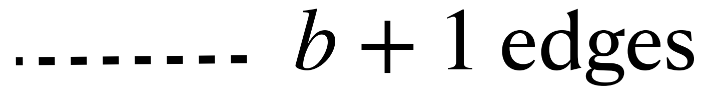




Selection Gadget

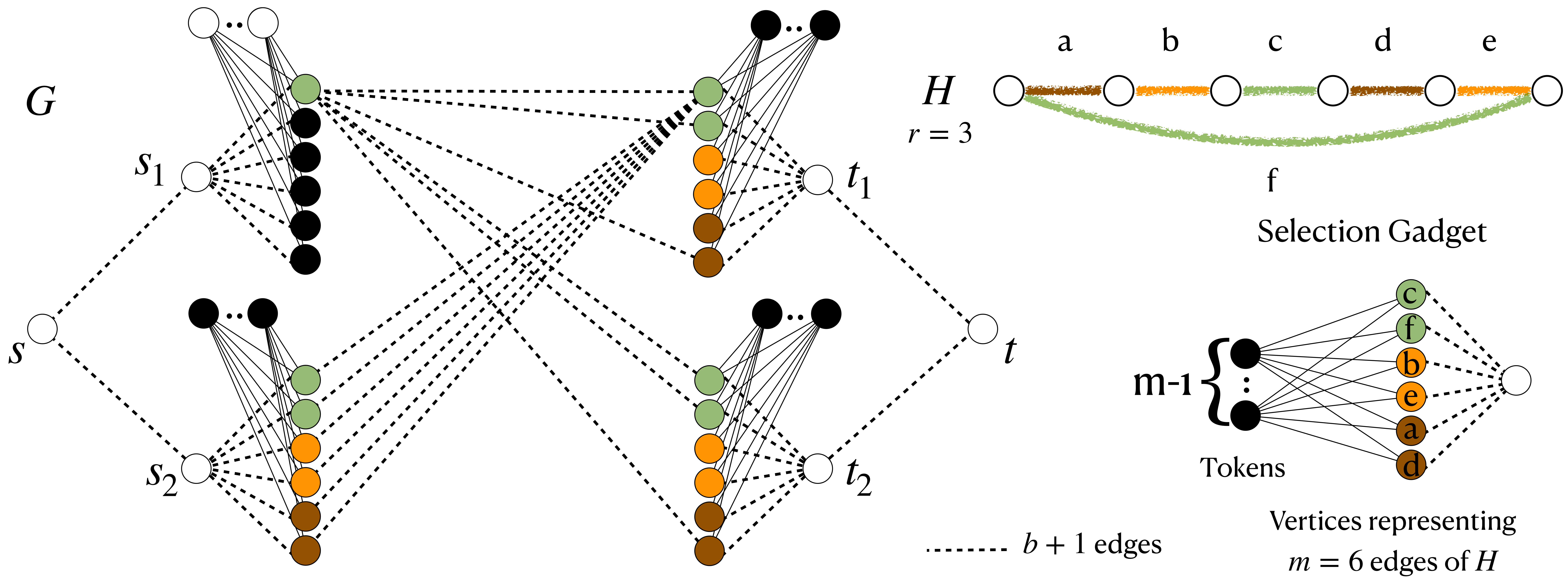


Vertices representing $m = 6$ edges of H



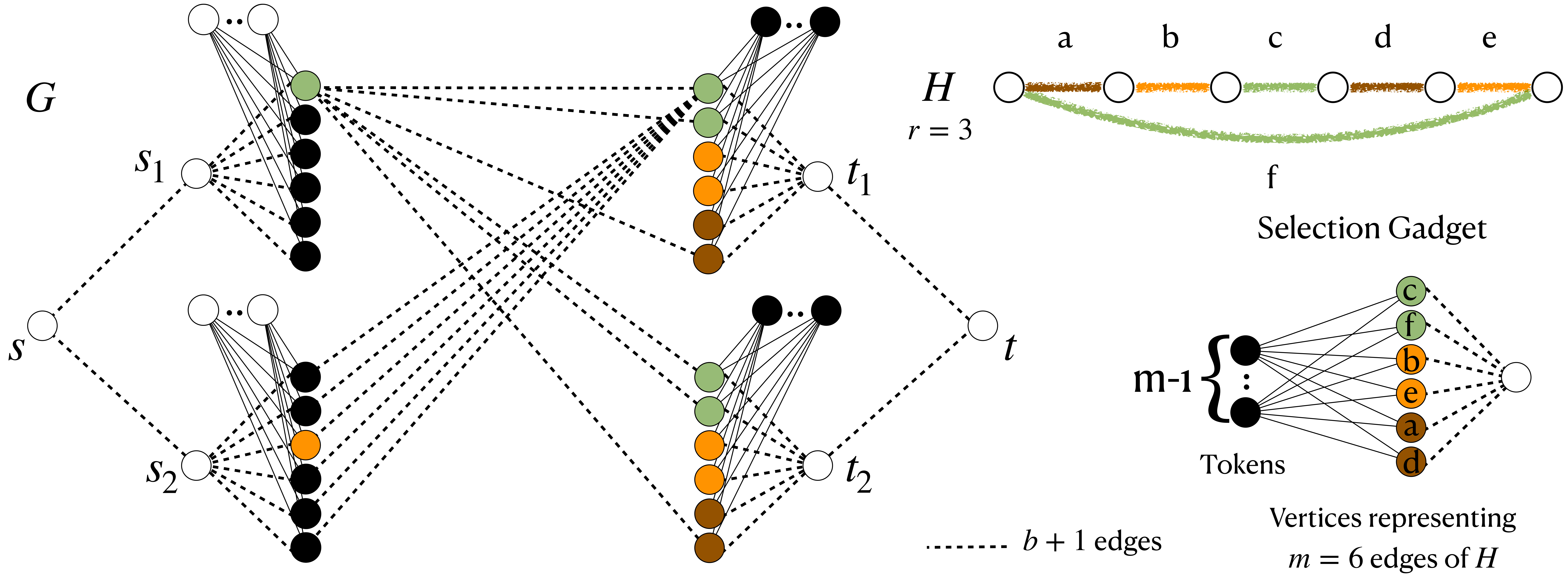
p: parameter x: instance for classical problem k: number of tokens b: number of token slides s, t: vertices to cut r: size of matching

Reduction from Rainbow Matching to st-Vertex Cut Discovery



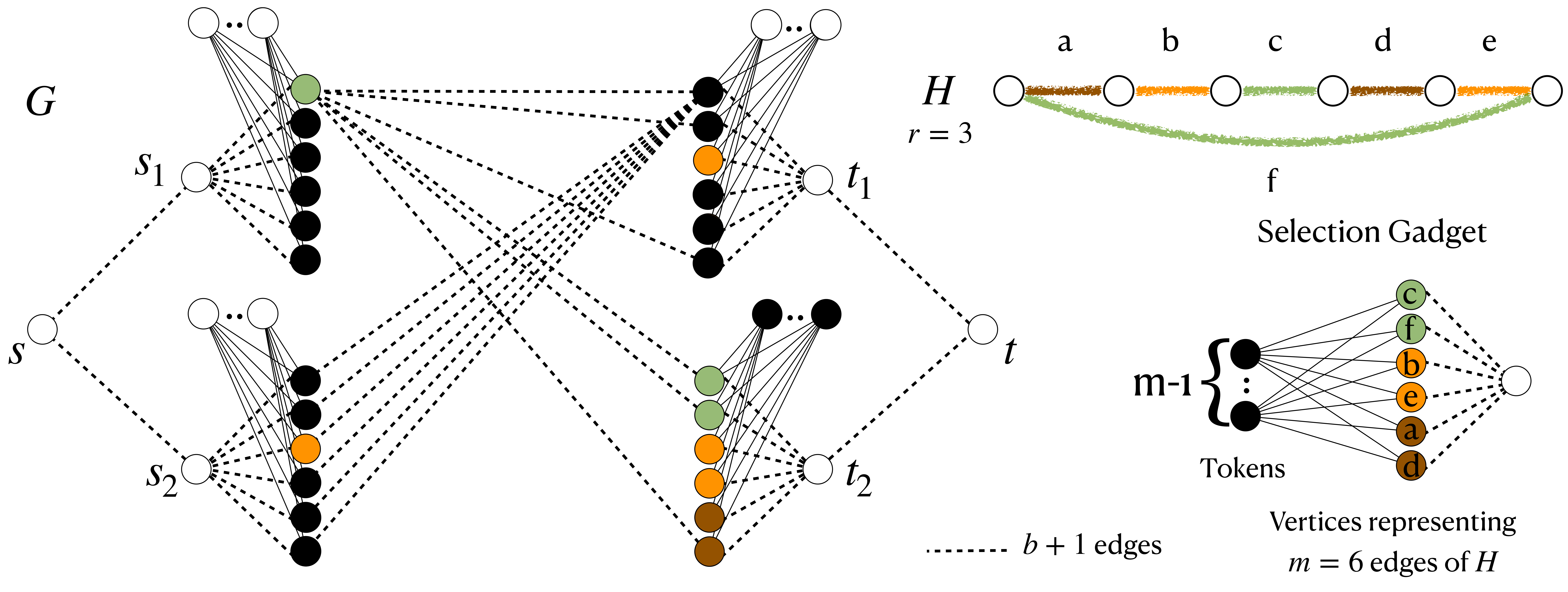
p: parameter x: instance for classical problem k: number of tokens b: number of token slides s, t: vertices to cut r: size of matching

Reduction from Rainbow Matching to st-Vertex Cut Discovery



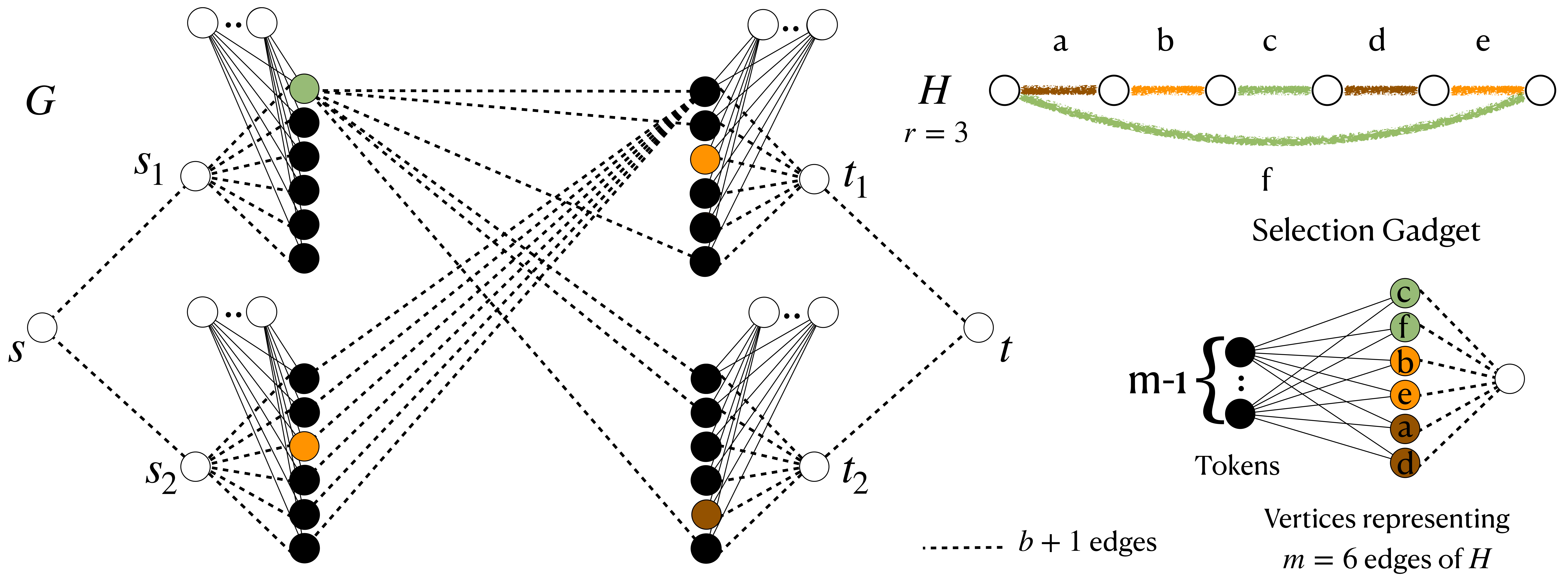
p: parameter x: instance for classical problem k: number of tokens b: number of token slides s, t: vertices to cut r: size of matching

Reduction from Rainbow Matching to st-Vertex Cut Discovery



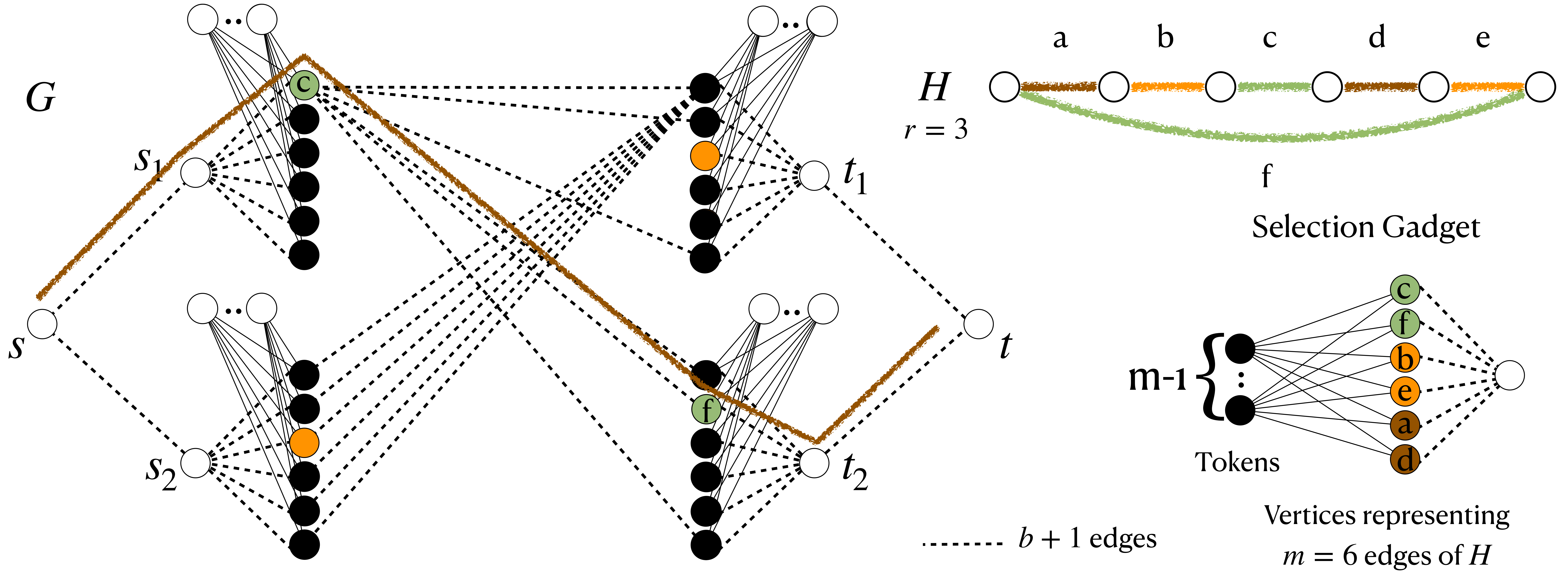
p: parameter x: instance for classical problem k: number of tokens b: number of token slides s, t: vertices to cut r: size of matching

Reduction from Rainbow Matching to st-Vertex Cut Discovery



p: parameter x: instance for classical problem k: number of tokens b: number of token slides s, t: vertices to cut r: size of matching

Reduction from Rainbow Matching to st-Vertex Cut Discovery



- If a cut can be reached in fewer than b token slides, the vertices that remain free in the gadgets represent the edges of the rainbow matching (and vice versa).

p: parameter x: instance for classical problem k: number of tokens b: number of token slides s, t: vertices to cut r: size of matching

UNIVERSITY OF WATERLOO

Naomi Nishimura

Mario Grobler

Thank you.

Vijayaragunathan Ramamoorthi

Stephanie Maaz
smaaz@uwaterloo.ca

Sebastian Siebertz

Amer E. Mouawad