Shortest Reconfiguration of Matchings

Nicolas Bousquet, Tatsuhiko Hatanaka, Takehiro Ito and Moritz Mühlenthaler

Presenters: Niranjan and Mano Prakash

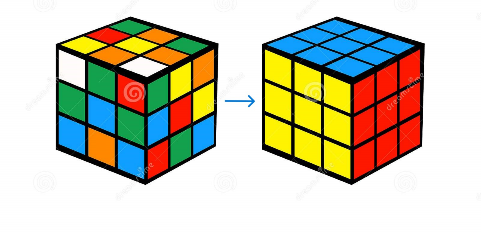
April 20, 2023

Reconfiguration Problem: In General

Reconfiguration Problem

Goal: Ask for the existence of a **step-by-step** transformation between two given configurations.

Example: Rubik's Cube Configurations.



- In this paper, we consider matchings in graphs as configurations.

Reconfiguration Problem: In Matchings

- Unlabelled tokens are placed on edges forming a matching of a graph.

Reconfiguration Problem: In Matchings

- Unlabelled tokens are placed on edges forming a matching of a graph.
- A token can be moved to another edge provided that the edges containing tokens remain a matching. This operation is known as **Token Jumping**.

Reconfiguration Problem: In Matchings

- Unlabelled tokens are placed on edges forming a matching of a graph.
- A token can be moved to another edge provided that the edges containing tokens remain a matching. This operation is known as **Token Jumping**.
- **Note:** If we only allow moving a token to an adjacent edge, this operation is called **Token Sliding**.

Reconfiguration Problem: In Matchings

- Unlabelled tokens are placed on edges forming a matching of a graph.
- A token can be moved to another edge provided that the edges containing tokens remain a matching. This operation is known as **Token Jumping**.
- **Note:** If we only allow moving a token to an adjacent edge, this operation is called **Token Sliding**.

Adjacency relation on matchings

Two matchings M and M' of a graph G are **adjacent** if one can be obtained from the other by relocating a single token.

Reconfiguration Problem: In Matchings

- Unlabelled tokens are placed on edges forming a matching of a graph.
- A token can be moved to another edge provided that the edges containing tokens remain a matching. This operation is known as **Token Jumping**.
- **Note:** If we only allow moving a token to an adjacent edge, this operation is called **Token Sliding**.

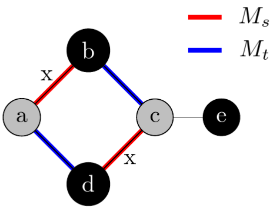
Adjacency relation on matchings

Two matchings M and M' of a graph G are **adjacent** if one can be obtained from the other by relocating a single token.

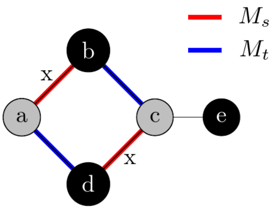
Reconfiguration sequence

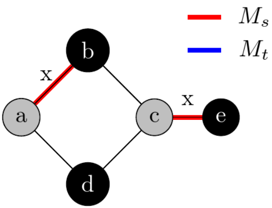
$M_0, M_1, ...M_l$ of matchings of G is a **reconfiguration sequence** of length l from M to M', if $M_0 = M, M_l = M'$, and the matchings M_{i-1} and M_i are adjacent for each $i \in \{1, 2, ...l\}$.

An Example of Reconfiguration Problem: In Matchings

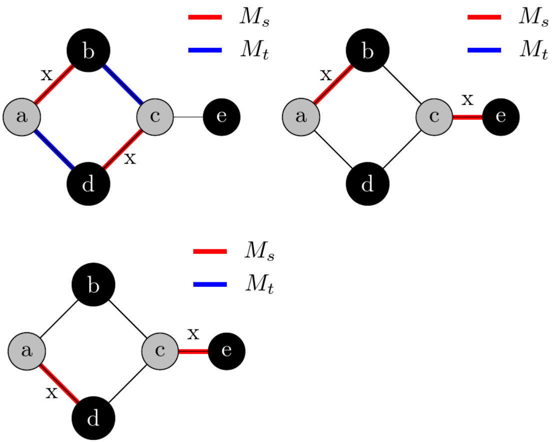


An Example of Reconfiguration Problem: In Matchings

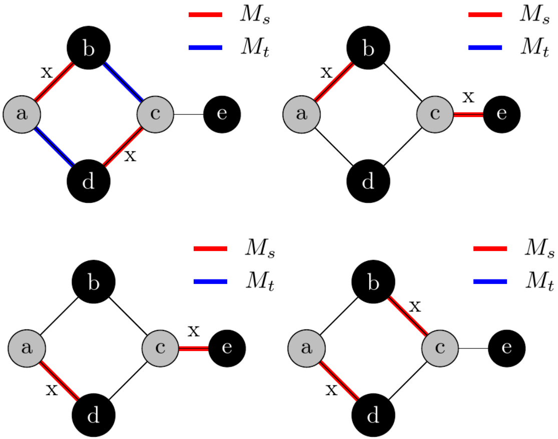




An Example of Reconfiguration Problem: In Matchings



An Example of Reconfiguration Problem: In Matchings



Reconfiguration Problem: In Matchings

- Two variants - *reachability variant* and *shortest variant*

Reconfiguration Problem: In Matchings

- Two variants - *reachability variant* and *shortest variant*

Reconfiguration of Matchings: Reachability Variant

Input Graph G and two matchings M_s, M_t of G.

Question Is there a reconfiguration sequence from M_s to M_t ?

Reconfiguration Problem: In Matchings

- Two variants - *reachability variant* and *shortest variant*

Reconfiguration of Matchings: Reachability Variant

Input: Graph G and two matchings M_s, M_t of G.

Question: Is there a reconfiguration sequence from M_s to M_t ?

Reconfiguration of Matchings: Shortest Variant

Input: Graph G and two matchings M_s, M_t of G.

Task: Compute the distance from M_s to M_t, where distance is the length of shortest reconfiguration sequence from M_s to M_t.

Reconfiguration Problem: In Matchings

- Two variants - *reachability variant* and *shortest variant*

Reconfiguration of Matchings: Reachability Variant

Input: Graph G and two matchings M_s, M_t of G.

Question: Is there a reconfiguration sequence from M_s to M_t ?

Reconfiguration of Matchings: Shortest Variant

Input: Graph G and two matchings M_s, M_t of G.

Task: Compute the distance from M_s to M_t, where distance is the length of shortest reconfiguration sequence from M_s to M_t.

- The reachability variant is solvable in polynomial time.

Reconfiguration Problem: In Matchings

- Two variants - *reachability variant* and *shortest variant*

Reconfiguration of Matchings: Reachability Variant

Input	Graph G and two matchings M_s, M_t of G.
Question	Is there a reconfiguration sequence from M_s to M_t ?

Reconfiguration of Matchings: Shortest Variant

Input	Graph G and two matchings M_s, M_t of G.
Task	Compute the distance from M_s to M_t, where distance is the length of shortest reconfiguration sequence from M_s to M_t.

- The reachability variant is solvable in polynomial time.
- This paper deals with the **shortest variant**.

Reconfiguration Problem: In Matchings

- Two variants - *reachability variant* and *shortest variant*

Reconfiguration of Matchings: Reachability Variant

Input Graph G and two matchings M_s, M_t of G.

Question Is there a reconfiguration sequence from M_s to M_t ?

Reconfiguration of Matchings: Shortest Variant

Input Graph G and two matchings M_s, M_t of G.

Task Compute the distance from M_s to M_t, where distance is the length of shortest reconfiguration sequence from M_s to M_t.

- The reachability variant is solvable in polynomial time.
- This paper deals with the **shortest variant**.
- This is an NP-hard problem.

Main Result of the paper

Theorem (Shorter Version)

Matching Distance in bipartite graphs is **fixed parameter tractable (FPT)**.

- Before looking into the complete version, we shall look into what fixed parameter tractability is.

Fixed Parameter Tractability

- Instead of expressing the running time as a function $T(n)$ of n, we express it as a function $T(n, k)$ of the input size n and some parameter k of the input.

Fixed Parameter Tractability

- Instead of expressing the running time as a function $T(n)$ of n, we express it as a function $T(n, k)$ of the input size n and some parameter k of the input.

Fixed Parameter Tractability

A problem is *fixed-parameter tractable* if there is an $f(k)n^c$ algorithm for some constant c and a parameter k.

Main Result of the paper

Theorem (Shorter Version)

Matching Distance in bipartite graphs is **fixed parameter tractable (FPT)**.

Main Result of the paper

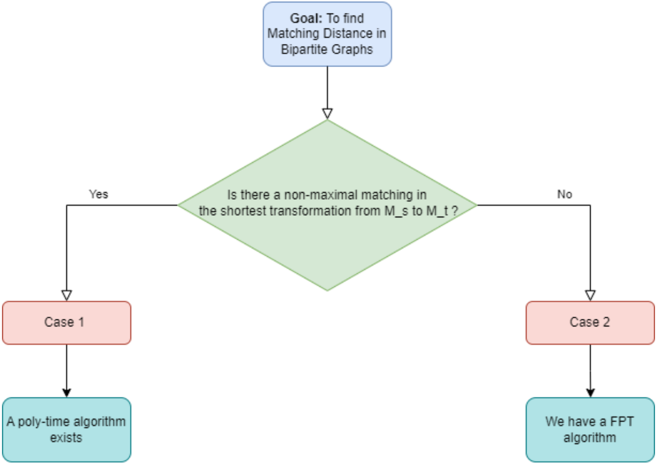
Theorem (Shorter Version)

Matching Distance in bipartite graphs is **fixed parameter tractable (FPT)**.

Theorem

Matching Distance in bipartite graphs can be solved in $2^d n^{\mathcal{O}(1)}$, where d is the size of the symmetric difference of the two given matchings.

Overview of the algorithm



Case-2 - Overview

- From the given Matching Distance instance, we construct an instance of Directed Steiner Tree (We will define it soon..).

Case-2 - Overview

- From the given Matching Distance instance, we construct an instance of Directed Steiner Tree (We will define it soon..).
- Directed Steiner Tree Problem is known to be FPT.

Case-2 - Overview

- From the given Matching Distance instance, we construct an instance of Directed Steiner Tree (We will define it soon..).
- Directed Steiner Tree Problem is known to be FPT.
- This will give us an FPT algorithm for matching distance in bipartite graphs.

The Directed Steiner Tree Problem

Directed Steiner Tree

Input Directed Graph $D = (V, A)$, integral arc weights c_a for each $a \in A$, root vertex $r \in V$, and terminals $T \subseteq V$

Task Find a minimum-cost directed tree in D that connects the root r to each terminal.

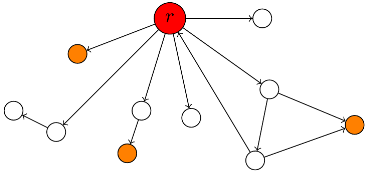
The Directed Steiner Tree Problem

Directed Steiner Tree

Input Directed Graph $D = (V, A)$, integral arc weights c_a for each $a \in A$, root vertex $r \in V$, and terminals $T \subseteq V$

Task Find a minimum-cost directed tree in D that connects the root r to each terminal.

An example:



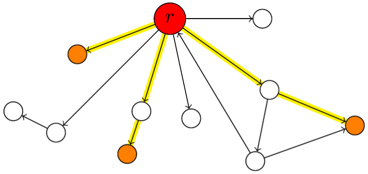
The Directed Steiner Tree Problem

Directed Steiner Tree

Input Directed Graph $D = (V, A)$, integral arc weights c_a for each $a \in A$, root vertex $r \in V$, and terminals $T \subseteq V$

Task Find a minimum-cost directed tree in D that connects the root r to each terminal.

An example:



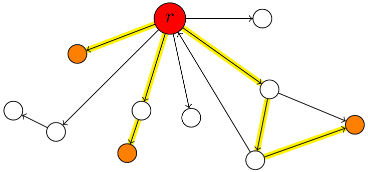
The Directed Steiner Tree Problem

Directed Steiner Tree

Input Directed Graph $D = (V, A)$, integral arc weights c_a for each $a \in A$, root vertex $r \in V$, and terminals $T \subseteq V$

Task Find a minimum-cost directed tree in D that connects the root r to each terminal.

An example:



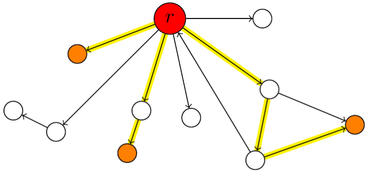
The Directed Steiner Tree Problem

Directed Steiner Tree

Input Directed Graph $D = (V, A)$, integral arc weights c_a for each $a \in A$, root vertex $r \in V$, and terminals $T \subseteq V$

Task Find a minimum-cost directed tree in D that connects the root r to each terminal.

An example:



Directed Steiner Tree has an algorithm which runs in $2^{|T|} n^{\mathcal{O}(1)} W$ where W is the maximum arc weight.

Reducing Matching Distance to DST

Input An instance $I := (G[U, W], M_s, M_t)$ of MATCHING DISTANCE where M_s, M_t are maximum matchings.

Reducing Matching Distance to DST

Input An instance $I := (G[U, W], M_s, M_t)$ of MATCHING DISTANCE where M_s, M_t are maximum matchings.

Goal We will convert I to an instance $I' := (D, c, r, T)$ of DIRECTED STEINER TREE such that given a Steiner Tree F for I', we can construct in polynomial time a transformation from M_s to M_t of cost at most $c(F)$.

Reducing Matching Distance to DST

Let X_s be the set of M_s-exposed vertices in G. We construct the digraph $D = (U', A)$ as follows.

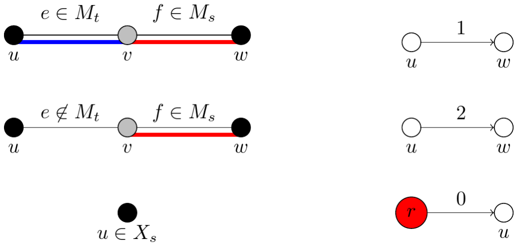
$U' := \mathcal{E}_G \cup \{r\}$, where r is a new vertex.

Reducing Matching Distance to DST

Let X_s be the set of M_s-exposed vertices in G. We construct the digraph $D = (U', A)$ as follows.

$U' := \mathcal{E}_G \cup \{r\}$, where r is a new vertex.

The arcs and arc weights of D are as follows.

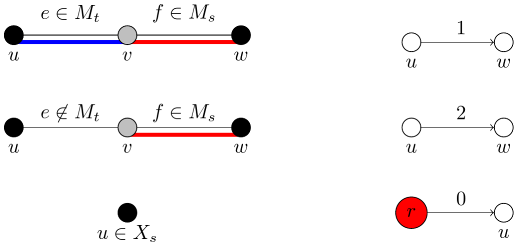


Reducing Matching Distance to DST

Let X_s be the set of M_s-exposed vertices in G. We construct the digraph $D = (U', A)$ as follows.

$U' := \mathcal{E}_G \cup \{r\}$, where r is a new vertex.

The arcs and arc weights of D are as follows.

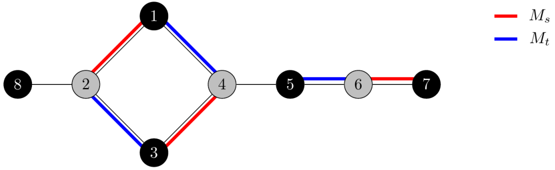


The terminals of the DIRECTED STEINER TREE instance are

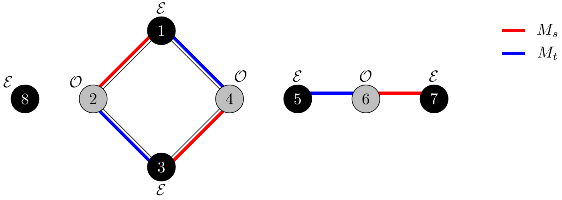
$$T := [\bigcup_{Z \in \mathcal{C}} V(Z) \bigcup_{Z \in \mathcal{P}} (V(Z) \setminus X_s)] \cap U'$$

where $\mathcal{P}$ and $\mathcal{C}$ are the paths and cycles respectively which form connected components of $M_s \oplus M_t$.

An example

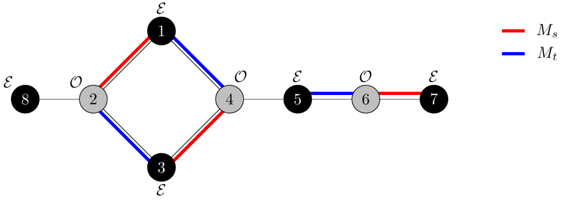


An example

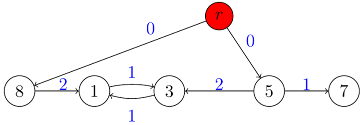


$\mathcal{E}$
1
$\mathcal{E}$
8
$\mathcal{O}$
2
$\mathcal{O}$
4
$\mathcal{E}$
5
$\mathcal{O}$
6
$\mathcal{E}$
7
3
$\mathcal{E}$
M_s
M_t

An example

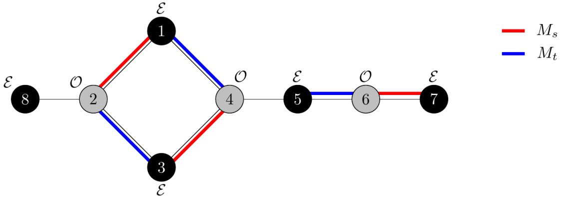


$\mathcal{E}$
1
$\mathcal{E}$
8
$\mathcal{O}$
2
$\mathcal{O}$
4
$\mathcal{E}$
5
$\mathcal{O}$
6
$\mathcal{E}$
7
3
$\mathcal{E}$
M_s
M_t

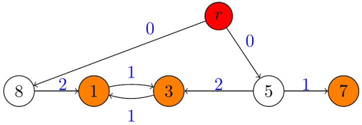


r
0
0
8
2
1
1
3
1
2
5
1
7

An example



$\mathcal{E}$
1
$\mathcal{E}$
8
$\mathcal{O}$
2
$\mathcal{O}$
4
$\mathcal{E}$
5
$\mathcal{O}$
6
$\mathcal{E}$
7
3
$\mathcal{E}$
M_s
M_t



r
0
0
8
2
1
1
3
2
5
1
7
1

We don't need to look at all Steiner trees!

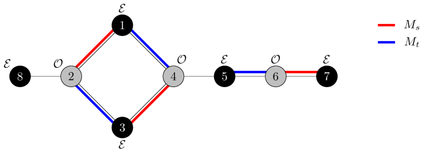
Proposition

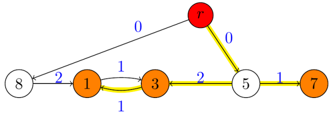
Given a Steiner Tree F for I', we can construct a Steiner Tree F' with $c(F') \leq c(F)$ which satisfies the following properties:

(i) For each $P \in \mathcal{P}$, F' contains all arcs of P.

(ii) For each $C \in \mathcal{C}$, F' misses exactly one arc of C.

(iii) For each $P \in \mathcal{P}$, r is joined to the M_s-exposed vertex of P.





The key idea

Lemma

Let F be a Steiner Tree for I'. Then we can construct in polynomial time a transformation from M_s to M_t of length at most $c(F)$.

The key idea

Lemma

Let F be a Steiner Tree for I'. Then we can construct in polynomial time a transformation from M_s to M_t of length at most $c(F)$.

- We can assume that F satisfies the properties listed in the previous proposition.

The key idea

Lemma

Let F be a Steiner Tree for I'. Then we can construct in polynomial time a transformation from M_s to M_t of length at most $c(F)$.

- We can assume that F satisfies the properties listed in the previous proposition.
- Perform DFS traversal of F giving preference to the largest weight arcs.

The key idea

Lemma

Let F be a Steiner Tree for I'. Then we can construct in polynomial time a transformation from M_s to M_t of length at most $c(F)$.

- We can assume that F satisfies the properties listed in the previous proposition.
- Perform DFS traversal of F giving preference to the largest weight arcs.
- Each arc of nonzero weight corresponds to a token.

The key idea

Lemma

Let F be a Steiner Tree for I'. Then we can construct in polynomial time a transformation from M_s to M_t of length at most $c(F)$.

- We can assume that F satisfies the properties listed in the previous proposition.
- Perform DFS traversal of F giving preference to the largest weight arcs.
- Each arc of nonzero weight corresponds to a token.
- When traversing down an arc of weight 1, move its token to target destination. No move when backtracking.

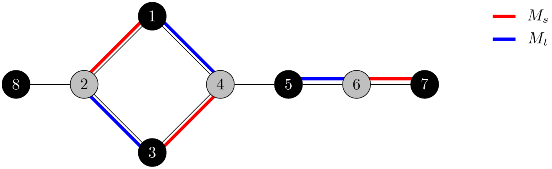
The key idea

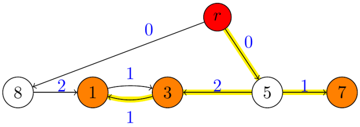
Lemma

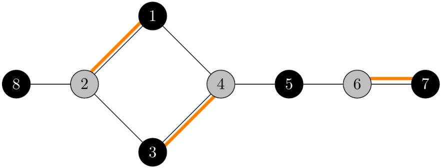
Let F be a Steiner Tree for I'. Then we can construct in polynomial time a transformation from M_s to M_t of length at most $c(F)$.

- We can assume that F satisfies the properties listed in the previous proposition.
- Perform DFS traversal of F giving preference to the largest weight arcs.
- Each arc of nonzero weight corresponds to a token.
- When traversing down an arc of weight 1, move its token to target destination. No move when backtracking.
- When traversing down an arc of weight 2, we move its token away from the target destination. When backtracking, move it to the target position.

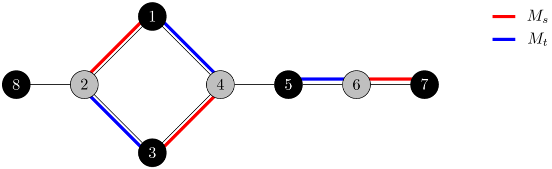
An example

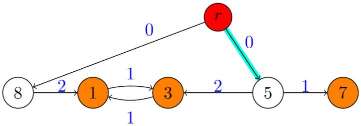


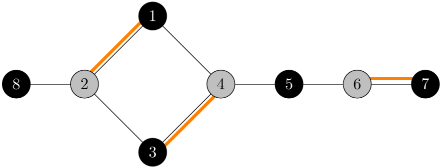




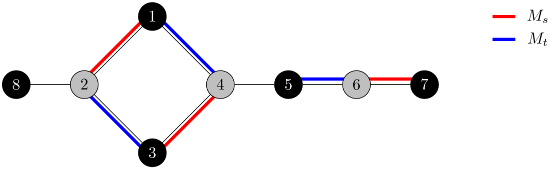
An example

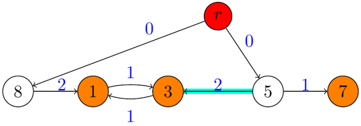


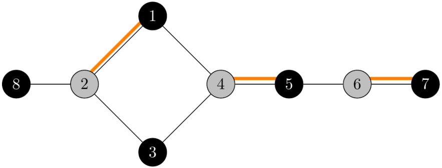




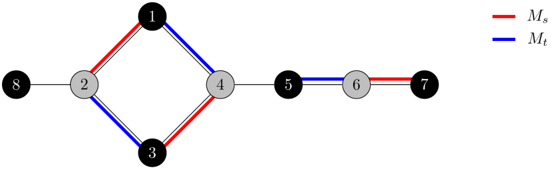
An example

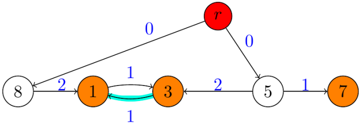


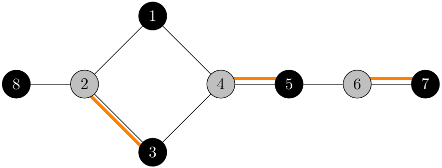




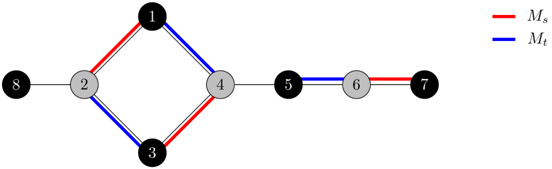
An example

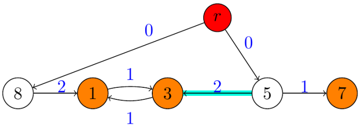


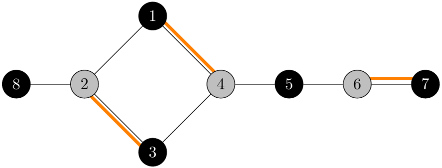




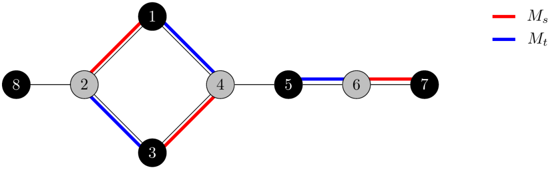
An example

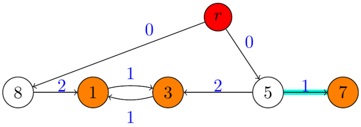


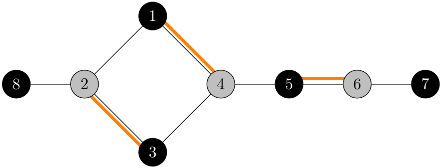




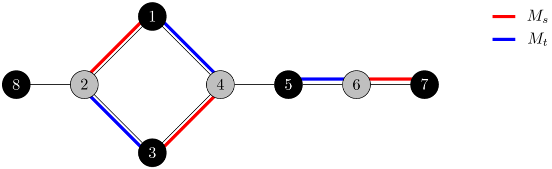
An example

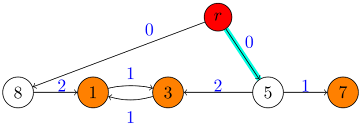


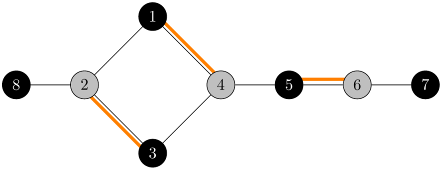




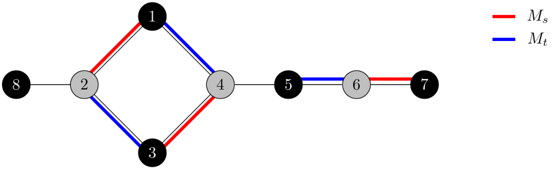
An example

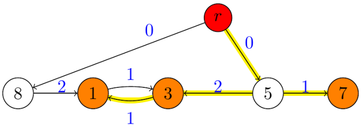


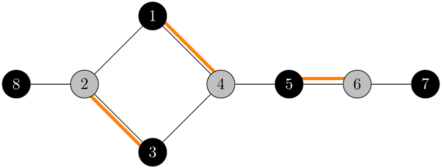




An example







A necessary and sufficient condition

Lemma

There is a transformation from M_s to M_t $\iff$ for each cycle $C \in \mathcal{C}$, there is an M_s-alternating path starting from an M_s-exposed vertex and ending at a vertex in C.

A necessary and sufficient condition

Lemma

There is a transformation from M_s to M_t $\iff$ for each cycle $C \in \mathcal{C}$, there is an M_s-alternating path starting from an M_s-exposed vertex and ending at a vertex in C.

Note: This is true for Case 2 in general, not just for maximum matchings.

A necessary and sufficient condition

Lemma

There is a transformation from M_s to M_t $\iff$ for each cycle $C \in \mathcal{C}$, there is an M_s-alternating path starting from an M_s-exposed vertex and ending at a vertex in C.

Note: This is true for Case 2 in general, not just for maximum matchings. For each cycle $C \in \mathcal{C}$, we have to slide the token along this M_s-alternating path and then keep moving the token to its target position.

The algorithm for Case 2

- $\mathcal{C}$ has at most $\frac{d}{4}$ cycles.

The algorithm for Case 2

- $\mathcal{C}$ has at most $\frac{d}{4}$ cycles.
- For each $C \in \mathcal{C}$, we have to use one of the M_s-alternating paths from the previous lemma.

The algorithm for Case 2

- $\mathcal{C}$ has at most $\frac{d}{4}$ cycles.
- For each $C \in \mathcal{C}$, we have to use one of the M_s-alternating paths from the previous lemma.
- For each cycle, we have at most two choices to reconfigure it - use an M-exposed vertex from U or from W.

The algorithm for Case 2

- $\mathcal{C}$ has at most $\frac{d}{4}$ cycles.
- For each $C \in \mathcal{C}$, we have to use one of the M_s-alternating paths from the previous lemma.
- For each cycle, we have at most two choices to reconfigure it - use an M-exposed vertex from U or from W.
- Branch over all of the at most $2^{\frac{d}{4}}$ choices.

The algorithm for Case 2

- $\mathcal{C}$ has at most $\frac{d}{4}$ cycles.
- For each $C \in \mathcal{C}$, we have to use one of the M_s-alternating paths from the previous lemma.
- For each cycle, we have at most two choices to reconfigure it - use an M-exposed vertex from U or from W.
- Branch over all of the at most $2^{\frac{d}{4}}$ choices.
- Create two sub-instances: one for the elements of $\mathcal{P} \cup \mathcal{C}$ to be reconfigured using a vertex from U and one for those that have to use a vertex from W.

The algorithm for Case 2

- $\mathcal{C}$ has at most $\frac{d}{4}$ cycles.
- For each $C \in \mathcal{C}$, we have to use one of the M_s-alternating paths from the previous lemma.
- For each cycle, we have at most two choices to reconfigure it - use an M-exposed vertex from U or from W.
- Branch over all of the at most $2^{\frac{d}{4}}$ choices.
- Create two sub-instances: one for the elements of $\mathcal{P} \cup \mathcal{C}$ to be reconfigured using a vertex from U and one for those that have to use a vertex from W.
- For the first sub-instance, delete all the exposed vertices in W. This gets rid of all the M_s-augmenting paths. Now modify the target matching appropriately to get a new matching distance instance for maximum matchings.

The algorithm for Case 2

- $\mathcal{C}$ has at most $\frac{d}{4}$ cycles.
- For each $C \in \mathcal{C}$, we have to use one of the M_s-alternating paths from the previous lemma.
- For each cycle, we have at most two choices to reconfigure it - use an M-exposed vertex from U or from W.
- Branch over all of the at most $2^{\frac{d}{4}}$ choices.
- Create two sub-instances: one for the elements of $\mathcal{P} \cup \mathcal{C}$ to be reconfigured using a vertex from U and one for those that have to use a vertex from W.
- For the first sub-instance, delete all the exposed vertices in W. This gets rid of all the M_s-augmenting paths. Now modify the target matching appropriately to get a new matching distance instance for maximum matchings.
- For the second sub-instance, delete the M_s-exposed vertices of W.

The algorithm for Case 2

- $\mathcal{C}$ has at most $\frac{d}{4}$ cycles.
- For each $C \in \mathcal{C}$, we have to use one of the M_s-alternating paths from the previous lemma.
- For each cycle, we have at most two choices to reconfigure it - use an M-exposed vertex from U or from W.
- Branch over all of the at most $2^{\frac{d}{4}}$ choices.
- Create two sub-instances: one for the elements of $\mathcal{P} \cup \mathcal{C}$ to be reconfigured using a vertex from U and one for those that have to use a vertex from W.
- For the first sub-instance, delete all the exposed vertices in W. This gets rid of all the M_s-augmenting paths. Now modify the target matching appropriately to get a new matching distance instance for maximum matchings.
- For the second sub-instance, delete the M_s-exposed vertices of W.
- Solve each sub-instance using the algorithm for maximum matchings and combine the optimal solutions of the two sub-instances.

Matching Distance in Bipartite Graphs is FPT

Lemma

Case 2 can be solved in $2^d n^{\mathcal{O}(1)}$.

Matching Distance in Bipartite Graphs is FPT

Lemma

Case 2 can be solved in $2^d n^{\mathcal{O}(1)}$.

Proof:

Matching Distance in Bipartite Graphs is FPT

Lemma

Case 2 can be solved in $2^d n^{\mathcal{O}(1)}$.

Proof:
There are at most $\frac{d}{2}$ terminals in the DIRECTED STEINER TREE instance obtained from the case where M_s is maximum.

Matching Distance in Bipartite Graphs is FPT

Lemma

Case 2 can be solved in $2^d n^{\mathcal{O}(1)}$.

Proof:
There are at most $\frac{d}{2}$ terminals in the DIRECTED STEINER TREE instance obtained from the case where M_s is maximum.
Hence it can be solved in time $2^{\frac{d}{2}} n^{\mathcal{O}(1)}$.

Matching Distance in Bipartite Graphs is FPT

Lemma

Case 2 can be solved in $2^d n^{\mathcal{O}(1)}$.

Proof:
There are at most $\frac{d}{2}$ terminals in the DIRECTED STEINER TREE instance obtained from the case where M_s is maximum.
Hence it can be solved in time $2^{\frac{d}{2}} n^{\mathcal{O}(1)}$.
For the general case 2, we branch over at most $2^{\frac{d}{4}}$ choices. Since each of these can be solved in $2^{\frac{d}{2}} n^{\mathcal{O}(1)}$, we can solve case 2 in time $2^{\frac{d}{4}} \times 2^{\frac{d}{2}} n^{\mathcal{O}(1)}$.
$\square$

Case 1 - Overview of the algorithm

- In this case, the shortest transformation visits a non-maximal matching.

Case 1 - Overview of the algorithm

- In this case, the shortest transformation visits a non-maximal matching.
- Such a transformation is possible only if M_s is not maximum.

Case 1 - Overview of the algorithm

- In this case, the shortest transformation visits a non-maximal matching.
- Such a transformation is possible only if M_s is not maximum.
- If one of M_s or M_t is non-maximal, the shortest transformation has length $\frac{d}{2}$ if $M_s \oplus M_t$ contains no cycles, or $\frac{d}{2} + 1$ otherwise.

Case 1 - Overview of the algorithm

- In this case, the shortest transformation visits a non-maximal matching.
- Such a transformation is possible only if M_s is not maximum.
- If one of M_s or M_t is non-maximal, the shortest transformation has length $\frac{d}{2}$ if $M_s \oplus M_t$ contains no cycles, or $\frac{d}{2} + 1$ otherwise.
- If M_s and M_t are maximal, we can transform M_s into a non-maximal matching by sliding tokens along an M_s-augmenting path.

Thank You !