Maximum Reachability Orientation of Mixed Graphs

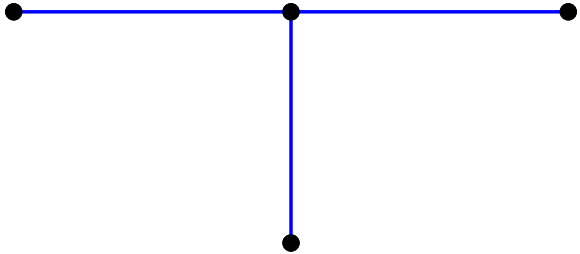
Florian Hörsch

Algorithms and Complexity Group, CISPA,
Saarbrücken, Germany

13th March, 2026

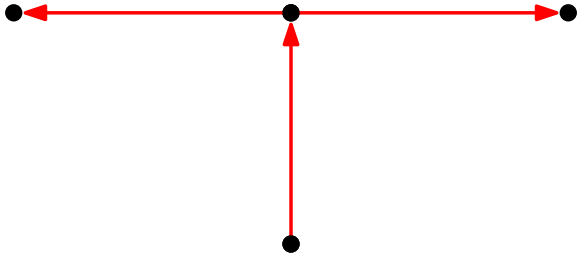
Basic Definitions

An orientation of a graph is a digraph obtained by assigning an orientation to every edge.



Basic Definitions

An orientation of a graph is a digraph obtained by assigning an orientation to every edge.



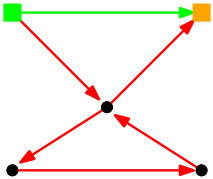
Basic definitions

Basic Definitions

An orientation of a graph is a digraph obtained by assigning an orientation to every edge.

Reachability in Digraphs

- we say that v is reachable from u if there is a directed path from u to v in D,



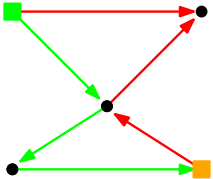
Basic definitions

Basic Definitions

An orientation of a graph is a digraph obtained by assigning an orientation to every edge.

Reachability in Digraphs

- we say that v is reachable from u if there is a directed path from u to v in D,



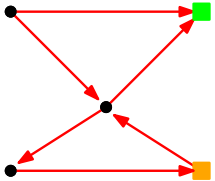
Basic definitions

Basic Definitions

An orientation of a graph is a digraph obtained by assigning an orientation to every edge.

Reachability in Digraphs

- we say that v is reachable from u if there is a directed path from u to v in D,



Basic definitions

Basic Definitions

An orientation of a graph is a digraph obtained by assigning an orientation to every edge.

Reachability in Digraphs

- we say that v is reachable from u if there is a directed path from u to v in D,
- we define $\kappa_D(u, v) = 1$ if v is reachable from u and $\kappa_D(u, v) = 0$, otherwise,

Basic definitions

Basic Definitions

An orientation of a graph is a digraph obtained by assigning an orientation to every edge.

Reachability in Digraphs

- we say that v is reachable from u if there is a directed path from u to v in D,
- we define $\kappa_D(u, v) = 1$ if v is reachable from u and $\kappa_D(u, v) = 0$, otherwise,
- we use $P(D)$ for the set of ordered pairs of distinct vertices,

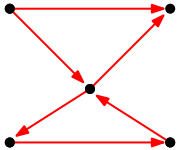
Basic definitions

Basic Definitions

An orientation of a graph is a digraph obtained by assigning an orientation to every edge.

Reachability in Digraphs

- we say that v is reachable from u if there is a directed path from u to v in D,
- we define $\kappa_D(u, v) = 1$ if v is reachable from u and $\kappa_D(u, v) = 0$, otherwise,
- we use $P(D)$ for the set of ordered pairs of distinct vertices,
- we use $R(D)$ for $\sum_{(u,v)\in P(D)} \kappa_D(u, v)$.



$R(D) = 13$

Basic definitions

Basic Definitions

An orientation of a graph is a digraph obtained by assigning an orientation to every edge.

Reachability in Digraphs

- we say that v is reachable from u if there is a directed path from u to v in D,
- we define $\kappa_D(u, v) = 1$ if v is reachable from u and $\kappa_D(u, v) = 0$, otherwise,
- we use $P(D)$ for the set of ordered pairs of distinct vertices,
- we use $R(D)$ for $\sum_{(u,v)\in P(D)} \kappa_D(u, v)$.

Maximum Reachability Orientation (MRO)

- Input: A graph G,
- Objective: Find an orientation $\vec{G}$ of G that maximizes $R(\vec{G})$.

Previous work

Theorem (Hakimi, Schmeichel, and Young; 1997)

MRO can be solved in polynomial time.

Previous work

Theorem (Hakimi, Schmeichel, and Young; 1997)

MRO can be solved in polynomial time.

Question (H,S,Y; 1997)

What if we already know the orientation of some edges?

Previous work

Theorem (Hakimi, Schmeichel, and Young; 1997)

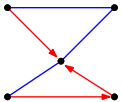
MRO can be solved in polynomial time.

Question (H,S,Y; 1997)

What if we already know the orientation of some edges?

Definitions

- a mixed graph G consists of $V(G)$, an edge set $E(G)$ and an arc set $A(G)$,



Previous work

Theorem (Hakimi, Schmeichel, and Young; 1997)

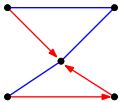
MRO can be solved in polynomial time.

Question (H,S,Y; 1997)

What if we already know the orientation of some edges?

Definitions

- a mixed graph G consists of $V(G)$, an edge set $E(G)$ and an arc set $A(G)$,
- an orientation of a mixed graph is a digraph obtained by assigning an orientation to every edge.



Previous work

Theorem (Hakimi, Schmeichel, and Young; 1997)

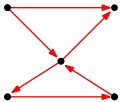
MRO can be solved in polynomial time.

Question (H,S,Y; 1997)

What if we already know the orientation of some edges?

Definitions

- a mixed graph G consists of $V(G)$, an edge set $E(G)$ and an arc set $A(G)$,
- an orientation of a mixed graph is a digraph obtained by assigning an orientation to every edge.



New results for mixed graphs

Mixed Maximum Reachability Orientation (MMRO)

- Input: A mixed graph G,
- Objective: Find an orientation $\vec{G}$ of G that maximizes $R(\vec{G})$.

New results for mixed graphs

Mixed Maximum Reachability Orientation (MMRO)

- Input: A mixed graph G,
- Objective: Find an orientation $\vec{G}$ of G that maximizes $R(\vec{G})$.

Theorem

MMRO is NP-complete.

New results for mixed graphs

Mixed Maximum Reachability Orientation (MMRO)

- Input: A mixed graph G,
- Objective: Find an orientation $\vec{G}$ of G that maximizes $R(\vec{G})$.

Theorem

MMRO is NP-complete.

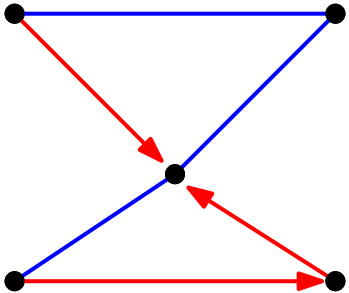
Theorem

Unless $P = NP$, there is no polynomial-time $\frac{581773}{581774}$-approximation algorithm for MMRO.

Small number of arcs

Definition

Given an instance G of MMRO, we use k for $|A(G)|$.



$k = 3$

Small number of arcs

Definition

Given an instance G of MMRO, we use k for $|A(G)|$.

Theorem

MMRO can be solved in $n^{O(k)}$.

Small number of arcs

Definition

Given an instance G of MMRO, we use k for $|A(G)|$.

Theorem

MMRO can be solved in $n^{O(k)}$.

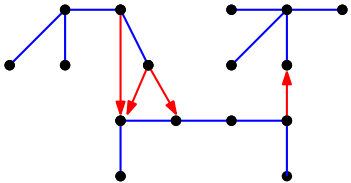
Theorem

For every $\epsilon > 0$, a $(1-\epsilon)$-approximation for MMRO can be computed in $f(k,\epsilon)n^{O(1)}$.

Preprocessing

Definition

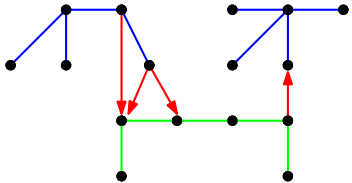
An undirected component of G is a component of $G \setminus A(G)$.



Preprocessing

Definition

An undirected component of G is a component of $G \setminus A(G)$.



Preprocessing

Definition

An undirected component of G is a component of $G \setminus A(G)$.

Preprocessed instances

We simplify so that undirected components

Preprocessing

Definition

An undirected component of G is a component of $G \setminus A(G)$.

Preprocessed instances

We simplify so that undirected components

- are trees (contract cycles),

Preprocessing

Definition

An undirected component of G is a component of $G \setminus A(G)$.

Preprocessed instances

We simplify so that undirected components

- are trees (contract cycles),
- contain at most two vertices incident to arcs (guess certain edges).

Algorithm for one Undirected Component

Objective

Want a small number of orientations one of which is always good.

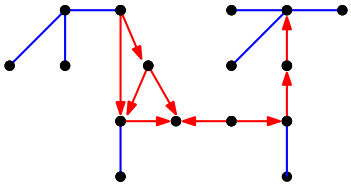
Algorithm for one Undirected Component

Objective

Want a small number of orientations one of which is always good.

Algorithm Sketch

- take a simplified instance and one undirected component,



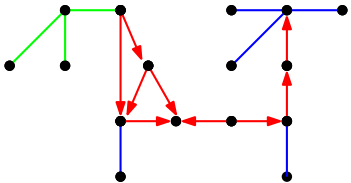
Algorithm for one Undirected Component

Objective

Want a small number of orientations one of which is always good.

Algorithm Sketch

- take a simplified instance and one undirected component,



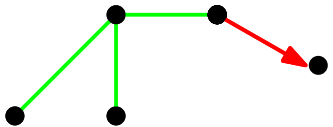
Algorithm for one Undirected Component

Objective

Want a small number of orientations one of which is always good.

Algorithm Sketch

- take a simplified instance and one undirected component,
- create gadget for influence of the remainder,



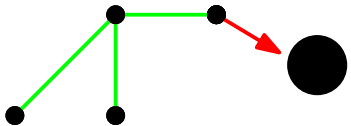
Algorithm for one Undirected Component

Objective

Want a small number of orientations one of which is always good.

Algorithm Sketch

- take a simplified instance and one undirected component,
- create gadget for influence of the remainder,
- create small number of simple instances,



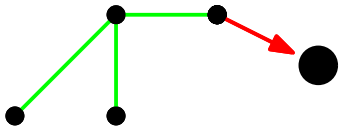
Algorithm for one Undirected Component

Objective

Want a small number of orientations one of which is always good.

Algorithm Sketch

- take a simplified instance and one undirected component,
- create gadget for influence of the remainder,
- create small number of simple instances,



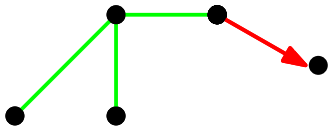
Algorithm for one Undirected Component

Objective

Want a small number of orientations one of which is always good.

Algorithm Sketch

- take a simplified instance and one undirected component,
- create gadget for influence of the remainder,
- create small number of simple instances,



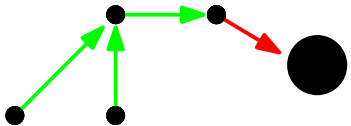
Algorithm for one Undirected Component

Objective

Want a small number of orientations one of which is always good.

Algorithm Sketch

- take a simplified instance and one undirected component,
- create gadget for influence of the remainder,
- create small number of simple instances,
- solve these instances and store orientations.



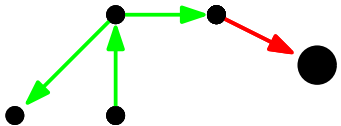
Algorithm for one Undirected Component

Objective

Want a small number of orientations one of which is always good.

Algorithm Sketch

- take a simplified instance and one undirected component,
- create gadget for influence of the remainder,
- create small number of simple instances,
- solve these instances and store orientations.



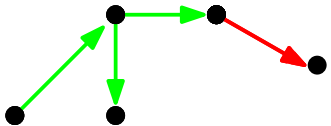
Algorithm for one Undirected Component

Objective

Want a small number of orientations one of which is always good.

Algorithm Sketch

- take a simplified instance and one undirected component,
- create gadget for influence of the remainder,
- create small number of simple instances,
- solve these instances and store orientations.



Final Algorithm

Objective

Compute optimal orientation overall.

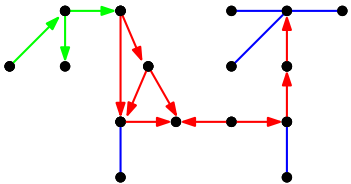
Final Algorithm

Objective

Compute optimal orientation overall.

Algorithm Sketch

- take a simplified instance,



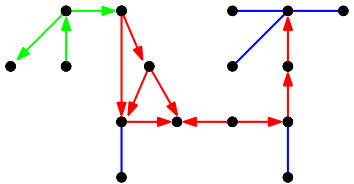
Final Algorithm

Objective

Compute optimal orientation overall.

Algorithm Sketch

- take a simplified instance,
- assume orientations for all undirected components are available,



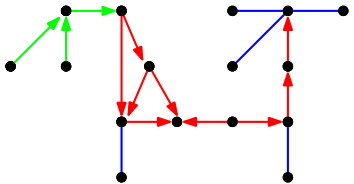
Final Algorithm

Objective

Compute optimal orientation overall.

Algorithm Sketch

- take a simplified instance,
- assume orientations for all undirected components are available,



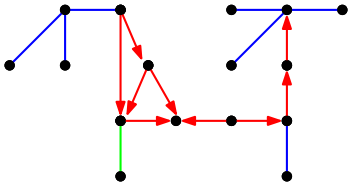
Final Algorithm

Objective

Compute optimal orientation overall.

Algorithm Sketch

- take a simplified instance,
- assume orientations for all undirected components are available,



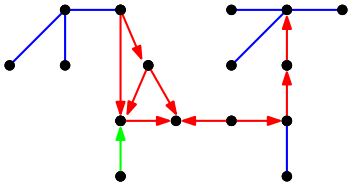
Final Algorithm

Objective

Compute optimal orientation overall.

Algorithm Sketch

- take a simplified instance,
- assume orientations for all undirected components are available,



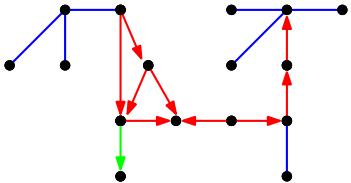
Final Algorithm

Objective

Compute optimal orientation overall.

Algorithm Sketch

- take a simplified instance,
- assume orientations for all undirected components are available,



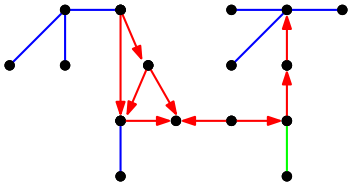
Final Algorithm

Objective

Compute optimal orientation overall.

Algorithm Sketch

- take a simplified instance,
- assume orientations for all undirected components are available,



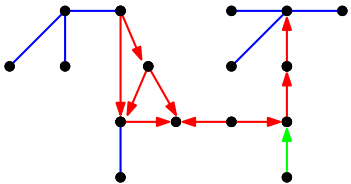
Final Algorithm

Objective

Compute optimal orientation overall.

Algorithm Sketch

- take a simplified instance,
- assume orientations for all undirected components are available,



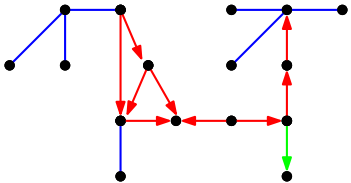
Final Algorithm

Objective

Compute optimal orientation overall.

Algorithm Sketch

- take a simplified instance,
- assume orientations for all undirected components are available,



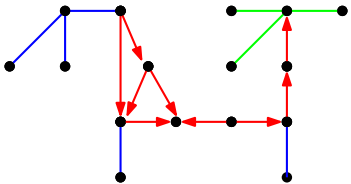
Final Algorithm

Objective

Compute optimal orientation overall.

Algorithm Sketch

- take a simplified instance,
- assume orientations for all undirected components are available,



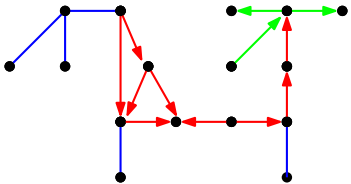
Final Algorithm

Objective

Compute optimal orientation overall.

Algorithm Sketch

- take a simplified instance,
- assume orientations for all undirected components are available,



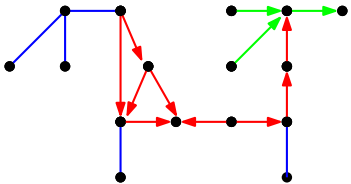
Final Algorithm

Objective

Compute optimal orientation overall.

Algorithm Sketch

- take a simplified instance,
- assume orientations for all undirected components are available,



Final Algorithm

Objective

Compute optimal orientation overall.

Algorithm Sketch

- take a simplified instance,
- assume orientations for all undirected components are available,
- try all possible combinations,

Final Algorithm

Objective

Compute optimal orientation overall.

Algorithm Sketch

- take a simplified instance,
- assume orientations for all undirected components are available,
- try all possible combinations,
- keep the best one.

Conclusion

Results

- NP-hardness,

Conclusion

Results

- NP-hardness,
- APX-hardness,

Conclusion

Results

- NP-hardness,
- APX-hardness,
- solution in $n^{O(k)}$,

Conclusion

Results

- NP-hardness,
- APX-hardness,
- solution in $n^{O(k)}$,
- $(1-\epsilon)$-approximation in $f(k,\epsilon)n^{O(1)}$.

Conclusion

Results

- NP-hardness,
- APX-hardness,
- solution in $n^{O(k)}$,
- $(1-\epsilon)$-approximation in $f(k,\epsilon)n^{O(1)}$.

Open problems

- constant factor approximation in polynomial time,

Conclusion

Results

- NP-hardness,
- APX-hardness,
- solution in $n^{O(k)}$,
- $(1-\epsilon)$-approximation in $f(k,\epsilon)n^{O(1)}$.

Open problems

- constant factor approximation in polynomial time,
- FPT algorithm (i.e. $f(k)n^{O(1)}$),

Conclusion

Results

- NP-hardness,
- APX-hardness,
- solution in $n^{O(k)}$,
- $(1-\epsilon)$-approximation in $f(k,\epsilon)n^{O(1)}$.

Open problems

- constant factor approximation in polynomial time,
- FPT algorithm (i.e. $f(k)n^{O(1)}$),
- similar parameterization for other problems.