Combinatorial matching algorithms

Peter Hajnal

Bolyai Institute,University of Szeged, Hungary

2023 fall

Introduction

Introduction

Reminder

Let G be a graphs $F \subseteq E(G)$ is an edge set.

$$V(F) := \{x \in V : x \text{ incident to at least one edge from } F\}.$$

Introduction

Reminder

Let G be a graphs $F \subseteq E(G)$ is an edge set.

$$V(F) := \{x \in V : x \text{ incident to at least one edge from } F\}.$$

- In the case of $v \in V(F)$ one says F covers the vertex v.

Introduction

Reminder

Let G be a graphs $F \subseteq E(G)$ is an edge set.

$$V(F) := \{x \in V : x \text{ incident to at least one edge from } F\}.$$

- In the case of $v \in V(F)$ one says F covers the vertex v.

Reminder: Matching

The edge set M is a matching, iff $|V(M)| = 2|M|$.

Introduction

Reminder

Let G be a graphs $F \subseteq E(G)$ is an edge set.

$$V(F) := \{x \in V : x \text{ incident to at least one edge from } F\}.$$

- In the case of $v \in V(F)$ one says F covers the vertex v.

Reminder: Matching

The edge set M is a matching, iff $|V(M)| = 2|M|$.

I.e. An edge set is a matching iff it doesn't contain loop, and adjacent edges.

Introduction

Reminder

Let G be a graphs $F \subseteq E(G)$ is an edge set.

$$V(F) := \{x \in V : x \text{ incident to at least one edge from } F\}.$$

- In the case of $v \in V(F)$ one says F covers the vertex v.

Reminder: Matching

The edge set M is a matching, iff $|V(M)| = 2|M|$.

I.e. An edge set is a matching iff it doesn't contain loop, and adjacent edges.

- If M is a matching, and it covers vertex v, then we say that v is matched.

Introduction

Reminder

Let G be a graphs $F \subseteq E(G)$ is an edge set.

$$V(F) := \{x \in V : x \text{ incident to at least one edge from } F\}.$$

- In the case of $v \in V(F)$ one says F covers the vertex v.

Reminder: Matching

The edge set M is a matching, iff $|V(M)| = 2|M|$.

I.e. An edge set is a matching iff it doesn't contain loop, and adjacent edges.

- If M is a matching, and it covers vertex v, then we say that v is matched.
- Let $\overline{V}(M) = V(G) - V(M)$ be the set of vertices that are not matched by M.

Introduction

Reminder

Let G be a graphs $F \subseteq E(G)$ is an edge set.

$$V(F) := \{x \in V : x \text{ incident to at least one edge from } F\}.$$

- In the case of $v \in V(F)$ one says F covers the vertex v.

Reminder: Matching

The edge set M is a matching, iff $|V(M)| = 2|M|$.

I.e. An edge set is a matching iff it doesn't contain loop, and adjacent edges.

- If M is a matching, and it covers vertex v, then we say that v is matched.
- Let $\overline{V}(M) = V(G) - V(M)$ be the set of vertices that are not matched by M. $|\overline{V}(M)| = |V(G)| - |V(M)| = |V(G)| - 2|M|$

Introduction

Introduction

Reminder: Perfect matching

If M is a matching, and $V(M) = V(G)$, then one says that M is a perfect matching.

Introduction

Reminder: Perfect matching

If M is a matching, and $V(M) = V(G)$, then one says that M is a perfect matching.

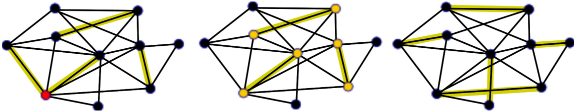
- We can have a perfect matching in G if $|V(G)|$ is even.

Introduction

Reminder: Perfect matching

If M is a matching, and $V(M) = V(G)$, then one says that M is a perfect matching.

- We can have a perfect matching in G if $|V(G)|$ is even.



A graph with an edges set (yellow edges), which is not a matching (the red vertex is covered by two edges). In the middle one can see a matching, which is not a perfect matching (the set of yellow vertices in the set of matched vertices). On the left there is perfect matching.

Optimization, algorithms

Optimization, algorithms

Definition

$\nu(G)$ is the maximum size among the matchings of G.

Optimization, algorithms

Definition

$\nu(G)$ is the maximum size among the matchings of G.

- $2\nu(G)$ is the maximum size of a vertex set, that can be matched.

$|V(G)| - 2\nu(G)$ is the minimum number of unmatched vertices.

Optimization, algorithms

Definition

$\nu(G)$ is the maximum size among the matchings of G.

• $2\nu(G)$ is the maximum size of a vertex set, that can be matched. $|V(G)| - 2\nu(G)$ is the minimum number of unmatched vertices.

• There are several natural algorithmic questions about matchings in graphs:

Algorithmic matching problems: Given a graph G.

(1) Find an optimal matching M.

(2) Determine the value $\nu(G)$.

(3) Decide whether G has a perfect matching or not.

(4) Find a "large" matching M.

The greedy algorithm, introduction

Greedy algorithm for finding large matchings

The greedy algorithm, introduction

Greedy algorithm for finding large matchings

(Initialization) Start with a matching M.

The greedy algorithm, introduction

Greedy algorithm for finding large matchings

(Initialization) Start with a matching M.
WHILE there is an edge $e \in E(G) - M$ such that $M \cup \{e\}$ is a matching too, do
(Greedy extension step) $M \leftarrow M \cup \{e\}$.

The greedy algorithm, introduction

Greedy algorithm for finding large matchings

(Initialization) Start with a matching M.
WHILE there is an edge $e \in E(G) - M$ such that $M \cup \{e\}$ is a matching too, do
(Greedy extension step) $M \leftarrow M \cup \{e\}$.
(Halting) The actual matching is the output.

The greedy algorithm, introduction

Greedy algorithm for finding large matchings

(Initialization) Start with a matching M.
WHILE there is an edge $e \in E(G) - M$ such that $M \cup \{e\}$ is a matching too, do
(Greedy extension step) $M \leftarrow M \cup \{e\}$.
(Halting) The actual matching is the output.
// At halting we have that each edge of $E(G) \setminus M$ is neighbors of an edge from M.

The greedy algorithm, introduction

Greedy algorithm for finding large matchings

(Initialization) Start with a matching M.
WHILE there is an edge $e \in E(G) - M$ such that $M \cup \{e\}$ is a matching too, do
(Greedy extension step) $M \leftarrow M \cup \{e\}$.
(Halting) The actual matching is the output.
// At halting we have that each edge of $E(G) \setminus M$ is neighbors of an edge from M.

- There is no fear if infinite cycle (cycling).

The greedy algorithm, introduction

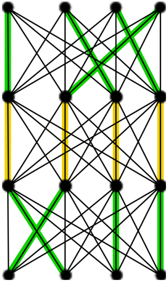
Greedy algorithm for finding large matchings

(Initialization) Start with a matching M.
WHILE there is an edge $e \in E(G) - M$ such that $M \cup \{e\}$ is a matching too, do
(Greedy extension step) $M \leftarrow M \cup \{e\}$.
(Halting) The actual matching is the output.
// At halting we have that each edge of $E(G) \setminus M$ is neighbors of an edge from M.

- There is no fear if infinite cycle (cycling).

- We know that int the case of halting the output can't be augmented by extensions.

Example



Our graph has four levels of common size (let n be the size of the levels, in our example $n = 4$). Between two adjacent levels all possible edges are resent and there are no further edges. It is possible that the greedy algorithm first chooses the yellow edges, matching the two middle levels. Then it halts. The green edges form a perfect matching.

Analysis

Analysis

Theorem

Let $\nu_{\text{greedy}}(G)$ denote the size of the output of the greedy algorithm. Then

$$\frac{\nu(G)}{2} \leq \nu_{\text{greedy}}(G) \leq \nu(G).$$

Analysis

Theorem

Let $\nu_{\text{greedy}}(G)$ denote the size of the output of the greedy algorithm. Then

$$\frac{\nu(G)}{2} \leq \nu_{\text{greedy}}(G) \leq \nu(G).$$

The second inequality is obvious since our algorithm computes a matching.

Analysis: the proof

Analysis: the proof

- Let M_{greedy} denote the output matching of the greedy algorithm.

Analysis: the proof

- Let M_{greedy} denote the output matching of the greedy algorithm.
- $L = V(M_{\text{greedy}})$ is the set of matched vertices.

Analysis: the proof

- Let M_{greedy} denote the output matching of the greedy algorithm.
- $L = V(M_{\text{greedy}})$ is the set of matched vertices.
- It is obvious that L is a covering vertex set, and $|L| = 2\nu_{\text{greedy}}(G)$.

Analysis: the proof

- Let M_{greedy} denote the output matching of the greedy algorithm.
- $L = V(M_{\text{greedy}})$ is the set of matched vertices.
- It is obvious that L is a covering vertex set, and $|L| = 2\nu_{\text{greedy}}(G)$.
- The size of L gives an upper bound on the size of an arbitrary matching, hence $\nu(G) \leq |L| = 2\nu_{\text{greedy}}(G)$.

Break

Algorithms based on augmentations

Algorithms based on augmentations

- Algorithms based on augmentations always define a way augmenting an actual feasible solution. Then from an initial solution they try to find an good/optimal solution.

Algorithms based on augmentations

• Algorithms based on augmentations always define a way augmenting an actual feasible solution. Then from an initial solution they try to find an good/optimal solution.

• When halting the actual solution can't be augmented in certain way.

Algorithms based on augmentations

- Algorithms based on augmentations always define a way augmenting an actual feasible solution. Then from an initial solution they try to find an good/optimal solution.

- When halting the actual solution can't be augmented in certain way.

- We apply this scheme for finding optimum matching.

Algorithms based on augmentations

• Algorithms based on augmentations always define a way augmenting an actual feasible solution. Then from an initial solution they try to find an good/optimal solution.

• When halting the actual solution can't be augmented in certain way.

• We apply this scheme for finding optimum matching. Now on we are always given a graph G and a matching M.

The notion of augmenting path

The notion of augmenting path

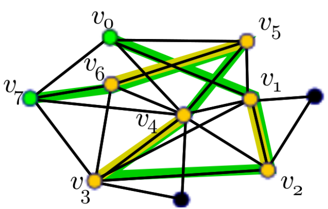
Definition

Let G be a graph and M a matching in it, $P : v_0, e_1, v_1, \ldots, e_k, v_k$ is a path of G. P is an augmenting path for M if v_0 and v_k unmatched, k is odd, furthermore the edges of M and $E(G) - M$ are alternating along P.

The notion of augmenting path

Definition

Let G be a graph and M a matching in it, $P : v_0, e_1, v_1, \ldots, e_k, v_k$ is a path of G. P is an augmenting path for M if v_0 and v_k unmatched, k is odd, furthermore the edges of M and $E(G) - M$ are alternating along P.



The yellow edges form a matching. The green path is an augmenting path for M. The right figure exhibits the augmented matching.

The augmentation

The augmentation

Observation

Let P be an augmenting path for M. Then

$$M' = (M \setminus E(P)) \cup (E(P) \setminus M) = M \triangle E(P)$$

is a matching.

The augmentation

Observation

Let P be an augmenting path for M. Then

$$M' = (M \setminus E(P)) \cup (E(P) \setminus M) = M \triangle E(P)$$

is a matching.

- Note that the size of M' is larger than the size of M. The $M \leftarrow M'$ step is called augmentation of M.

The algorithmic scheme to find large matching

The algorithmic scheme to find large matching

The scheme

The algorithmic scheme to find large matching

The scheme

Given a graph G and a matching M in it.

The algorithmic scheme to find large matching

The scheme

Given a graph G and a matching M in it.

WHILE we find an augmentation path P for M do
(Augmentation) $M \leftarrow M \Delta E(P)$.

The algorithmic scheme to find large matching

The scheme

Given a graph G and a matching M in it.

WHILE we find an augmentation path P for M do
(Augmentation) $M \leftarrow M \Delta E(P)$.

(Halting) We output the actual matching.
// In the case of halting we know that there is no augmenting path for M.

Remarks

Remarks

- There is no fear of infinite cycle.

Remarks

- There is no fear of infinite cycle.
- Augmenting along a path of length 1 is the greedy extension.

Remarks

- There is no fear of infinite cycle.
- Augmenting along a path of length 1 is the greedy extension.
- The major questions are:

(1) What one can say about the size of the output?

(2) How to find augmenting path for given matching M?

(1): Theorem of Berge

(1): Theorem of Berge

Theorem of Berge (1957)

Let G be a graph and M a matching. If M is non-optimal, then there exists an augmenting path.

(1): Theorem of Berge

Theorem of Berge (1957)

Let G be a graph and M a matching. If M is non-optimal, then there exists an augmenting path.

We answered (1): Our scheme

(1): Theorem of Berge

Theorem of Berge (1957)

Let G be a graph and M a matching. If M is non-optimal, then there exists an augmenting path.

We answered (1): Our scheme

„Corollary"

Furthermore we have a new problem: For input (G, M) we search for an augmenting path for matching M in the graph G.

(2) Greedy search for augmenting path, warm up

(2) Greedy search for augmenting path, warm up

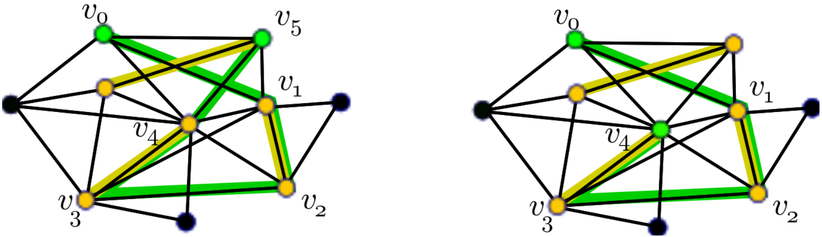
Definition

P is called a partial augmenting path, if $P: v_0, e_1, v_1, \ldots, e_k, v_k$ is a path, and $v_0 \notin V(M)$, furthermore $e_{2i} \in M$ $(i = 1, 2, \ldots)$.

(2) Greedy search for augmenting path, warm up

Definition

P is called a partial augmenting path, if $P: v_0, e_1, v_1, \ldots, e_k, v_k$ is a path, and $v_0 \notin V(M)$, furthermore $e_{2i} \in M$ $(i = 1, 2, \ldots)$.



A partial augmenting path of length 5 (v_5 is matched). On the right hand side we have a partial augmenting path of length 4 (it has even length).

The basics

The basics

- We search for partial augmenting paths.

The basics

- We search for partial augmenting paths.
- If we find one partial augmenting path leading to v, then we label v.

The basics

- We search for partial augmenting paths.

- If we find one partial augmenting path leading to v, then we label v. The label contains the information that we found a partial augmenting path leading to v.

The basics

- We search for partial augmenting paths.

- If we find one partial augmenting path leading to v, then we label v. The label contains the information that we found a partial augmenting path leading to v. We also maintain an additional information: the parity of the length of the path, we found.

The basics

- We search for partial augmenting paths.

- If we find one partial augmenting path leading to v, then we label v. The label contains the information that we found a partial augmenting path leading to v. We also maintain an additional information: the parity of the length of the path, we found.

- Let O be the set of vertices that are reached by a partial augmenting path of even length.

The basics

- We search for partial augmenting paths.

- If we find one partial augmenting path leading to v, then we label v. The label contains the information that we found a partial augmenting path leading to v. We also maintain an additional information: the parity of the length of the path, we found.

- Let O be the set of vertices that are reached by a partial augmenting path of even length. The elements of O are called outer vertices.

The basics

- We search for partial augmenting paths.

- If we find one partial augmenting path leading to v, then we label v. The label contains the information that we found a partial augmenting path leading to v. We also maintain an additional information: the parity of the length of the path, we found.

- Let O be the set of vertices that are reached by a partial augmenting path of even length. The elements of O are called outer vertices. Let I be the set of vertices that are reached by a partial augmenting path of odd length.

The basics

- We search for partial augmenting paths.

- If we find one partial augmenting path leading to v, then we label v. The label contains the information that we found a partial augmenting path leading to v. We also maintain an additional information: the parity of the length of the path, we found.

- Let O be the set of vertices that are reached by a partial augmenting path of even length. The elements of O are called outer vertices. Let I be the set of vertices that are reached by a partial augmenting path of odd length. The elements of I are called inner vertices.

The basics

- We search for partial augmenting paths.

- If we find one partial augmenting path leading to v, then we label v. The label contains the information that we found a partial augmenting path leading to v. We also maintain an additional information: the parity of the length of the path, we found.

- Let O be the set of vertices that are reached by a partial augmenting path of even length. The elements of O are called outer vertices. Let I be the set of vertices that are reached by a partial augmenting path of odd length. The elements of I are called inner vertices. Let L be the set of labeled vertices, i.e. $L = O \cup I$.

The basics (cont'd)

The basics (cont'd)

- Our search will be greedy.

The basics (cont'd)

- Our search will be greedy. If we label a vertex then we won't change it (even if we find an alternative partial augmenting path to it).

The basics (cont'd)

- Our search will be greedy. If we label a vertex then we won't change it (even if we find an alternative partial augmenting path to it).
- We will choose a subset R of $\overline{V}(M)$.

The basics (cont'd)

- Our search will be greedy. If we label a vertex then we won't change it (even if we find an alternative partial augmenting path to it).
- We will choose a subset R of $\overline{V}(M)$. We will consider these vertices as partial augmenting paths of length 0.

The basics (cont'd)

• Our search will be greedy. If we label a vertex then we won't change it (even if we find an alternative partial augmenting path to it).

• We will choose a subset R of $\overline{V}(M)$. We will consider these vertices as partial augmenting paths of length 0. These vertices will be the roots of our search.

The basics (cont'd)

• Our search will be greedy. If we label a vertex then we won't change it (even if we find an alternative partial augmenting path to it).

• We will choose a subset R of $\overline{V}(M)$. We will consider these vertices as partial augmenting paths of length 0. These vertices will be the roots of our search. The roots will be outer vertices.

The greedy search

Greedy search for augmenting path

The greedy search

Greedy search for augmenting path

(Initialization) Start with a subset of $\overline{V}(M)$, called R.
// First, $O = R$, $I = \emptyset$, $L = K$.

The greedy search

Greedy search for augmenting path

(Initialization) Start with a subset of $\overline{V}(M)$, called R.
// First, $O = R$, $I = \emptyset$, $L = K$.

WHILE there is $o \in O$ and an unlabeled neighbor s of it

The greedy search

Greedy search for augmenting path

(Initialization) Start with a subset of $\overline{V}(M)$, called R.
// First, $O = R$, $I = \emptyset$, $L = K$.

WHILE there is $o \in O$ and an unlabeled neighbor s of it

If s is a unmatched vertex

The greedy search

Greedy search for augmenting path

(Initialization) Start with a subset of $\overline{V}(M)$, called R.
// First, $O = R$, $I = \emptyset$, $L = K$.

WHILE there is $o \in O$ and an unlabeled neighbor s of it

If s is a unmatched vertex
(Success) // Our search found a partial augmenting path to o.

The greedy search

Greedy search for augmenting path

(Initialization) Start with a subset of $\overline{V}(M)$, called R.
// First, $O = R$, $I = \emptyset$, $L = K$.

WHILE there is $o \in O$ and an unlabeled neighbor s of it

If s is a unmatched vertex
(Success) // Our search found a partial augmenting path to o.
We can extend this path to an augmenting path.

The greedy search

Greedy search for augmenting path

(Initialization) Start with a subset of $\overline{V}(M)$, called R.
// First, $O = R$, $I = \emptyset$, $L = K$.

WHILE there is $o \in O$ and an unlabeled neighbor s of it

If s is a unmatched vertex
(Success) // Our search found a partial augmenting path to o.
We can extend this path to an augmenting path.

If s is matched, them
(Greedy extension of labels) let s' be the vertex matched to s by M. $O \leftarrow O \cup \{s'\}$, $I \leftarrow I \cup \{s\}$, $L \leftarrow L \cup \{s, s'\}$.

The greedy search

Greedy search for augmenting path

(Initialization) Start with a subset of $\overline{V}(M)$, called R.
// First, $O = R$, $I = \emptyset$, $L = K$.

WHILE there is $o \in O$ and an unlabeled neighbor s of it

If s is a unmatched vertex
(Success) // Our search found a partial augmenting path to o.
We can extend this path to an augmenting path.

If s is matched, them
(Greedy extension of labels) let s' be the vertex matched to s by M. $O \leftarrow O \cup \{s'\}$, $I \leftarrow I \cup \{s\}$, $L \leftarrow L \cup \{s, s'\}$.

(Halting) If we exit the above while loop without success then we say that the search is unsuccessful.
// In this case all neighbors of the vertices of O are labeled.

Search/alternating forest

Search/alternating forest

- We can enrich the above algorithm. In the case of extending the labeling we point out an edge, that is responsible for the label: the vertex s will obtain its label through the edge os, s' will get label because of the edge $ss' \in M$.

Search/alternating forest

- We can enrich the above algorithm. In the case of extending the labeling we point out an edge, that is responsible for the label: the vertex s will obtain its label through the edge os, s' will get label because of the edge $ss' \in M$.

- The responsible edges and the labeled vertices form a rooted forest.

Search/alternating forest

• We can enrich the above algorithm. In the case of extending the labeling we point out an edge, that is responsible for the label: the vertex s will obtain its label through the edge os, s' will get label because of the edge $ss' \in M$.

• The responsible edges and the labeled vertices form a rooted forest. The vertices of R are the roots of the forest.

Search/alternating forest

- We can enrich the above algorithm. In the case of extending the labeling we point out an edge, that is responsible for the label: the vertex s will obtain its label through the edge os, s' will get label because of the edge $ss' \in M$.

- The responsible edges and the labeled vertices form a rooted forest. The vertices of R are the roots of the forest.

- The extensions of the labels/our search is done by double outgrowth operations.

Search/alternating forest

• We can enrich the above algorithm. In the case of extending the labeling we point out an edge, that is responsible for the label: the vertex s will obtain its label through the edge os, s' will get label because of the edge $ss' \in M$.

• The responsible edges and the labeled vertices form a rooted forest. The vertices of R are the roots of the forest.

• The extensions of the labels/our search is done by double outgrowth operations.

• The branches of length 2 has a middle vertex. These vertices get the "inner" label.

Search/alternating forest

• We can enrich the above algorithm. In the case of extending the labeling we point out an edge, that is responsible for the label: the vertex s will obtain its label through the edge os, s' will get label because of the edge $ss' \in M$.

• The responsible edges and the labeled vertices form a rooted forest. The vertices of R are the roots of the forest.

• The extensions of the labels/our search is done by double outgrowth operations.

• The branches of length 2 has a middle vertex. These vertices get the "inner" label. The last vertex of the branches of length 2 is a vertex where the search can be continued to any unlabeled neighbor.

Search/alternating forest

- We can enrich the above algorithm. In the case of extending the labeling we point out an edge, that is responsible for the label: the vertex s will obtain its label through the edge os, s' will get label because of the edge $ss' \in M$.

- The responsible edges and the labeled vertices form a rooted forest. The vertices of R are the roots of the forest.

- The extensions of the labels/our search is done by double outgrowth operations.

- The branches of length 2 has a middle vertex. These vertices get the "inner" label. The last vertex of the branches of length 2 is a vertex where the search can be continued to any unlabeled neighbor.

- Every vertex v of the search forest has a component, and the component has a root r.

Search/alternating forest

• We can enrich the above algorithm. In the case of extending the labeling we point out an edge, that is responsible for the label: the vertex s will obtain its label through the edge os, s' will get label because of the edge $ss' \in M$.

• The responsible edges and the labeled vertices form a rooted forest. The vertices of R are the roots of the forest.

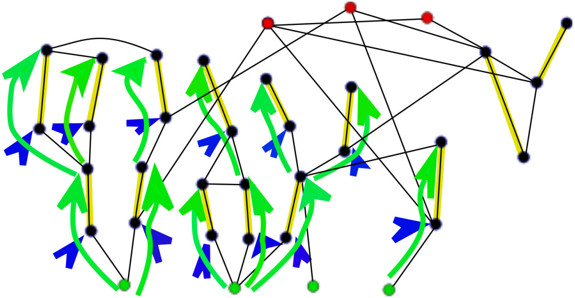
• The extensions of the labels/our search is done by double outgrowth operations.

• The branches of length 2 has a middle vertex. These vertices get the "inner" label. The last vertex of the branches of length 2 is a vertex where the search can be continued to any unlabeled neighbor.

• Every vertex v of the search forest has a component, and the component has a root r. The unique rv path in the forest is the

A picture of a search forest

A picture of a search forest



The yellow edges form our matching. The green vertices are the roots. The red vertices are unmatched, unlabeled (at the beginning) vertices.

An important remark

An important remark

- Remember: At the beginning every labeled vertex is an outer one. At each extension exactly one inner label and exactly one outer label is assigned. The two new labeled vertices are matched to each other.

An important remark

• Remember: At the beginning every labeled vertex is an outer one. At each extension exactly one inner label and exactly one outer label is assigned. The two new labeled vertices are matched to each other.

Observation

(1) Our promise, that s' is automatically unlabeled is true.

An important remark

• Remember: At the beginning every labeled vertex is an outer one. At each extension exactly one inner label and exactly one outer label is assigned. The two new labeled vertices are matched to each other.

Observation

(1) Our promise, that s' is automatically unlabeled is true.

(2) Any point of our algorithm we have $|O| - |I| = |R|$.

An important remark

• Remember: At the beginning every labeled vertex is an outer one. At each extension exactly one inner label and exactly one outer label is assigned. The two new labeled vertices are matched to each other.

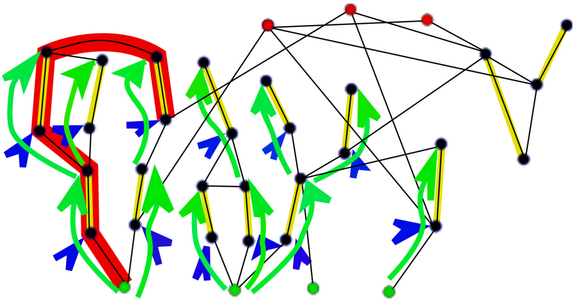
Observation

(1) Our promise, that s' is automatically unlabeled is true.

(2) Any point of our algorithm we have $|O| - |I| = |R|$. // Easy induction

An important example

An important example



The red path is a partial augmenting path, that reaches its endvertex in a wrong phase.

Break

Theorem

Theorem

Theorem (Harold Kuhn, Dénes Kőnig—Jenő Egerváry/Hungarian method)

Let G be a bipartite graph with color classes L and U. Let M be a matching in G. Let $R = A \cap \overline{V}(M)$. $R = A \cap \overline{V}(M)$.

Theorem

Theorem (Harold Kuhn, Dénes Kőnig—Jenő Egerváry/Hungarian method)

Let G be a bipartite graph with color classes L and U. Let M be a matching in G. Let $R = A \cap \overline{V}(M)$. $R = A \cap \overline{V}(M)$.

If the greedy search for augmenting path is unsuccessful, then the matching is optimal (i.e. there is no augmenting path).

Greedy search for augmenting path: The case of bipartite graphs

Greedy search for augmenting path: The case of bipartite graphs

- At the beginning $O \subseteq L$, and $I \subseteq U$.

Greedy search for augmenting path: The case of bipartite graphs

- At the beginning $O \subseteq L$, and $I \subseteq U$.

Observation

The above property is fulfilled during the complete run of the algorithm.

Greedy search for augmenting path: The case of bipartite graphs

- At the beginning $O \subseteq L$, and $I \subseteq U$.

Observation

The above property is fulfilled during the complete run of the algorithm.

- In other words the outer and lower categories are the same.

Greedy search for augmenting path: The case of bipartite graphs

- At the beginning $O \subseteq L$, and $I \subseteq U$.

Observation

The above property is fulfilled during the complete run of the algorithm.

- In other words the outer and lower categories are the same. Of course, the outer and lower categories are the same too.

Greedy search for augmenting path: initial remarks

Greedy search for augmenting path: initial remarks

- We assume that at the end of the run of the algorithm the search is unsuccessful,

Greedy search for augmenting path: initial remarks

- We assume that at the end of the run of the algorithm the search is unsuccessful, hence the neighbors of outer vertices are labeled.

Greedy search for augmenting path: initial remarks

- We assume that at the end of the run of the algorithm the search is unsuccessful, hence the neighbors of outer vertices are labeled.

Notation

The neighbors of X are denoted as
$N(X) = \{s \in V,\ s \text{ connected to a vertex in } X\}$.

Greedy search for augmenting path: initial remarks

- We assume that at the end of the run of the algorithm the search is unsuccessful, hence the neighbors of outer vertices are labeled.

Notation

The neighbors of X are denoted as
$N(X) = \{s \in V,\ s \text{ connected to a vertex in } X\}$.

- At the end of the algorithm we have $N(K) \subset C$.

First proof: Notations

First proof: Notations

Notations

Let $S \subset L$. Let $\epsilon(S) = |S| - |N(S)|$, i.e. the surplus of S compared to the number of neighbors.

First proof: Notations

Notations

Let $S \subset L$. Let $\epsilon(S) = |S| - |N(S)|$, i.e. the surplus of S compared to the number of neighbors.

Notation

$\delta_L(P) = |L| - |P|$, i.e. the number of unmatched lower vertices.

First proof: An observation

First proof: An observation

Observation

Let S be an arbitrary set of lower vertices ($S \subset L$). Let M be a matching. Then we have

$$\delta_L(P) \geq \epsilon(S).$$

First proof: An observation

Observation

Let S be an arbitrary set of lower vertices ($S \subset L$). Let M be a matching. Then we have

$$\delta_L(P) \geq \epsilon(S).$$

- The observation has mathematical content if $\epsilon(S) > 0$.

First proof

First proof

Corollary

Let $S \subset L$, and assume $\epsilon(S) > 0$. Then there is no matching that matches each lower vertex.

First proof

Corollary

Let $S \subset L$, and assume $\epsilon(S) > 0$. Then there is no matching that matches each lower vertex.

Definition

S, a set of lower vertices is called Kőnig obstruction iff $\epsilon(S) > 0$.

First proof

First proof

Corollary

Let S be an arbitrary set of lower vertices. Let P be an arbitrary matching. If $\epsilon(S) = \delta_L(P)$, then P is an optimal matching.

First proof

Corollary

Let S be an arbitrary set of lower vertices. Let P be an arbitrary matching. If $\epsilon(S) = \delta_L(P)$, then P is an optimal matching.

In the above case one says S lower-Kőnig-set, that proves the optimality of P.

First proof

First proof

- Remember: We assume that we run the Hungarian method on G, M, and it ends with an unsuccessful search.

First proof

- Remember: We assume that we run the Hungarian method on G, M, and it ends with an unsuccessful search.
- We know, that $N(O) \subset C$,

First proof

- Remember: We assume that we run the Hungarian method on G, M, and it ends with an unsuccessful search.
- We know, that $N(O) \subset C$, $O \subset L$,

First proof

- Remember: We assume that we run the Hungarian method on G, M, and it ends with an unsuccessful search.
- We know, that $N(O) \subset C$, $O \subset L$, $N(O) \subset U$,

First proof

- Remember: We assume that we run the Hungarian method on G, M, and it ends with an unsuccessful search.

- We know, that $N(O) \subset C$, $O \subset L$, $N(O) \subset U$, $N(O) \subset C \cap U = I$.

First proof

- Remember: We assume that we run the Hungarian method on G, M, and it ends with an unsuccessful search.
- We know, that $N(O) \subset C$, $O \subset L$, $N(O) \subset U$, $N(O) \subset C \cap U = I$.
- With some additional thought we have $N(O) = I$.

First proof

- Remember: We assume that we run the Hungarian method on G, M, and it ends with an unsuccessful search.

- We know, that $N(O) \subset C$, $O \subset L$, $N(O) \subset U$, $N(O) \subset C \cap U = I$.

- With some additional thought we have $N(O) = I$.

- Hence

$$\epsilon(O) = |O| - |N(O)| = |O| - |I| = |R| = |L \cap \overline{V}(M)| = \delta_L(M).$$

First proof

- Remember: We assume that we run the Hungarian method on G, M, and it ends with an unsuccessful search.

- We know, that $N(O) \subset C$, $O \subset L$, $N(O) \subset U$, $N(O) \subset C \cap U = I$.

- With some additional thought we have $N(O) = I$.

- Hence

$$\epsilon(O) = |O| - |N(O)| = |O| - |I| = |R| = |L \cap \overline{V}(M)| = \delta_L(M).$$

- So O is a lower-Kőnig-set that proves M optimality.

A mathematical corollary of the first proof

A mathematical corollary of the first proof

Kőnig's theorem

$$\min\{\delta_L(P) : \ P \text{ is a matching}\} = \max\{\epsilon(S) : S \subset L\}.$$

Second proof: Initial notation

Second proof: Initial notation

Reminder

$C \subset V$ is a covering set of any edge has at least one endvertex in C.

Second proof: The main observation

Second proof: The main observation

Observation

Assume that C is a covering set, and P is a matching. Then $|C| \geq |P|$.

Second proof: The main observation

Observation

Assume that C is a covering set, and P is a matching. Then $|C| \geq |P|$.

Corollary

Let C be a covering set and M be a matching. If $|C| = |P|$, then P is an optimal matching in G, and C is an optimal covering set of G.

Second proof: The main observation

Observation

Assume that C is a covering set, and P is a matching. Then $|C| \geq |P|$.

Corollary

Let C be a covering set and M be a matching. If $|C| = |P|$, then P is an optimal matching in G, and C is an optimal covering set of G.

In the case of sets C, P as above, one say that C proves that P is optimal. Or one can say that P proves that L is optimal.

Second proof

Second proof

- Now we return to our bipartite graph and matching M, where the Hungarian method stops with an unsuccessful search.

Second proof

- Now we return to our bipartite graph and matching M, where the Hungarian method stops with an unsuccessful search.

- We can classify the edges of M into two classes: labeled and unlabeled.

Second proof

- Now we return to our bipartite graph and matching M, where the Hungarian method stops with an unsuccessful search.

- We can classify the edges of M into two classes: labeled and unlabeled.

- Let U_{labeled} be the set of upper vertices of labeled edges of M.

Second proof

• Now we return to our bipartite graph and matching M, where the Hungarian method stops with an unsuccessful search.

• We can classify the edges of M into two classes: labeled and unlabeled.

• Let U_{labeled} be the set of upper vertices of labeled edges of M. Let $L_{\text{unlabeled}}$ be the set of lower vertices of the unlabeled edges of M.

Second proof

- Now we return to our bipartite graph and matching M, where the Hungarian method stops with an unsuccessful search.

- We can classify the edges of M into two classes: labeled and unlabeled.

- Let U_{labeled} be the set of upper vertices of labeled edges of M. Let $L_{\text{unlabeled}}$ be the set of lower vertices of the unlabeled edges of M.

- Let $C = U_{\text{labeled}} \cup L_{\text{unlabeled}}$. Note that $|C| = |M|$.

Second proof

Second proof

Observation

C is a covering set in G.

Second proof

Observation

C is a covering set in G.

- The observation gives us that, C proves the optimality of M.

Mathematical proof of the second proof

Mathematical proof of the second proof

Kőnig's theorem

For a bipartite graph G

$$\nu(G) = \tau(G).$$

Final remarks

Final remarks

• The Hungarian method gives an efficient algorithm to find an optimal matching in a bipartite graph.

Final remarks

- The Hungarian method gives an efficient algorithm to find an optimal matching in a bipartite graph.
- We will see that this algorithm can be extended to general graphs.

Final remarks

• The Hungarian method gives an efficient algorithm to find an optimal matching in a bipartite graph.

• We will see that this algorithm can be extended to general graphs.

• As a byproduct we obtain an efficient algorithm that finds an optimal covering set in a given BIPARTITE graph.

Final remarks

- The Hungarian method gives an efficient algorithm to find an optimal matching in a bipartite graph.
- We will see that this algorithm can be extended to general graphs.
- As a byproduct we obtain an efficient algorithm that finds an optimal covering set in a given BIPARTITE graph.
- No one can extend this algorithm to the general case.

Break

The general case: Initial steps

The general case: Initial steps

- We start with $O = \overline{V}(M)$, $B = \emptyset$.

The general case: Initial steps

• We start with $O = \overline{V}(M)$, $B = \emptyset$.

• Strange start. Success is impossible, there are no room to complete an augmenting path by the greedy approach.

Edmonds' algorithm: 0th version

Edmonds' algorithm: 0th version

Edmonds' algorithm: 0th version

Edmonds' algorithm: 0th version

Edmonds' algorithm: 0th version

(Initialization) Let $R = \overline{V}(M)$.

Edmonds' algorithm: 0th version

Edmonds' algorithm: 0th version

(Initialization) Let $R = \overline{V}(M)$. $O = R$, $I = \emptyset$, $L = O = R$.

Edmonds' algorithm: 0th version

Edmonds' algorithm: 0th version

(Initialization) Let $R = \overline{V}(M)$. $O = R$, $I = \emptyset$, $L = O = R$.
(Label extension) Greedy extension till it gets stuck.
// After this step we have a search forest. The forest has $|R|$ components. Any tree in the forest is rooted tree, with a root in R.

Edmonds' algorithm: 0th version

Edmonds' algorithm: 0th version

(Initialization) Let $R = \overline{V}(M)$. $O = R$, $I = \emptyset$, $L = O = R$.
(Label extension) Greedy extension till it gets stuck.
// After this step we have a search forest. The forest has $|R|$ components. Any tree in the forest is rooted tree, with a root in R.
If there is no edge inside O, then we stop: Unsuccessful search.

Edmonds' algorithm: 0th version

Edmonds' algorithm: 0th version

(Initialization) Let $R = \overline{V}(M)$. $O = R$, $I = \emptyset$, $L = O = R$.
(Label extension) Greedy extension till it gets stuck.
// After this step we have a search forest. The forest has $|R|$ components. Any tree in the forest is rooted tree, with a root in R.
If there is no edge inside O, then we stop: Unsuccessful search.
If $e = kk' \in E$, where $k, k' \in O$:
// Let $r, r' \in R$ those roots where the search trees reaching k, and k' are rooted.

Edmonds' algorithm: 0th version

Edmonds' algorithm: 0th version

(Initialization) Let $R = \overline{V}(M)$. $O = R$, $I = \emptyset$, $L = O = R$.
(Label extension) Greedy extension till it gets stuck.
// After this step we have a search forest. The forest has $|R|$ components. Any tree in the forest is rooted tree, with a root in R.
If there is no edge inside O, then we stop: Unsuccessful search.
If $e = kk' \in E$, where $k, k' \in O$:
// Let $r, r' \in R$ those roots where the search trees reaching k, and k' are rooted.
If $r \neq r'$, (Successful search) We take the augmenting path to k, we cross e, finally from k'we follow the $r'k'$ augmenting path in revers order.

Edmonds' algorithm: 0th version

Edmonds' algorithm: 0th version

(Initialization) Let $R = \overline{V}(M)$. $O = R$, $I = \emptyset$, $L = O = R$.
(Label extension) Greedy extension till it gets stuck.
// After this step we have a search forest. The forest has $|R|$ components. Any tree in the forest is rooted tree, with a root in R.
If there is no edge inside O, then we stop: Unsuccessful search.
If $e = kk' \in E$, where $k, k' \in O$:
// Let $r, r' \in R$ those roots where the search trees reaching k, and k' are rooted.
If $r \neq r'$, (Successful search) We take the augmenting path to k, we cross e, finally from k'we follow the $r'k'$ augmenting path in revers order.
If $r = r'$, then (Case Edmonds) $\star$ // We will discuss it later.

Case Edmonds: The notions

Case Edmonds: The notions

- Let P be the augmenting path from r to k that is found by the greedy search.

Case Edmonds: The notions

- Let P be the augmenting path from r to k that is found by the greedy search. Similarly let P' be the augmenting path from r to k' that is found by the greedy search.

Case Edmonds: The notions

- Let P be the augmenting path from r to k that is found by the greedy search. Similarly let P' be the augmenting path from r to k' that is found by the greedy search.
- Let a_e be the branching point, where the two paths part each other.

Case Edmonds: The notions

- Let P be the augmenting path from r to k that is found by the greedy search. Similarly let P' be the augmenting path from r to k' that is found by the greedy search.

- Let a_e be the branching point, where the two paths part each other.

Observation

a_e is an outer vertex.

Case Edmonds: The notions

- Let P be the augmenting path from r to k that is found by the greedy search. Similarly let P' be the augmenting path from r to k' that is found by the greedy search.

- Let a_e be the branching point, where the two paths part each other.

Observation

a_e is an outer vertex.

Let C_e be the following cycle: Following P we walk from a_e to k, we cross e, furthermore we walk from k' to a_e following P'.

Case Edmonds: The notions

- Let P be the augmenting path from r to k that is found by the greedy search. Similarly let P' be the augmenting path from r to k' that is found by the greedy search.

- Let a_e be the branching point, where the two paths part each other.

Observation

a_e is an outer vertex.

Let C_e be the following cycle: Following P we walk from a_e to k, we cross e, furthermore we walk from k' to a_e following P'.

Observation

C_e is an odd cycle.

Case Edmonds: An observation

Case Edmonds: An observation

Observation

In G we have an augmenting path of even length to each vertex of C_e .

Case Edmonds: An observation

Observation

In G we have an augmenting path of even length to each vertex of C_e .

Definition: Contracting the cycle C_e

We define a $\widetilde{G}$ graph:

Case Edmonds: An observation

Observation

In G we have an augmenting path of even length to each vertex of C_e .

Definition: Contracting the cycle C_e

We define a $\widetilde{G}$ graph:
Informally: We substitute all nodes of the cycle with a "supernode".

Case Edmonds: An observation

Observation

In G we have an augmenting path of even length to each vertex of C_e .

Definition: Contracting the cycle C_e

We define a $\widetilde{G}$ graph:
Informally: We substitute all nodes of the cycle with a "supernode".
Formally:

Case Edmonds: An observation

Observation

In G we have an augmenting path of even length to each vertex of C_e .

Definition: Contracting the cycle C_e

We define a $\widetilde{G}$ graph:
Informally: We substitute all nodes of the cycle with a "supernode".
Formally: $V(\widetilde{G}) = V(G) \setminus V(C_e) \dot{\cup} \{c\}$,

Case Edmonds: An observation

Observation

In G we have an augmenting path of even length to each vertex of C_e .

Definition: Contracting the cycle C_e

We define a $\widetilde{G}$ graph:
Informally: We substitute all nodes of the cycle with a "supernode".
Formally: $V(\widetilde{G}) = V(G) \setminus V(C_e) \dot{\cup} \{c\}$,
$E(\widetilde{G}) = E(G) \setminus \{$ edges within $C_e\}$, $I(\widetilde{G})$ is the natural incidence.

Case Edmonds: An observation

Observation

In G we have an augmenting path of even length to each vertex of C_e .

Definition: Contracting the cycle C_e

We define a $\widetilde{G}$ graph:
Informally: We substitute all nodes of the cycle with a "supernode".
Formally: $V(\widetilde{G}) = V(G) \setminus V(C_e) \dot{\cup} \{c\}$,
$E(\widetilde{G}) = E(G) \setminus \{$ edges within $C_e\}$, $I(\widetilde{G})$ is the natural incidence.
$\widetilde{M} = M \cap E(\widetilde{G})$.

Case Edmonds: An observation

Observation

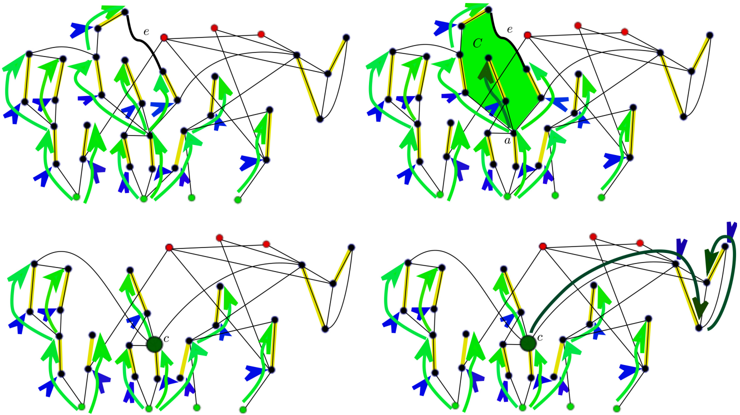
In G we have an augmenting path of even length to each vertex of C_e .

Definition: Contracting the cycle C_e

We define a $\widetilde{G}$ graph:
Informally: We substitute all nodes of the cycle with a "supernode".
Formally: $V(\widetilde{G}) = V(G) \setminus V(C_e) \dot{\cup} \{c\}$,
$E(\widetilde{G}) = E(G) \setminus \{$ edges within $C_e\}$, $I(\widetilde{G})$ is the natural incidence.
$\widetilde{M} = M \cap E(\widetilde{G})$.
$\widetilde{F}$: We have the "same" set of roots and we take all double outgrowth operations outside C_e.

Example

Example



On the left the picture shows the search and the edge e. In the middle we can see the cycle, the vertex a_e. On the right we see the extension of labeling after the contraction.

Lemma

Lemma

Lemma

(i) $\widetilde{M}$ is a matching in $\widetilde{G}$,

(ii) $\widetilde{F}$ is a search forest for $\widetilde{M}$,

(iii) c is an outer vertex in $\widetilde{F}$.

The case Edmonds

The case Edmonds

Case Edmonds

The case Edmonds

Case Edmonds

(Finding C_e) We arrive to this step with a suitable edge e. We find a_e and "build" C_e

The case Edmonds

Case Edmonds

(Finding C_e) We arrive to this step with a suitable edge e. We find a_e and "build" C_e
(Contraction) We "build" the graph $\widetilde{G}$.

The case Edmonds

Case Edmonds

(Finding C_e) We arrive to this step with a suitable edge e. We find a_e and "build" C_e
(Contraction) We "build" the graph $\widetilde{G}$.
(Saving the information) We construct $\widetilde{M}$ and $\widetilde{F}$.

The case Edmonds

Case Edmonds

(Finding C_e) We arrive to this step with a suitable edge e. We find a_e and "build" C_e
(Contraction) We "build" the graph $\widetilde{G}$.
(Saving the information) We construct $\widetilde{M}$ and $\widetilde{F}$.
(Iteration) We go back to the step (Label extension).

Where are we?

Where are we?

• A "generic" run of our algorithm look like sequence of contractions

$$(G, M) = (G_0, M_0) \to (G_1, M_1) \to \ldots \to (G_\ell, M_\ell)$$

and a final successful or unsuccessful search.

What's next?

What's next?

A missing Theorem

Let P be an augmenting path for M_ℓ in G_ℓ. Then there is one for M in G.

What's next?

A missing Theorem

Let P be an augmenting path for M_ℓ in G_ℓ. Then there is one for M in G.

A missing Lemma

Let P be an augmenting path for M_ℓ in G_ℓ. Then there is one for $M_{\ell-1}$ in $G_{\ell-1}$.

What's next?

A missing Theorem

Let P be an augmenting path for M_ℓ in G_ℓ. Then there is one for M in G.

A missing Lemma

Let P be an augmenting path for M_ℓ in G_ℓ. Then there is one for $M_{\ell-1}$ in $G_{\ell-1}$.

A missing Theorem

If the search in G_ℓ is unsuccessful, then the original matching M is optimal.

The end of the Edmonds' algorithm

The end of the Edmonds' algorithm

The end of the Edmonds' algorithm

The end of the Edmonds' algorithm

The end of the Edmonds' algorithm

(Successful search in G_ℓ) We apply the Lemma ℓ times to construct an augmenting path for the original M.

The end of the Edmonds' algorithm

The end of the Edmonds' algorithm

(Successful search in G_ℓ) We apply the Lemma ℓ times to construct an augmenting path for the original M.
This path P is the output of our algorithm.

The end of the Edmonds' algorithm

The end of the Edmonds' algorithm

(Successful search in G_ℓ) We apply the Lemma ℓ times to construct an augmenting path for the original M.
This path P is the output of our algorithm.

(Unsuccessful search in G_ℓ) We output: "There is no augmenting path for M".

The missing Theorem: Definitions

The missing Theorem: Definitions

Definition

Let$R \subset V(G)$. We define

$$\beta(R) = c_1(G - R) - |R|,$$

where c_1 give the number of components with odd vertices.

The missing Theorem: Definitions

Definition

Let $R \subset V(G)$. We define

$$\beta(R) = c_1(G - R) - |R|,$$

where c_1 give the number of components with odd vertices.
I.e. The surplus of the components of $G - R$ with odd vertices compared to the size of R.

We call this parameter of R the Berge-Tutte parameter.

The missing Theorem: Definitions

Definition

Let $R \subset V(G)$. We define

$$\beta(R) = c_1(G - R) - |R|,$$

where c_1 give the number of components with odd vertices. I.e. The surplus of the components of $G - R$ with odd vertices compared to the size of R.

We call this parameter of R the Berge-Tutte parameter.

Definition

Let P be a matching.

$$\delta(P) = |V(G)| - 2|P|.$$

The missing Theorem: An observation

The missing Theorem: An observation

- In each component of $G - R$ with odd vertices there will be at least one vertex that is not matched inside the component.

The missing Theorem: An observation

- In each component of $G - R$ with odd vertices there will be at least one vertex that is not matched inside the component.

- They can be matched in G. They can be matched only to vertices in R.

The missing Theorem: An observation

- In each component of $G - R$ with odd vertices there will be at least one vertex that is not matched inside the component.
- They can be matched in G. They can be matched only to vertices in R.
- If the surplus, defined above β is positive, then it gives a lower bound on the number of unmatched vertices.

The missing Theorem: An observation

- In each component of $G-R$ with odd vertices there will be at least one vertex that is not matched inside the component.
- They can be matched in G. They can be matched only to vertices in R.
- If the surplus, defined above β is positive, then it gives a lower bound on the number of unmatched vertices.
- In general, for arbitrary $R \subset V(G)$, and matching P

$$\beta(R) \leq \delta(P).$$

The missing Theorem: The consequences of the Observation

The missing Theorem: The consequences of the Observation

Definition

$T \subset V(G)$ vertex set is called Tutte obstruction iff $\beta(T) > 0$.

The missing Theorem: The consequences of the Observation

Definition

$T \subset V(G)$ vertex set is called Tutte obstruction iff $\beta(T) > 0$.

Observation

If there is a Tutte obstruction in G, then there is no perfect matching in G.

The missing Theorem: The consequences of the Observation

Definition

$T \subset V(G)$ vertex set is called Tutte obstruction iff $\beta(T) > 0$.

Observation

If there is a Tutte obstruction in G, then there is no perfect matching in G.

Observation

For arbitrary $R \subset V(G)$, and arbitrary matching P we have

$$\beta(R) = \delta(P),$$

then P is an optimal matching.

The missing Theorem: The consequences of the Observation

Definition

$T \subset V(G)$ vertex set is called Tutte obstruction iff $\beta(T) > 0$.

Observation

If there is a Tutte obstruction in G, then there is no perfect matching in G.

Observation

For arbitrary $R \subset V(G)$, and arbitrary matching P we have

$$\beta(R) = \delta(P),$$

then P is an optimal matching. One says that R, P is a Berge pair, and R is a Berge proof for the optimality of P.

The missing Theorem: Reminder, notations

The missing Theorem: Reminder, notations

- The run of Edmonds' algorithm is a sequence of contractions

$$(G, M) = (G_0, M_0) \to (G_1, M_1) \to \ldots \to (G_\ell, M_\ell).$$

The missing Theorem: Reminder, notations

- The run of Edmonds' algorithm is a sequence of contractions

$$(G, M) = (G_0, M_0) \to (G_1, M_1) \to \ldots \to (G_\ell, M_\ell).$$

- In G_i we have a search forest for M_i. We defined I_i, the set of inner vertices, and O_i the set of outer vertices.

The missing Theorem: Reminder, notations

- The run of Edmonds' algorithm is a sequence of contractions

$$(G, M) = (G_0, M_0) \to (G_1, M_1) \to \ldots \to (G_\ell, M_\ell).$$

- In G_i we have a search forest for M_i. We defined I_i, the set of inner vertices, and O_i the set of outer vertices.

- Note that there is a vertex in O_{i+1}, that is not present in O_i: the vertex representing the contracting cycle.

The missing Theorem: Reminder, notations

- The run of Edmonds' algorithm is a sequence of contractions

$$(G, M) = (G_0, M_0) \to (G_1, M_1) \to \ldots \to (G_\ell, M_\ell).$$

- In G_i we have a search forest for M_i. We defined I_i, the set of inner vertices, and O_i the set of outer vertices.

- Note that there is a vertex in O_{i+1}, that is not present in O_i: the vertex representing the contracting cycle. On the other side the vertices of I_{i+1} are present in G_i.

The missing Theorem: The essence

The missing Theorem: The essence

Theorem

In the case of unsuccessful search in each G_i the pair I_ℓ, M_i is a Berge pair.

The missing Theorem: The essence

Theorem

In the case of unsuccessful search in each G_i the pair I_ℓ, M_i is a Berge pair.

The Theorem gives us that M_i is optimal in G_i. The case $i = 0$ proves the missing Theorem and completes the proof of correctness of Edmonds' algorithm.

The missing Theorem: The essence

Theorem

In the case of unsuccessful search in each G_i the pair I_ℓ, M_i is a Berge pair.

The Theorem gives us that M_i is optimal in G_i. The case $i = 0$ proves the missing Theorem and completes the proof of correctness of Edmonds' algorithm.

We need to prove a sequence of claims. We do induction and follow the order $i = \ell, \ell - 1, \ell - 2, \ldots, 2, 1, 0$.

The Theorem: The start of the induction

The Theorem: The start of the induction

- We discuss the case of $i = \ell$.

The Theorem: The start of the induction

- We discuss the case of $i = \ell$.

Observation

The vertices of O_ℓ in $G_\ell - I_\ell$ are isolated vertices.

The Theorem: The start of the induction

- We discuss the case of $i = \ell$.

Observation

The vertices of O_ℓ in $G_\ell - I_\ell$ are isolated vertices.

- Specially $c_1(G_\ell - I_\ell) \geq |O_\ell|$, furthermore

$$\beta(I_\ell) = c_1(G_\ell - I_\ell) - |I_\ell| \geq |O_\ell| - |I_\ell| = \delta(M_\ell).$$

The Theorem: The induction step

The Theorem: The induction step

- When we do the step $G_{i+1} \to G_i$ we blow up a vertex $c_{i+1} \in O_{i+1}$ to an odd cycle.

The Theorem: The induction step

- When we do the step $G_{i+1} \to G_i$ we blow up a vertex $c_{i+1} \in O_{i+1}$ to an odd cycle.

- This effects only one component: With one exception the components$G_{i+1} - I_\ell$ are "untouched" in $G_i - I_\ell$.

The Theorem: The induction step

- When we do the step $G_{i+1} \to G_i$ we blow up a vertex $c_{i+1} \in O_{i+1}$ to an odd cycle.

- This effects only one component: With one exception the components$G_{i+1} - I_\ell$ are "untouched" in $G_i - I_\ell$.

- The number of vertices in the new component has the same parity as its ancestor. Specially $c_1(G_i - I_\ell) = c_1(G_{i+1} - I_\ell)$.

The Theorem: The induction step (cont'd)

The Theorem: The induction step (cont'd)

- The Berge-Tutte parameter of I_ℓ in the graph G_i

$$c_1(G_i - I_\ell) - |I_\ell| = c_1(G_{i+1} - I_\ell) - |I_\ell| = \delta(M_{i+1}).$$

The Theorem: The induction step (cont'd)

- The Berge-Tutte parameter of I_ℓ in the graph G_i

$$c_1(G_i - I_\ell) - |I_\ell| = c_1(G_{i+1} - I_\ell) - |I_\ell| = \delta(M_{i+1}).$$

- To complete the proof we need to note that $\delta(M_i) = |V(G_i)| - 2|M_i|$ doesn't change during the algorithm.

Corollaries

Corollaries

Tutte's Theorem (1947)

A graph G contains perfect matching if and only if it has no Tutte's obstruction.

Corollaries

Tutte's Theorem (1947)

A graph G contains perfect matching if and only if it has no Tutte's obstruction.

Berge's formula (1958)

For any graph G

$$\max\{\beta(T) : \ T \subset V(G)\} = \min\{\delta(P) : \ P \text{ is a matching}\}.$$

Corollaries

Tutte's Theorem (1947)

A graph G contains perfect matching if and only if it has no Tutte's obstruction.

Berge's formula (1958)

For any graph G

$$\max\{\beta(T) : T \subset V(G)\} = \min\{\delta(P) : P \text{ is a matching}\}.$$

The above two theorems have two directions. They are not symmetric. One direction is easy, straight forward. The other direction is the essence of the mathematical content.

Petersen's Theorem

Petersen's Theorem

Petersen's Theorem (1891)

If G is 3-regular 2-edge.connected graph then it has a perfect matching.

This is the end!

Thank you for your attention!