NEW CS 473: Theory II, Fall 2015

Matchings I

Lecture 16 October 20, 2015

16.1: Matchings

16.1.1: Definitions

Matching, perfect, maximal

Definition

For a graph $\mathbf{G}=(\mathbf{V},\mathbf{E})$ a set $M\subseteq\mathbf{E}$ is a **matching** if no pair of edges of M has a common vertex.

Definition

A matching is **perfect** if it covers all the vertices of G. For a weight function w, which assigns real weight to the edges of G, a matching M is a **maximal weight matching**, if M is a matching and $w(M) = \sum_{e \in M} w(e)$ is maximal.

Definition

If there is no weight on the edges, we consider the weight of every edge to be one, and in this case, we are trying to compute a **maximum size matching**.

The problem

Problem

Given a graph **G** and a weight function on the edges, compute the maximum weight matching in **G**.

 $16.2.2: \ \mathsf{Matchings} \ \mathsf{and} \ \mathsf{alternating} \ \mathsf{paths}$

- 1 M: matching.
- $oldsymbol{e} e \in M$ is a **matching edge**matching!matching edge.
- $e' \in E(G) \setminus M$ is free.
- \circ unmatched vertex v' is free.
- alternating path: a simple path edges alternating between matched and free edges.
- alternating cycle...
- length of a path/cycle is the number of edges in it.

- M: matching.
- $e \in M$ is a **matching edge**matching!matching edge.
- $e' \in E(G) \setminus M$ is free.
- \circ unmatched vertex v' is free.
- alternating path: a simple path edges alternating between matched and free edges.
- alternating cycle...
- length of a path/cycle is the number of edges in it.

- M: matching.
- $oldsymbol{e} e \in M$ is a **matching edge**matching!matching edge.
- $e' \in E(G) \setminus M$ is free.

- alternating path: a simple path edges alternating between matched and free edges.
- alternating cycle...
- length of a path/cycle is the number of edges in it.

- M: matching.
- $oldsymbol{e} e \in M$ is a **matching edge**matching!matching edge.
- $e' \in E(G) \setminus M$ is free.
- $0 v \in V(G)$ matched \iff adjacent to edge in M.
- alternating path: a simple path edges alternating between matched and free edges.
- alternating cycle...
- length of a path/cycle is the number of edges in it.

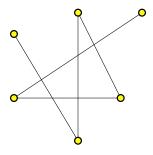
- M: matching.
- $oldsymbol{e} e \in M$ is a **matching edge**matching!matching edge.
- $e' \in E(G) \setminus M$ is free.
- \bullet $v \in V(G)$ matched \iff adjacent to edge in M.
- \circ unmatched vertex v' is free.
- alternating path: a simple path edges alternating between matched and free edges.
- alternating cycle...
- length of a path/cycle is the number of edges in it.

- M: matching.
- $oldsymbol{e} e \in M$ is a **matching edge**matching!matching edge.
- $e' \in E(G) \setminus M$ is free.
- $lackbox{0} v \in V(G)$ matched \iff adjacent to edge in M.
- \bullet unmatched vertex v' is **free**.
- alternating path: a simple path edges alternating between matched and free edges.
- alternating cycle...
- length of a path/cycle is the number of edges in it.

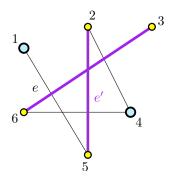
- M: matching.
- $oldsymbol{e} e \in M$ is a **matching edge**matching!matching edge.
- $e' \in E(G) \setminus M$ is free.
- \bullet $v \in V(G)$ matched \iff adjacent to edge in M.
- \bullet unmatched vertex v' is **free**.
- alternating path: a simple path edges alternating between matched and free edges.
- alternating cycle...
- length of a path/cycle is the number of edges in it.

- M: matching.
- $e \in M$ is a **matching edge**matching!matching edge.
- $e' \in E(G) \setminus M$ is free.
- \bullet $v \in V(G)$ matched \iff adjacent to edge in M.
- \bullet unmatched vertex v' is **free**.
- alternating path: a simple path edges alternating between matched and free edges.
- alternating cycle...
- length of a path/cycle is the number of edges in it.

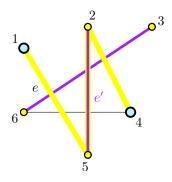
- M: matching.
- $e \in M$ is a **matching edge**matching!matching edge.
- $e' \in E(G) \setminus M$ is free.
- \bullet $v \in V(G)$ matched \iff adjacent to edge in M.
- ullet unmatched vertex $oldsymbol{v'}$ is **free**.
- alternating path: a simple path edges alternating between matched and free edges.
- alternating cycle...
- length of a path/cycle is the number of edges in it.



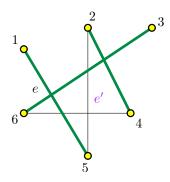
(A) The input graph.



(B) A maximal matching in ${\bf G}$. The edge ${\bf e}$ is free, and vertices ${\bf 1}$ and ${\bf 4}$ are free.



(C) An alternating path.



(D) The resulting matching from applying the augmenting path.

Definition

Path $\pi = v_1 v_2, \ldots, v_{2k+2}$ is **augmenting** path for matching M (for graph **G**):

- (i) π is simple,
- (ii) for all i, $e_i = v_i v_{i+1} \in \mathsf{E}(\mathsf{G})$,
- (iii) v_1 and v_{2k+2} are free vertices for M,
- (iv) $e_1,e_3,\ldots,e_{2k+1}
 otin M$, and
- $(\mathsf{v})\ e_2, e_4, \ldots, e_{2k} \in M.$

Definition

Path $\pi = v_1 v_2, \ldots, v_{2k+2}$ is **augmenting** path for matching M (for graph **G**):

- (i) π is simple,
- (ii) for all i, $e_i = v_i v_{i+1} \in \mathsf{E}(\mathsf{G})$,
- (iii) v_1 and v_{2k+2} are free vertices for M,
- (iv) $e_1,e_3,\ldots,e_{2k+1}
 otin M$, and
- (v) $e_2, e_4, \dots, e_{2k} \in M$.



Definition

Path $\pi = v_1 v_2, \dots, v_{2k+2}$ is **augmenting** path for matching M(for graph G):

- (i) π is simple,
- (ii) for all $i, e_i = v_i v_{i+1} \in \mathsf{E}(\mathsf{G})$,
- (iii) v_1 and v_{2k+2} are free vertices for M,
- (iv) $e_1, e_3, \ldots, e_{2k+1} \notin M$, and
- $(\mathsf{v})\ e_2, e_4, \ldots, e_{2k} \in M.$



Definition

Path $\pi = v_1 v_2, \ldots, v_{2k+2}$ is **augmenting** path for matching M (for graph **G**):

- (i) π is simple,
- (ii) for all i, $e_i = v_i v_{i+1} \in \mathsf{E}(\mathsf{G})$,
- (iii) v_1 and v_{2k+2} are free vertices for M,
- (iv) $e_1, e_3, \ldots, e_{2k+1} \notin M$, and
- $(v) e_2, e_4, \ldots, e_{2k} \in M.$



Definition

Path $\pi = v_1 v_2, \ldots, v_{2k+2}$ is **augmenting** path for matching M (for graph ${f G}$):

- (i) π is simple,
- (ii) for all i, $e_i = v_i v_{i+1} \in \mathsf{E}(\mathsf{G})$,
- (iii) v_1 and v_{2k+2} are free vertices for M,
- (iv) $e_1,e_3,\ldots,e_{2k+1}
 otin M$, and
- (v) $e_2,e_4,\ldots,e_{2k}\in M$.



After applying both augmenting path, we end up with maximum matching here.

Augmenting paths improve things

Lemma

M: matching. π : augmenting path relative to M. Then

$$M' = M \oplus \pi = \{e \in \mathsf{E} \mid e \in (M \setminus \pi) \cup (\pi \setminus M)\}$$

is a matching of size |M|+1.

- **1** Remove π from graph.
- 2 Leftover matching: $|M| |M \cap \pi|$.
- **3** Add back π . Add free edges of π to matching.

Augmenting paths improve things

Lemma

M: matching. π : augmenting path relative to M. Then

$$M' = M \oplus \pi = \{e \in \mathsf{E} \mid e \in (M \setminus \pi) \cup (\pi \setminus M)\}$$

is a matching of size |M|+1.

- **1** Remove π from graph.
- 2 Leftover matching: $|M| |M \cap \pi|$.
- **3** Add back π . Add free edges of π to matching.
- M': New set of edges... a matching.
- $|M'| = |M| |M \cap \pi| + |\pi \setminus M| = |M| + 1.$

Augmenting paths improve things

Lemma

M: matching. π : augmenting path relative to M. Then

$$M' = M \oplus \pi = \{e \in \mathsf{E} \mid e \in (M \setminus \pi) \cup (\pi \setminus M)\}$$

is a matching of size |M|+1.

- **1** Remove π from graph.
- 2 Leftover matching: $|M| |M \cap \pi|$.
- **3** Add back π . Add free edges of π to matching.
- \bullet M': New set of edges... a matching.
- $|M'| = |M| |M \cap \pi| + |\pi \setminus M| = |M| + 1.$

Lemma

M: matching. T: maximum matching. k = |T| - |M|. Then M has k vertex disjoint augmenting paths.

Proof.

- **1** $E' = M \oplus T$. H = (V, E').
- $oldsymbol{3}$ $oldsymbol{H}$: collection of alternating paths and cycles.
- cycles are even length.
- $\bullet \ \, \text{For any cycle} \,\, C \in H \colon |C \cap M| = |C \cap T|.$

For augmenting path π : $|\pi| |I| = |\pi| |I|$

Lemma

M: matching. T: maximum matching. k = |T| - |M|. Then M has k vertex **disjoint** augmenting paths.

Proof.

- **1** $E' = M \oplus T$. H = (V, E').
- $oldsymbol{0}$ $oldsymbol{H}$: collection of alternating paths and cycles.
- cycles are even length.
- $lacksymbol{\circ}$ k more edges of T in $M\oplus T$ than of M.
- $\bullet \ \, \text{For any cycle} \,\, C \in H \colon |C \cap M| = |C \cap T|.$
- $m{0}$ For a path $\pi \in H$: $|\pi \cap M| \leq |\pi \cap T| \leq |\pi \cap M| + 1$.

of or augmenting path π . $|\pi| |\Pi| = |\pi| |\Pi |\Pi| = 1$

Lemma

M: matching. T: maximum matching. k = |T| - |M|. Then M has k vertex **disjoint** augmenting paths.

Proof.

- **1** $E' = M \oplus T$. H = (V, E').
- $\forall v \in \mathsf{V}(H): \ d(v) \leq 2.$
- $oldsymbol{0}$ $oldsymbol{H}$: collection of alternating paths and cycles.
- cycles are even length.
- $lacksymbol{\circ}$ k more edges of T in $M\oplus T$ than of M.
- $\bullet \ \, \text{For any cycle} \,\, C \in H \colon |C \cap M| = |C \cap T|.$
- \P For a path $\pi \in H$: $|\pi \cap M| \leq |\pi \cap T| \leq |\pi \cap M| + 1$.

Lemma

M: matching. T: maximum matching. k = |T| - |M|. Then M has k vertex **disjoint** augmenting paths.

- **1** $E' = M \oplus T$. H = (V, E').
- 2 *H*: collection of alternating paths and cycles.
- 3 cycles are even length.
- @ k more edges of T in $M \oplus T$ than of M.
- ullet For any cycle $C \in H$: $|C \cap M| = |C \cap T|$.
- ullet For a path $\pi \in H$: $|\pi \cap M| \leq |\pi \cap T| \leq |\pi \cap M| + 1$.
- extstyle ext
- $@> \Longrightarrow \mathsf{Must} \mathsf{be} \ k$ augmenting paths in H

Lemma

M: matching. T: maximum matching. k = |T| - |M|. Then M has k vertex **disjoint** augmenting paths.

- **1** $E' = M \oplus T$. H = (V, E').
- 2 *H*: collection of alternating paths and cycles.
- 3 cycles are even length.
- $oldsymbol{eta}$ k more edges of T in $M\oplus T$ than of M.
- ullet For any cycle $C \in H$: $|C \cap M| = |C \cap T|$.
- \bullet For a path $\pi \in H$: $|\pi \cap M| \leq |\pi \cap T| \leq |\pi \cap M| + 1$.
- $m{0}$ For augmenting path $\pi\colon |\pi\cap T|=|\pi\cap M|+1$
- $@> \Longrightarrow \mathsf{Must} \mathsf{be} \ k$ augmenting paths in H

Lemma

M: matching. T: maximum matching. k = |T| - |M|. Then M has k vertex **disjoint** augmenting paths.

- **1** $E' = M \oplus T$. H = (V, E').
- 2 *H*: collection of alternating paths and cycles.
- Ocycles are even length.
- lacksquare k more edges of T in $M\oplus T$ than of M .
- ullet For any cycle $C \in H$: $|C \cap M| = |C \cap T|$.
- ullet For a path $\pi \in H$: $|\pi \cap M| \leq |\pi \cap T| \leq |\pi \cap M| + 1$.
- ${m 0}$ For augmenting path $\pi\colon |\pi\cap T|=|\pi\cap M|+1$

Lemma

M: matching. T: maximum matching. k = |T| - |M|. Then M has k vertex **disjoint** augmenting paths.

- **1** $E' = M \oplus T$. H = (V, E').
- 2 *H*: collection of alternating paths and cycles.
- cycles are even length.
- lacksquare k more edges of T in $M\oplus T$ than of M.
- $\textbf{ § For any cycle } C \in H \text{: } |C \cap M| = |C \cap T|.$
- ullet For a path $\pi \in H \colon |\pi \cap M| \leq |\pi \cap T| \leq |\pi \cap M| + 1$.
- ${m 0}$ For augmenting path $\pi\colon |\pi\cap T|=|\pi\cap M|+1$
- $| \Longrightarrow$ Must be k augmenting paths in H .

Lemma

M: matching. T: maximum matching. k = |T| - |M|. Then M has k vertex **disjoint** augmenting paths.

- **1** $E' = M \oplus T$. H = (V, E').
- ② H: collection of alternating paths and cycles.
- $lacksquare{1}{3} k$ more edges of T in $M \oplus T$ than of M.
- $\bullet \ \text{ For any cycle } C \in H \colon |C \cap M| = |C \cap T|.$
- $lacksquare{0}$ For augmenting path $\pi\colon |\pi\cap T|=|\pi\cap M|+1$
- $\bigcirc \implies$ Must be k augmenting paths in H.

Lemma

M: matching. T: maximum matching. k = |T| - |M|. Then M has k vertex **disjoint** augmenting paths.

- **1** $E' = M \oplus T$. H = (V, E').
- ② H: collection of alternating paths and cycles.
- $lacksquare{1}{3} k$ more edges of T in $M \oplus T$ than of M.
- $\bullet \ \text{ For any cycle } C \in H \colon |C \cap M| = |C \cap T|.$
- For augmenting path π : $|\pi \cap T| = |\pi \cap M| + 1$.
- \bigcirc \Longrightarrow Must be k augmenting paths in H.

Lemma

M: matching. T: maximum matching. k = |T| - |M|. Then M has k vertex **disjoint** augmenting paths.

- **1** $E' = M \oplus T$. H = (V, E').
- ② H: collection of alternating paths and cycles.
- $oldsymbol{0}$ k more edges of T in $M\oplus T$ than of M.
- $\bullet \ \text{ For any cycle } C \in H \colon |C \cap M| = |C \cap T|.$
- **o** For augmenting path π : $|\pi \cap T| = |\pi \cap M| + 1$.
- \bigcirc \Longrightarrow Must be k augmenting paths in H.

Lemma

M: matching. T: maximum matching. k=|T|-|M|. At least one augmenting path for M of length $\leq u/k-1$, where u=2(|T|+|M|).

- **1** $E' = M \oplus T$. H = (V, E').
- $u = |V(H)| \le 2(|T| + |M|).$
- ullet By previous lemma: There are k augmenting paths in $oldsymbol{H}$.
- ① If all augmenting paths were of length $\geq u/k$
- lacksquare total number of vertices in $H \geq (u/k+1)u > u$
- $oldsymbol{0}$... since a path of length ℓ has $\ell+1$ vertices. A contradiction.

Lemma

M: matching. T: maximum matching. k=|T|-|M|. At least one augmenting path for M of length $\leq u/k-1$, where u=2(|T|+|M|).

- **1** $E' = M \oplus T$. H = (V, E').
- $u = |V(H)| \le 2(|T| + |M|).$
- ullet By previous lemma: There are k augmenting paths in H.
- $ext{ } ext{ }$
- lacksquare total number of vertices in $H \geq (u/k+1)u > u$
- $oldsymbol{0}$... since a path of length ℓ has $\ell+1$ vertices. A contradiction.

Lemma

M: matching. T: maximum matching. k=|T|-|M|. At least one augmenting path for M of length $\leq u/k-1$, where u=2(|T|+|M|).

- **1** $E' = M \oplus T$. H = (V, E').
- $u = |V(H)| \le 2(|T| + |M|).$
- lacksquare By previous lemma: There are $m{k}$ augmenting paths in $m{H}$.
- $ext{ } ext{ }$
- lacksquare total number of vertices in $H \geq (u/k+1)u > u$
- $ext{ } ext{ } ext$

Lemma

M: matching. T: maximum matching. k=|T|-|M|. At least one augmenting path for M of length $\leq u/k-1$, where u=2(|T|+|M|).

- \bullet $E' = M \oplus T$. H = (V, E').
- $u = |V(H)| \le 2(|T| + |M|).$
- lacksquare By previous lemma: There are $m{k}$ augmenting paths in $m{H}$.
- lacksquare If all augmenting paths were of length $\geq u/k$
- lacksquare total number of vertices in $H \geq (u/k+1)u > u$
- ullet ... since a path of length ℓ has $\ell+1$ vertices. A contradiction.

Lemma

M: matching. T: maximum matching. k=|T|-|M|. At least one augmenting path for M of length $\leq u/k-1$, where u=2(|T|+|M|).

- \bullet $E' = M \oplus T$. H = (V, E').
- $u = |V(H)| \le 2(|T| + |M|).$
- $oldsymbol{0}$ By previous lemma: There are $oldsymbol{k}$ augmenting paths in $oldsymbol{H}$.
- lacktriangledown If all augmenting paths were of length $\geq u/k$
- lacksquare total number of vertices in $H \geq (u/k+1)u > u$
- ullet ... since a path of length ℓ has $\ell+1$ vertices. A contradiction.

Lemma

M: matching. T: maximum matching. k=|T|-|M|. At least one augmenting path for M of length $\leq u/k-1$, where u=2(|T|+|M|).

Proof.

- \bullet $E' = M \oplus T$. H = (V, E').
- $u = |V(H)| \le 2(|T| + |M|).$
- $oldsymbol{0}$ By previous lemma: There are $oldsymbol{k}$ augmenting paths in $oldsymbol{H}$.
- lacktriangledown If all augmenting paths were of length $\geq u/k$
- lacksquare total number of vertices in $H \geq (u/k+1)u > u$
- $oldsymbol{0}$... since a path of length ℓ has $\ell+1$ vertices. A contradiction.

12 / 37

No augmenting path, no cry

Or: Having a maximum matching.

Corollary

A matching M is maximum \iff there is no augmenting path for M.

Sariel (UIUC) New CS473 13 Fall 2015 13 / 37

16.3: Unweighted matching in bipartite graph

 $16.3.1: \ \, \mathsf{The} \,\, \mathsf{slow} \,\, \mathsf{algorithm}$

- **0** $G = (L \cup R, E)$: bipartite graph.
- ② Task: Compute maximum size matching in G.
- ullet $M_0=\emptyset$ empty matching.
- In ith iteration of algSlowMatch:
 - $oldsymbol{0}$ $L_i\subseteq L$ and $R_i\subseteq R$: set of free vertices for matching M_{i-1} .
 - **②** Graph H_i : Orient all edges of $E \setminus M_{i-1}$ from left to the right.
 - **③** $\forall lr \in M_{i-1}$ oriented from the right to left, as the new directed edge (r, l).
 - **g** BFS: compute shortest path π_i from a vertex of L_i to a vertex of R_i .

 - $M_i = M_{i-1} \oplus \pi_i$

- **1** $G = (L \cup R, E)$: bipartite graph.
- Task: Compute maximum size matching in G.
- **3** $M_0 = \emptyset$ empty matching.
- In ith iteration of algSlowMatch:
 - ullet $L_i\subseteq L$ and $R_i\subseteq R$: set of free vertices for matching M_{i-1} .
 - ② Graph H_i : Orient all edges of $E \setminus M_{i-1}$ from left to the right
 - $\forall lr \in M_{i-1}$ oriented from the right to left, as the new directed edge (r,l).
 - **BFS**: compute shortest path π_i from a vertex of L_i to a vertex of R_i .

 - $M_i = M_{i-1} \oplus \pi_i$

- **1** $G = (L \cup R, E)$: bipartite graph.
- Task: Compute maximum size matching in G.
- **3** $M_0 = \emptyset$ empty matching.
- In ith iteration of algSlowMatch:
 - ullet $L_i\subseteq L$ and $R_i\subseteq R$: set of free vertices for matching M_{i-1} .
 - **2** Graph H_i : Orient all edges of $E \setminus M_{i-1}$ from left to the right.
 - $\forall lr \in M_{i-1}$ oriented from the right to left, as the new directed edge (r,l).
 - **BFS**: compute shortest path π_i from a vertex of L_i to a vertex of R_i .

 - $M_i = M_{i-1} \oplus \pi_i$

- **1** $G = (L \cup R, E)$: bipartite graph.
- Task: Compute maximum size matching in G.
- **3** $M_0 = \emptyset$ empty matching.
- In ith iteration of algSlowMatch:
 - **1** $L_i \subseteq L$ and $R_i \subseteq R$: set of free vertices for matching M_{i-1} .
 - **2** Graph H_i : Orient all edges of $\mathsf{E}\setminus M_{i-1}$ from left to the right.
 - $\forall lr \in M_{i-1}$ oriented from the right to left, as the new directed edge (r,l).
 - **BFS**: compute shortest path π_i from a vertex of L_i to a vertex of R_i .

16 / 37

- \bigcirc If no such path \Longrightarrow no augmenting path \Longrightarrow stop.
- $\bullet \ M_i = M_{i-1} \oplus \pi_i.$

- **1** $G = (L \cup R, E)$: bipartite graph.
- Task: Compute maximum size matching in G.
- $oldsymbol{0} M_0 = \emptyset$ empty matching.
- In ith iteration of algSlowMatch:
 - ullet $L_i\subseteq L$ and $R_i\subseteq R$: set of free vertices for matching M_{i-1} .
 - **2** Graph H_i : Orient all edges of $E \setminus M_{i-1}$ from left to the right.
 - $\forall lr \in M_{i-1}$ oriented from the right to left, as the new directed edge (r,l).
 - **3** BFS: compute shortest path π_i from a vertex of L_i to a vertex of R_i .

 - $M_i = M_{i-1} \oplus \pi_i$

- **1** $G = (L \cup R, E)$: bipartite graph.
- Task: Compute maximum size matching in G.
- $oldsymbol{0} M_0 = \emptyset$ empty matching.
- In ith iteration of algSlowMatch:
 - ullet $L_i\subseteq L$ and $R_i\subseteq R$: set of free vertices for matching M_{i-1} .
 - **2** Graph H_i : Orient all edges of $E \setminus M_{i-1}$ from left to the right.
 - $\forall lr \in M_{i-1}$ oriented from the right to left, as the new directed edge (r,l).
 - **3** BFS: compute shortest path π_i from a vertex of L_i to a vertex of R_i .
 - **1** If no such path \implies no augmenting path \implies stop.
 - $\mathbf{0} \ \ M_i = M_{i-1} \oplus \pi_i.$

- **1** $G = (L \cup R, E)$: bipartite graph.
- Task: Compute maximum size matching in G.
- $\mathbf{M}_0 = \emptyset$ empty matching.
- In ith iteration of algSlowMatch:
 - ullet $L_i\subseteq L$ and $R_i\subseteq R$: set of free vertices for matching M_{i-1} .
 - **Q** Graph H_i : Orient all edges of $E \setminus M_{i-1}$ from left to the right.
 - $\forall lr \in M_{i-1}$ oriented from the right to left, as the new directed edge (r,l).
 - **3** BFS: compute shortest path π_i from a vertex of L_i to a vertex of R_i .
 - **6** If no such path \implies no augmenting path \implies stop.
 - $\mathbf{0} \ M_i = M_{i-1} \oplus \pi_i.$

- augmenting path has an odd number of edges.
- 2 starts free vertex on left side: ends in free vertex on right side.
- $lacksquare{0}$ augmenting path: path between vertex L_i to vertex of R_i in $oldsymbol{\mathsf{H}}_i$.
- By corollary: algorithm matching not maximum matching yet...,
- $\Rightarrow \exists augmenting path.$
- Using augmenting path: increases size of matching by one.
- ② any shortest path found in H_i between L_i and R_i is an augmenting path.
- $\ \ \exists$ augmenting path for $M_{i-1} \implies$ path from vertex of L_i to vertex of R_i in H_i .
- algorithm computes shortest such path.

- augmenting path has an odd number of edges.
- 2 starts free vertex on left side: ends in free vertex on right side.
- $lacksquare{3}$ augmenting path: path between vertex L_i to vertex of R_i in H_i .
- By corollary: algorithm matching not maximum matching yet...,
- $\implies \exists$ augmenting path.
- Using augmenting path: increases size of matching by one.
- $extbf{ iny any shortest}$ path found in $extbf{ iny H}_i$ between L_i and R_i is an augmenting path.
- $\ \ \exists$ augmenting path for $M_{i-1} \implies$ path from vertex of L_i to vertex of R_i in H_i .
- algorithm computes shortest such path.

- augmenting path has an odd number of edges.
- 2 starts free vertex on left side: ends in free vertex on right side.
- $lacksquare{0}$ augmenting path: path between vertex L_i to vertex of R_i in $oldsymbol{\mathsf{H}}_i$.
- By corollary: algorithm matching not maximum matching yet...,
- $\Rightarrow \exists augmenting path.$
- Using augmenting path: increases size of matching by one
- $m{0}$ any shortest path found in $m{H}_i$ between $m{L}_i$ and $m{R}_i$ is an augmenting path.
- $\ \ \exists$ augmenting path for $M_{i-1} \implies$ path from vertex of L_i to vertex of R_i in H_i .
- algorithm computes shortest such path.

- augmenting path has an odd number of edges.
- 2 starts free vertex on left side: ends in free vertex on right side.
- lacksquare augmenting path: path between vertex L_i to vertex of R_i in $oldsymbol{\mathsf{H}}_i$.
- By corollary: algorithm matching not maximum matching yet...,
- $\Rightarrow \exists$ augmenting path.
- Using augmenting path: increases size of matching by one.
- $oldsymbol{0}$ any shortest path found in $oldsymbol{\mathsf{H}}_i$ between $oldsymbol{L}_i$ and $oldsymbol{R}_i$ is an augmenting path.
- $exttt{3}$ \exists augmenting path for $M_{i-1} \implies$ path from vertex of L_i to vertex of R_i in H_i .
- algorithm computes shortest such path.

- augmenting path has an odd number of edges.
- 2 starts free vertex on left side: ends in free vertex on right side.
- $lacksquare{3}$ augmenting path: path between vertex L_i to vertex of R_i in H_i .
- By corollary: algorithm matching not maximum matching yet...,
- $\Rightarrow \exists$ augmenting path.
- Using augmenting path: increases size of matching by one.
- $oldsymbol{0}$ any shortest path found in $oldsymbol{\mathsf{H}}_i$ between L_i and R_i is an augmenting path.
- $\ \ \exists$ augmenting path for $M_{i-1} \implies$ path from vertex of L_i to vertex of R_i in H_i .
- algorithm computes shortest such path.

- augmenting path has an odd number of edges.
- 2 starts free vertex on left side: ends in free vertex on right side.
- $oldsymbol{\circ}$ augmenting path: path between vertex L_i to vertex of R_i in $oldsymbol{\mathsf{H}}_i$.
- By corollary: algorithm matching not maximum matching yet...,
- Using augmenting path: increases size of matching by one.
- ② any shortest path found in H_i between L_i and R_i is an augmenting path.
- $egin{array}{ll} \exists \ \, ext{augmenting path for } M_{i-1} \implies ext{path from vertex of } L_i \ \, ext{to} \ \, ext{vertex of } R_i \ \, ext{in } \mathsf{H}_i. \end{array}$
- algorithm computes shortest such path.

- augmenting path has an odd number of edges.
- starts free vertex on left side: ends in free vertex on right side.
- $oldsymbol{\circ}$ augmenting path: path between vertex L_i to vertex of R_i in $oldsymbol{\mathsf{H}}_i$.
- By corollary: algorithm matching not maximum matching yet...,
- Using augmenting path: increases size of matching by one.
- ② any shortest path found in H_i between L_i and R_i is an augmenting path.
- $egin{array}{ll} \exists \ \, ext{augmenting path for } M_{i-1} \implies ext{path from vertex of } L_i \ \, ext{to} \ \, ext{vertex of } R_i \ \, ext{in } \mathsf{H}_i. \end{array}$
- algorithm computes shortest such path.

- \bullet After at most n iterations...
- algorithm would be done.
- ullet Iteration of algorithm can be implemented in linear time O(m).
- We have:

Lemma

Given a bipartite undirected graph ${f G}=(L\cup R,{f E})$, with n vertices and m edges, one can compute the maximum matching in ${f G}$ in O(nm) time.

- After at most n iterations...
- 2 algorithm would be done.
- ullet Iteration of algorithm can be implemented in linear time O(m).
- We have:

Lemma

Given a bipartite undirected graph ${\bf G}=(L\cup R,{\bf E})$, with n vertices and m edges, one can compute the maximum matching in ${\bf G}$ in O(nm) time.

- After at most n iterations...
- algorithm would be done.
- lacksquare Iteration of algorithm can be implemented in linear time O(m).
- We have:

Lemma

Given a bipartite undirected graph ${f G}=(L\cup R,{f E})$, with n vertices and m edges, one can compute the maximum matching in ${f G}$ in O(nm) time.

- After at most n iterations...
- 2 algorithm would be done.
- **1** Iteration of algorithm can be implemented in linear time O(m).
- We have:

Lemma

Given a bipartite undirected graph ${\bf G}=(L\cup R,{\bf E}),$ with n vertices and m edges, one can compute the maximum matching in ${\bf G}$ in O(nm) time.

 $16.3.2: \ \, \mathsf{The} \,\, \mathsf{Hopcroft\text{-}Karp} \,\, \mathsf{algorithm}$

16.3.2.1: Some more structural observations

Sariel (UIUC) New CS473 20 Fall 2015 20 / 37

- If we augmenting along a shortest path, then the next augmenting path must be longer (or at least not shorter).
- If always augment along shortest paths, then the augmenting paths get longer as the algorithm progress.
- All the augmenting paths of the same length used by the algorithm are vertex-disjoint (!).
- Main idea of the faster algorithm: compute this block of vertex-disjoint paths of the same length in one go, thus getting the improved running time.

- If we augmenting along a shortest path, then the next augmenting path must be longer (or at least not shorter).
- If always augment along shortest paths, then the augmenting paths get longer as the algorithm progress.
- All the augmenting paths of the same length used by the algorithm are vertex-disjoint (!).
- Main idea of the faster algorithm: compute this block of vertex-disjoint paths of the same length in one go, thus getting the improved running time.

- If we augmenting along a shortest path, then the next augmenting path must be longer (or at least not shorter).
- If always augment along shortest paths, then the augmenting paths get longer as the algorithm progress.
- All the augmenting paths of the same length used by the algorithm are vertex-disjoint (!).
- Main idea of the faster algorithm: compute this block of vertex-disjoint paths of the same length in one go, thus getting the improved running time.

- If we augmenting along a shortest path, then the next augmenting path must be longer (or at least not shorter).
- If always augment along shortest paths, then the augmenting paths get longer as the algorithm progress.
- All the augmenting paths of the same length used by the algorithm are vertex-disjoint (!).
- Main idea of the faster algorithm: compute this block of vertex-disjoint paths of the same length in one go, thus getting the improved running time.

Shortest augmenting paths get longer...

Lemma

Let M be a matching, and π be the shortest augmenting path for M, and let π' be any augmenting path for $M' = M \oplus \pi$. Then $|\pi'| \geq |\pi|$. Specifically, we have $|\pi'| \geq |\pi| + 2 |\pi \cap \pi'|$.

- **①** Consider the matching $N = M \oplus \pi \oplus \pi'$.
- |N| = |M| + 2.
- $oxed{3} \ M \oplus N$ contains two augmenting paths, say σ_1 and σ_2 (relative to M).
- $egin{aligned} oldsymbol{M} & M \oplus N = \pi \oplus \pi', ext{ and } \ |\pi \oplus \pi'| = |M \oplus N| \geq |\sigma_1| + |\sigma_2| \,. \end{aligned}$
- $oldsymbol{\sigma}$ π : shortest augmenting path $(M) \implies |\sigma_1| \geq |\pi|$ and $|\sigma_2| \geq |\pi|$.
- extstyle ext
- $\begin{array}{c} \textbf{ (ombining with the above, we have} \\ |\pi| + |\pi'| 2 \, |\pi \cap \pi'| \geq 2 \, |\pi| \\ \Longrightarrow \quad |\pi'| \geq |\pi| + 2 \, |\pi \cap \pi'| \, . \end{array}$

- lacksquare Consider the matching $N=M\oplus\pi\oplus\pi'$.
- |N| = |M| + 2.
- $oxed{3} \ M \oplus N$ contains two augmenting paths, say σ_1 and σ_2 (relative to M).
- $egin{aligned} oldsymbol{M} & M \oplus N = \pi \oplus \pi', ext{ and } \ |\pi \oplus \pi'| = |M \oplus N| \geq |\sigma_1| + |\sigma_2| \,. \end{aligned}$
- $oldsymbol{\sigma}$ π : shortest augmenting path $(M) \implies |\sigma_1| \geq |\pi|$ and $|\sigma_2| \geq |\pi|$.
- $exttt{0}$ By definition: $|\pi \oplus \pi'| = |\pi| + |\pi'| 2 |\pi \cap \pi'|$.
- $\begin{array}{c} \textbf{ (Sombining with the above, we have} \\ |\pi| + |\pi'| 2 \, |\pi \cap \pi'| \geq 2 \, |\pi| \\ \Longrightarrow \quad |\pi'| \geq |\pi| + 2 \, |\pi \cap \pi'| \, . \end{array}$

- lacksquare Consider the matching $N=M\oplus\pi\oplus\pi'$.
- |N| = |M| + 2.
- $oldsymbol{0} M \oplus N$ contains two augmenting paths, say $oldsymbol{\sigma_1}$ and $oldsymbol{\sigma_2}$ (relative to M).
- $egin{aligned} oldsymbol{M} & M \oplus N = \pi \oplus \pi', ext{ and } \ |\pi \oplus \pi'| = |M \oplus N| \geq |\sigma_1| + |\sigma_2| \,. \end{aligned}$
- $oldsymbol{\sigma}$ π : shortest augmenting path $(M) \implies |\sigma_1| \geq |\pi|$ and $|\sigma_2| \geq |\pi|$.
- lacksquare By definition: $|\pi \oplus \pi'| = |\pi| + |\pi'| 2 \, |\pi \cap \pi'|$.
- $\begin{array}{c} \texttt{Oombining with the above, we have} \\ |\pi| + |\pi'| 2 \, |\pi \cap \pi'| \geq 2 \, |\pi| \\ \Longrightarrow \quad |\pi'| \geq |\pi| + 2 \, |\pi \cap \pi'| \, . \end{array}$

- ① Consider the matching $N=M\oplus \pi \oplus \pi'$.
- |N| = |M| + 2.
- $lacksquare{oldsymbol{0}}{oldsymbol{M}} \oplus oldsymbol{N}$ contains two augmenting paths, say $oldsymbol{\sigma_1}$ and $oldsymbol{\sigma_2}$ (relative to $oldsymbol{M}$).
- $egin{aligned} oldsymbol{M} & M \oplus N = \pi \oplus \pi', ext{ and } \ |\pi \oplus \pi'| = |M \oplus N| \geq |\sigma_1| + |\sigma_2| \,. \end{aligned}$
- $oldsymbol{\sigma}$ π : shortest augmenting path $(M) \implies |\sigma_1| \geq |\pi|$ and $|\sigma_2| \geq |\pi|$.
- ① By definition: $|\pi \oplus \pi'| = |\pi| + |\pi'| 2 |\pi \cap \pi'|$.
- $\begin{array}{c} \texttt{Oombining with the above, we have} \\ |\pi| + |\pi'| 2 \, |\pi \cap \pi'| \geq 2 \, |\pi| \\ \Longrightarrow \quad |\pi'| \geq |\pi| + 2 \, |\pi \cap \pi'| \, . \end{array}$

- ① Consider the matching $N=M\oplus \pi \oplus \pi'$.
- |N| = |M| + 2.
- $lacksquare{oldsymbol{0}}{oldsymbol{M}} \oplus oldsymbol{N}$ contains two augmenting paths, say $oldsymbol{\sigma_1}$ and $oldsymbol{\sigma_2}$ (relative to $oldsymbol{M}$).
- $egin{aligned} oldsymbol{M} & M \oplus N = \pi \oplus \pi', ext{ and } \ |\pi \oplus \pi'| = |M \oplus N| \geq |\sigma_1| + |\sigma_2| \,. \end{aligned}$
- ullet π : shortest augmenting path $(M) \implies |\sigma_1| \geq |\pi|$ and $|\sigma_2| \geq |\pi|$.
- lacksquare By definition: $|\pi \oplus \pi'| = |\pi| + |\pi'| 2\,|\pi \cap \pi'|$.
- $\begin{array}{c} \text{Ombining with the above, we have} \\ |\pi| + |\pi'| 2 |\pi \cap \pi'| \geq 2 |\pi| \\ \Longrightarrow |\pi'| \geq |\pi| + 2 |\pi \cap \pi'| \,. \end{array}$

- ① Consider the matching $N=M\oplus \pi \oplus \pi'$.
- |N| = |M| + 2.
- $oldsymbol{0} M \oplus N$ contains two augmenting paths, say $oldsymbol{\sigma_1}$ and $oldsymbol{\sigma_2}$ (relative to M).
- $egin{aligned} oldsymbol{M} & M \oplus N = \pi \oplus \pi', ext{ and } \ |\pi \oplus \pi'| = |M \oplus N| \geq |\sigma_1| + |\sigma_2| \,. \end{aligned}$
- ullet π : shortest augmenting path $(M) \implies |\sigma_1| \geq |\pi|$ and $|\sigma_2| \geq |\pi|$.
- $\bullet \implies |\pi \oplus \pi'| \geq |\sigma_1| + |\sigma_2| \geq |\pi| + |\pi| = 2|\pi|.$
- ① By definition: $|\pi \oplus \pi'| = |\pi| + |\pi'| 2 |\pi \cap \pi'|$.
- $\begin{array}{c} \textbf{ 0} \quad \text{Combining with the above, we have} \\ |\pi| + |\pi'| 2 \, |\pi \cap \pi'| \geq 2 \, |\pi| \\ \implies \quad |\pi'| \geq |\pi| + 2 \, |\pi \cap \pi'| \, . \end{array}$

- ① Consider the matching $N=M\oplus \pi \oplus \pi'$.
- |N| = |M| + 2.
- $oldsymbol{0} M \oplus N$ contains two augmenting paths, say σ_1 and σ_2 (relative to M).
- $egin{aligned} oldsymbol{M} & M \oplus N = \pi \oplus \pi', ext{ and } \ & |\pi \oplus \pi'| = |M \oplus N| \geq |\sigma_1| + |\sigma_2| \,. \end{aligned}$
- $oldsymbol{\sigma}$ π : shortest augmenting path $(M) \implies |\sigma_1| \geq |\pi|$ and $|\sigma_2| \geq |\pi|$.
- $\bullet \implies |\pi \oplus \pi'| \geq |\sigma_1| + |\sigma_2| \geq |\pi| + |\pi| = 2|\pi|.$
- lacksquare By definition: $|\pi \oplus \pi'| = |\pi| + |\pi'| 2 |\pi \cap \pi'|$.
- © Combining with the above, we have
 $$\begin{split} |\pi| + |\pi'| 2 \, |\pi \cap \pi'| &\geq 2 \, |\pi| \\ \Longrightarrow & |\pi'| \geq |\pi| + 2 \, |\pi \cap \pi'| \, . \end{split}$$

- ① Consider the matching $N=M\oplus \pi \oplus \pi'$.
- |N| = |M| + 2.
- $oldsymbol{0} M \oplus N$ contains two augmenting paths, say $oldsymbol{\sigma_1}$ and $oldsymbol{\sigma_2}$ (relative to M).
- $egin{aligned} oldsymbol{M} & M \oplus N = \pi \oplus \pi', ext{ and } \ & |\pi \oplus \pi'| = |M \oplus N| \geq |\sigma_1| + |\sigma_2| \,. \end{aligned}$
- $oldsymbol{\sigma}$ π : shortest augmenting path $(M) \implies |\sigma_1| \geq |\pi|$ and $|\sigma_2| \geq |\pi|$.
- $\bullet \implies |\pi \oplus \pi'| \geq |\sigma_1| + |\sigma_2| \geq |\pi| + |\pi| = 2|\pi|.$
- lacksquare By definition: $|\pi \oplus \pi'| = |\pi| + |\pi'| 2 |\pi \cap \pi'|$.
- $\begin{array}{c|c} \textbf{0} & \text{Combining with the above, we have} \\ |\pi| + |\pi'| 2 \, |\pi \cap \pi'| \geq 2 \, |\pi| \\ \implies |\pi'| \geq |\pi| + 2 \, |\pi \cap \pi'| \end{array}$

- ① Consider the matching $N=M\oplus \pi\oplus \pi'$.
- |N| = |M| + 2.
- $oldsymbol{0} M \oplus N$ contains two augmenting paths, say $oldsymbol{\sigma_1}$ and $oldsymbol{\sigma_2}$ (relative to M).
- $egin{aligned} oldsymbol{M} & M \oplus N = \pi \oplus \pi', ext{ and } \ & |\pi \oplus \pi'| = |M \oplus N| \geq |\sigma_1| + |\sigma_2| \,. \end{aligned}$
- $oldsymbol{\sigma}$ π : shortest augmenting path $(M) \implies |\sigma_1| \geq |\pi|$ and $|\sigma_2| \geq |\pi|$.
- lacksquare By definition: $|\pi \oplus \pi'| = |\pi| + |\pi'| 2 |\pi \cap \pi'|$.
- $\begin{array}{c} \textbf{0} \quad \text{Combining with the above, we have} \\ |\pi| + |\pi'| 2 \, |\pi \cap \pi'| \geq 2 \, |\pi| \\ \implies |\pi'| \geq |\pi| + 2 \, |\pi \cap \pi'| \, . \end{array}$



Corollary

Corollary

.For sequence of augmenting paths used algorithm (always augment the matching along the shortest augmenting path). We have:

$$|\pi_1| \leq |\pi_2| \leq \ldots \leq |\pi_t|.$$

 $oldsymbol{t}$: number of augmenting paths computed by the algorithm.

 $\pi_1, \pi_2, \ldots, \pi_t$: sequence augmenting paths used by algorithm.

Augmenting paths of same length are disjoint

Lemma

For all i and j, such that $|\pi_i| = \cdots = |\pi_j|$, we have that the paths π_i and π_j are vertex disjoint.

Sariel (UIUC) New CS473 25 Fall 2015 25 / 37

- **1** Assume for contradiction: $|\pi_i| = |\pi_j|$, i < j, π_i and π_j are not vertex disjoint j i is minimal.
- ② $\forall k, i < k < j$: π_k is disjoint from π_i and π_j .
- ullet M_i : matching after π_i was applied.
- lacksquare π_j not using any of the edges of $\pi_{i+1},\ldots,\pi_{j-1}$.
- \bullet π_j is an augmenting path for M_i .
- \bullet π_j and π_i share vertices.
 - lacksquare can not be the two endpoints of π_j (since they are free)
 - 2 must be some interval vertex of π_j
 - $\Longrightarrow \pi_i \text{ and } \pi_j \text{ must share an edge}.$
- $|\pi_i \cap \pi_j| \ge 1.$
- By lemma: $|\pi_j| \geq |\pi_i| + 2|\pi_i \cap \pi_j| > |\pi_i|.$
- A contradiction.

- Assume for contradiction: $|\pi_i| = |\pi_j|$, i < j, π_i and π_j are not vertex disjoint j i is minimal.
- ullet M_i : matching after π_i was applied.
- \bullet π_j not using any of the edges of $\pi_{i+1}, \ldots, \pi_{j-1}$.
- \bullet π_j is an augmenting path for M_i .
- \odot π_j and π_i share vertices.
 - lacksquare can not be the two endpoints of π_j (since they are free)
 - 2 must be some interval vertex of π_j .
 - $\Longrightarrow \pi_i \text{ and } \pi_j \text{ must share an edge}.$
- $|\pi_i \cap \pi_j| \ge 1.$
- **1** By lemma: $|\pi_i| \ge |\pi_i| + 2|\pi_i \cap \pi_i| > |\pi_i|$.
- A contradiction.

- **1** Assume for contradiction: $|\pi_i| = |\pi_j|$, i < j, π_i and π_j are not vertex disjoint j i is minimal.
- $oldsymbol{0}$ M_i : matching after π_i was applied.
- \bullet π_j not using any of the edges of $\pi_{i+1},\ldots,\pi_{j-1}$.
- $lacksquare{1}{3}\pi_{j}$ is an augmenting path for M_{i} .
- \bullet π_j and π_i share vertices.
 - lacksquare can not be the two endpoints of π_j (since they are free)
 - 2 must be some interval vertex of π_j .
 - $\Longrightarrow \pi_i \text{ and } \pi_j \text{ must share an edge}.$
- $|\pi_i \cap \pi_j| \ge 1.$
- ullet By lemma: $|\pi_j| \geq |\pi_i| + 2|\pi_i \cap \pi_j| > |\pi_i|$.
- A contradiction.

- Assume for contradiction: $|\pi_i| = |\pi_j|$, i < j, π_i and π_j are not vertex disjoint j i is minimal.
- $oldsymbol{0}$ M_i : matching after π_i was applied.
- $lacktriangledown \pi_j$ not using any of the edges of $\pi_{i+1},\ldots,\pi_{j-1}$.
- \bullet π_j and π_i share vertices.
 - lacksquare can not be the two endpoints of π_j (since they are free)
 - 2 must be some interval vertex of π_j .
 - $\Longrightarrow \pi_i \text{ and } \pi_j \text{ must share an edge}.$
- $|\pi_i \cap \pi_j| \ge 1.$
- ullet By lemma: $|\pi_j| \geq |\pi_i| + 2|\pi_i \cap \pi_j| > |\pi_i|$.
- A contradiction.

- **1** Assume for contradiction: $|\pi_i| = |\pi_j|$, i < j, π_i and π_j are not vertex disjoint j i is minimal.
- $oldsymbol{0}$ M_i : matching after π_i was applied.
- $lacktriangledown \pi_j$ not using any of the edges of $\pi_{i+1},\ldots,\pi_{j-1}$.
- lacktriangledown π_j is an augmenting path for M_i .
- \odot π_j and π_i share vertices.
 - lacksquare can not be the two endpoints of π_j (since they are free)
 - 2 must be some interval vertex of π_j .
 - $\Longrightarrow \pi_i \text{ and } \pi_j \text{ must share an edge}.$
- $|\pi_i \cap \pi_j| \geq 1.$
- ullet By lemma: $|\pi_j| \geq |\pi_i| + 2|\pi_i \cap \pi_j| > |\pi_i|$.
- A contradiction.

- **1** Assume for contradiction: $|\pi_i| = |\pi_j|$, i < j, π_i and π_j are not vertex disjoint j i is minimal.
- $oldsymbol{0}$ M_i : matching after π_i was applied.
- $lacktriangledown \pi_j$ not using any of the edges of $\pi_{i+1},\ldots,\pi_{j-1}$.
- $lackbox{1}{\sigma} \pi_j$ is an augmenting path for M_i .
- \bullet π_j and π_i share vertices.
 - lacksquare can not be the two endpoints of π_j (since they are free)
 - 2 must be some interval vertex of π_j
 - $\Longrightarrow \pi_i \text{ and } \pi_j \text{ must share an edge}.$
- $|\pi_i \cap \pi_j| \ge 1.$
- ullet By lemma: $|\pi_j| \geq |\pi_i| + 2|\pi_i \cap \pi_j| > |\pi_i|$.
 - A contradiction.

- Assume for contradiction: $|\pi_i| = |\pi_j|$, i < j, π_i and π_j are not vertex disjoint j i is minimal.
- $oldsymbol{0}$ M_i : matching after π_i was applied.
- $lacktriangledown_j$ not using any of the edges of $\pi_{i+1},\ldots,\pi_{j-1}$.
- $lackbox{1}{\sigma} \pi_j$ is an augmenting path for M_i .
- \bullet π_j and π_i share vertices.
 - lacksquare can not be the two endpoints of π_j (since they are free)
 - 2 must be some interval vertex of π_j
 - $\Longrightarrow \pi_i \text{ and } \pi_j \text{ must share an edge}.$
- $|\pi_i \cap \pi_j| \ge 1.$
- | By lemma: $|\pi_j| \geq |\pi_i| + 2|\pi_i \cap \pi_j| > |\pi_i|$.
 - A contradiction.

- **1** Assume for contradiction: $|\pi_i| = |\pi_j|$, i < j, π_i and π_j are not vertex disjoint j i is minimal.
- **1** M_i : matching after π_i was applied.
- $lacktriangledown_j$ not using any of the edges of $\pi_{i+1},\ldots,\pi_{j-1}$.
- $lackbox{1}{\sigma} \pi_j$ is an augmenting path for M_i .
- \bullet π_j and π_i share vertices.
 - lacksquare can not be the two endpoints of π_j (since they are free)
 - 2 must be some interval vertex of π_j .
 - $\Longrightarrow \pi_i \text{ and } \pi_j \text{ must share an edge}.$
- $|\pi_i \cap \pi_j| \ge 1.$
- | By lemma: $|\pi_j| \geq |\pi_i| + 2|\pi_i \cap \pi_j| > |\pi_i|$.
 - A contradiction.

- Assume for contradiction: $|\pi_i| = |\pi_j|$, i < j, π_i and π_j are not vertex disjoint j-i is minimal.
- **1** M_i : matching after π_i was applied.
- $lacktriangledown_j$ not using any of the edges of $\pi_{i+1},\ldots,\pi_{j-1}$.
- $lackbox{1}{\sigma} \pi_j$ is an augmenting path for M_i .
- \bullet π_j and π_i share vertices.
 - lacksquare can not be the two endpoints of π_j (since they are free)
 - 2 must be some interval vertex of π_j .
 - $\Longrightarrow \pi_i$ and π_j must share an edge.
- $|\pi_i \cap \pi_j| \geq 1.$
- **1** By lemma: $|\pi_j| \ge |\pi_i| + 2|\pi_i \cap \pi_j| > |\pi_i|$.
 - A contradiction.

- Assume for contradiction: $|\pi_i| = |\pi_j|$, i < j, π_i and π_j are not vertex disjoint j i is minimal.
- $oldsymbol{0}$ M_i : matching after π_i was applied.
- ullet π_j not using any of the edges of $\pi_{i+1},\ldots,\pi_{j-1}$.
- \bullet π_j is an augmenting path for M_i .
- \bullet π_j and π_i share vertices.
 - lacksquare can not be the two endpoints of π_j (since they are free)
 - 2 must be some interval vertex of π_j .
 - $\Longrightarrow \pi_i$ and π_j must share an edge.
- $|\pi_i \cap \pi_j| \geq 1.$
- **1** By lemma: $|\pi_j| \ge |\pi_i| + 2|\pi_i \cap \pi_j| > |\pi_i|$.
 - A contradiction.

- **1** Assume for contradiction: $|\pi_i| = |\pi_j|$, i < j, π_i and π_j are not vertex disjoint j i is minimal.
- $oldsymbol{0}$ M_i : matching after π_i was applied.
- $lacktriangledown \pi_j$ not using any of the edges of $\pi_{i+1},\ldots,\pi_{j-1}$.
- $lackbox{1}{\sigma} \pi_j$ is an augmenting path for M_i .
- \bullet π_j and π_i share vertices.
 - lacksquare can not be the two endpoints of π_j (since they are free)
 - 2 must be some interval vertex of π_j .
 - $\Longrightarrow \pi_i$ and π_j must share an edge.
- $|\pi_i \cap \pi_j| \geq 1.$
- **3** By lemma: $|\pi_j| \ge |\pi_i| + 2|\pi_i \cap \pi_j| > |\pi_i|$.
 - A contradiction.

- **1** Assume for contradiction: $|\pi_i| = |\pi_j|$, i < j, π_i and π_j are not vertex disjoint j i is minimal.
- **1** M_i : matching after π_i was applied.
- $lacktriangledown_j$ not using any of the edges of $\pi_{i+1},\ldots,\pi_{j-1}$.
- $lackbox{1}{\sigma} \pi_j$ is an augmenting path for M_i .
- \bullet π_j and π_i share vertices.
 - lacksquare can not be the two endpoints of π_j (since they are free)
 - 2 must be some interval vertex of π_j .
 - $\Longrightarrow \pi_i$ and π_j must share an edge.
- $|\pi_i \cap \pi_j| \geq 1.$
- lacksquare By lemma: $|\pi_j| \geq |\pi_i| + 2|\pi_i \cap \pi_j| > |\pi_i|$.
- A contradiction.

16.3.2.2:Improved algorithm for bipartite maximum size matching

- extract all possible augmenting shortest paths of a certain length in one iteration.
- 2 Assume: given a matching can exact all augmenting paths of length k for M in G in O(m) time, for $k = 1, 3, 5, \ldots$
- Apply this extraction algorithm, till $k=1+2\left\lceil \sqrt{n} \right\rceil$.
- \bigcirc Take $O(km)=O(\sqrt{n}m)$ time.
- lacksquare By the end of this process, matching is of size $|T|-\Omega(\sqrt{n})$. (See below why.)
- Resume regular algorithm that augments one augmenting path at a time.
- **3** After $O(\sqrt{n})$ regular iterations we would be done.

- extract all possible augmenting shortest paths of a certain length in one iteration.
- ② Assume: given a matching can exact all augmenting paths of length k for M in G in O(m) time, for $k = 1, 3, 5, \ldots$
- ① Apply this extraction algorithm, till $k=1+2\left\lceil \sqrt{n} \right\rceil$.
- $ext{ } ext{ }$
- lacksquare By the end of this process, matching is of size $|T|-\Omega(\sqrt{n})$. (See below why.)
- Resume regular algorithm that augments one augmenting path at a time.
- 3 After $O(\sqrt{n})$ regular iterations we would be done.

- extract all possible augmenting shortest paths of a certain length in one iteration.
- ② Assume: given a matching can exact all augmenting paths of length k for M in G in O(m) time, for $k = 1, 3, 5, \ldots$
- **3** Apply this extraction algorithm, till $k=1+2\left\lceil \sqrt{n} \right\rceil$.
- $ext{ } ext{ }$
- lacksquare By the end of this process, matching is of size $|T|-\Omega(\sqrt{n})$. (See below why.)
- Resume regular algorithm that augments one augmenting path at a time.
- **3** After $O(\sqrt{n})$ regular iterations we would be done.

- extract all possible augmenting shortest paths of a certain length in one iteration.
- 2 Assume: given a matching can exact all augmenting paths of length k for M in G in O(m) time, for $k = 1, 3, 5, \ldots$
- **3** Apply this extraction algorithm, till $k=1+2\left\lceil \sqrt{n} \right\rceil$.
- $\ \, \textbf{ Take } O(km) = O(\sqrt{n}m) \text{ time.}$
- T: maximum matching.
- lacksquare By the end of this process, matching is of size $|T|-\Omega(\sqrt{n})$. (See below why.)
- Resume regular algorithm that augments one augmenting path at a time.
- ① After $O(\sqrt{n})$ regular iterations we would be done.

- extract all possible augmenting shortest paths of a certain length in one iteration.
- ② Assume: given a matching can exact all augmenting paths of length k for M in G in O(m) time, for $k = 1, 3, 5, \ldots$
- **3** Apply this extraction algorithm, till $k=1+2\left\lceil \sqrt{n} \right\rceil$.
- lacksquare Take $O(km) = O(\sqrt{n}m)$ time.
- T: maximum matching.
- lacksquare By the end of this process, matching is of size $|T|-\Omega(\sqrt{n})$. (See below why.)
- Resume regular algorithm that augments one augmenting path at a time.
- \bigcirc After $O(\sqrt{n})$ regular iterations we would be done.

- extract all possible augmenting shortest paths of a certain length in one iteration.
- ② Assume: given a matching can exact all augmenting paths of length k for M in G in O(m) time, for $k = 1, 3, 5, \ldots$
- lacksquare Apply this extraction algorithm, till $k=1+2\left\lceil \sqrt{n}
 ight
 ceil.$
- lacksquare Take $O(km) = O(\sqrt{n}m)$ time.
- T: maximum matching.
- lacktriangledown By the end of this process, matching is of size $|T| \Omega(\sqrt{n})$. (See below why.)
- Resume regular algorithm that augments one augmenting path at a time.
- ① After $O(\sqrt{n})$ regular iterations we would be done.

- extract all possible augmenting shortest paths of a certain length in one iteration.
- ② Assume: given a matching can exact all augmenting paths of length k for M in G in O(m) time, for $k = 1, 3, 5, \ldots$
- **3** Apply this extraction algorithm, till $k=1+2\left\lceil \sqrt{n} \right\rceil$.
- lacksquare Take $O(km) = O(\sqrt{n}m)$ time.
- T: maximum matching.
- ullet By the end of this process, matching is of size $|T| \Omega(\sqrt{n})$. (See below why.)
- Resume regular algorithm that augments one augmenting path at a time.
- 3 After $O(\sqrt{n})$ regular iterations we would be done.

- extract all possible augmenting shortest paths of a certain length in one iteration.
- ② Assume: given a matching can exact all augmenting paths of length k for M in G in O(m) time, for $k = 1, 3, 5, \ldots$
- lacksquare Apply this extraction algorithm, till $k=1+2\left\lceil \sqrt{n}
 ight
 ceil.$
- lacksquare Take $O(km) = O(\sqrt{n}m)$ time.
- 5 T: maximum matching.
- lacktriangledown By the end of this process, matching is of size $|T| \Omega(\sqrt{n})$. (See below why.)
- Resume regular algorithm that augments one augmenting path at a time.
- **3** After $O(\sqrt{n})$ regular iterations we would be done.

Analysis...

Lemma

Consider the iterative algorithm that applies shortest path augmenting path to the current matching, and let M be the first matching such that the shortest path augmenting path for it is of length $\geq \sqrt{n}$, where n is the number of vertices in the input graph \mathbf{G} . Let \mathbf{T} be the maximum matching. Then $|\mathbf{T}| \leq |\mathbf{M}| + O(\sqrt{n})$.

Proof.

- ① Shortest augmenting path for the current matching M is of length at $\geq \sqrt{n}$.
- ② T: the maximum matching.
- ullet We proved: \exists augmenting path of length $\leq 2n/(|T|-|M|)+1.$
- Together:

$$\sqrt{n} \leq rac{2n}{|T|-|M|}+1$$



Proof.

- Shortest augmenting path for the current matching M is of length at $\geq \sqrt{n}$.
- $oldsymbol{2}$ $oldsymbol{T}$: the maximum matching.
- ullet We proved: \exists augmenting path of length $\leq 2n/(|T|-|M|)+1.$
- Together:

$$\sqrt{n} \le rac{2n}{|T| - |M|} + 1$$

Proof.

- Shortest augmenting path for the current matching M is of length at $\geq \sqrt{n}$.
- 2 T: the maximum matching.
- ullet We proved: \exists augmenting path of length $\leq 2n/(|T|-|M|)+1.$
- Together:

$$\sqrt{n} \leq rac{2n}{|T|-|M|}+1$$

Proof.

- Shortest augmenting path for the current matching M is of length at $\geq \sqrt{n}$.
- $oldsymbol{2}$ T: the maximum matching.
- ullet We proved: \exists augmenting path of length $\leq 2n/(|T|-|M|)+1.$
- Together:

$$\sqrt{n} \le \frac{2n}{|T| - |M|} + 1,$$

Proof...

Proof.

- Shortest augmenting path for the current matching M is of length at $\geq \sqrt{n}$.
- $oldsymbol{2}$ $oldsymbol{T}$: the maximum matching.
- **③** We proved: \exists augmenting path of length $\leq 2n/(|T|-|M|)+1$.
- Together:

$$\sqrt{n} \leq \frac{2n}{|T|-|M|}+1,$$

Proof...

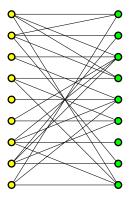
Proof.

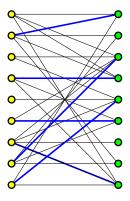
- Shortest augmenting path for the current matching M is of length at $\geq \sqrt{n}$.
- $oldsymbol{2}$ $oldsymbol{T}$: the maximum matching.
- **③** We proved: \exists augmenting path of length $\leq 2n/(|T|-|M|)+1$.
- Together:

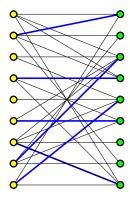
$$\sqrt{n} \leq \frac{2n}{|T| - |M|} + 1,$$

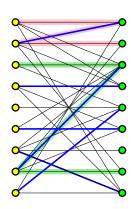


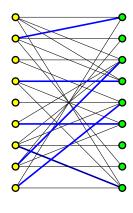
 $16.3.2.3 \\ \vdots \\ \text{Extracting many augmenting paths}$

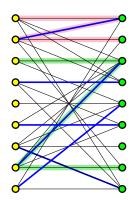


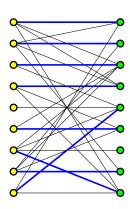


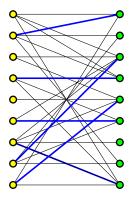


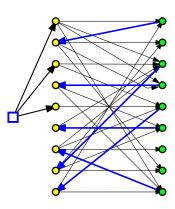


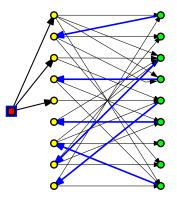


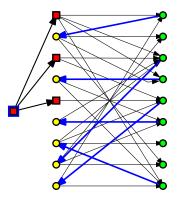


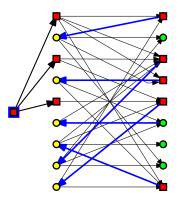


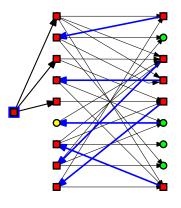


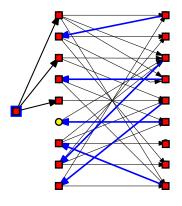


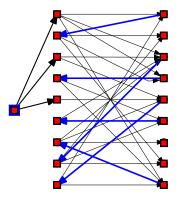




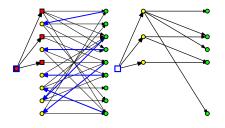




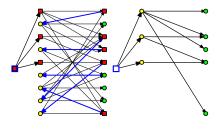




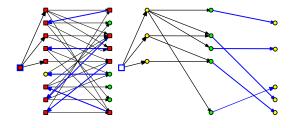
The layered graph



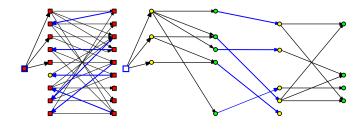
The layered graph



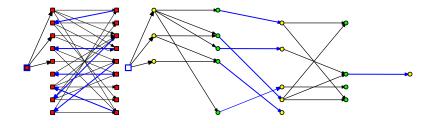
The layered graph



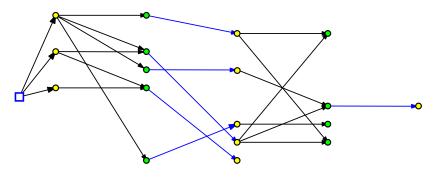
The layered graph



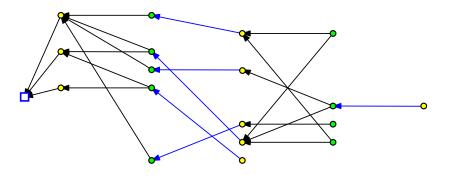
The layered graph



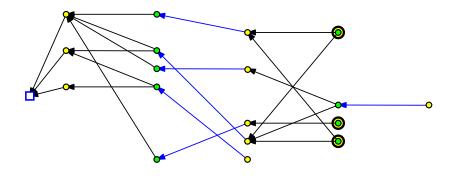
The reverse layered graph and extracting paths



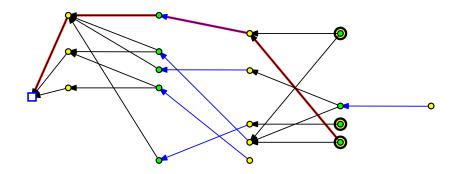
The reverse layered graph and extracting paths



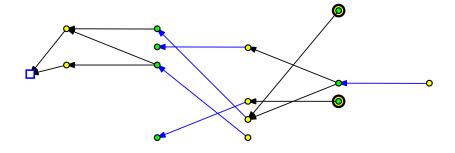
The reverse layered graph and extracting paths



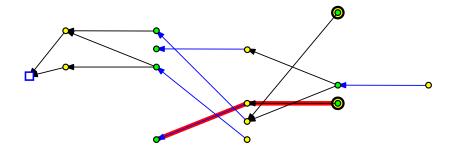
The reverse layered graph and extracting paths



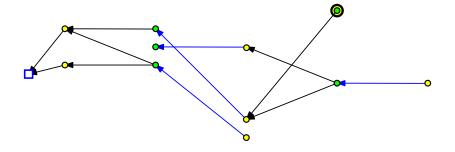
The reverse layered graph and extracting paths



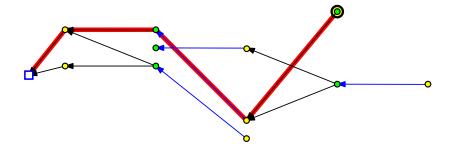
The reverse layered graph and extracting paths



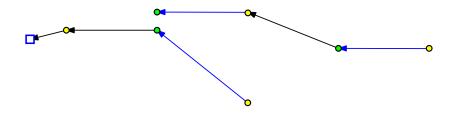
The reverse layered graph and extracting paths

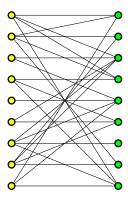


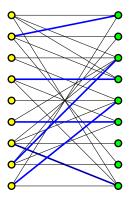
The reverse layered graph and extracting paths

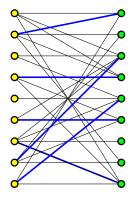


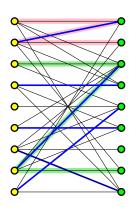
The reverse layered graph and extracting paths

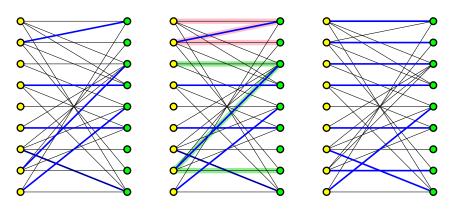












- Idea: build data-structure that is similar to BFS tree.
- ② Input: G, a matching M, and a parameter k, where k odd integer.
- ullet Assumption: Length shortest augmenting path for M is k.
- Task: Extract as many augmenting paths as possible. Vertex disjoint. Of length k
- **F**: set of free vertices in <math> **G**.
- Build directed graph:
 - $lacksymbol{0}$ s: source vertex connected to all vertices of $L_1=L\cap F$.
 - Q direct edges of G from left to right, and matching edges from right to left.
 - H: resulting graph.
- ② Compute **BFS** on the graph **H** starting at s, and let $\mathfrak T$ be the resulting tree.
- lacksquare $L_1, R_1, L_2, R_2, L_3, \ldots$ be the layers of the **BFS**.

- Idea: build data-structure that is similar to BFS tree.
- ② Input: G, a matching M, and a parameter k, where k odd integer.
- $lacksquare{3}$ Assumption: Length shortest augmenting path for M is k.
- Task: Extract as many augmenting paths as possible. Vertex disjoint. Of length k
- O Build directed graph:
 - $lacksymbol{0}$ s: source vertex connected to all vertices of $L_1=L\cap F$.
 - Q direct edges of G from left to right, and matching edges from right to left.
 - 6 H: resulting graph.
- ② Compute **BFS** on the graph **H** starting at s, and let $\mathfrak T$ be the resulting tree.
- lacksquare $L_1, R_1, L_2, R_2, L_3, \ldots$ be the layers of the **BFS**.

- Idea: build data-structure that is similar to BFS tree.
- ② Input: G, a matching M, and a parameter k, where k odd integer.
- $lacksquare{3}$ Assumption: Length shortest augmenting path for M is k.
- ullet Task: Extract as many augmenting paths as possible. Vertex disjoint. Of length ${m k}$
- \bullet F: set of free vertices in \bullet .
- Build directed graph:
 - $oldsymbol{0}$ s: source vertex connected to all vertices of $L_1=L\cap F$.
 - Q direct edges of G from left to right, and matching edges from right to left.
 - 6 H: resulting graph.
- ② Compute **BFS** on the graph **H** starting at s, and let $\mathfrak T$ be the resulting tree.
- lacksquare $L_1, R_1, L_2, R_2, L_3, \ldots$ be the layers of the **BFS**.

- Idea: build data-structure that is similar to BFS tree.
- ② Input: G, a matching M, and a parameter k, where k odd integer.
- **3** Assumption: Length shortest augmenting path for M is k.
- ullet Task: Extract as many augmenting paths as possible. Vertex disjoint. Of length ${m k}$
- **5** F: set of free vertices in **G**.
- Build directed graph:
 - $oldsymbol{0}$ s: source vertex connected to all vertices of $L_1=L\cap F$.
 - Q direct edges of G from left to right, and matching edges from right to left.
 - 6 H: resulting graph.
- ② Compute **BFS** on the graph **H** starting at s, and let $\mathfrak T$ be the resulting tree.
- lacksquare $L_1, R_1, L_2, R_2, L_3, \ldots$ be the layers of the **BFS**.

- Idea: build data-structure that is similar to BFS tree.
- ② Input: G, a matching M, and a parameter k, where k odd integer.
- **3** Assumption: Length shortest augmenting path for M is k.
- ullet Task: Extract as many augmenting paths as possible. Vertex disjoint. Of length ${m k}$
- F: set of free vertices in G.
- Build directed graph:
 - lacksquare s: source vertex connected to all vertices of $L_1=L\cap F$.
 - direct edges of G from left to right, and matching edges from right to left.
 - H: resulting graph.
- O Compute **BFS** on the graph **H** starting at s, and let \mathfrak{T} be the resulting tree.
- $L_1, R_1, L_2, R_2, L_3, \dots$ be the layers of the BFS.

Sariel (UIUC) New CS473 37 Fall 2015 37 / 37

- Idea: build data-structure that is similar to BFS tree.
- ② Input: G, a matching M, and a parameter k, where k odd integer.
- **3** Assumption: Length shortest augmenting path for M is k.
- ullet Task: Extract as many augmenting paths as possible. Vertex disjoint. Of length ${m k}$
- **5** F: set of free vertices in G.
- Build directed graph:
 - lacksquare s: source vertex connected to all vertices of $L_1=L\cap F$.
 - Q direct edges of G from left to right, and matching edges from right to left.
 - H: resulting graph.
- **②** Compute **BFS** on the graph **H** starting at s, and let \mathfrak{T} be the resulting tree.
- $lacksquare L_1, R_1, L_2, R_2, L_3, \ldots$ be the layers of the **BFS**.

Sariel (UIUC) New CS473 37 Fall 2015 37 / 37

- Idea: build data-structure that is similar to BFS tree.
- ② Input: G, a matching M, and a parameter k, where k odd integer.
- **3** Assumption: Length shortest augmenting path for M is k.
- lacksquare Task: Extract as many augmenting paths as possible. Vertex disjoint. Of length $m{k}$
- **5F**: set of free vertices in**G**.
- Build directed graph:
 - $oldsymbol{s}$: source vertex connected to all vertices of $L_1=L\cap F$.
 - Q direct edges of G from left to right, and matching edges from right to left.
 - **H**: resulting graph.
- **②** Compute **BFS** on the graph **H** starting at s, and let \mathfrak{T} be the resulting tree.
- \bullet $L_1, R_1, L_2, R_2, L_3, \ldots$ be the layers of the BFS.

Sariel (UIUC) New CS473 37 Fall 2015 37 / 37

- ① By assumption: first free vertex below L_1 encountered is at level $R_ au$, where $au=\lceil k/2
 ceil$.
- Scan edges of H.
- Add forward edges to tree.
- \bullet ... edge between two vertices that belong to two consecutive levels of the **BFS** tree \Im .
- 5 J be the resulting graph.
- \bullet **J** is a DAG (which is an enrichment of the original tree \circ).
- Compute also the reverse graph J^{rev} (where, we just reverse the edges).

- lacktriangledown By assumption: first free vertex below L_1 encountered is at level $R_ au$, where $au=\lceil k/2
 ceil$.
- Scan edges of H.
- Add forward edges to tree.
- ullet ... edge between two vertices that belong to two consecutive levels of the **BFS** tree ${\mathfrak T}$.
- 5 J be the resulting graph.
- \bullet **J** is a DAG (which is an enrichment of the original tree \circ).
- Compute also the reverse graph J^{rev} (where, we just reverse the edges).

- lacksquare By assumption: first free vertex below L_1 encountered is at level $R_ au$, where $au=\lceil k/2
 ceil$.
- Scan edges of H.
- Add forward edges to tree.
- edge between two vertices that belong to two consecutive levels of the BFS tree T.
- 5 J be the resulting graph.
- \bullet **J** is a DAG (which is an enrichment of the original tree \circ).
- Compute also the reverse graph J^{rev} (where, we just reverse the edges).

- lacksquare By assumption: first free vertex below L_1 encountered is at level $R_ au$, where $au=\lceil k/2
 ceil$.
- Scan edges of H.
- 3 Add forward edges to tree.
- \bullet ... edge between two vertices that belong to two consecutive levels of the **BFS** tree \Im .
- 5 J be the resulting graph.
- \bullet **J** is a DAG (which is an enrichment of the original tree \circ).
- \bigcirc Compute also the reverse graph $\mathbf{J}^{\mathrm{rev}}$ (where, we just reverse the edges).

- f 0 By assumption: first free vertex below L_1 encountered is at level $R_ au$, where $au=\lceil k/2
 ceil$.
- Scan edges of H.
- 3 Add forward edges to tree.
- ullet ... edge between two vertices that belong to two consecutive levels of the BFS tree ${\mathfrak T}$.
- 5 J be the resulting graph.
- \bullet **J** is a DAG (which is an enrichment of the original tree \circ).
- \odot Compute also the reverse graph $\mathbf{J}^{\mathrm{rev}}$ (where, we just reverse the edges).

- f 0 By assumption: first free vertex below L_1 encountered is at level $R_ au$, where $au=\lceil k/2
 ceil$.
- Scan edges of H.
- 3 Add forward edges to tree.
- ullet ... edge between two vertices that belong to two consecutive levels of the BFS tree ${\mathfrak T}$.
- **5** J be the resulting graph.
- \odot **J** is a DAG (which is an enrichment of the original tree \Im).
- \bigcirc Compute also the reverse graph $\mathbf{J}^{\mathrm{rev}}$ (where, we just reverse the edges).

- lacksquare By assumption: first free vertex below L_1 encountered is at level $R_ au$, where $au=\lceil k/2
 ceil$.
- Scan edges of H.
- Add forward edges to tree.
- ullet ... edge between two vertices that belong to two consecutive levels of the BFS tree ${\mathfrak T}$.
- **5** J be the resulting graph.
- **5** J is a DAG (which is an enrichment of the original tree \mathfrak{T}).
- \bigcirc Compute also the reverse graph $\mathbf{J}^{\mathrm{rev}}$ (where, we just reverse the edges).

Sariel (UIUC) New CS473 38 Fall 2015 38 / 37

- f 0 By assumption: first free vertex below L_1 encountered is at level $R_ au$, where $au=\lceil k/2
 ceil$.
- Scan edges of H.
- Add forward edges to tree.
- ullet ... edge between two vertices that belong to two consecutive levels of the BFS tree ${\mathfrak T}$.
- **5** J be the resulting graph.
- **5** J is a DAG (which is an enrichment of the original tree \mathfrak{T}).
- Compute also the reverse graph J^{rev} (where, we just reverse the edges).

- $oldsymbol{0} F_{ au}=R_{ au}\cap F$: free vertices of distance k from free vertices of $L_1.$
- ② $\forall v \in F_{ au}$ do a <code>DFS</code> in <code>J^{rev}</code> till the <code>DFS</code> reaches a vertex of L_1 .
- Mark all the vertices visited by the DFS as "used" thus not allowing any future DFS to use these vertices (i.e., the DFS ignore edges leading to used vertices).
- If the DFS succeeds, extract shortest path found, and add it to the collection of augmenting paths.
- Otherwise, move on to the next vertex in $F_{ au}$, till visit all such vertices.
- $exttt{ iny Results: collection of augmenting paths <math>P_{ au_t}$
 - vertex disjoint.
 - All of length k

- $oldsymbol{0} F_{ au}=R_{ au}\cap F$: free vertices of distance k from free vertices of $L_1.$
- extstyle ext
- Mark all the vertices visited by the DFS as "used" thus not allowing any future DFS to use these vertices (i.e., the DFS ignore edges leading to used vertices).
- If the DFS succeeds, extract shortest path found, and add it to the collection of augmenting paths.
- ullet Otherwise, move on to the next vertex in $m{F_{ au}}$, till visit all such vertices.
- $exttt{ iny Results: collection of augmenting paths } P_ au,$
 - vertex disjoint.
 - $oldsymbol{0}$ All of length $oldsymbol{k}$

- $oldsymbol{0} F_{ au}=R_{ au}\cap F$: free vertices of distance k from free vertices of $L_1.$
- extstyle ext
- Mark all the vertices visited by the DFS as "used" thus not allowing any future DFS to use these vertices (i.e., the DFS ignore edges leading to used vertices).
- If the DFS succeeds, extract shortest path found, and add it to the collection of augmenting paths.
- Otherwise, move on to the next vertex in $F_{ au}$, till visit all such vertices.
- $exttt{0}$ Results: collection of augmenting paths $P_{ au}$,
 - vertex disjoint.
 - $oldsymbol{o}$ All of length $oldsymbol{k}$

- $oldsymbol{0} F_{ au}=R_{ au}\cap F$: free vertices of distance k from free vertices of $L_1.$
- ② $\forall v \in F_{ au}$ do a <code>DFS</code> in <code>J^{rev}</code> till the <code>DFS</code> reaches a vertex of L_1 .
- Mark all the vertices visited by the DFS as "used" thus not allowing any future DFS to use these vertices (i.e., the DFS ignore edges leading to used vertices).
- If the DFS succeeds, extract shortest path found, and add it to the collection of augmenting paths.
- Otherwise, move on to the next vertex in $F_{ au}$, till visit all such vertices.
- $exttt{0}$ Results: collection of augmenting paths $P_{ au}$,
 - vertex disjoint.
 - All of length k

- $oldsymbol{0} F_{ au}=R_{ au}\cap F$: free vertices of distance k from free vertices of $L_1.$
- ② $orall v \in F_ au$ do a <code>DFS</code> in <code>J^{rev}</code> till the <code>DFS</code> reaches a vertex of L_1 .
- Mark all the vertices visited by the DFS as "used" thus not allowing any future DFS to use these vertices (i.e., the DFS ignore edges leading to used vertices).
- If the DFS succeeds, extract shortest path found, and add it to the collection of augmenting paths.
- **1** Otherwise, move on to the next vertex in F_{τ} , till visit all such vertices.
- $exttt{0}$ Results: collection of augmenting paths $P_{ au}$,
 - vertex disjoint.
 - All of length k

- $oldsymbol{0} F_{ au} = R_{ au} \cap F$: free vertices of distance k from free vertices of $L_1.$
- ② $\forall v \in F_{ au}$ do a <code>DFS</code> in <code>J^{rev}</code> till the <code>DFS</code> reaches a vertex of L_1 .
- Mark all the vertices visited by the DFS as "used" thus not allowing any future DFS to use these vertices (i.e., the DFS ignore edges leading to used vertices).
- If the DFS succeeds, extract shortest path found, and add it to the collection of augmenting paths.
- **1** Otherwise, move on to the next vertex in F_{τ} , till visit all such vertices.
- **o** Results: collection of augmenting paths $P_{ au}$,
 - vertex disjoint.
 - All of length k

- $oldsymbol{0} F_{ au}=R_{ au}\cap F$: free vertices of distance k from free vertices of $L_1.$
- ② $\forall v \in F_{ au}$ do a <code>DFS</code> in <code>J^{rev}</code> till the <code>DFS</code> reaches a vertex of L_1 .
- Mark all the vertices visited by the DFS as "used" thus not allowing any future DFS to use these vertices (i.e., the DFS ignore edges leading to used vertices).
- If the DFS succeeds, extract shortest path found, and add it to the collection of augmenting paths.
- **1** Otherwise, move on to the next vertex in F_{τ} , till visit all such vertices.
- **o** Results: collection of augmenting paths $P_{ au}$,
 - vertex disjoint.
 - All of length k

- $oldsymbol{0} F_{ au}=R_{ au}\cap F$: free vertices of distance k from free vertices of $L_1.$
- extstyle ext
- Mark all the vertices visited by the DFS as "used" thus not allowing any future DFS to use these vertices (i.e., the DFS ignore edges leading to used vertices).
- If the DFS succeeds, extract shortest path found, and add it to the collection of augmenting paths.
- **1** Otherwise, move on to the next vertex in F_{τ} , till visit all such vertices.
- **1** Results: collection of augmenting paths $P_{ au}$,
 - vertex disjoint.
 - \bigcirc All of length k.

- lacksquare Building initial graphs lacksquare and lacksquare takes O(m) time.
- Charge running time of the second stage to the edges and vertices visited.
- Any vertex visited by any DFS is never going to be visited again...
- edge of J^{rev} is going to be considered only once by algorithm.
- \bigcirc running time of the algorithm is O(n+m).

- lacksquare Building initial graphs lacksquare and lacksquare takes O(m) time.
- Charge running time of the second stage to the edges and vertices visited.
- Any vertex visited by any DFS is never going to be visited again...
- edge of J^{rev} is going to be considered only once by algorithm.
- \bigcirc \Longrightarrow running time of the algorithm is O(n+m).

- lacksquare Building initial graphs lacksquare and lacksquare takes O(m) time.
- Charge running time of the second stage to the edges and vertices visited.
- Any vertex visited by any DFS is never going to be visited again...
- edge of J^{rev} is going to be considered only once by algorithm.
- \Longrightarrow running time of the algorithm is O(n+m).

- **1** Building initial graphs **J** and J^{rev} takes O(m) time.
- Charge running time of the second stage to the edges and vertices visited.
- Any vertex visited by any DFS is never going to be visited again...
- edge of J^{rev} is going to be considered only once by algorithm.
- lacksquare running time of the algorithm is O(n+m).

Maximal set of disjoint augmenting paths

Lemma

The set P_k is a maximal set of vertex-disjoint augmenting paths of length k for M.

Proof.

- $oldsymbol{0} M'$ be the result of augmenting M with the paths of P_k .
- ② Assume for sake of contradiction: P_k is not maximal.
- $lacksymbol{\circ}$ That is: eta augmenting path $oldsymbol{\sigma}$ of length $oldsymbol{k}$ disjoint from paths of $oldsymbol{P_k}.$
- Algorithm could traverse σ in J,
- 5 ... would go through unused vertices.
- lacktriangle Indeed, if any vertices of $oldsymbol{\sigma}$ were used by any of the back DFS,
- \bigcirc \Longrightarrow resulted in a path that goes to a free vertex in L_1 .
- \circledcirc \Longrightarrow a contradiction: σ is supposedly disjoint from the paths of P_k .

Sariel (UIUC) New CS473 42 Fall 2015 42 / 37

- $oldsymbol{0} M'$ be the result of augmenting M with the paths of P_k .
- ② Assume for sake of contradiction: P_k is not maximal.
- ${}^{\circ}$ That is: \exists augmenting path σ of length k disjoint from paths of P_k .
- Algorithm could traverse σ in J,
- 5 ... would go through unused vertices.
- \odot Indeed, if any vertices of σ were used by any of the back **DFS**,
- $extstyle igoplus ext{resulted}$ in a path that goes to a free vertex in L_1 .
- \Longrightarrow a contradiction: σ is supposedly disjoint from the paths of P_k .



- $oldsymbol{0} M'$ be the result of augmenting M with the paths of P_k .
- ② Assume for sake of contradiction: P_k is not maximal.
- **3** That is: \exists augmenting path σ of length k disjoint from paths of P_k .
- Algorithm could traverse σ in J,
- 5 ... would go through unused vertices.
- \odot Indeed, if any vertices of σ were used by any of the back **DFS**,
- $extcolor{o} \implies$ resulted in a path that goes to a free vertex in L_1 .
- \circledcirc \Longrightarrow a contradiction: σ is supposedly disjoint from the paths of P_k .



- $oldsymbol{0} M'$ be the result of augmenting M with the paths of P_k .
- ② Assume for sake of contradiction: P_k is not maximal.
- **3** That is: \exists augmenting path σ of length k disjoint from paths of P_k .
- **4** Algorithm could traverse σ in **J**,
- 5 ... would go through unused vertices.
- lacktriangle Indeed, if any vertices of $oldsymbol{\sigma}$ were used by any of the back DFS,
- $@\Longrightarrow$ resulted in a path that goes to a free vertex in $L_1.$
- \Longrightarrow a contradiction: σ is supposedly disjoint from the paths of P_k .



- $oldsymbol{0} M'$ be the result of augmenting M with the paths of P_k .
- ② Assume for sake of contradiction: P_k is not maximal.
- **3** That is: \exists augmenting path σ of length k disjoint from paths of P_k .
- 4 Algorithm could traverse σ in J,
- ... would go through unused vertices.
- lacktriangle Indeed, if any vertices of $oldsymbol{\sigma}$ were used by any of the back DFS,
- \bigcirc \Longrightarrow resulted in a path that goes to a free vertex in L_1 .
- \Longrightarrow a contradiction: σ is supposedly disjoint from the paths of P_k .



Proof.

- **1** M' be the result of augmenting M with the paths of P_k .
- ② Assume for sake of contradiction: P_k is not maximal.
- **3** That is: \exists augmenting path σ of length k disjoint from paths of P_k .
- 4 Algorithm could traverse σ in J,
- ... would go through unused vertices.
- **1** Indeed, if any vertices of σ were used by any of the back **DFS**,
- \bigcirc \Longrightarrow resulted in a path that goes to a free vertex in L_1 .
- \Longrightarrow a contradiction: σ is supposedly disjoint from the paths of P_k .

Sariel (UIUC) New CS473 42 Fall 2015 42 / 37

Proof.

- **1** M' be the result of augmenting M with the paths of P_k .
- 2 Assume for sake of contradiction: P_k is not maximal.
- **3** That is: \exists augmenting path σ of length k disjoint from paths of P_{k} .
- 4 Algorithm could traverse σ in J,
- ... would go through unused vertices.
- Indeed, if any vertices of σ were used by any of the back DFS,
- \implies resulted in a path that goes to a free vertex in L_1 .

Sariel (UIUC) New CS473 42 Fall 2015 42 / 37

Proof.

- **1** M' be the result of augmenting M with the paths of P_k .
- 2 Assume for sake of contradiction: P_k is not maximal.
- **3** That is: \exists augmenting path σ of length k disjoint from paths of P_{k} .
- 4 Algorithm could traverse σ in J,
- ... would go through unused vertices.
- **1** Indeed, if any vertices of σ were used by any of the back **DFS**,
- \implies resulted in a path that goes to a free vertex in L_1 .
- \implies a contradiction: σ is supposedly disjoint from the paths of P_{ν} .

Fall 2015 42 / 37 16.3.2.4:The result

The result

Theorem

Given a bipartite unweighted graph ${\bf G}$ with ${\bf n}$ vertices and ${\bf m}$ edges, one can compute maximum matching in ${\bf G}$ in $O(\sqrt{n}{\bf m})$ time.

Sariel (UIUC) New CS473 44 Fall 2015 44 / 37

The proof...

The $\operatorname{algMatching}_{HK}$ algorithm was described, and the running time analysis was also done.

The main challenge is the correctness.

The proof...

The $\operatorname{algMatching}_{HK}$ algorithm was described, and the running time analysis was also done.

The main challenge is the correctness.

Sariel (UIUC) New CS473 45 Fall 2015 45 / 37

- interpret execution of algorithm as simulating the slower and simpler algorithm.
- **2** algMatching $_{HK}$: computes sequence of sets of augmenting paths P_1, P_3, P_5, \ldots
- order augmenting paths in an arbitrary order inside each such set.
- Results: in sequence of augmenting paths that are shortest augmenting paths for the current matching.
- ullet By lemma: each P_k maximal set of vertex-disjoint augmenting paths of length k.
- \odot Other lemma: all aug. paths of len k computed: vertex disjoint.
- Now by induction: argue that if $\operatorname{algMatching}_{HK}$ simulates correctly $\operatorname{algSlowMatch}$, for the augmenting paths in $P_1 \cup P_3 \cup \ldots P_i$, then it simulates it correctly for $P_1 \cup P_3 \cup \ldots P_i \cup P_{i+1}$. Done.

- interpret execution of algorithm as simulating the slower and simpler algorithm.
- **2** algMatching $_{HK}$: computes sequence of sets of augmenting paths P_1, P_3, P_5, \ldots
- order augmenting paths in an arbitrary order inside each such set.
- Results: in sequence of augmenting paths that are shortest augmenting paths for the current matching.
- ullet By lemma: each P_k maximal set of vertex-disjoint augmenting paths of length k.
- ullet Other lemma: all aug. paths of len k computed: vertex disjoint.
- Now by induction: argue that if $\operatorname{algMatching}_{HK}$ simulates correctly $\operatorname{algSlowMatch}$, for the augmenting paths in $P_1 \cup P_3 \cup \ldots P_i$, then it simulates it correctly for $P_1 \cup P_3 \cup \ldots P_i \cup P_{i+1}$. Done.

- interpret execution of algorithm as simulating the slower and simpler algorithm.
- **2** algMatching $_{HK}$: computes sequence of sets of augmenting paths P_1, P_3, P_5, \ldots
- order augmenting paths in an arbitrary order inside each such set.
- Results: in sequence of augmenting paths that are shortest augmenting paths for the current matching.
- ullet By lemma: each P_k maximal set of vertex-disjoint augmenting paths of length k.
- Now by induction: argue that if $\operatorname{algMatching}_{HK}$ simulates correctly $\operatorname{algSlowMatch}$, for the augmenting paths in $P_1 \cup P_3 \cup \ldots P_i$, then it simulates it correctly for $P_1 \cup P_3 \cup \ldots P_i \cup P_{i+1}$. Done.

- interpret execution of algorithm as simulating the slower and simpler algorithm.
- **2** algMatching $_{HK}$: computes sequence of sets of augmenting paths P_1, P_3, P_5, \ldots
- order augmenting paths in an arbitrary order inside each such set.
- Results: in sequence of augmenting paths that are shortest augmenting paths for the current matching.
- ullet By lemma: each P_k maximal set of vertex-disjoint augmenting paths of length k.
- ullet Other lemma: all aug. paths of len $oldsymbol{k}$ computed: vertex disjoint.
- Now by induction: argue that if $\operatorname{algMatching}_{HK}$ simulates correctly $\operatorname{algSlowMatch}$, for the augmenting paths in $P_1 \cup P_3 \cup \ldots P_i$, then it simulates it correctly for $P_1 \cup P_3 \cup \ldots P_i \cup P_{i+1}$. Done.

- interpret execution of algorithm as simulating the slower and simpler algorithm.
- **2** algMatching $_{HK}$: computes sequence of sets of augmenting paths P_1, P_3, P_5, \ldots
- order augmenting paths in an arbitrary order inside each such set.
- Results: in sequence of augmenting paths that are shortest augmenting paths for the current matching.
- lacktriangledown By lemma: each P_k maximal set of vertex-disjoint augmenting paths of length k.
- ullet Other lemma: all aug. paths of len $oldsymbol{k}$ computed: vertex disjoint.
- Now by induction: argue that if $\operatorname{algMatching}_{HK}$ simulates correctly $\operatorname{algSlowMatch}$, for the augmenting paths in $P_1 \cup P_3 \cup \ldots P_i$, then it simulates it correctly for $P_1 \cup P_3 \cup \ldots P_i \cup P_{i+1}$. Done.

- interpret execution of algorithm as simulating the slower and simpler algorithm.
- **2** algMatching $_{HK}$: computes sequence of sets of augmenting paths P_1, P_3, P_5, \ldots
- order augmenting paths in an arbitrary order inside each such set.
- Results: in sequence of augmenting paths that are shortest augmenting paths for the current matching.
- lacktriangledown By lemma: each P_k maximal set of vertex-disjoint augmenting paths of length k.
- lacktriangle Other lemma: all aug. paths of len k computed: vertex disjoint.
- Now by induction: argue that if $\operatorname{algMatching}_{HK}$ simulates correctly $\operatorname{algSlowMatch}$, for the augmenting paths in $P_1 \cup P_3 \cup \ldots P_i$, then it simulates it correctly for $P_1 \cup P_3 \cup \ldots P_i \cup P_{i+1}$. Done.

- interpret execution of algorithm as simulating the slower and simpler algorithm.
- **2** algMatching $_{HK}$: computes sequence of sets of augmenting paths P_1, P_3, P_5, \ldots
- order augmenting paths in an arbitrary order inside each such set.
- Results: in sequence of augmenting paths that are shortest augmenting paths for the current matching.
- lacktriangledown By lemma: each P_k maximal set of vertex-disjoint augmenting paths of length k.
- lacktriangle Other lemma: all aug. paths of len $m{k}$ computed: vertex disjoint.
- Now by induction: argue that if $\operatorname{algMatching}_{HK}$ simulates correctly $\operatorname{algSlowMatch}$, for the augmenting paths in $P_1 \cup P_3 \cup \ldots P_i$, then it simulates it correctly for $P_1 \cup P_3 \cup \ldots P_i \cup P_{i+1}$. Done.

Bibliographical notes

The description here follows the original and reasonably well written paper of Hopcroft and Karp Hopcroft and Karp [1973]. Both won the Turing award.

Sariel (UIUC) New CS473 48 Fall 2015 48 / 37

Sariel (UIUC) New CS473 49 Fall 2015 49 / 37

Sariel (UIUC) New CS473 50 Fall 2015 50 / 37

Sariel (UIUC) New CS473 51 Fall 2015 51 / 37

J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2:225–231, 1973.

Sariel (UIUC) New CS473 51 Fall 2015 51 / 37