CMPT 307: Shortest Paths & Maximum Matchings

Week 12-2

Xian Qiu

Simon Fraser University

xianq@sfu.ca

All-pairs Shortest Path Problem

- given: digraph $G = (V, E)$ and cost function $c : E \to \mathbb{R}$
- goal: find shortest s-t paths, for all $(s, t) \in V \times V$

All-pairs Shortest Path Problem

- given: digraph $G = (V, E)$ and cost function $c : E \to \mathbb{R}$
- goal: find shortest s-t paths, for all $(s, t) \in V \times V$
- run Bellman-Ford algorithm n times, with running time $O(n^2 m)$ ($\Theta(n^4)$ for dense graphs)

All-pairs Shortest Path Problem

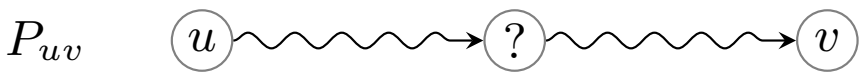
- given: digraph $G = (V, E)$ and cost function $c : E \to \mathbb{R}$
- goal: find shortest s-t paths, for all $(s, t) \in V \times V$
- run Bellman-Ford algorithm n times, with running time $O(n^2 m)$ ($\Theta(n^4)$ for dense graphs)
- can we do better?

All-pairs Shortest Path Problem

- given: digraph $G = (V, E)$ and cost function $c : E \to \mathbb{R}$
- goal: find shortest s-t paths, for all $(s, t) \in V \times V$
- run Bellman-Ford algorithm n times, with running time $O(n^2 m)$ ($\Theta(n^4)$ for dense graphs)
- can we do better?
- dynamic programming

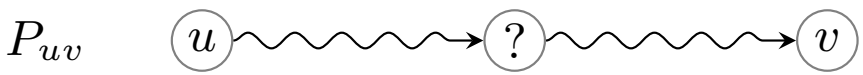
Optimal Substructure

- consider a shortest (u, v)-path P_{uv}
- with internal nodes $\subseteq V_k := \{1, 2, \ldots, k\}$
- $\delta_k(u, v) =$ cost of path P_{uv}



Optimal Substructure

- consider a shortest (u, v)-path P_{uv}
- with internal nodes $\subseteq V_k := \{1, 2, \ldots, k\}$
- $\delta_k(u, v) =$ cost of path P_{uv}

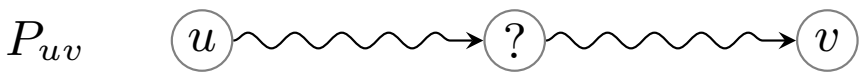


case 1: k is not an interior node of P_{uv}

case 2: k is an interior node of P_{uv}

Optimal Substructure

- consider a shortest (u, v)-path P_{uv}
- with internal nodes $\subseteq V_k := \{1, 2, \ldots, k\}$
- $\delta_k(u, v) =$ cost of path P_{uv}



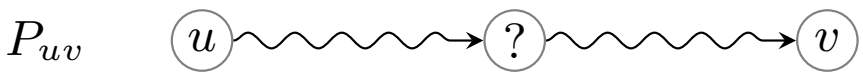
case 1: k is not an interior node of P_{uv}

- $\delta_k(u, v) = \delta_{k-1}(u, v)$

case 2: k is an interior node of P_{uv}

Optimal Substructure

- consider a shortest (u, v)-path P_{uv}
- with internal nodes $\subseteq V_k := \{1, 2, \ldots, k\}$
- $\delta_k(u, v) =$ cost of path P_{uv}



case 1: k is not an interior node of P_{uv}

- $\delta_k(u, v) = \delta_{k-1}(u, v)$

case 2: k is an interior node of P_{uv}

- $\delta_k(u, v) = \delta_{k-1}(u, k) + \delta_{k-1}(k, v)$

Recursive Formula

$$\delta_k(u,v) = \min\left\{\delta_{k-1}(u,v), \delta_{k-1}(u,k) + \delta_{k-1}(k,v)\right\}, \quad k \geq 1$$

boundary conditions

$$\delta_0(u,v) = \begin{cases} 0, & \text{if } u = v \\ c(u,v) & \text{if } (u,v) \in E \\ \infty & \text{otherwise} \end{cases}$$

Recursive Formula

$$\delta_k(u,v) = \min\left\{\delta_{k-1}(u,v), \delta_{k-1}(u,k) + \delta_{k-1}(k,v)\right\}, \quad k \geq 1$$

boundary conditions

$$\delta_0(u,v) = \begin{cases} 0, & \text{if } u = v \\ c(u,v) & \text{if } (u,v) \in E \\ \infty & \text{otherwise} \end{cases}$$

running time: $O(n^3)$

Floyd-Warshall Algorithm

- input: a graph represented by (weighted) adjacency matrix W

FLOYD-WARSHALL(W)

```
1 n = W.rows;
2 D^(0) = W;                                          // boundary conditions
3 for k = 1 to n do
4     let D^k = (d_ij^(k)) be a new n × n matrix;
5     for i = 1 to n do
6         for j = 1 to n do
7             d_ij^(k) = min{ d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1) };
```

Floyd-Warshall Algorithm

- ▷ input: a graph represented by (weighted) adjacency matrix W

FLOYD-WARSHALL(W)

```
1 n = W.rows;
2 D^(0) = W;                                          // boundary conditions
3 for k = 1 to n do
4     let D^k = (d_ij^(k)) be a new n × n matrix;
5     for i = 1 to n do
6         for j = 1 to n do
7             d_ij^(k) = min{ d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1) };
```

- ▷ given $i, j \in V$, how to construct a shortest (i, j)-path?

Matching

given an undirected graph $G = (V, E)$

- ▷ a matching is a set $M \subseteq E$ of pairwise non-incident edges

Matching

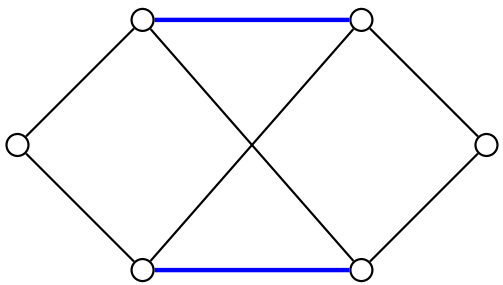
given an undirected graph $G = (V, E)$

- a matching is a set $M \subseteq E$ of pairwise non-incident edges
- perfect matching: $|M| = |V|/2$

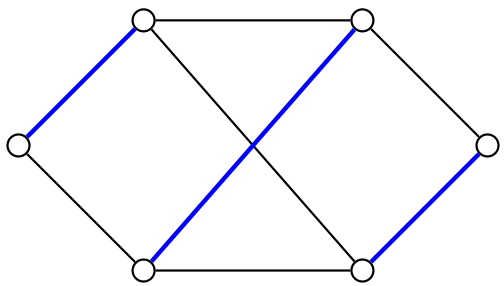
Matching

given an undirected graph $G = (V, E)$

- ▷ a matching is a set $M \subseteq E$ of pairwise non-incident edges
- ▷ perfect matching: $|M| = |V|/2$
- ▷ maximum matching is matching of maximal cardinality



maximal

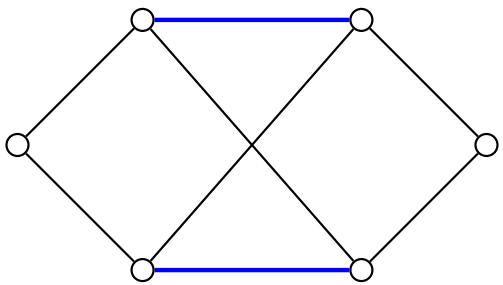


maximum

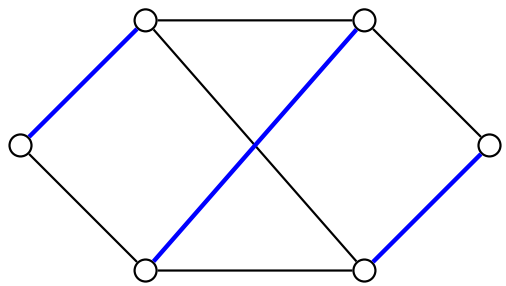
Matching

given an undirected graph $G = (V, E)$

- ▷ a matching is a set $M \subseteq E$ of pairwise non-incident edges
- ▷ perfect matching: $|M| = |V|/2$
- ▷ maximum matching is matching of maximal cardinality



maximal



maximum

Maximum (cardinality) matching

- ▷ given: undirected graph $G = (V, E)$
- ▷ goal: compute a maximum matching M

Basic Idea

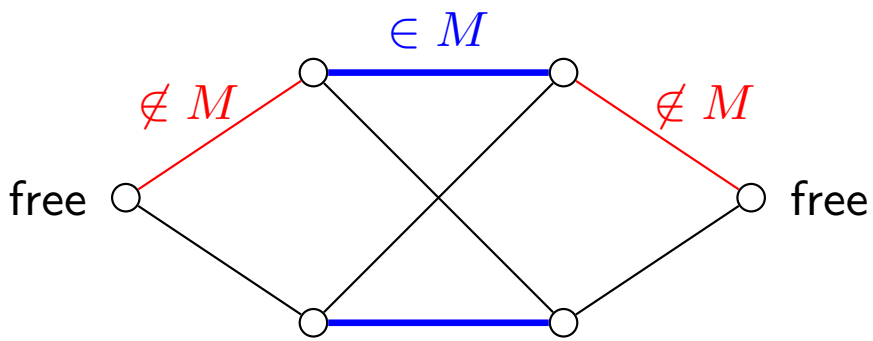
- greedy yields maximal matching, but needn't be maximum

Basic Idea

- greedy yields maximal matching, but needn't be maximum
- idea: start with maximal matching, look for augmenting path

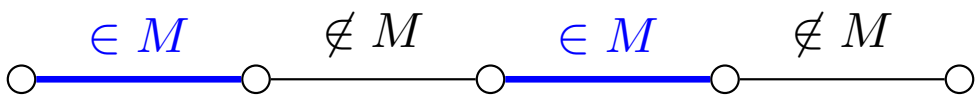
Basic Idea

- ▷ greedy yields maximal matching, but needn't be maximum
- ▷ idea: start with maximal matching, look for augmenting path



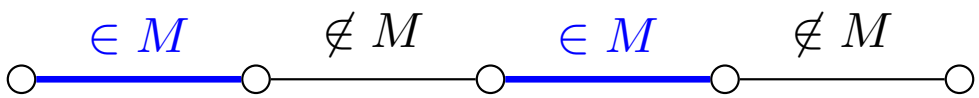
Augmenting Paths

M-alternating path

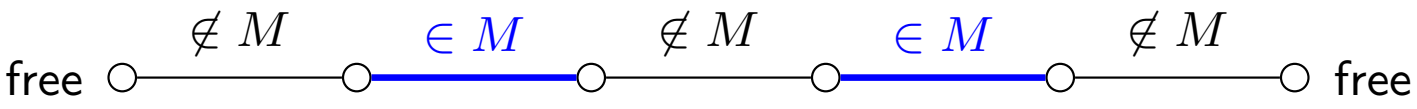


Augmenting Paths

M-alternating path

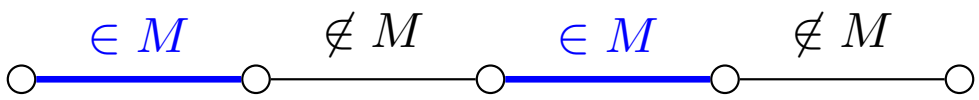


M-augmenting path

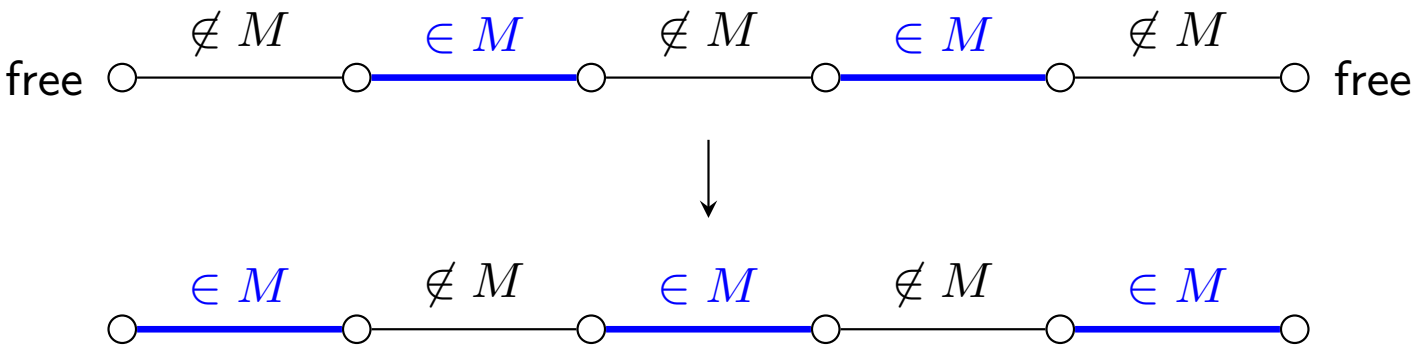


Augmenting Paths

M-alternating path

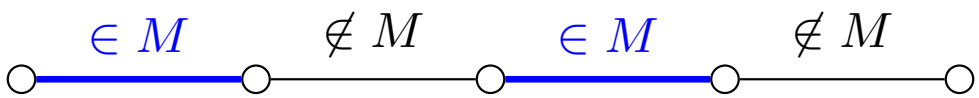


M-augmenting path

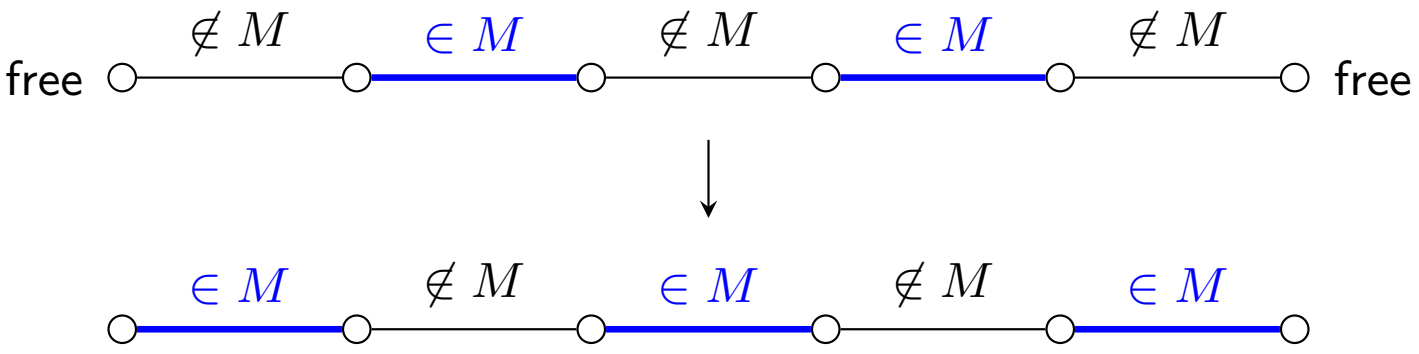


Augmenting Paths

M-alternating path



M-augmenting path



symmetric difference: $M \Delta N = (M \backslash N) \cup (N \backslash M),\ M, N \subseteq E$

Optimality Condition

Theorem

Matching M is maximum (cardinality) iff there is no M-augmenting path.

Optimality Condition

Theorem

Matching M is maximum (cardinality) iff there is no M-augmenting path.

Proof

($\Rightarrow$) trivial

Optimality Condition

Theorem

Matching M is maximum (cardinality) iff there is no M-augmenting path.

Proof

($\Rightarrow$) trivial ($\Leftarrow$) assume $\exists M^*$ with $|M^*| > |M|$ and consider $M \Delta M^*$

Optimality Condition

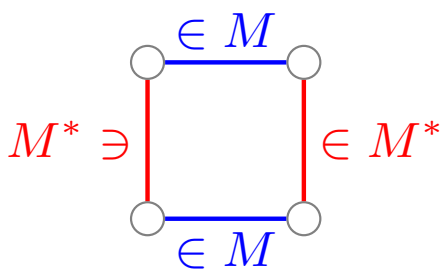
Theorem

Matching M is maximum (cardinality) iff there is no M-augmenting path.

Proof

($\Rightarrow$) trivial ($\Leftarrow$) assume $\exists M^*$ with $|M^*| > |M|$ and consider $M \Delta M^*$

- $M \Delta M^*$ contains node disjoint cycles and paths



Optimality Condition

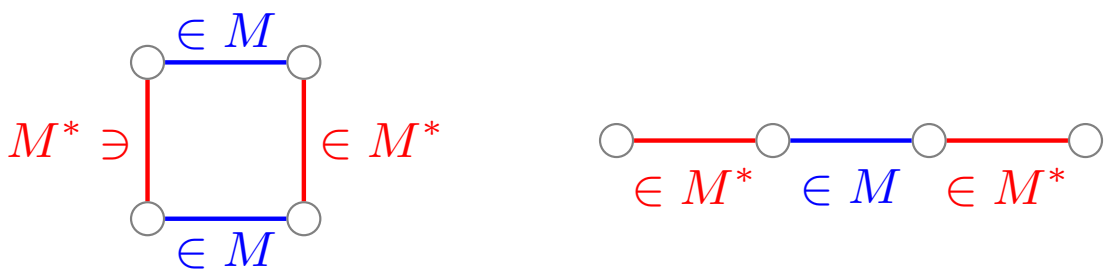
Theorem

Matching M is maximum (cardinality) iff there is no M-augmenting path.

Proof

($\Rightarrow$) trivial ($\Leftarrow$) assume $\exists M^*$ with $|M^*| > |M|$ and consider $M \Delta M^*$

- $M \Delta M^*$ contains node disjoint cycles and paths



- each cycle has even number of edges

Optimality Condition

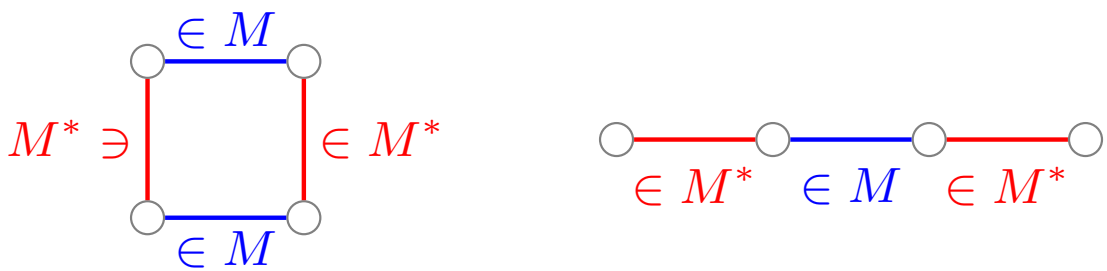
Theorem

Matching M is maximum (cardinality) iff there is no M-augmenting path.

Proof

($\Rightarrow$) trivial ($\Leftarrow$) assume $\exists M^*$ with $|M^*| > |M|$ and consider $M \Delta M^*$

- $M \Delta M^*$ contains node disjoint cycles and paths



- each cycle has even number of edges
- exist a path P having $|P \cap M^*| > |P \cap M|$ by assumption

Optimality Condition

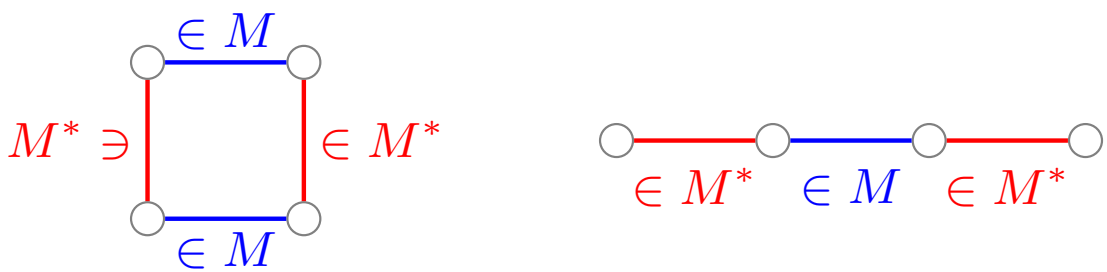
Theorem

Matching M is maximum (cardinality) iff there is no M-augmenting path.

Proof

($\Rightarrow$) trivial ($\Leftarrow$) assume $\exists M^*$ with $|M^*| > |M|$ and consider $M \Delta M^*$

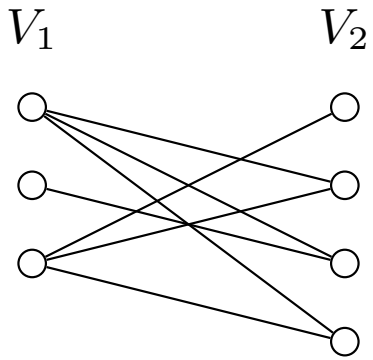
- $M \Delta M^*$ contains node disjoint cycles and paths



- each cycle has even number of edges
- exist a path P having $|P \cap M^*| > |P \cap M|$ by assumption
- P is an M-augmenting path

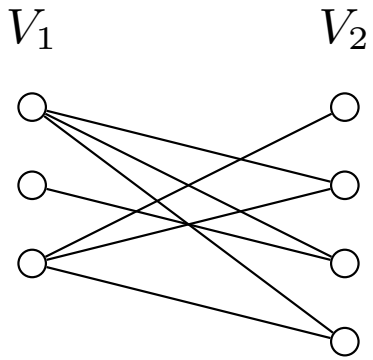
Maximum Matching on Bipartite Graphs

find maximum matching for bipartite graph $G = (V_1, V_2, E)$



Maximum Matching on Bipartite Graphs

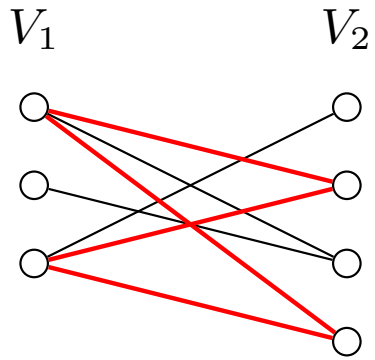
find maximum matching for bipartite graph $G = (V_1, V_2, E)$



how to find M-augmenting paths?

Maximum Matching on Bipartite Graphs

find maximum matching for bipartite graph $G = (V_1, V_2, E)$

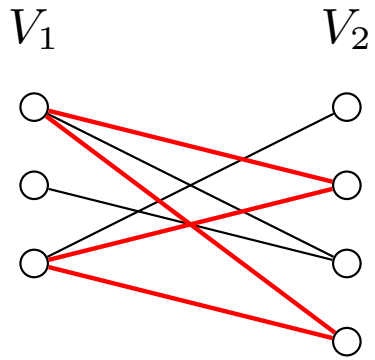


how to find M-augmenting paths?

- ▷ bipartite graphs do not have odd cycles (augmenting paths have odd number of edges)

Maximum Matching on Bipartite Graphs

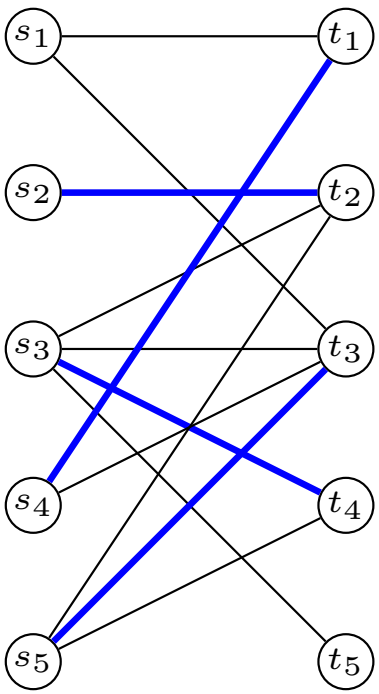
find maximum matching for bipartite graph $G = (V_1, V_2, E)$



how to find M-augmenting paths?

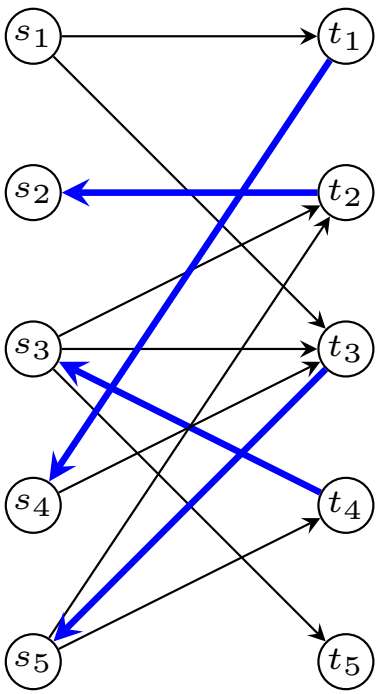
- ▷ bipartite graphs do not have odd cycles (augmenting paths have odd number of edges)
- ▷ start with a free node $s \in V_1$ and search alternating path until finding another free node $t \in V_2$

Finding Augmenting Paths



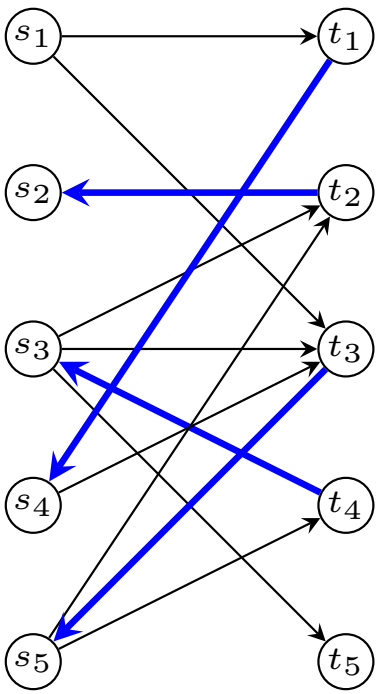
1. construct an auxiliary digraph $D = (V, A)$, with $(s, t) \in A$ if $(s, t) \notin M$, $(t, s) \in A$ otherwise

Finding Augmenting Paths

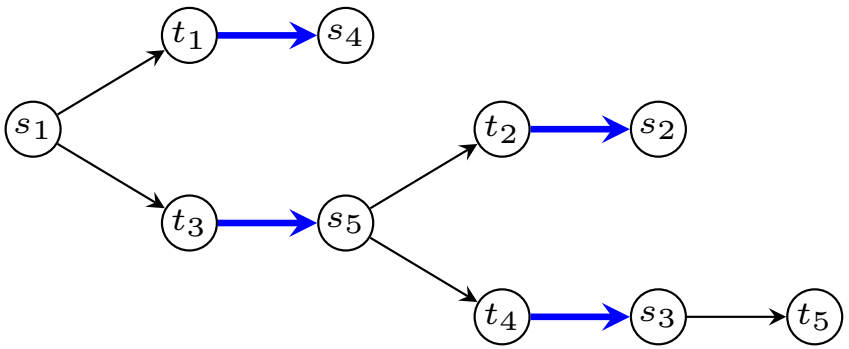


1. construct an auxiliary digraph $D = (V, A)$, with $(s, t) \in A$ if $(s, t) \notin M$, $(t, s) \in A$ otherwise

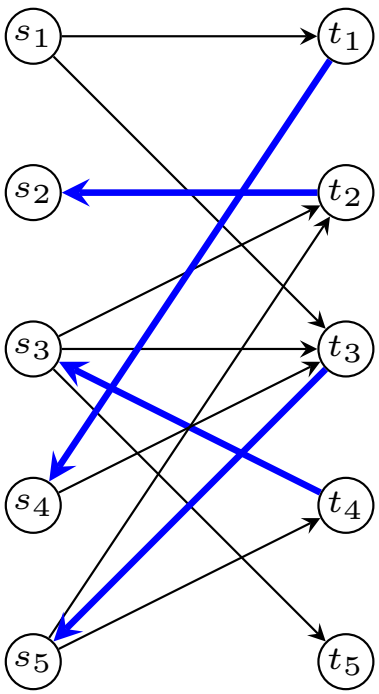
Finding Augmenting Paths



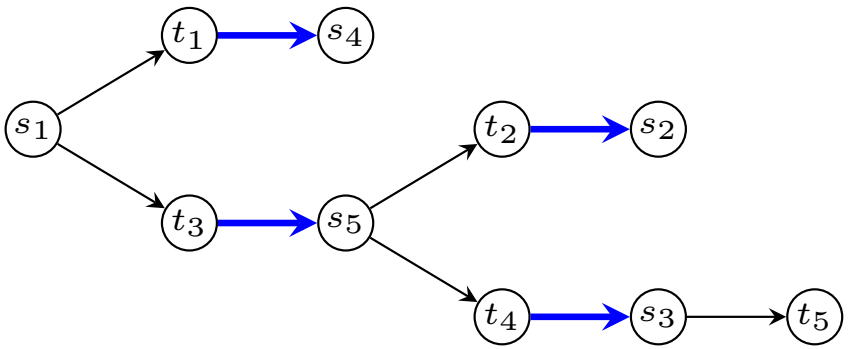
1. construct an auxiliary digraph $D = (V, A)$, with $(s, t) \in A$ if $(s, t) \notin M$, $(t, s) \in A$ otherwise
2. make BFS from any free node $s_1 \in V_1$



Finding Augmenting Paths



1. construct an auxiliary digraph $D = (V, A)$, with $(s, t) \in A$ if $(s, t) \notin M$, $(t, s) \in A$ otherwise
2. make BFS from any free node $s_1 \in V_1$



3. an M-augmenting path is found if exists

Augmenting Path Algorithm

BIPARTITE-MATHCING(G)

```
1 M = ∅;
2 for all free node s ∈ V_1 do
3     construct the auxiliary digraph D;                          // O(m)
4     apply BFS(D, s) to find an augmenting path P ;              // O(m)
5     if P ≠ nil then
6         M := MΔE(P);                                  // augment along P  O(m)
```

Augmenting Path Algorithm

BIPARTITE-MATHCING(G)

```
1 M = ∅;
2 for all free node s ∈ V_1 do
3     construct the auxiliary digraph D;                    // O(m)
4     apply BFS(D, s) to find an augmenting path P ;        // O(m)
5     if P ≠ nil then
6         M := MΔE(P);                                      // augment along P  O(m)
```

- after augmentation: free nodes are matched and matched nodes are still matched

Augmenting Path Algorithm

BIPARTITE-MATHCING(G)

```
1 M = ∅;
2 for all free node s ∈ V_1 do
3     construct the auxiliary digraph D;                          // O(m)
4     apply BFS(D, s) to find an augmenting path P ;              // O(m)
5     if P ≠ nil then
6         M := MΔE(P);                          // augment along P  O(m)
```

- ▷ after augmentation: free nodes are matched and matched nodes are still matched
- ▷ thus no augmenting paths at termination

Augmenting Path Algorithm

BIPARTITE-MATHCING(G)

```
1 M = ∅;
2 for all free node s ∈ V_1 do
3     construct the auxiliary digraph D;                    // O(m)
4     apply BFS(D, s) to find an augmenting path P ;        // O(m)
5     if P ≠ nil then
6         M := MΔE(P);                                      // augment along P  O(m)
```

- ▷ after augmentation: free nodes are matched and matched nodes are still matched
- ▷ thus no augmenting paths at termination
- ▷ running time $O(nm)$