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 $\mathbf{G}$ *and a weight function on the edges, compute the maximum weight matching in* $\mathbf{G}$*.*

16.2.2: Matchings and alternating paths

Some definitions

1. M: matching.
2. $e \in M$ is a **matching edge**matching!matching edge.
3. $e' \in \mathbf{E(G)} \setminus M$ is **free**.
4. $v \in \mathbf{V(G)}$ **matched** $\iff$ adjacent to edge in M.
5. unmatched vertex v' is **free**.
6. **alternating path**: a simple path edges alternating between matched and free edges.
7. **alternating cycle**...
8. **length** of a path/cycle is the number of edges in it.

Some definitions

1. M: matching.
2. $e \in M$ is a **matching edge**matching!matching edge.
3. $e' \in \mathbf{E(G)} \setminus M$ is **free**.
4. $v \in \mathbf{V(G)}$ **matched** $\iff$ adjacent to edge in M.
5. unmatched vertex v' is **free**.
6. **alternating path**: a simple path edges alternating between matched and free edges.
7. **alternating cycle**...
8. **length** of a path/cycle is the number of edges in it.

Some definitions

1. M: matching.
2. $e \in M$ is a **matching edge**matching!matching edge.
3. $e' \in \mathsf{E}(\mathsf{G}) \setminus M$ is **free**.
4. $v \in \mathsf{V}(\mathsf{G})$ **matched** $\iff$ adjacent to edge in M.
5. unmatched vertex v' is **free**.
6. **alternating path**: a simple path edges alternating between matched and free edges.
7. **alternating cycle**...
8. **length** of a path/cycle is the number of edges in it.

Some definitions

1. M: matching.
2. $e \in M$ is a **matching edge**matching!matching edge.
3. $e' \in \mathsf{E}(\mathsf{G}) \setminus M$ is **free**.
4. $v \in \mathsf{V}(\mathsf{G})$ **matched** $\iff$ adjacent to edge in M.
5. unmatched vertex v' is **free**.
6. **alternating path**: a simple path edges alternating between matched and free edges.
7. **alternating cycle**...
8. **length** of a path/cycle is the number of edges in it.

Some definitions

1. M: matching.
2. $e \in M$ is a **matching edge**matching!matching edge.
3. $e' \in \mathbf{E}(\mathbf{G}) \setminus M$ is **free**.
4. $v \in \mathbf{V}(\mathbf{G})$ **matched** $\iff$ adjacent to edge in M.
5. unmatched vertex v' is **free**.
6. **alternating path**: a simple path edges alternating between matched and free edges.
7. **alternating cycle**...
8. **length** of a path/cycle is the number of edges in it.

Some definitions

1. M: matching.
2. $e \in M$ is a **matching edge**matching!matching edge.
3. $e' \in \mathsf{E}(\mathsf{G}) \setminus M$ is **free**.
4. $v \in \mathsf{V}(\mathsf{G})$ **matched** $\iff$ adjacent to edge in M.
5. unmatched vertex v' is **free**.
6. **alternating path**: a simple path edges alternating between matched and free edges.
7. **alternating cycle**...
8. **length** of a path/cycle is the number of edges in it.

Some definitions

1. M: matching.
2. $e \in M$ is a **matching edge**matching!matching edge.
3. $e' \in \mathbf{E}(\mathbf{G}) \setminus M$ is **free**.
4. $v \in \mathbf{V}(\mathbf{G})$ **matched** $\iff$ adjacent to edge in M.
5. unmatched vertex v' is **free**.
6. **alternating path**: a simple path edges alternating between matched and free edges.
7. **alternating cycle**...
8. **length** of a path/cycle is the number of edges in it.

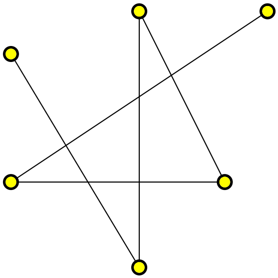
Some definitions

1. M: matching.
2. $e \in M$ is a **matching edge**matching!matching edge.
3. $e' \in \mathbf{E}(\mathbf{G}) \setminus M$ is **free**.
4. $v \in \mathbf{V}(\mathbf{G})$ **matched** $\iff$ adjacent to edge in M.
5. unmatched vertex v' is **free**.
6. **alternating path**: a simple path edges alternating between matched and free edges.
7. **alternating cycle**...
8. **length** of a path/cycle is the number of edges in it.

Some definitions

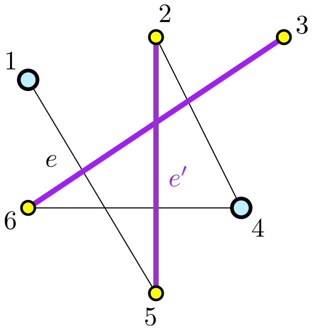
1. M: matching.
2. $e \in M$ is a **matching edge**matching!matching edge.
3. $e' \in \mathbf{E}(\mathbf{G}) \setminus M$ is **free**.
4. $v \in \mathbf{V}(\mathbf{G})$ **matched** $\iff$ adjacent to edge in M.
5. unmatched vertex v' is **free**.
6. **alternating path**: a simple path edges alternating between matched and free edges.
7. **alternating cycle**...
8. **length** of a path/cycle is the number of edges in it.

Example



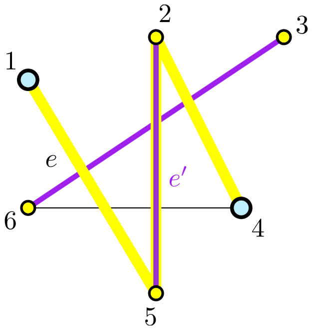
(A) The input graph.

Example



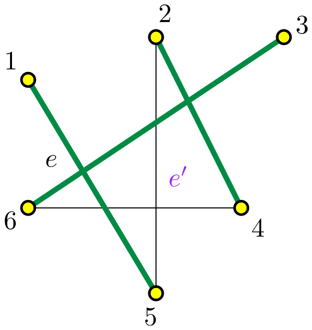
(B) A maximal matching in $\mathbf{G}$. The edge e is free, and vertices 1 and 4 are free.

Example



(C) An alternating path.

Example



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

Augmenting paths

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, $e_i = v_i v_{i+1} \in \mathbf{E}(\mathbf{G})$,

(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) $e_2, e_4, \ldots, e_{2k} \in M$.

Augmenting paths

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, $e_i = v_i v_{i+1} \in \mathbf{E}(\mathbf{G})$,

(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) $e_2, e_4, \ldots, e_{2k} \in M$.

Augmenting paths

Definition

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

(i) π is simple,
(ii) for all i, $e_i = v_i v_{i+1} \in \mathbf{E}(\mathbf{G})$,
(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) $e_2, e_4, \ldots, e_{2k} \in M$.

Augmenting paths

Definition

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

(i) π is simple,

(ii) for all i, $e_i = v_i v_{i+1} \in \mathbf{E}(\mathbf{G})$,

(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) $e_2, e_4, \ldots, e_{2k} \in M$.

Augmenting paths

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) $e_2, e_4, \ldots, e_{2k} \in M$.

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 $|M| + 1$.

Proof.

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

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 $|M| + 1$.

Proof.

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

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 $|M| + 1$.

Proof.

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

□

Many augmenting paths

Lemma

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

Proof.

1. $E' = M \oplus T$. $H = (V, E')$.
2. $\forall v \in \mathsf{V}(H)$: $d(v) \leq 2$.
3. H: collection of alternating paths and cycles.
4. cycles are even length.
5. k more edges of T in $M \oplus T$ than of M.
6. For any cycle $C \in H$: $|C \cap M| = |C \cap T|$.
7. For a path $\pi \in H$: $|\pi \cap M| \leq |\pi \cap T| \leq |\pi \cap M| + 1$.
8. For augmenting path π: $|\pi \cap T| = |\pi \cap M| + 1$.

Many augmenting paths

Lemma

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

Proof.

1. $E' = M \oplus T$. $H = (V, E')$.
2. $\forall v \in \mathsf{V}(H)$: $d(v) \leq 2$.
3. H: collection of alternating paths and cycles.
4. cycles are even length.
5. k more edges of T in $M \oplus T$ than of M.
6. For any cycle $C \in H$: $|C \cap M| = |C \cap T|$.
7. For a path $\pi \in H$: $|\pi \cap M| \leq |\pi \cap T| \leq |\pi \cap M| + 1$.
8. For augmenting path π: $|\pi \cap T| = |\pi \cap M| + 1$.

Many augmenting paths

Lemma

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

Proof.

1. $E' = M \oplus T$. $H = (V, E')$.
2. $\forall v \in \mathsf{V}(H)$: $d(v) \leq 2$.
3. H: collection of alternating paths and cycles.
4. cycles are even length.
5. k more edges of T in $M \oplus T$ than of M.
6. For any cycle $C \in H$: $|C \cap M| = |C \cap T|$.
7. For a path $\pi \in H$: $|\pi \cap M| \leq |\pi \cap T| \leq |\pi \cap M| + 1$.
8. For augmenting path π: $|\pi \cap T| = |\pi \cap M| + 1$.

Many augmenting paths

Lemma

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

Proof.

1. $E' = M \oplus T$. $H = (V, E')$.
2. H: collection of alternating paths and cycles.
3. cycles are even length.
4. k more edges of T in $M \oplus T$ than of M.
5. For any cycle $C \in H$: $|C \cap M| = |C \cap T|$.
6. For a path $\pi \in H$: $|\pi \cap M| \le |\pi \cap T| \le |\pi \cap M| + 1$.
7. For augmenting path π: $|\pi \cap T| = |\pi \cap M| + 1$.
8. $\implies$ Must be k augmenting paths in H.

Many augmenting paths

Lemma

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

Proof.

1. $E' = M \oplus T$. $H = (V, E')$.
2. H: collection of alternating paths and cycles.
3. cycles are even length.
4. k more edges of T in $M \oplus T$ than of M.
5. For any cycle $C \in H$: $|C \cap M| = |C \cap T|$.
6. For a path $\pi \in H$: $|\pi \cap M| \leq |\pi \cap T| \leq |\pi \cap M| + 1$.
7. For augmenting path π: $|\pi \cap T| = |\pi \cap M| + 1$.
8. $\implies$ Must be k augmenting paths in H.

Many augmenting paths

Lemma

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

Proof.

1. $E' = M \oplus T$. $H = (V, E')$.
2. H: collection of alternating paths and cycles.
3. cycles are even length.
4. k more edges of T in $M \oplus T$ than of M.
5. For any cycle $C \in H$: $|C \cap M| = |C \cap T|$.
6. For a path $\pi \in H$: $|\pi \cap M| \leq |\pi \cap T| \leq |\pi \cap M| + 1$.
7. For augmenting path π: $|\pi \cap T| = |\pi \cap M| + 1$.
8. $\implies$ Must be k augmenting paths in H.

Many augmenting paths

Lemma

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

Proof.

1. $E' = M \oplus T$. $H = (V, E')$.
2. H: collection of alternating paths and cycles.
3. cycles are even length.
4. k more edges of T in $M \oplus T$ than of M.
5. For any cycle $C \in H$: $|C \cap M| = |C \cap T|$.
6. For a path $\pi \in H$: $|\pi \cap M| \leq |\pi \cap T| \leq |\pi \cap M| + 1$.
7. For augmenting path π: $|\pi \cap T| = |\pi \cap M| + 1$.
8. $\implies$ Must be k augmenting paths in H.

Many augmenting paths

Lemma

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

Proof.

1. $E' = M \oplus T$. $H = (V, E')$.
2. H: collection of alternating paths and cycles.
3. k more edges of T in $M \oplus T$ than of M.
4. For any cycle $C \in H$: $|C \cap M| = |C \cap T|$.
5. For a path $\pi \in H$: $|\pi \cap M| \leq |\pi \cap T| \leq |\pi \cap M| + 1$.
6. For augmenting path π: $|\pi \cap T| = |\pi \cap M| + 1$.
7. $\implies$ Must be k augmenting paths in H.

Many augmenting paths

Lemma

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

Proof.

1. $E' = M \oplus T$. $H = (V, E')$.
2. H: collection of alternating paths and cycles.
3. k more edges of T in $M \oplus T$ than of M.
4. For any cycle $C \in H$: $|C \cap M| = |C \cap T|$.
5. For a path $\pi \in H$: $|\pi \cap M| \leq |\pi \cap T| \leq |\pi \cap M| + 1$.
6. For augmenting path π: $|\pi \cap T| = |\pi \cap M| + 1$.
7. $\implies$ Must be k augmenting paths in H.

Many augmenting paths

Lemma

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

Proof.

1. $E' = M \oplus T$. $H = (V, E')$.
2. H: collection of alternating paths and cycles.
3. k more edges of T in $M \oplus T$ than of M.
4. For any cycle $C \in H$: $|C \cap M| = |C \cap T|$.
5. For a path $\pi \in H$: $|\pi \cap M| \leq |\pi \cap T| \leq |\pi \cap M| + 1$.
6. For augmenting path π: $|\pi \cap T| = |\pi \cap M| + 1$.
7. $\implies$ Must be k augmenting paths in H.

□

Many augmenting paths

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.

1. $E' = M \oplus T$. $H = (V, E')$.
2. $u = |\mathsf{V}(H)| \leq 2\big(|T| + |M|\big)$.
3. By previous lemma: There are k augmenting paths in H.
4. If all augmenting paths were of length $\geq u/k$
5. $\implies$ total number of vertices in $H \geq (u/k + 1)u > u$
6. ... since a path of length ℓ has $\ell + 1$ vertices. A contradiction.

Many augmenting paths

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.

1. $E' = M \oplus T$. $H = (V, E')$.
2. $u = |\mathsf{V}(H)| \leq 2\big(|T| + |M|\big)$.
3. By previous lemma: There are k augmenting paths in H.
4. If all augmenting paths were of length $\geq u/k$
5. $\implies$ total number of vertices in $H \geq (u/k + 1)u > u$
6. ... since a path of length ℓ has $\ell + 1$ vertices. A contradiction.

Many augmenting paths

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.

1. $E' = M \oplus T$. $H = (V, E')$.
2. $u = |\mathsf{V}(H)| \leq 2\big(|T| + |M|\big)$.
3. By previous lemma: There are k augmenting paths in H.
4. If all augmenting paths were of length $\geq u/k$
5. $\implies$ total number of vertices in $H \geq (u/k + 1)u > u$
6. ... since a path of length ℓ has $\ell + 1$ vertices. A contradiction.

Many augmenting paths

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.

1. $E' = M \oplus T$. $H = (V, E')$.
2. $u = |\mathsf{V}(H)| \leq 2\big(|T| + |M|\big)$.
3. By previous lemma: There are k augmenting paths in H.
4. If all augmenting paths were of length $\geq u/k$
5. $\implies$ total number of vertices in $H \geq (u/k + 1)u > u$
6. ... since a path of length ℓ has $\ell + 1$ vertices. A contradiction.

Many augmenting paths

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.

1. $E' = M \oplus T$. $H = (V, E')$.
2. $u = |\mathsf{V}(H)| \leq 2\big(|T| + |M|\big)$.
3. By previous lemma: There are k augmenting paths in H.
4. If all augmenting paths were of length $\geq u/k$
5. $\implies$ total number of vertices in $H \geq (u/k + 1)u > u$
6. ... since a path of length ℓ has $\ell + 1$ vertices. A contradiction.

□

Many augmenting paths

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.

1. $E' = M \oplus T$. $H = (V, E')$.
2. $u = |\mathsf{V}(H)| \leq 2\big(|T| + |M|\big)$.
3. By previous lemma: There are k augmenting paths in H.
4. If all augmenting paths were of length $\geq u/k$
5. $\implies$ total number of vertices in $H \geq (u/k + 1)u > u$
6. ... since a path of length ℓ has $\ell + 1$ vertices. A contradiction.

□

No augmenting path, no cry

Or: Having a maximum matching.

Corollary

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

16.3: Unweighted matching in bipartite graph

16.3.1: The slow algorithm

The algorithm

1. $\mathsf{G} = (L \cup R, \mathsf{E})$: bipartite graph.
2. Task: Compute maximum size matching in G.
3. $M_0 = \emptyset$ empty matching.
4. 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.
    3. $\forall lr \in M_{i-1}$ oriented from the right to left, as the new directed edge (r, l).
    4. **BFS**: compute *shortest path* π_i from a vertex of L_i to a vertex of R_i.
    5. If no such path $\implies$ no augmenting path $\implies$ stop.
    6. $M_i = M_{i-1} \oplus \pi_i$.

The algorithm

1. $\mathsf{G} = (L \cup R, \mathsf{E})$: bipartite graph.
2. Task: Compute maximum size matching in G.
3. $M_0 = \emptyset$ empty matching.
4. 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.
   3. $\forall lr \in M_{i-1}$ oriented from the right to left, as the new directed edge (r, l).
   4. **BFS**: compute *shortest path* π_i from a vertex of L_i to a vertex of R_i.
   5. If no such path $\implies$ no augmenting path $\implies$ stop.
   6. $M_i = M_{i-1} \oplus \pi_i$.

The algorithm

1. $\mathsf{G} = (L \cup R, \mathsf{E})$: bipartite graph.
2. Task: Compute maximum size matching in **G**.
3. $M_0 = \emptyset$ empty matching.
4. 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.
   3. $\forall lr \in M_{i-1}$ oriented from the right to left, as the new directed edge (r, l).
   4. **BFS**: compute *shortest path* π_i from a vertex of L_i to a vertex of R_i.
   5. If no such path $\implies$ no augmenting path $\implies$ stop.
   6. $M_i = M_{i-1} \oplus \pi_i$.

The algorithm

1. $\mathsf{G} = (L \cup R, \mathsf{E})$: bipartite graph.
2. Task: Compute maximum size matching in G.
3. $M_0 = \emptyset$ empty matching.
4. 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.
    3. $\forall lr \in M_{i-1}$ oriented from the right to left, as the new directed edge (r, l).
    4. **BFS**: compute *shortest path* π_i from a vertex of L_i to a vertex of R_i.
    5. If no such path $\implies$ no augmenting path $\implies$ stop.
    6. $M_i = M_{i-1} \oplus \pi_i$.

The algorithm

1. $\mathbf{G} = (L \cup R, \mathbf{E})$: bipartite graph.
2. Task: Compute maximum size matching in **G**.
3. $M_0 = \emptyset$ empty matching.
4. 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 $\mathbf{H}_i$: Orient all edges of $\mathbf{E} \setminus M_{i-1}$ 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).
    4. **BFS**: compute *shortest path* π_i from a vertex of L_i to a vertex of R_i.
    5. If no such path $\implies$ no augmenting path $\implies$ stop.
    6. $M_i = M_{i-1} \oplus \pi_i$.

The algorithm

1. $\mathsf{G} = (L \cup R, \mathsf{E})$: bipartite graph.
2. Task: Compute maximum size matching in **G**.
3. $M_0 = \emptyset$ empty matching.
4. 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.
    3. $\forall lr \in M_{i-1}$ oriented from the right to left, as the new directed edge (r, l).
    4. **BFS**: compute *shortest path* π_i from a vertex of L_i to a vertex of R_i.
    5. If no such path $\implies$ no augmenting path $\implies$ stop.
    6. $M_i = M_{i-1} \oplus \pi_i$.

The algorithm

1. $\mathsf{G} = (L \cup R, \mathsf{E})$: bipartite graph.
2. Task: Compute maximum size matching in G.
3. $M_0 = \emptyset$ empty matching.
4. 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.
    3. $\forall lr \in M_{i-1}$ oriented from the right to left, as the new directed edge (r, l).
    4. **BFS**: compute *shortest path* π_i from a vertex of L_i to a vertex of R_i.
    5. If no such path $\implies$ no augmenting path $\implies$ stop.
    6. $M_i = M_{i-1} \oplus \pi_i$.

Analysis.

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

Analysis.

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

Analysis.

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

Analysis.

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

Analysis.

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

Analysis.

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

Analysis.

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

Result

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

Lemma

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

Result

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

Lemma

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

Result

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

Lemma

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

Result

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

Lemma

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

16.3.2: The Hopcroft-Karp algorithm

16.3.2.1: Some more structural observations

Observations:

1. If we augmenting along a shortest path, then the next augmenting path must be longer (or at least not shorter).
2. If always augment along shortest paths, then the augmenting paths get longer as the algorithm progress.
3. All the augmenting paths of the same length used by the algorithm are vertex-disjoint (!).
4. 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.

Observations:

1. If we augmenting along a shortest path, then the next augmenting path must be longer (or at least not shorter).
2. If always augment along shortest paths, then the augmenting paths get longer as the algorithm progress.
3. All the augmenting paths of the same length used by the algorithm are vertex-disjoint (!).
4. 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.

Observations:

1. If we augmenting along a shortest path, then the next augmenting path must be longer (or at least not shorter).
2. If always augment along shortest paths, then the augmenting paths get longer as the algorithm progress.
3. All the augmenting paths of the same length used by the algorithm are vertex-disjoint (!).
4. 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.

Observations:

1. If we augmenting along a shortest path, then the next augmenting path must be longer (or at least not shorter).
2. If always augment along shortest paths, then the augmenting paths get longer as the algorithm progress.
3. All the augmenting paths of the same length used by the algorithm are vertex-disjoint (!).
4. 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* $|\pi'| \geq |\pi| + 2\,|\pi \cap \pi'|$*.*

Proof

1. Consider the matching $N = M \oplus \pi \oplus \pi'$.
2. $|N| = |M| + 2$.
3. $M \oplus N$ contains two augmenting paths, say σ_1 and σ_2 (relative to M).
4. $M \oplus N = \pi \oplus \pi'$, and $|\pi \oplus \pi'| = |M \oplus N| \geq |\sigma_1| + |\sigma_2|$.
5. π: shortest augmenting path $(M) \implies |\sigma_1| \geq |\pi|$ and $|\sigma_2| \geq |\pi|$.
6. $\implies |\pi \oplus \pi'| \geq |\sigma_1| + |\sigma_2| \geq |\pi| + |\pi| = 2|\pi|$.
7. By definition: $|\pi \oplus \pi'| = |\pi| + |\pi'| - 2|\pi \cap \pi'|$.
8. Combining with the above, we have
$$|\pi| + |\pi'| - 2|\pi \cap \pi'| \geq 2|\pi| \implies |\pi'| \geq |\pi| + 2|\pi \cap \pi'|.$$

■

Proof

1. Consider the matching $N = M \oplus \pi \oplus \pi'$.
2. $|N| = |M| + 2$.
3. $M \oplus N$ contains two augmenting paths, say σ_1 and σ_2 (relative to M).
4. $M \oplus N = \pi \oplus \pi'$, and $|\pi \oplus \pi'| = |M \oplus N| \geq |\sigma_1| + |\sigma_2|$.
5. π: shortest augmenting path (M) $\implies$ $|\sigma_1| \geq |\pi|$ and $|\sigma_2| \geq |\pi|$.
6. $\implies |\pi \oplus \pi'| \geq |\sigma_1| + |\sigma_2| \geq |\pi| + |\pi| = 2|\pi|$.
7. By definition: $|\pi \oplus \pi'| = |\pi| + |\pi'| - 2|\pi \cap \pi'|$.
8. Combining with the above, we have
$$|\pi| + |\pi'| - 2|\pi \cap \pi'| \geq 2|\pi| \implies |\pi'| \geq |\pi| + 2|\pi \cap \pi'|.$$

■

Proof

1. Consider the matching $N = M \oplus \pi \oplus \pi'$.
2. $|N| = |M| + 2$.
3. $M \oplus N$ contains two augmenting paths, say σ_1 and σ_2 (relative to M).
4. $M \oplus N = \pi \oplus \pi'$, and $|\pi \oplus \pi'| = |M \oplus N| \geq |\sigma_1| + |\sigma_2|$.
5. π: shortest augmenting path $(M) \implies |\sigma_1| \geq |\pi|$ and $|\sigma_2| \geq |\pi|$.
6. $\implies |\pi \oplus \pi'| \geq |\sigma_1| + |\sigma_2| \geq |\pi| + |\pi| = 2|\pi|$.
7. By definition: $|\pi \oplus \pi'| = |\pi| + |\pi'| - 2|\pi \cap \pi'|$.
8. Combining with the above, we have
$$|\pi| + |\pi'| - 2|\pi \cap \pi'| \geq 2|\pi| \implies |\pi'| \geq |\pi| + 2|\pi \cap \pi'|.$$

■

Proof

1. Consider the matching $N = M \oplus \pi \oplus \pi'$.
2. $|N| = |M| + 2$.
3. $M \oplus N$ contains two augmenting paths, say σ_1 and σ_2 (relative to M).
4. $M \oplus N = \pi \oplus \pi'$, and $|\pi \oplus \pi'| = |M \oplus N| \geq |\sigma_1| + |\sigma_2|$.
5. π: shortest augmenting path (M) $\implies$ $|\sigma_1| \geq |\pi|$ and $|\sigma_2| \geq |\pi|$.
6. $\implies |\pi \oplus \pi'| \geq |\sigma_1| + |\sigma_2| \geq |\pi| + |\pi| = 2|\pi|$.
7. By definition: $|\pi \oplus \pi'| = |\pi| + |\pi'| - 2|\pi \cap \pi'|$.
8. Combining with the above, we have
$$|\pi| + |\pi'| - 2|\pi \cap \pi'| \geq 2|\pi| \implies |\pi'| \geq |\pi| + 2|\pi \cap \pi'|.$$

■

Proof

1. Consider the matching $N = M \oplus \pi \oplus \pi'$.
2. $|N| = |M| + 2$.
3. $M \oplus N$ contains two augmenting paths, say σ_1 and σ_2 (relative to M).
4. $M \oplus N = \pi \oplus \pi'$, and $|\pi \oplus \pi'| = |M \oplus N| \geq |\sigma_1| + |\sigma_2|$.
5. π: shortest augmenting path (M) $\implies$ $|\sigma_1| \geq |\pi|$ and $|\sigma_2| \geq |\pi|$.
6. $\implies$ $|\pi \oplus \pi'| \geq |\sigma_1| + |\sigma_2| \geq |\pi| + |\pi| = 2|\pi|$.
7. By definition: $|\pi \oplus \pi'| = |\pi| + |\pi'| - 2|\pi \cap \pi'|$.
8. Combining with the above, we have $|\pi| + |\pi'| - 2|\pi \cap \pi'| \geq 2|\pi|$ $\implies$ $|\pi'| \geq |\pi| + 2|\pi \cap \pi'|$.

■

Proof

1. Consider the matching $N = M \oplus \pi \oplus \pi'$.
2. $|N| = |M| + 2$.
3. $M \oplus N$ contains two augmenting paths, say σ_1 and σ_2 (relative to M).
4. $M \oplus N = \pi \oplus \pi'$, and $|\pi \oplus \pi'| = |M \oplus N| \geq |\sigma_1| + |\sigma_2|$.
5. π: shortest augmenting path $(M) \implies |\sigma_1| \geq |\pi|$ and $|\sigma_2| \geq |\pi|$.
6. $\implies |\pi \oplus \pi'| \geq |\sigma_1| + |\sigma_2| \geq |\pi| + |\pi| = 2|\pi|$.
7. By definition: $|\pi \oplus \pi'| = |\pi| + |\pi'| - 2|\pi \cap \pi'|$.
8. Combining with the above, we have
$$|\pi| + |\pi'| - 2|\pi \cap \pi'| \geq 2|\pi|$$
$$\implies \quad |\pi'| \geq |\pi| + 2|\pi \cap \pi'|.$$

Proof

1. Consider the matching $N = M \oplus \pi \oplus \pi'$.
2. $|N| = |M| + 2$.
3. $M \oplus N$ contains two augmenting paths, say σ_1 and σ_2 (relative to M).
4. $M \oplus N = \pi \oplus \pi'$, and $|\pi \oplus \pi'| = |M \oplus N| \geq |\sigma_1| + |\sigma_2|$.
5. π: shortest augmenting path (M) $\implies$ $|\sigma_1| \geq |\pi|$ and $|\sigma_2| \geq |\pi|$.
6. $\implies$ $|\pi \oplus \pi'| \geq |\sigma_1| + |\sigma_2| \geq |\pi| + |\pi| = 2|\pi|$.
7. By definition: $|\pi \oplus \pi'| = |\pi| + |\pi'| - 2|\pi \cap \pi'|$.
8. Combining with the above, we have

$$|\pi| + |\pi'| - 2|\pi \cap \pi'| \geq 2|\pi|$$
$$\implies \quad |\pi'| \geq |\pi| + 2|\pi \cap \pi'|.$$

Proof

1. Consider the matching $N = M \oplus \pi \oplus \pi'$.
2. $|N| = |M| + 2$.
3. $M \oplus N$ contains two augmenting paths, say σ_1 and σ_2 (relative to M).
4. $M \oplus N = \pi \oplus \pi'$, and $|\pi \oplus \pi'| = |M \oplus N| \geq |\sigma_1| + |\sigma_2|$.
5. π: shortest augmenting path $(M) \implies |\sigma_1| \geq |\pi|$ and $|\sigma_2| \geq |\pi|$.
6. $\implies |\pi \oplus \pi'| \geq |\sigma_1| + |\sigma_2| \geq |\pi| + |\pi| = 2|\pi|$.
7. By definition: $|\pi \oplus \pi'| = |\pi| + |\pi'| - 2|\pi \cap \pi'|$.
8. Combining with the above, we have $|\pi| + |\pi'| - 2|\pi \cap \pi'| \geq 2|\pi|$ $\implies |\pi'| \geq |\pi| + 2|\pi \cap \pi'|$.

Proof

1. Consider the matching $N = M \oplus \pi \oplus \pi'$.
2. $|N| = |M| + 2$.
3. $M \oplus N$ contains two augmenting paths, say σ_1 and σ_2 (relative to M).
4. $M \oplus N = \pi \oplus \pi'$, and $|\pi \oplus \pi'| = |M \oplus N| \geq |\sigma_1| + |\sigma_2|$.
5. π: shortest augmenting path (M) $\implies$ $|\sigma_1| \geq |\pi|$ and $|\sigma_2| \geq |\pi|$.
6. $\implies |\pi \oplus \pi'| \geq |\sigma_1| + |\sigma_2| \geq |\pi| + |\pi| = 2|\pi|$.
7. By definition: $|\pi \oplus \pi'| = |\pi| + |\pi'| - 2|\pi \cap \pi'|$.
8. Combining with the above, we have
$$|\pi| + |\pi'| - 2|\pi \cap \pi'| \geq 2|\pi| \implies |\pi'| \geq |\pi| + 2|\pi \cap \pi'|.$$

■

Corollary

Corollary

.For sequence of augmenting paths used algorithm (always augment the matching along the shortest augmenting path). We have:
$|\pi_1| \leq |\pi_2| \leq \ldots \leq |\pi_t|$.
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.*

Proof

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.
3. M_i: matching after π_i was applied.
4. π_j not using any of the edges of $\pi_{i+1}, \ldots, \pi_{j-1}$.
5. π_j is an augmenting path for M_i.
6. π_j and π_i share vertices.
   1. can not be the two endpoints of π_j (since they are free)
   2. must be some interval vertex of π_j.
   3. $\implies$ π_i and π_j must share an edge.
7. $|\pi_i \cap \pi_j| \geq 1$.
8. By lemma: $|\pi_j| \geq |\pi_i| + 2|\pi_i \cap \pi_j| > |\pi_i|$.
9. A contradiction.

Proof

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.
3. M_i: matching after π_i was applied.
4. π_j not using any of the edges of $\pi_{i+1}, \ldots, \pi_{j-1}$.
5. π_j is an augmenting path for M_i.
6. π_j and π_i share vertices.
    1. can not be the two endpoints of π_j (since they are free)
    2. must be some interval vertex of π_j.
    3. $\implies$ π_i and π_j must share an edge.
7. $|\pi_i \cap \pi_j| \geq 1$.
8. By lemma: $|\pi_j| \geq |\pi_i| + 2|\pi_i \cap \pi_j| > |\pi_i|$.
9. A contradiction.

Proof

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.
3. M_i: matching after π_i was applied.
4. π_j not using any of the edges of $\pi_{i+1}, \ldots, \pi_{j-1}$.
5. π_j is an augmenting path for M_i.
6. π_j and π_i share vertices.
   1. can not be the two endpoints of π_j (since they are free)
   2. must be some interval vertex of π_j.
   3. $\implies$ π_i and π_j must share an edge.
7. $|\pi_i \cap \pi_j| \geq 1$.
8. By lemma: $|\pi_j| \geq |\pi_i| + 2|\pi_i \cap \pi_j| > |\pi_i|$.
9. A contradiction.

Proof

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.
3. M_i: matching after π_i was applied.
4. π_j not using any of the edges of $\pi_{i+1}, \ldots, \pi_{j-1}$.
5. π_j is an augmenting path for M_i.
6. π_j and π_i share vertices.
   1. can not be the two endpoints of π_j (since they are free)
   2. must be some interval vertex of π_j.
   3. $\implies$ π_i and π_j must share an edge.
7. $|\pi_i \cap \pi_j| \geq 1$.
8. By lemma: $|\pi_j| \geq |\pi_i| + 2|\pi_i \cap \pi_j| > |\pi_i|$.
9. A contradiction.

Proof

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.
3. M_i: matching after π_i was applied.
4. π_j not using any of the edges of $\pi_{i+1}, \ldots, \pi_{j-1}$.
5. π_j is an augmenting path for M_i.
6. π_j and π_i share vertices.
   1. can not be the two endpoints of π_j (since they are free)
   2. must be some interval vertex of π_j.
   3. $\implies$ π_i and π_j must share an edge.
7. $|\pi_i \cap \pi_j| \geq 1$.
8. By lemma: $|\pi_j| \geq |\pi_i| + 2|\pi_i \cap \pi_j| > |\pi_i|$.
9. A contradiction.

Proof

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.
3. M_i: matching after π_i was applied.
4. π_j not using any of the edges of $\pi_{i+1}, \ldots, \pi_{j-1}$.
5. π_j is an augmenting path for M_i.
6. π_j and π_i share vertices.
   1. can not be the two endpoints of π_j (since they are free)
   2. must be some interval vertex of π_j.
   3. $\implies$ π_i and π_j must share an edge.
7. $|\pi_i \cap \pi_j| \geq 1$.
8. By lemma: $|\pi_j| \geq |\pi_i| + 2|\pi_i \cap \pi_j| > |\pi_i|$.
9. A contradiction.

Proof

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.
3. M_i: matching after π_i was applied.
4. π_j not using any of the edges of $\pi_{i+1}, \ldots, \pi_{j-1}$.
5. π_j is an augmenting path for M_i.
6. π_j and π_i share vertices.
   1. can not be the two endpoints of π_j (since they are free)
   2. must be some interval vertex of π_j.
   3. $\implies$ π_i and π_j must share an edge.
7. $|\pi_i \cap \pi_j| \geq 1$.
8. By lemma: $|\pi_j| \geq |\pi_i| + 2|\pi_i \cap \pi_j| > |\pi_i|$.
9. A contradiction.

Proof

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.
3. M_i: matching after π_i was applied.
4. π_j not using any of the edges of $\pi_{i+1}, \ldots, \pi_{j-1}$.
5. π_j is an augmenting path for M_i.
6. π_j and π_i share vertices.
   1. can not be the two endpoints of π_j (since they are free)
   2. must be some interval vertex of π_j.
   3. $\implies$ π_i and π_j must share an edge.
7. $|\pi_i \cap \pi_j| \geq 1$.
8. By lemma: $|\pi_j| \geq |\pi_i| + 2|\pi_i \cap \pi_j| > |\pi_i|$.
9. A contradiction.

Proof

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.
3. M_i: matching after π_i was applied.
4. π_j not using any of the edges of $\pi_{i+1}, \ldots, \pi_{j-1}$.
5. π_j is an augmenting path for M_i.
6. π_j and π_i share vertices.
   1. can not be the two endpoints of π_j (since they are free)
   2. must be some interval vertex of π_j.
   3. $\implies$ π_i and π_j must share an edge.
7. $|\pi_i \cap \pi_j| \geq 1$.
8. By lemma: $|\pi_j| \geq |\pi_i| + 2|\pi_i \cap \pi_j| > |\pi_i|$.
9. A contradiction.

Proof

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.
3. M_i: matching after π_i was applied.
4. π_j not using any of the edges of $\pi_{i+1}, \ldots, \pi_{j-1}$.
5. π_j is an augmenting path for M_i.
6. π_j and π_i share vertices.
   1. can not be the two endpoints of π_j (since they are free)
   2. must be some interval vertex of π_j.
   3. $\implies$ π_i and π_j must share an edge.
7. $|\pi_i \cap \pi_j| \geq 1$.
8. By lemma: $|\pi_j| \geq |\pi_i| + 2|\pi_i \cap \pi_j| > |\pi_i|$.
9. A contradiction.

Proof

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.
3. M_i: matching after π_i was applied.
4. π_j not using any of the edges of $\pi_{i+1}, \ldots, \pi_{j-1}$.
5. π_j is an augmenting path for M_i.
6. π_j and π_i share vertices.
    1. can not be the two endpoints of π_j (since they are free)
    2. must be some interval vertex of π_j.
    3. $\implies$ π_i and π_j must share an edge.
7. $|\pi_i \cap \pi_j| \geq 1$.
8. By lemma: $|\pi_j| \geq |\pi_i| + 2|\pi_i \cap \pi_j| > |\pi_i|$.
9. A contradiction.

Proof

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.
3. M_i: matching after π_i was applied.
4. π_j not using any of the edges of $\pi_{i+1}, \ldots, \pi_{j-1}$.
5. π_j is an augmenting path for M_i.
6. π_j and π_i share vertices.
   1. can not be the two endpoints of π_j (since they are free)
   2. must be some interval vertex of π_j.
   3. $\implies$ π_i and π_j must share an edge.
7. $|\pi_i \cap \pi_j| \geq 1$.
8. By lemma: $|\pi_j| \geq |\pi_i| + 2|\pi_i \cap \pi_j| > |\pi_i|$.
9. A contradiction.

16.3.2.2: Improved algorithm for bipartite maximum size matching

Better algorithm

1. 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 $k = 1 + 2\left\lceil \sqrt{n} \right\rceil$.
4. Take $O(km) = O(\sqrt{n}m)$ time.
5. T: maximum matching.
6. By the end of this process, matching is of size $|T| - \Omega(\sqrt{n})$. (See below why.)
7. Resume regular algorithm that augments one augmenting path at a time.
8. After $O(\sqrt{n})$ regular iterations we would be done.

Better algorithm

1. 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 $k = 1 + 2\left\lceil \sqrt{n} \right\rceil$.
4. Take $O(km) = O(\sqrt{n}m)$ time.
5. T: maximum matching.
6. By the end of this process, matching is of size $|T| - \Omega(\sqrt{n})$. (See below why.)
7. Resume regular algorithm that augments one augmenting path at a time.
8. After $O(\sqrt{n})$ regular iterations we would be done.

Better algorithm

1. 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 $k = 1 + 2\left\lceil \sqrt{n} \right\rceil$.
4. Take $O(km) = O(\sqrt{n}m)$ time.
5. T: maximum matching.
6. By the end of this process, matching is of size $|T| - \Omega(\sqrt{n})$. (See below why.)
7. Resume regular algorithm that augments one augmenting path at a time.
8. After $O(\sqrt{n})$ regular iterations we would be done.

Better algorithm

1. 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 $k = 1 + 2\left\lceil \sqrt{n} \right\rceil$.
4. Take $O(km) = O(\sqrt{n}m)$ time.
5. T: maximum matching.
6. By the end of this process, matching is of size $|T| - \Omega(\sqrt{n})$. (See below why.)
7. Resume regular algorithm that augments one augmenting path at a time.
8. After $O(\sqrt{n})$ regular iterations we would be done.

Better algorithm

1. 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 $k = 1 + 2\left\lceil\sqrt{n}\right\rceil$.
4. Take $O(km) = O(\sqrt{n}m)$ time.
5. T: maximum matching.
6. By the end of this process, matching is of size $|T| - \Omega(\sqrt{n})$. (See below why.)
7. Resume regular algorithm that augments one augmenting path at a time.
8. After $O(\sqrt{n})$ regular iterations we would be done.

Better algorithm

1. 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 $k = 1 + 2\left\lceil\sqrt{n}\right\rceil$.
4. Take $O(km) = O(\sqrt{n}m)$ time.
5. T: maximum matching.
6. By the end of this process, matching is of size $|T| - \Omega(\sqrt{n})$. (See below why.)
7. Resume regular algorithm that augments one augmenting path at a time.
8. After $O(\sqrt{n})$ regular iterations we would be done.

Better algorithm

1. 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 $k = 1 + 2\left\lceil \sqrt{n} \right\rceil$.
4. Take $O(km) = O(\sqrt{n}m)$ time.
5. T: maximum matching.
6. By the end of this process, matching is of size $|T| - \Omega(\sqrt{n})$. (See below why.)
7. Resume regular algorithm that augments one augmenting path at a time.
8. After $O(\sqrt{n})$ regular iterations we would be done.

Better algorithm

1. 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 $k = 1 + 2\left\lceil \sqrt{n} \right\rceil$.
4. Take $O(km) = O(\sqrt{n}m)$ time.
5. T: maximum matching.
6. By the end of this process, matching is of size $|T| - \Omega(\sqrt{n})$. (See below why.)
7. Resume regular algorithm that augments one augmenting path at a time.
8. 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* **G**. *Let* T *be the maximum matching. Then* $|T| \leq |M| + O(\sqrt{n})$.

Proof...

Proof.

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

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

5. $\implies |T| - |M| \leq 3\sqrt{n}$, for $n \geq 4$.

Proof...

Proof.

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

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

5. $\implies |T| - |M| \leq 3\sqrt{n}$, for $n \geq 4$.

□

Proof...

Proof.

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

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

5. $\implies |T| - |M| \leq 3\sqrt{n}$, for $n \geq 4$.

Proof...

Proof.

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

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

5. $\implies |T| - |M| \leq 3\sqrt{n}$, for $n \geq 4$.

Proof...

Proof.

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

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

5. $\implies |T| - |M| \leq 3\sqrt{n}$, for $n \geq 4$.

Proof...

Proof.

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

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

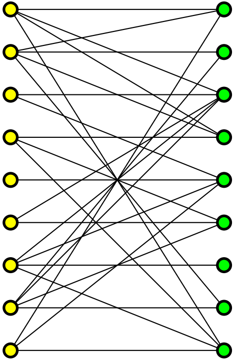
5. $\implies |T| - |M| \leq 3\sqrt{n}$, for $n \geq 4$.

□

16.3.2.3: Extracting many augmenting paths

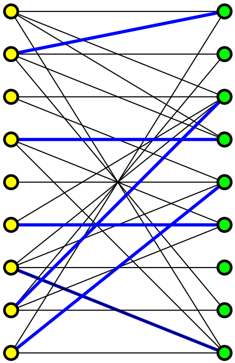
Algorithm via animation

Find many disjoint augmenting paths



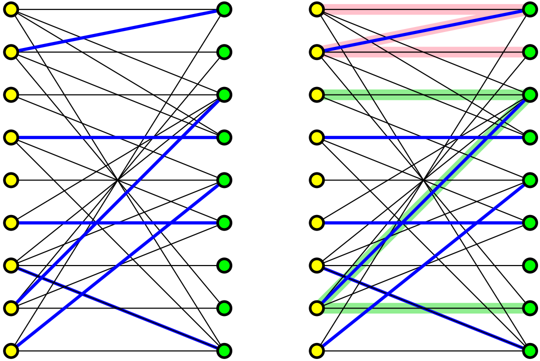
Algorithm via animation

Find many disjoint augmenting paths



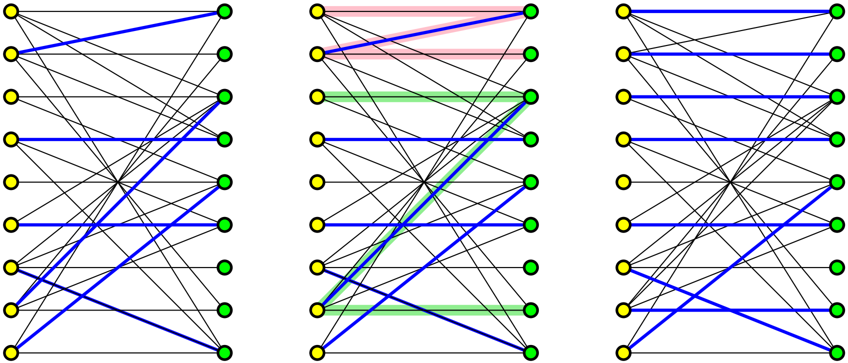
Algorithm via animation

Find many disjoint augmenting paths



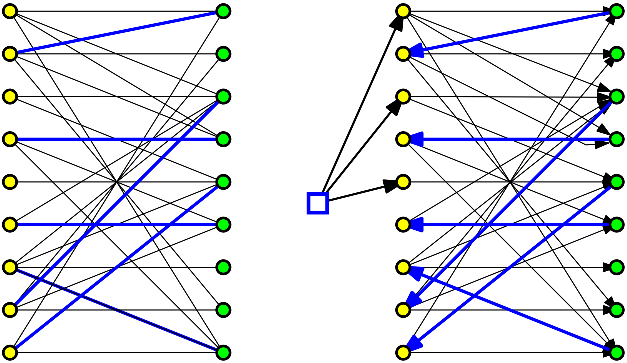
Algorithm via animation

Find many disjoint augmenting paths



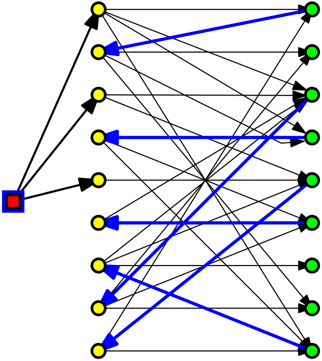
Algorithm via animation

Layering the graph - via BFS



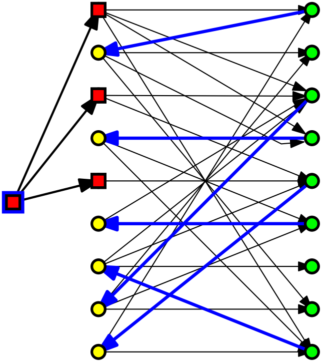
Algorithm via animation

Layering the graph - via BFS



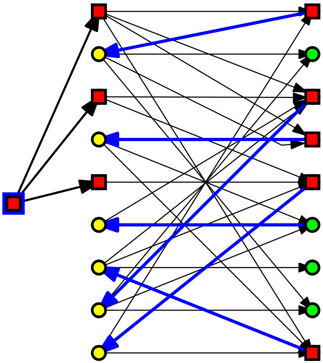
Algorithm via animation

Layering the graph - via BFS



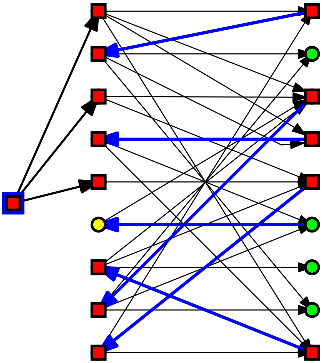
Algorithm via animation

Layering the graph - via BFS



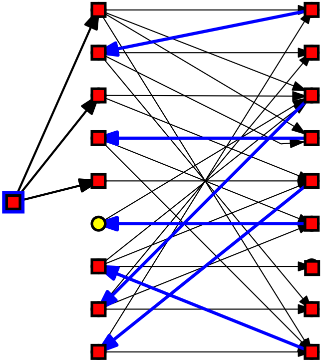
Algorithm via animation

Layering the graph - via BFS



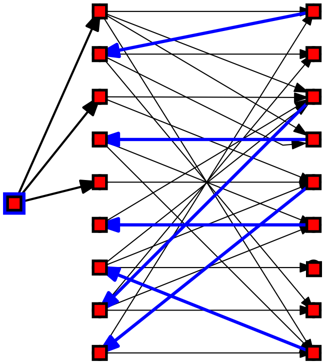
Algorithm via animation

Layering the graph - via BFS



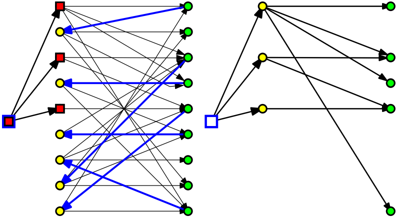
Algorithm via animation

Layering the graph - via BFS



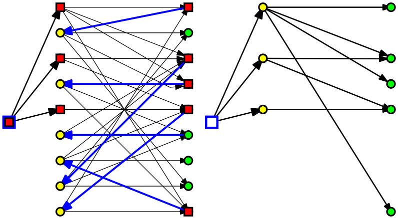
Algorithm via animation

The layered graph



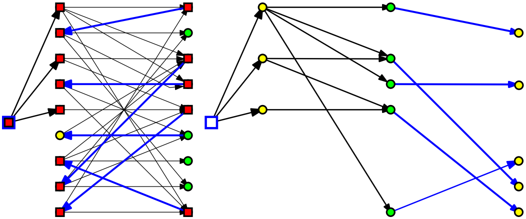
Algorithm via animation

The layered graph



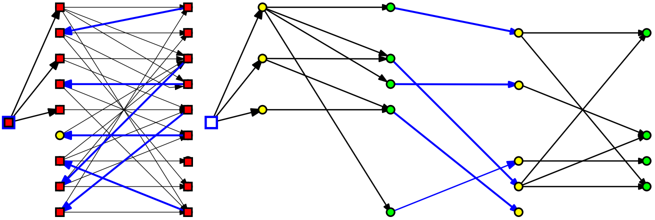
Algorithm via animation

The layered graph



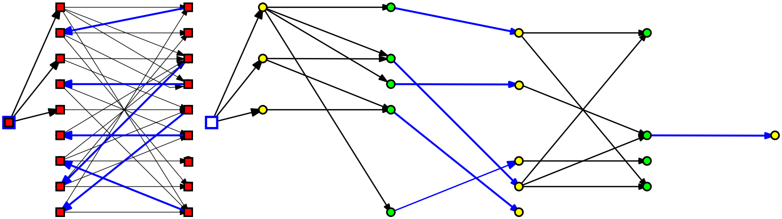
Algorithm via animation

The layered graph



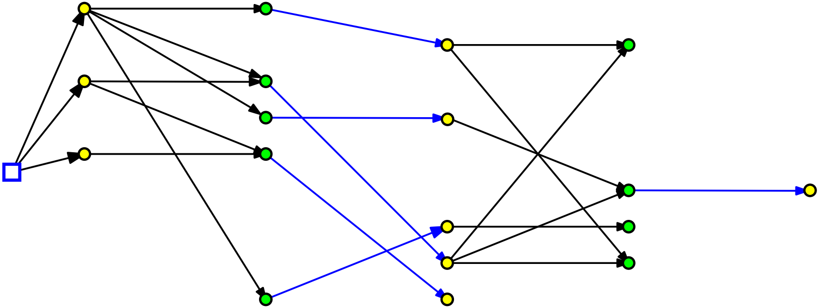
Algorithm via animation

The layered graph



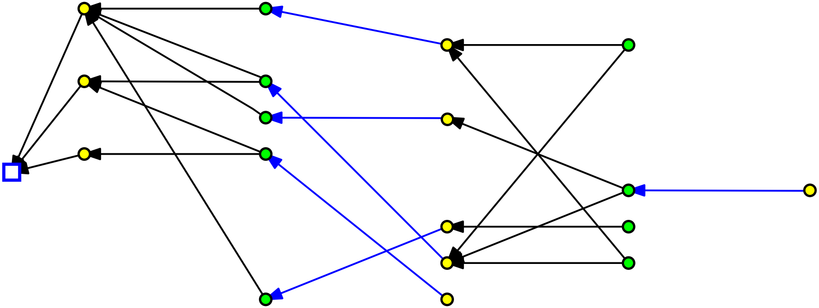
Algorithm via animation

The reverse layered graph and extracting paths



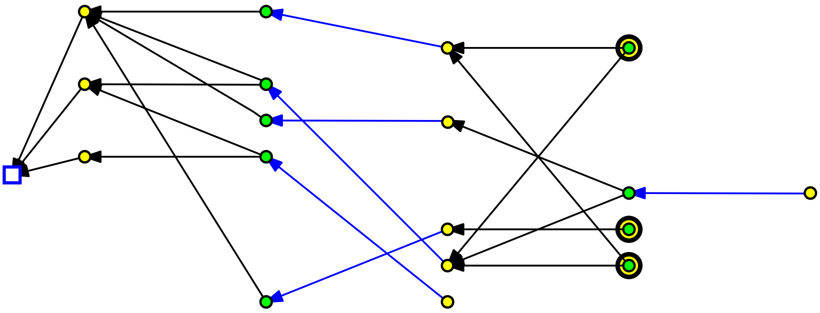
Algorithm via animation

The reverse layered graph and extracting paths



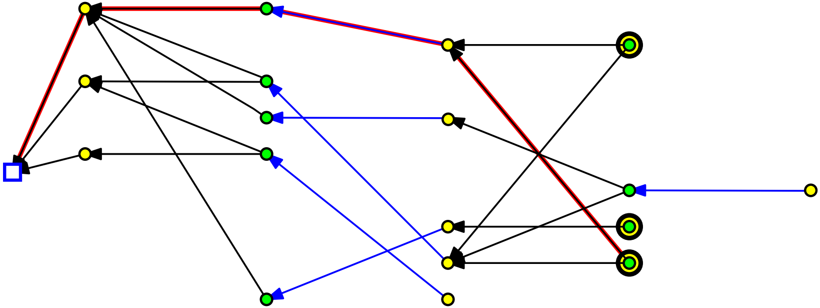
Algorithm via animation

The reverse layered graph and extracting paths



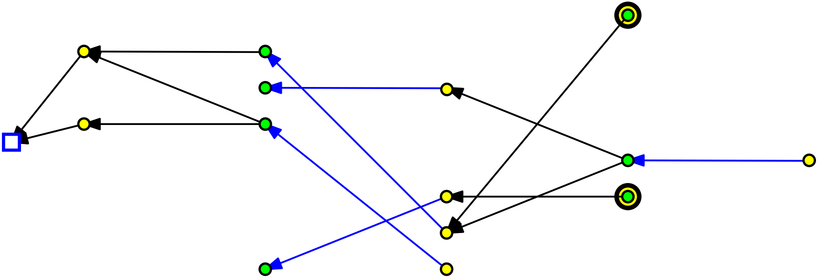
Algorithm via animation

The reverse layered graph and extracting paths



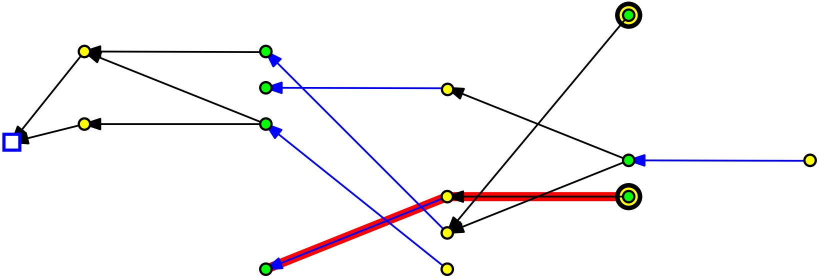
Algorithm via animation

The reverse layered graph and extracting paths



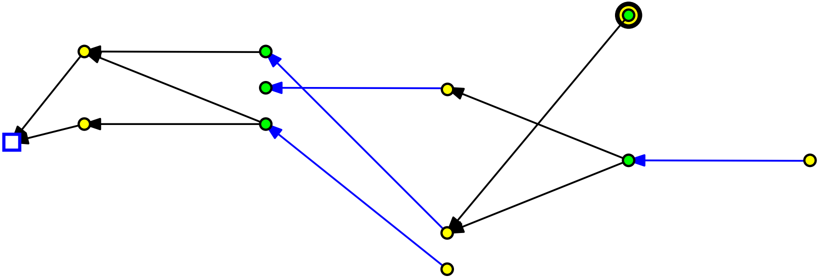
Algorithm via animation

The reverse layered graph and extracting paths



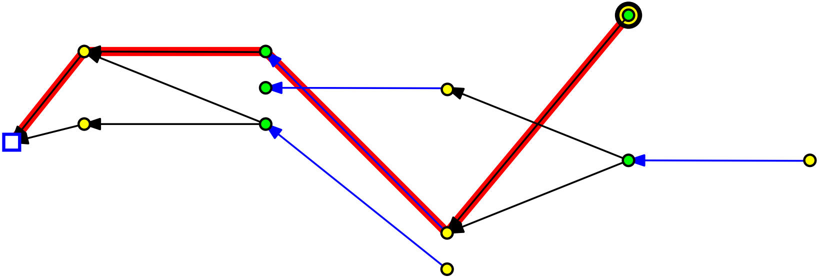
Algorithm via animation

The reverse layered graph and extracting paths



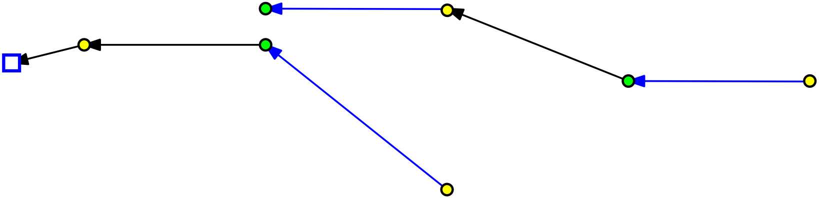
Algorithm via animation

The reverse layered graph and extracting paths



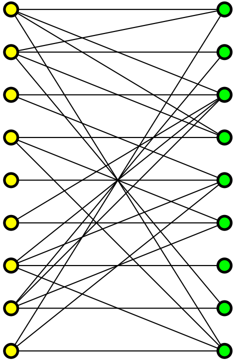
Algorithm via animation

The reverse layered graph and extracting paths



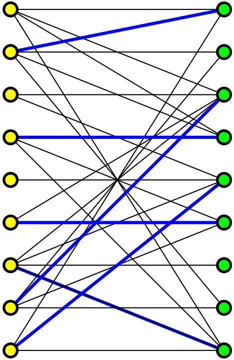
Algorithm via animation

Recall: Now we use these augmenting paths to improve the matching



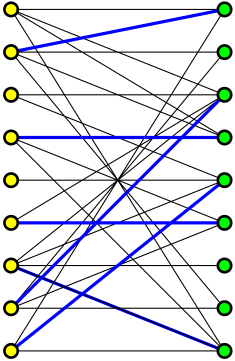
Algorithm via animation

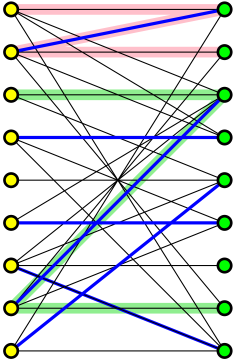
Recall: Now we use these augmenting paths to improve the matching



Algorithm via animation

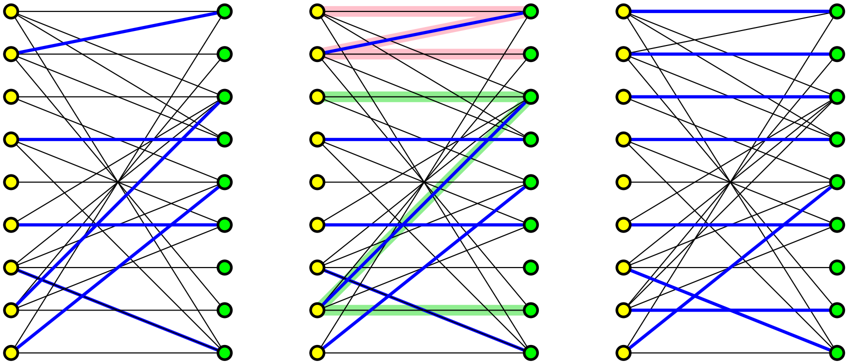
Recall: Now we use these augmenting paths to improve the matching





Algorithm via animation

Recall: Now we use these augmenting paths to improve the matching



Algorithm to extract many augmenting path

1. 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.
4. Task: Extract as many augmenting paths as possible. Vertex disjoint. Of length k
5. F: set of free vertices in **G**.
6. Build directed graph:
   1. s: source vertex connected to all vertices of $L_1 = L \cap F$.
   2. direct edges of **G** from left to right, and matching edges from right to left.
   3. **H**: resulting graph.
7. Compute **BFS** on the graph **H** starting at s, and let $\mathcal{T}$ be the resulting tree.
8. $L_1, R_1, L_2, R_2, L_3, \ldots$ be the layers of the **BFS**.

Algorithm to extract many augmenting path

1. 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.
4. Task: Extract as many augmenting paths as possible. Vertex disjoint. Of length k
5. F: set of free vertices in **G**.
6. Build directed graph:
    1. s: source vertex connected to all vertices of $L_1 = L \cap F$.
    2. direct edges of **G** from left to right, and matching edges from right to left.
    3. **H**: resulting graph.
7. Compute **BFS** on the graph **H** starting at s, and let $\mathcal{T}$ be the resulting tree.
8. $L_1, R_1, L_2, R_2, L_3, \ldots$ be the layers of the **BFS**.

Algorithm to extract many augmenting path

1. 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.
4. Task: Extract as many augmenting paths as possible. Vertex disjoint. Of length k
5. F: set of free vertices in **G**.
6. Build directed graph:
    1. s: source vertex connected to all vertices of $L_1 = L \cap F$.
    2. direct edges of **G** from left to right, and matching edges from right to left.
    3. **H**: resulting graph.
7. Compute **BFS** on the graph **H** starting at s, and let $\mathcal{T}$ be the resulting tree.
8. $L_1, R_1, L_2, R_2, L_3, \ldots$ be the layers of the **BFS**.

Algorithm to extract many augmenting path

1. 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.
4. Task: Extract as many augmenting paths as possible. Vertex disjoint. Of length k
5. F: set of free vertices in **G**.
6. Build directed graph:
    1. s: source vertex connected to all vertices of $L_1 = L \cap F$.
    2. direct edges of **G** from left to right, and matching edges from right to left.
    3. **H**: resulting graph.
7. Compute **BFS** on the graph **H** starting at s, and let $\mathcal{T}$ be the resulting tree.
8. $L_1, R_1, L_2, R_2, L_3, \ldots$ be the layers of the **BFS**.

Algorithm to extract many augmenting path

1. 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.
4. Task: Extract as many augmenting paths as possible. Vertex disjoint. Of length k
5. F: set of free vertices in **G**.
6. Build directed graph:
   1. s: source vertex connected to all vertices of $L_1 = L \cap F$.
   2. direct edges of **G** from left to right, and matching edges from right to left.
   3. **H**: resulting graph.
7. Compute **BFS** on the graph **H** starting at s, and let $\mathcal{T}$ be the resulting tree.
8. $L_1, R_1, L_2, R_2, L_3, \ldots$ be the layers of the **BFS**.

Algorithm to extract many augmenting path

1. 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.
4. Task: Extract as many augmenting paths as possible. Vertex disjoint. Of length k
5. F: set of free vertices in **G**.
6. Build directed graph:
   1. s: source vertex connected to all vertices of $L_1 = L \cap F$.
   2. direct edges of **G** from left to right, and matching edges from right to left.
   3. **H**: resulting graph.
7. Compute **BFS** on the graph **H** starting at s, and let $\mathcal{T}$ be the resulting tree.
8. $L_1, R_1, L_2, R_2, L_3, \ldots$ be the layers of the **BFS**.

Algorithm to extract many augmenting path

1. 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.
4. Task: Extract as many augmenting paths as possible. Vertex disjoint. Of length k
5. F: set of free vertices in **G**.
6. Build directed graph:
   1. s: source vertex connected to all vertices of $L_1 = L \cap F$.
   2. direct edges of **G** from left to right, and matching edges from right to left.
   3. **H**: resulting graph.
7. Compute **BFS** on the graph **H** starting at s, and let $\mathcal{T}$ be the resulting tree.
8. $L_1, R_1, L_2, R_2, L_3, \ldots$ be the layers of the **BFS**.

Algorithm to extract many augmenting path

1. By assumption: first free vertex below L_1 encountered is at level R_τ, where $\tau = \lceil k/2 \rceil$.
2. Scan edges of **H**.
3. Add forward edges to tree.
4. ... edge between two vertices that belong to two consecutive levels of the **BFS** tree $\mathcal{T}$.
5. **J** be the resulting graph.
6. **J** is a DAG (which is an enrichment of the original tree $\mathcal{T}$).
7. Compute also the reverse graph $\mathsf{J}^{\mathrm{rev}}$ (where, we just reverse the edges).

Algorithm to extract many augmenting path

1. By assumption: first free vertex below L_1 encountered is at level R_τ, where $\tau = \lceil k/2 \rceil$.
2. Scan edges of **H**.
3. Add forward edges to tree.
4. ... edge between two vertices that belong to two consecutive levels of the **BFS** tree $\mathcal{T}$.
5. **J** be the resulting graph.
6. **J** is a DAG (which is an enrichment of the original tree $\mathcal{T}$).
7. Compute also the reverse graph $\mathbf{J}^{\mathrm{rev}}$ (where, we just reverse the edges).

Algorithm to extract many augmenting path

1. By assumption: first free vertex below L_1 encountered is at level R_τ, where $\tau = \lceil k/2 \rceil$.
2. Scan edges of **H**.
3. Add forward edges to tree.
4. ... edge between two vertices that belong to two consecutive levels of the **BFS** tree $\mathcal{T}$.
5. **J** be the resulting graph.
6. **J** is a DAG (which is an enrichment of the original tree $\mathcal{T}$).
7. Compute also the reverse graph $\mathbf{J}^{\mathrm{rev}}$ (where, we just reverse the edges).

Algorithm to extract many augmenting path

1. By assumption: first free vertex below L_1 encountered is at level R_τ, where $\tau = \lceil k/2 \rceil$.
2. Scan edges of **H**.
3. Add forward edges to tree.
4. ... edge between two vertices that belong to two consecutive levels of the **BFS** tree $\mathcal{T}$.
5. **J** be the resulting graph.
6. **J** is a DAG (which is an enrichment of the original tree $\mathcal{T}$).
7. Compute also the reverse graph $\mathbf{J}^{\mathrm{rev}}$ (where, we just reverse the edges).

Algorithm to extract many augmenting path

1. By assumption: first free vertex below L_1 encountered is at level R_τ, where $\tau = \lceil k/2 \rceil$.
2. Scan edges of **H**.
3. Add forward edges to tree.
4. ... edge between two vertices that belong to two consecutive levels of the **BFS** tree $\mathcal{T}$.
5. **J** be the resulting graph.
6. **J** is a DAG (which is an enrichment of the original tree $\mathcal{T}$).
7. Compute also the reverse graph $\mathsf{J}^{\mathrm{rev}}$ (where, we just reverse the edges).

Algorithm to extract many augmenting path

1. By assumption: first free vertex below L_1 encountered is at level R_τ, where $\tau = \lceil k/2 \rceil$.
2. Scan edges of **H**.
3. Add forward edges to tree.
4. ... edge between two vertices that belong to two consecutive levels of the **BFS** tree $\mathcal{T}$.
5. **J** be the resulting graph.
6. **J** is a DAG (which is an enrichment of the original tree $\mathcal{T}$).
7. Compute also the reverse graph $\mathbf{J}^{\mathrm{rev}}$ (where, we just reverse the edges).

Algorithm to extract many augmenting path

1. By assumption: first free vertex below L_1 encountered is at level R_τ, where $\tau = \lceil k/2 \rceil$.
2. Scan edges of **H**.
3. Add forward edges to tree.
4. ... edge between two vertices that belong to two consecutive levels of the **BFS** tree $\mathcal{T}$.
5. **J** be the resulting graph.
6. **J** is a DAG (which is an enrichment of the original tree $\mathcal{T}$).
7. Compute also the reverse graph $\mathbf{J}^{\mathrm{rev}}$ (where, we just reverse the edges).

Algorithm to extract many augmenting path

1. By assumption: first free vertex below L_1 encountered is at level R_τ, where $\tau = \lceil k/2 \rceil$.
2. Scan edges of **H**.
3. Add forward edges to tree.
4. ... edge between two vertices that belong to two consecutive levels of the **BFS** tree $\mathcal{T}$.
5. **J** be the resulting graph.
6. **J** is a DAG (which is an enrichment of the original tree $\mathcal{T}$).
7. Compute also the reverse graph $\mathbf{J}^{\mathbf{rev}}$ (where, we just reverse the edges).

Back to extracting paths...

1. $F_\tau = R_\tau \cap F$: free vertices of distance k from free vertices of L_1.
2. $\forall v \in F_\tau$ do a **DFS** in J^{rev} till the **DFS** reaches a vertex of L_1.
3. 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).
4. If the **DFS** succeeds, extract shortest path found, and add it to the collection of augmenting paths.
5. Otherwise, move on to the next vertex in F_τ, till visit all such vertices.
6. Results: collection of augmenting paths P_τ,
    1. vertex disjoint.
    2. All of length k.

Back to extracting paths...

1. $F_\tau = R_\tau \cap F$: free vertices of distance k from free vertices of L_1.
2. $\forall v \in F_\tau$ do a **DFS** in $\mathsf{J}^{\mathbf{rev}}$ till the **DFS** reaches a vertex of L_1.
3. 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).
4. If the **DFS** succeeds, extract shortest path found, and add it to the collection of augmenting paths.
5. Otherwise, move on to the next vertex in F_τ, till visit all such vertices.
6. Results: collection of augmenting paths P_τ,
    1. vertex disjoint.
    2. All of length k.

Back to extracting paths...

1. $F_\tau = R_\tau \cap F$: free vertices of distance k from free vertices of L_1.
2. $\forall v \in F_\tau$ do a **DFS** in $\mathsf{J}^{\mathbf{rev}}$ till the **DFS** reaches a vertex of L_1.
3. 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).
4. If the **DFS** succeeds, extract shortest path found, and add it to the collection of augmenting paths.
5. Otherwise, move on to the next vertex in F_τ, till visit all such vertices.
6. Results: collection of augmenting paths P_τ,
    1. vertex disjoint.
    2. All of length k.

Back to extracting paths...

1. $F_\tau = R_\tau \cap F$: free vertices of distance k from free vertices of L_1.
2. $\forall v \in F_\tau$ do a **DFS** in $\mathsf{J}^{\mathrm{rev}}$ till the **DFS** reaches a vertex of L_1.
3. 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).
4. If the **DFS** succeeds, extract shortest path found, and add it to the collection of augmenting paths.
5. Otherwise, move on to the next vertex in F_τ, till visit all such vertices.
6. Results: collection of augmenting paths P_τ,
    1. vertex disjoint.
    2. All of length k.

Back to extracting paths...

1. $F_\tau = R_\tau \cap F$: free vertices of distance k from free vertices of L_1.
2. $\forall v \in F_\tau$ do a **DFS** in $\mathsf{J}^{\mathrm{rev}}$ till the **DFS** reaches a vertex of L_1.
3. 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).
4. If the **DFS** succeeds, extract shortest path found, and add it to the collection of augmenting paths.
5. Otherwise, move on to the next vertex in F_τ, till visit all such vertices.
6. Results: collection of augmenting paths P_τ,
    1. vertex disjoint.
    2. All of length k.

Back to extracting paths...

1. $F_\tau = R_\tau \cap F$: free vertices of distance k from free vertices of L_1.
2. $\forall v \in F_\tau$ do a **DFS** in J^{rev} till the **DFS** reaches a vertex of L_1.
3. 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).
4. If the **DFS** succeeds, extract shortest path found, and add it to the collection of augmenting paths.
5. Otherwise, move on to the next vertex in F_τ, till visit all such vertices.
6. Results: collection of augmenting paths P_τ,
    1. vertex disjoint.
    2. All of length k.

Back to extracting paths...

1. $F_\tau = R_\tau \cap F$: free vertices of distance k from free vertices of L_1.
2. $\forall v \in F_\tau$ do a **DFS** in $\mathsf{J}^{\mathrm{rev}}$ till the **DFS** reaches a vertex of L_1.
3. 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).
4. If the **DFS** succeeds, extract shortest path found, and add it to the collection of augmenting paths.
5. Otherwise, move on to the next vertex in F_τ, till visit all such vertices.
6. Results: collection of augmenting paths P_τ,
    1. vertex disjoint.
    2. All of length k.

Back to extracting paths...

1. $F_\tau = R_\tau \cap F$: free vertices of distance k from free vertices of L_1.
2. $\forall v \in F_\tau$ do a **DFS** in J^{rev} till the **DFS** reaches a vertex of L_1.
3. 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).
4. If the **DFS** succeeds, extract shortest path found, and add it to the collection of augmenting paths.
5. Otherwise, move on to the next vertex in F_τ, till visit all such vertices.
6. Results: collection of augmenting paths P_τ,
    1. vertex disjoint.
    2. All of length k.

Analysis...

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

Analysis...

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

Analysis...

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

Analysis...

1. Building initial graphs **J** and $\mathbf{J}^{\mathrm{rev}}$ takes $O(m)$ time.
2. Charge running time of the second stage to the edges and vertices visited.
3. Any vertex visited by any **DFS** is never going to be visited again...
4. $\implies$ edge of $\mathbf{J}^{\mathrm{rev}}$ is going to be considered only once by algorithm.
5. $\implies$ 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...

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**,
5. ... would go through unused vertices.
6. Indeed, if any vertices of σ were used by any of the back **DFS**,
7. $\implies$ resulted in a path that goes to a free vertex in L_1.
8. $\implies$ a contradiction: σ is supposedly disjoint from the paths of P_k.

□

Proof...

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**,
5. ... would go through unused vertices.
6. Indeed, if any vertices of σ were used by any of the back **DFS**,
7. $\implies$ resulted in a path that goes to a free vertex in L_1.
8. $\implies$ a contradiction: σ is supposedly disjoint from the paths of P_k.

Proof...

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**,
5. ... would go through unused vertices.
6. Indeed, if any vertices of σ were used by any of the back **DFS**,
7. $\implies$ resulted in a path that goes to a free vertex in L_1.
8. $\implies$ a contradiction: σ is supposedly disjoint from the paths of P_k.

Proof...

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**,
5. ... would go through unused vertices.
6. Indeed, if any vertices of σ were used by any of the back **DFS**,
7. $\implies$ resulted in a path that goes to a free vertex in L_1.
8. $\implies$ a contradiction: σ is supposedly disjoint from the paths of P_k.

Proof...

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**,
5. ... would go through unused vertices.
6. Indeed, if any vertices of σ were used by any of the back **DFS**,
7. $\implies$ resulted in a path that goes to a free vertex in L_1.
8. $\implies$ a contradiction: σ is supposedly disjoint from the paths of P_k.

Proof...

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**,
5. ... would go through unused vertices.
6. Indeed, if any vertices of σ were used by any of the back **DFS**,
7. $\implies$ resulted in a path that goes to a free vertex in L_1.
8. $\implies$ a contradiction: σ is supposedly disjoint from the paths of P_k.

Proof...

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**,
5. ... would go through unused vertices.
6. Indeed, if any vertices of σ were used by any of the back **DFS**,
7. $\implies$ resulted in a path that goes to a free vertex in L_1.
8. $\implies$ a contradiction: σ is supposedly disjoint from the paths of P_k.

Proof...

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**,
5. ... would go through unused vertices.
6. Indeed, if any vertices of σ were used by any of the back **DFS**,
7. $\implies$ resulted in a path that goes to a free vertex in L_1.
8. $\implies$ a contradiction: σ is supposedly disjoint from the paths of P_k.

□

16.3.2.4: The result

The result

Theorem

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

The proof...

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

The main challenge is the correctness.

The proof...

The **algMatching**$_{HK}$ algorithm was described, and the running time analysis was also done.
The main challenge is the correctness.

Proof of correctness...

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

Proof of correctness...

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

Proof of correctness...

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

Proof of correctness...

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

Proof of correctness...

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

Proof of correctness...

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

Proof of correctness...

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

Notes

Notes

Notes

Notes

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.