A polynomial time algorithm for the k-disjoint shortest paths problem.

William Lochet

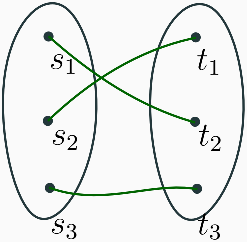
University of Bergen, Norway

Disjoint paths problem and optimization version

Disjoint paths problem

Let G be a graph, $S = (s_1, \ldots, s_k)$ and $T = (t_1, \ldots, t_k)$ two sets of vertices.

Question
Does there exist k disjoint paths $P_1, \ldots, P_k$ linking S and T?

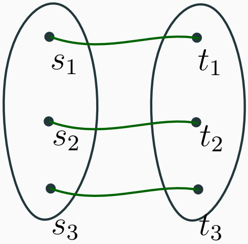


- Without constraints on the extremity: flow.

Disjoint paths problem

Let G be a graph, $S = (s_1, \dots, s_k)$ and $T = (t_1, \dots, t_k)$ two sets of vertices.

Question
Does there exist k disjoint paths $P_1, \dots, P_k$ linking S and T?



- Without constraints on the extremity: flow.
- If each P_i must link s_i to t_i: **k-disjoint paths problem**.

Result and hardness

Theorem (Robertson and Seymour 1995)
The k-disjoint paths problem admits an algorithm in $f(k)n^3$.

- Allows to decide the existence of a **minor**.
- Has been extensively studied since then.

Result and hardness

Theorem (Robertson and Seymour 1995)
The k-disjoint paths problem admits an algorithm in $f(k)n^3$.

- Allows to decide the existence of a **minor**.
- Has been extensively studied since then.

Theorem (Fortune et al. 1980)
The 2-disjoint paths problem in directed graphs in NP-hard.

- There exists an $n^{O(k)}$ algorithm on DAGs (Fortune et al.) .
- W[1]-hard on DAGs (Slivkins. 2010)

Optimal version

Question
Can we find a solution which ***minimises*** *the size of the paths?*

Optimal version

Question
Can we find a solution which ***minimises*** *the size of the paths?*

- Random algorithm for $k = 2$ (Björklund, Husfeldt 2015).
- Some results in the planar case.
- The case $k = 3$ is open.

Optimal version

Question
*Can we find a solution which **minimises** the size of the paths?*

- Random algorithm for $k = 2$ (Björklund, Husfeldt 2015).
- Some results in the planar case.
- The case $k = 3$ is open.

Problem (k-disjoint shortest path problem)
*Can we find a solution where each path P_i between s_i and t_i is a **shortest path**?*

Optimal version

Question
*Can we find a solution which **minimises** the size of the paths?*

- Random algorithm for $k = 2$ (Björklund, Husfeldt 2015).
- Some results in the planar case.
- The case $k = 3$ is open.

Problem (k-disjoint shortest path problem)
*Can we find a solution where each path P_i between s_i and t_i is a **shortest path**?*

- Problem posed by Eilam-Tzoreff in 1998.
- She proved the case $k = 2$ has a polynomial algorithm.
- The case $k \geq 3$ was open until now.

Recent interest and results

Theorem (Bérczi, Kobayashi 2017)
The directed 2 disjoint shortest paths problem admits a polynomial time algorithm.

Recent interest and results

Theorem (Bérczi, Kobayashi 2017)
The directed 2 disjoint shortest paths problem admits a polynomial time algorithm.

- Also show that it exists for planar graphs.
- A lot of extensions of this result (non negative weights, slightly longer than shortest etc.)

Recent interest and results

Theorem (Bérczi, Kobayashi 2017)
The directed 2 disjoint shortest paths problem admits a polynomial time algorithm.

- Also show that it exists for planar graphs.
- A lot of extensions of this result (non negative weights, slightly longer than shortest etc.)

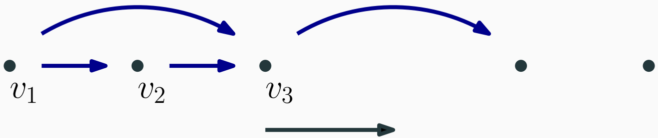
Theorem (L. 2021)
The k disjoint shortest path problem admits an $n^{f(k)}$ algorithm.

Works for both edge and vertex-disjoint version. The talk focus on **edge-disjoint**.

W[1]-hardness

Theorem
The k-DSP (directed or not) is W[1]-hard (no $f(k)n^{O(1)}$ algorithm).

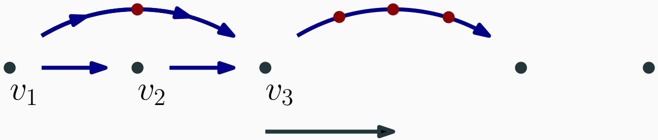
- Let $(D, (s_i, t_i)_{i \in [k]})$ be an instance of k-DP on DAGs and $v_1, \ldots v_n$ the topological order of D.
- By subdividing each arc (v_i, v_j) $j - i - 1$ times, every directed path is a shortest path.



W[1]-hardness

Theorem
The k-DSP (directed or not) is W[1]-hard (no $f(k)n^{O(1)}$ algorithm).

- Let $(D, (s_i, t_i)_{i \in [k]})$ be an instance of k-DP on DAGs and $v_1, \ldots v_n$ the topological order of D.
- By subdividing each arc (v_i, v_j) $j - i - 1$ times, every directed path is a shortest path.



General ideas

BFS and shortest path

Consider a BFS starting from s_1:

- It defines levels $L_1, \ldots, L_r$

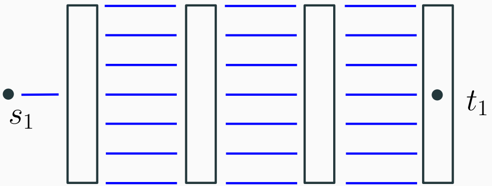
BFS and shortest path

Consider a BFS starting from s_1:

- It defines levels $L_1, \ldots, L_r$
- The shortest paths from s_1 "follow" the levels

BFS and shortest path

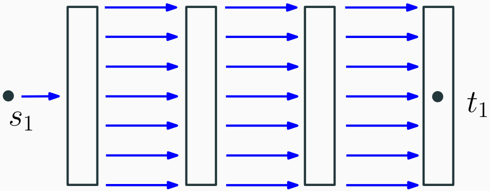
Consider a BFS starting from s_1:



- It defines levels $L_1, \ldots, L_r$
- The shortest paths from s_1 "follow" the levels
- We will call the edges between levels blue

BFS and shortest path

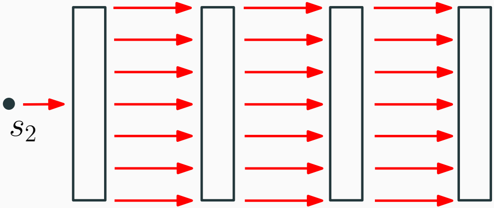
Consider a BFS starting from s_1:



- It defines levels $L_1, \ldots, L_r$
- The shortest paths from s_1 "follow" the levels
- We will call the edges between levels blue
- It defines also an **orientation** on these edges
- Paths in this digraph will be called blue paths

Coloured disjoint paths

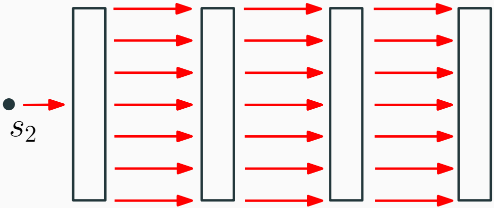
We can define red edges similarly by doing a BFS from s_2:



One edge can be of **both colours** (and oriented differently)

Coloured disjoint paths

We can define red edges similarly by doing a BFS from s_2:



One edge can be of **both colours** (and oriented differently)

Definition (k-coloured Graph)

A k-coloured graph is a graph G as well as k colours on some edges obtained by doing k BFS.

Multi-coloured graph

Problem ((k, l)-DSP)
Let G be a k-coloured graph, $(s_1, t_1), \dots (s_l, t_l)$ l pairs of vertices of G and $c : [l] \to [k]$. Does there exists a set of disjoint paths $P_1, \dots, P_l$ such that for every i:

- *P_i is a path of colour $c(i)$ from s_i to t_i*

Multi-coloured graph

Problem ((k, l)-DSP)
Let G be a k-coloured graph, $(s_1, t_1), \ldots (s_l, t_l)$ l pairs of vertices of G and $c : [l] \to [k]$. Does there exists a set of disjoint paths $P_1, \ldots, P_l$ such that for every i:

- *P_i is a path of colour $c(i)$ from s_i to t_i*

Multi-coloured graph

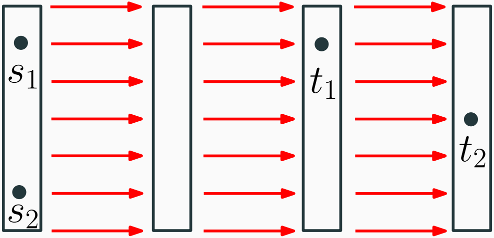
Problem ((k,l)-DSP)
Let G be a k-coloured graph, $(s_1, t_1), \dots (s_l, t_l)$ l pairs of vertices of G and $c : [l] \to [k]$. Does there exists a set of disjoint paths $P_1, \dots, P_l$ such that for every i:

- *P_i is a path of colour $c(i)$ from s_i to t_i*

We show by **induction on k** that (k,l)-DSP admits a algorithm in $n^{f(k,l)}$

Case $k = 1$

The case $k = 1$ corresponds to the disjoint-paths problem in **directed acyclic graphs**.



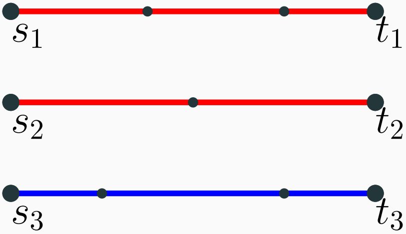
Theorem (Fortune et al. 1980)
*ℓ-disjoint-paths on **DAGs** admits an $n^{O(\ell)}$ algorithm.*

Algorithm

The algorithm is then a generalization of Fortune et al. algorithm for DAGs.

Main Idea

Guess some intermediate points on the solution paths such that paths of different colours **cannot intersect**.

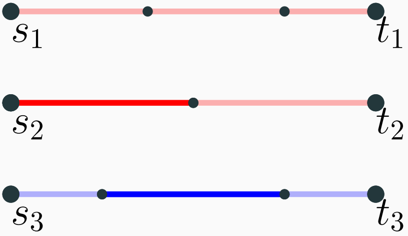


Algorithm

The algorithm is then a generalization of Fortune et al. algorithm for DAGs.

Main Idea

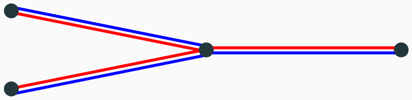
Guess some intermediate points on the solution paths such that paths of different colours **cannot intersect**.



Bi-coloured components

Definition

- G^+: graph of red/blue edges with the **same orientation**. G^-: **opposit** one.
- A set of vertices is said to be a **bi-colored** component if it is a connected component of either G^+ or G^-.



Some remarks

- It is the main ingredient in Bérczi and Kobayashi's proof
- As the role of s_i and t_i are symmetrical, G^+ and G^- have the same properties

Some remarks

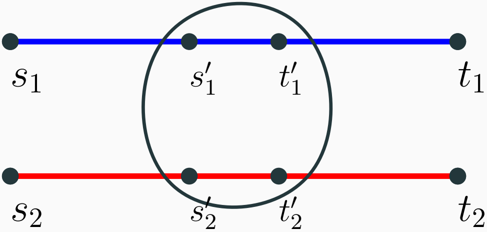
- It is the main ingredient in Bérczi and Kobayashi's proof
- As the role of s_i and t_i are symmetrical, G^+ and G^- have the same properties

Lemma
If P is a blue path and C_i a bi-coloured component, $P \cap C_i$ is a ***sub-path*** *of P.*

Main lemma

Lemma

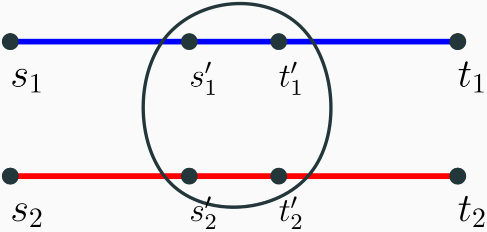
Let P_1 *and* P_2 *be two shortest paths. There exists a finite number of* ***bi-coloured components*** *such that* P_1 *and* P_2 *cannot intersect outside these components.*



Main lemma

Lemma

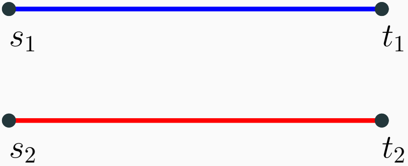
Let P_1 and P_2 be two shortest paths. There exists a finite number of ***bi-coloured components*** *such that P_1 and P_2 cannot intersect outside these components.*



- One is enough (Bentert et al. 2020+)

Case $k = \ell = 2$

- Let G, (s_1, t_1), (s_2, t_2) be an instance of $(2, 2)$-DSP

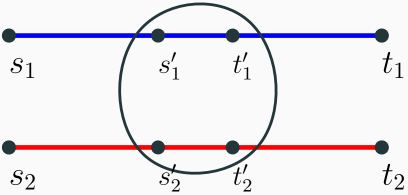


The algorithm then:

- Guess (s_i', t_i') for $i \in [2]$. (n^4 possible choices)

Case $k = \ell = 2$

- Let G, (s_1, t_1), (s_2, t_2) be an instance of $(2,2)$-DSP

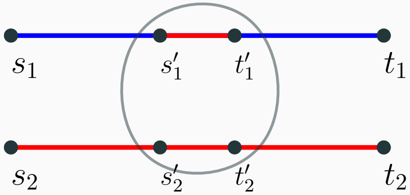


The algorithm then:

- Guess (s'_i, t'_i) for $i \in [2]$. (n^4 possible choices)
- Consider s'_1, t'_1 as red.

Case $k = \ell = 2$

- Let G, (s_1, t_1), (s_2, t_2) be an instance of $(2, 2)$-DSP

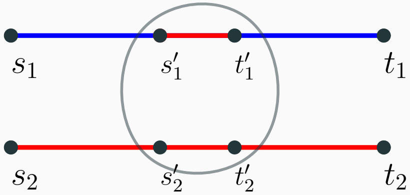


The algorithm then:

- Guess (s'_i, t'_i) for $i \in [2]$. (n^4 possible choices)
- Consider s'_1, t'_1 as red.
- Solve this new instance where paths of different colours **cannot** intersect.

Case $k = \ell = 2$

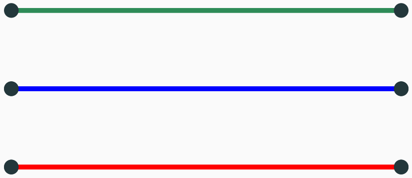
- Let G, (s_1, t_1), (s_2, t_2) be an instance of $(2, 2)$-DSP



The algorithm then:

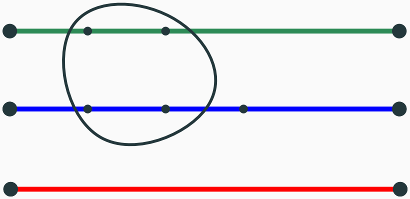
- Guess (s'_i, t'_i) for $i \in [2]$. (n^4 possible choices)
- Consider s'_1, t'_1 as red.
- Solve this new instance where paths of different colours **cannot** intersect.

Induction step



In the case with 3 colours:

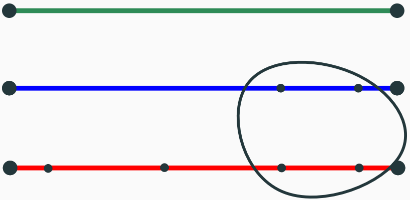
Induction step



In the case with 3 colours:

- Guess the components for blue/ red and blue/ green

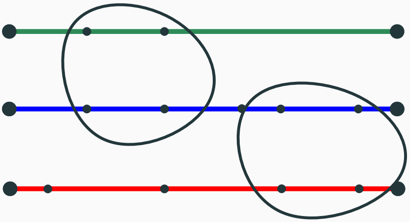
Induction step



In the case with 3 colours:

- Guess the components for blue/ red and blue/ green

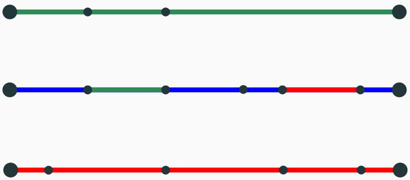
Induction step



In the case with 3 colours:

- Guess the components for blue/ red and blue/ green
- Take the intersections and change the colour of the blue paths on the bi-colored components

Induction step

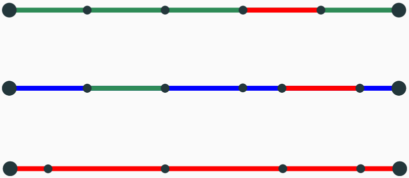


In the case with 3 colours:

- Guess the components for blue/ red and blue/ green
- Take the intersections and change the colour of the blue paths on the bi-colored components
- The blue paths remaining do not **intersect** with paths of other colours.

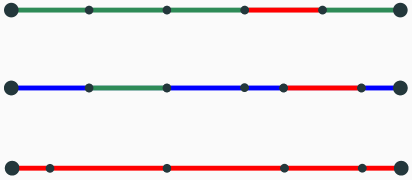
Final step

There is a **decomposition** of each path into $f(k, l)$ paths (with changes of colour) s.t each pair of paths of different colours cannot intersect.



Final step

There is a **decomposition** of each path into $f(k,l)$ paths (with changes of colour) s.t each pair of paths of different colours cannot intersect.



The algorithm consists then of:

1. Try all decompositions and colours ($n^{f(k,l)}$ tries)
2. Solve each colour using Fortune's type algorithm in $n^{f(k,l)}$

Proof of the main lemma

Role of bi-coloured components

Lemma
Let P_1 and P_2 be two shortest paths. There exists a finite number of ***bi-coloured components*** *such that P_1 and P_2 cannot intersect outside these components.*

Role of bi-coloured components

Lemma
*Let P_1 and P_2 be two shortest paths. There exists a finite number of **bi-coloured components** such that P_1 and P_2 cannot intersect outside these components.*

- We will start with the following lemma:

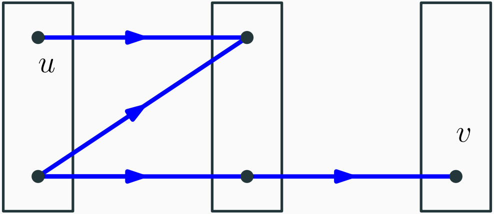
Lemma
*If P is a blue path and C_i a bi-coloured component, $P \cap C_i$ is a **sub-path** of P.*

Remember that G^+ and G^- are symmetrical, so we assume that C is a component of G^+.

Properties of bi-coloured components

Lemma
Let C be a component of G^+, then the two BFS induce an identical partition on C.

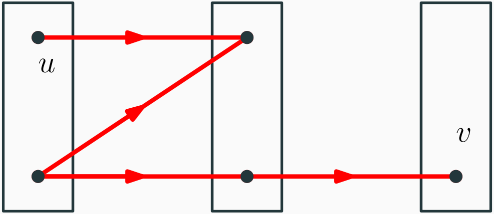
For $u, v \in C$, the difference of levels are the same in both colour.



Properties of bi-coloured components

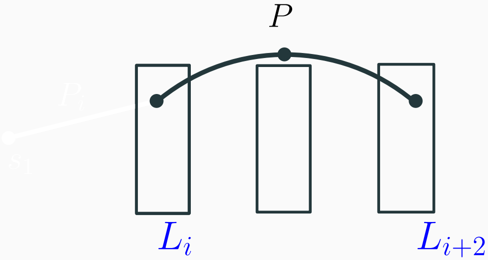
Lemma
Let C be a component of G^+, then the two BFS induce an identical partition on C.

For $u, v \in C$, the difference of levels are the same in both colour.



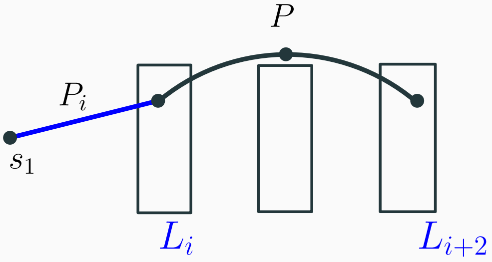
Lemma

If $x \in L_i$, $y \in L_{i+t}$ and there is a path P of length t between x and y in G, then P is a blue path.



Some properties of blue paths

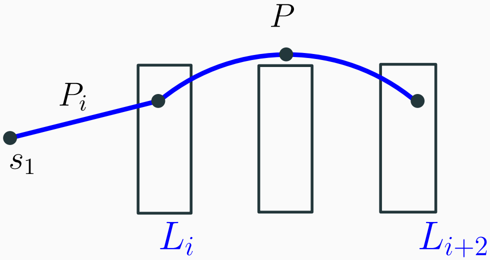
Lemma
If $x \in L_i$, $y \in L_{i+t}$ and there is a path P of length t between x and y in G, then P is a blue path.



- Consider P_i of length i from s_1 to x.

Some properties of blue paths

Lemma
If $x \in L_i$, $y \in L_{i+t}$ and there is a path P of length t between x and y in G, then P is a blue path.

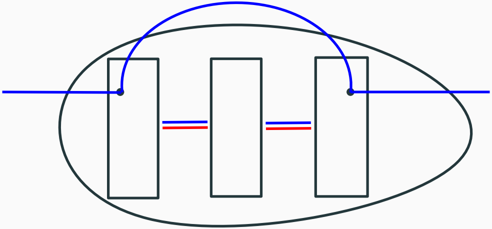


- Consider P_i of length i from s_1 to x.
- $P' = P_i \odot P$ is a path of length $i + t$ from s_1 to t and so is a blue path.

Shortest path and components

Lemma
If P is a blue path and C_i a bi-coloured component, $P \cap C_i$ is a ***sub-path*** *of P.*

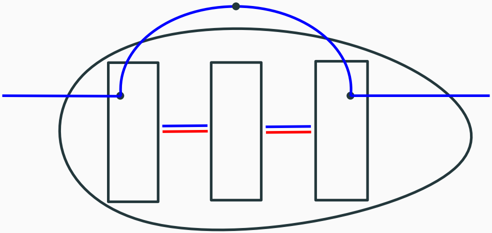
Suppose this is not true:



Shortest path and components

Lemma
*If P is a blue path and C_i a bi-coloured component, $P \cap C_i$ is a **sub-path** of P.*

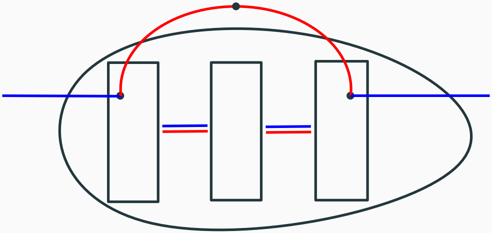
Suppose this is not true:



Shortest path and components

Lemma
If P is a blue path and C_i a bi-coloured component, $P \cap C_i$ is a ***sub-path*** *of P.*

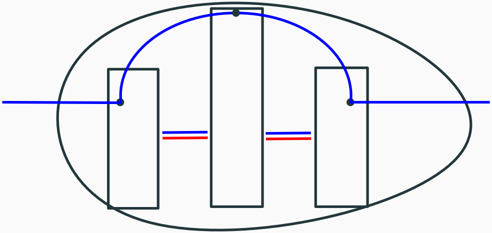
Suppose this is not true:



Shortest path and components

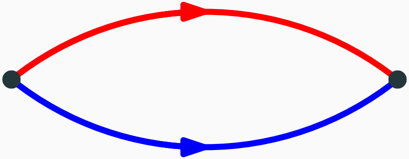
Lemma
If P is a blue path and C_i a bi-coloured component, $P \cap C_i$ is a ***sub-path*** *of P.*

Suppose this is not true:



Some properties of bi-coloured paths

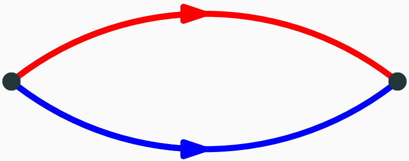
Lemma
If P_1 is a red (x, y)-path and P_2 is a blue (x, y)-path, then P_1 and P_2 are both red and blue paths.



- By definition of blue/red paths: $|P_1| = |P_2|$.

Some properties of bi-coloured paths

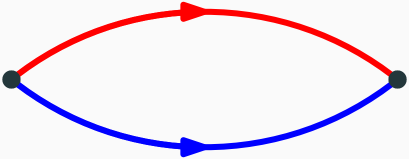
Lemma
If P_1 is a red (x, y)-path and P_2 is a blue (x, y)-path, then P_1 and P_2 are both red and blue paths.



- By definition of blue/red paths: $|P_1| = |P_2|$.
- If $x \in L_i$ and $y \in L_{i+t}$, then $|P_2| = t$
- $|P_1|$ is equal to the difference of blue levels between x and y.

Some properties of bi-coloured paths

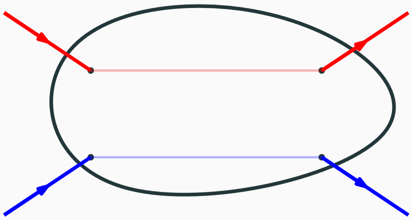
Lemma
If P_1 is a red (x, y)-path and P_2 is a blue (x, y)-path, then P_1 and P_2 are both red and blue paths.



- By definition of blue/red paths: $|P_1| = |P_2|$.
- If $x \in L_i$ and $y \in L_{i+t}$, then $|P_2| = t$
- $|P_1|$ is equal to the difference of blue levels between x and y.
- P_1 is blue by previous lemma.

Conflict component

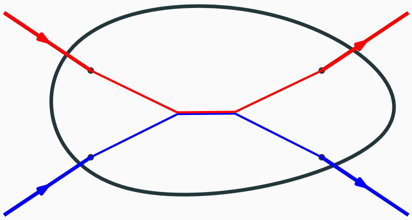
Definition
Let P_1, P_2 be two (possibly intersecting) paths and C a bi-coloured component. We say that P_1 and P_2 are in **conflict** on C if $C \cap P_1$ and $C \cap P_2$ can be replaced by intersecting paths.



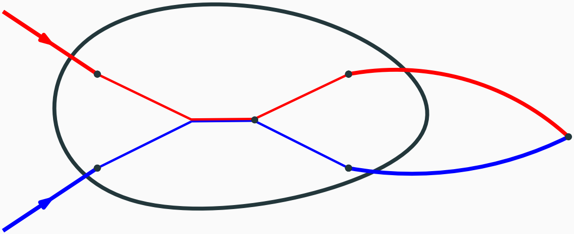
Conflict component

Definition

Let P_1, P_2 be two (possibly intersecting) paths and C a bi-coloured component. We say that P_1 and P_2 are in **conflict** on C if $C \cap P_1$ and $C \cap P_2$ can be replaced by intersecting paths.

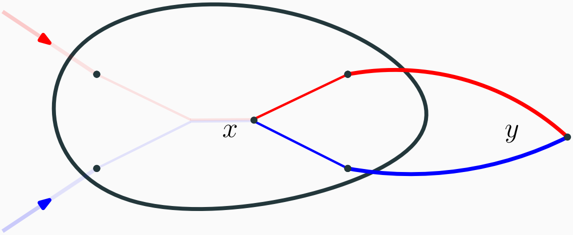


Lemma
Let P_1 and P_2 be two paths such that C is a conflicting component, then $P_1 \cap P_2$ is empty outside of C.



Conflict component and intersection

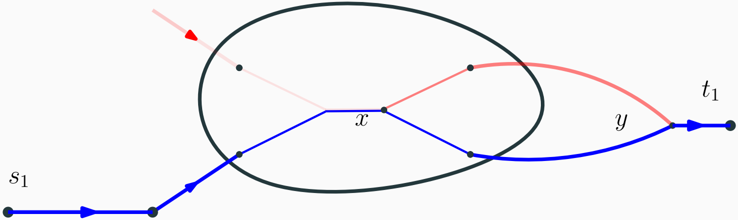
Lemma
Let P_1 and P_2 be two paths such that C is a conflicting component, then $P_1 \cap P_2$ is empty outside of C.



- Both paths between x and y are red/blue path.
- y belongs to the component.

Conflict component and intersection

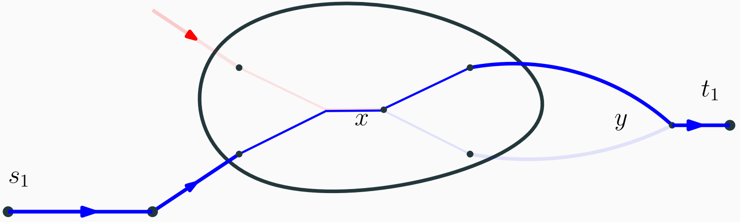
Lemma
Let P_1 and P_2 be two paths such that C is a conflicting component, then $P_1 \cap P_2$ is empty outside of C.



- Both paths between x and y are red/blue path.
- y belongs to the component.

Conflict component and intersection

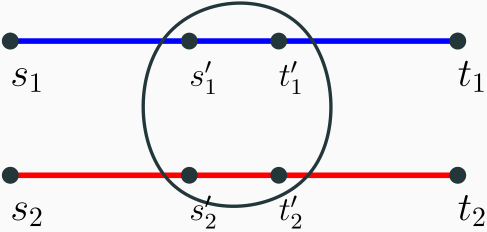
Lemma
Let P_1 and P_2 be two paths such that C is a conflicting component, then $P_1 \cap P_2$ is empty outside of C.



- Both paths between x and y are red/blue path.
- y belongs to the component.

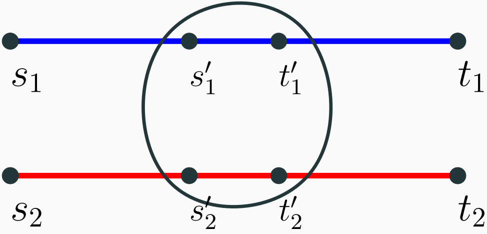
conflicting case

If P_1 and P_2 are **conflicting**, then the conflicting component has the properties we want.



conflicting case

If P_1 and P_2 are **conflicting**, then the conflicting component has the properties we want.

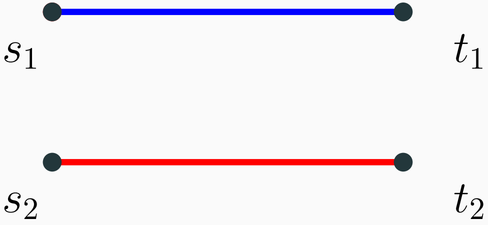


Unfortunately, this is not always the case and we have to look for something **weaker**.

Intersection between red/blue paths

Definition
Let P_1 be a blue (s_1, t_1)-path and P_2 a red (s_2, t_2)-path. We say that P_1 **does not see** P_2 if:

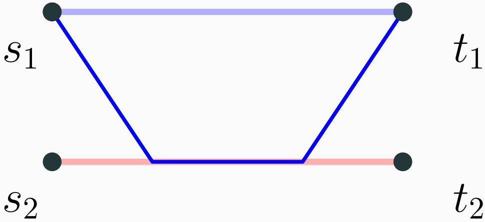
- No blue (s_1, t_1)-path can intersect P_2



Intersection between red/blue paths

Definition
Let P_1 be a blue (s_1, t_1)-path and P_2 a red (s_2, t_2)-path. We say that P_1 **does not see** P_2 if:

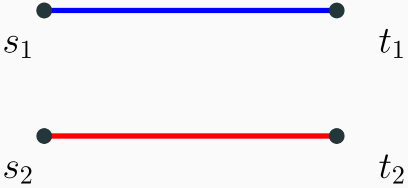
- No blue (s_1, t_1)-path can intersect P_2



Blind paths

Definition
Let P_1 be a blue (s_1, t_1)-path and P_2 a red (s_2, t_2)-path. We say that P_1 and P_2 are **blind** if:

- P_1 does not see P_2
- P_2 does not see P_1

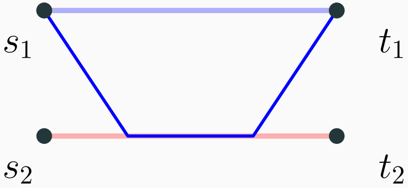


- Goal: reduce to an instance where pairs of paths of different colours are blind.
- Fortune's algorithm can be adapted in that case.

Blind paths

Definition
Let P_1 be a blue (s_1, t_1)-path and P_2 a red (s_2, t_2)-path. We say that P_1 and P_2 are **blind** if:

- P_1 does not see P_2
- P_2 does not see P_1

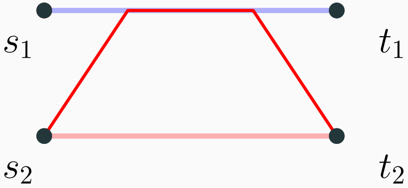


- Goal: reduce to an instance where pairs of paths of different colours are blind.
- Fortune's algorithm can be adapted in that case.

Blind paths

Definition
Let P_1 be a blue (s_1, t_1)-path and P_2 a red (s_2, t_2)-path. We say that P_1 and P_2 are **blind** if:

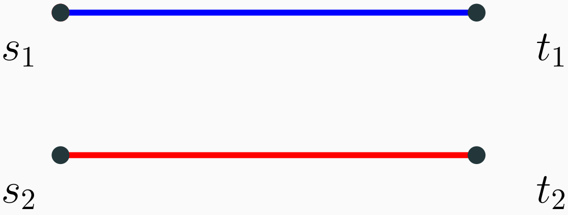
- P_1 does not see P_2
- P_2 does not see P_1



- Goal: reduce to an instance where pairs of paths of different colours are blind.
- Fortune's algorithm can be adapted in that case.

Finding a blind decomposition

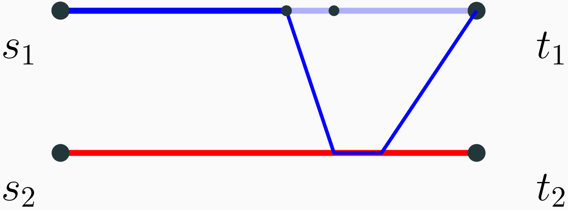
Let P_1 and P_2 be two disjoint paths such that P_1 sees P_2:



- Consider the last vertex x on P_1 s.t $\exists$ a blue (x, t_1)-path intersecting P_2. $(x \neq t_1)$.

Finding a blind decomposition

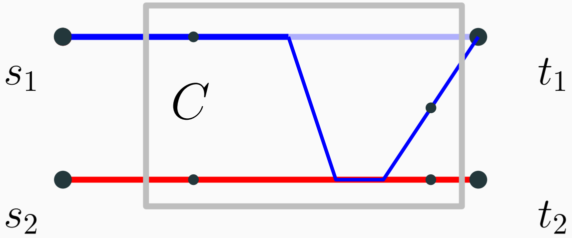
Let P_1 and P_2 be two disjoint paths such that P_1 sees P_2:



- Consider the last vertex x on P_1 s.t $\exists$ a blue (x, t_1)-path intersecting P_2. $(x \neq t_1)$.
- This defines P_1' intersecting P_2.

Finding a blind decomposition

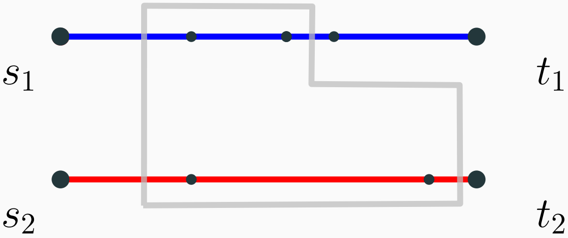
Let P_1 and P_2 be two disjoint paths such that P_1 sees P_2:



- Consider the last vertex x on P_1 s.t $\exists$ a blue (x, t_1)-path intersecting P_2. $(x \neq t_1)$.
- This defines P_1' intersecting P_2.
- $\exists$ C such that P_1' and P_2 cannot intersect outside.

Finding a blind decomposition

Let P_1 and P_2 be two disjoint paths such that P_1 sees P_2:



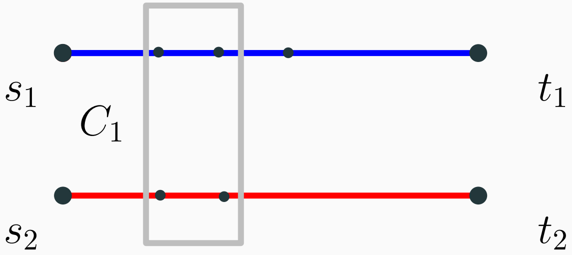
- Consider the last vertex x on P_1 s.t $\exists$ a blue (x, t_1)-path intersecting P_2. $(x \neq t_1)$.
- This defines P_1' intersecting P_2.
- $\exists$ C such that P_1' and P_2 cannot intersect outside.

Finding a blind decomposition

Lemma

There exists a decomposition of P_1 into a finite set $\mathcal{H}_1$ of blue paths and decomposition of P_2 into a finite set $\mathcal{H}_2$ of red paths s.t for any pair $L_1 \in \mathcal{H}_1$ and $L_2 \in \mathcal{H}_2$:

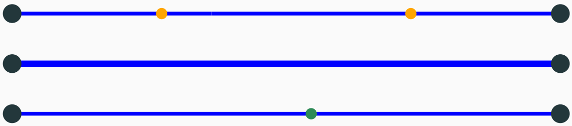
- *Either L_1 **does not see** L_2; or*
- *They belong to the **same** bi-coloured component C_1.*



Intersection of path-decomposition

Definition
Let $\mathcal{H}$ and $\mathcal{Q}$ be two path-partitions of P. The intersection $\mathcal{H} \cap \mathcal{Q}$ is the minimal path partition $\mathcal{L}$ of P such that for every $L \in \mathcal{L}$:

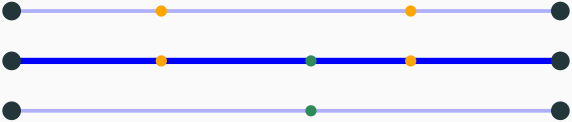
- L is a subpath of some $H \in \mathcal{H}$
- L is a subpath of some $Q \in \mathcal{Q}$



Intersection of path-decomposition

Definition
Let $\mathcal{H}$ and $\mathcal{Q}$ be two path-partitions of P. The intersection $\mathcal{H} \cap \mathcal{Q}$ is the minimal path partition $\mathcal{L}$ of P such that for every $L \in \mathcal{L}$:

- L is a subpath of some $H \in \mathcal{H}$
- L is a subpath of some $Q \in \mathcal{Q}$

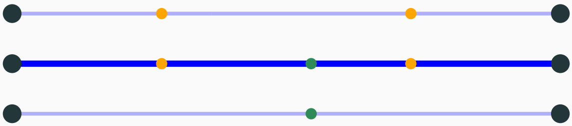


- $|\mathcal{H} \cap \mathcal{Q}| \leq |\mathcal{H}| + |\mathcal{Q}|$

Intersection of path-decomposition

Definition
Let $\mathcal{H}$ and $\mathcal{Q}$ be two path-partitions of P. The intersection $\mathcal{H} \cap \mathcal{Q}$ is the minimal path partition $\mathcal{L}$ of P such that for every $L \in \mathcal{L}$:

- L is a subpath of some $H \in \mathcal{H}$
- L is a subpath of some $Q \in \mathcal{Q}$

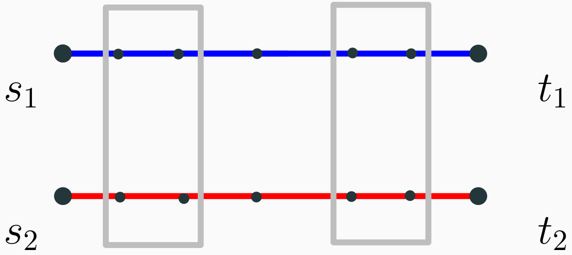


- $|\mathcal{H} \cap \mathcal{Q}| \leq |\mathcal{H}| + |\mathcal{Q}|$

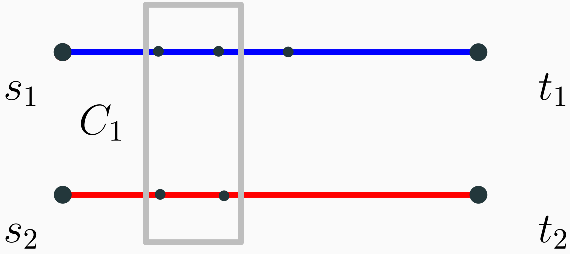
Main Lemma

Lemma
There exists a decomposition of P_1 into a finite set $\mathcal{H}_1$ of blue paths and decomposition of P_2 into a finite set $\mathcal{H}_2$ of red paths as well as two components s.t, for any pair $L_1 \in \mathcal{H}_1$ and $L_2 \in \mathcal{H}_2$:

- *Either L_1 and L_2 are **blind**; or*
- *They belong to the **same** bi-coloured component.*

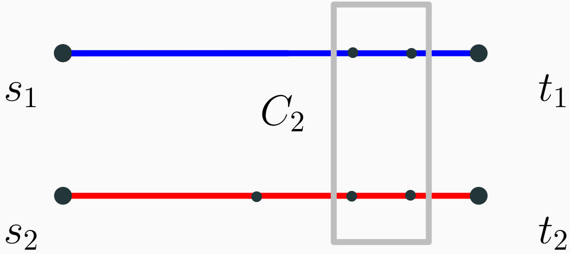


Proof of the main lemma



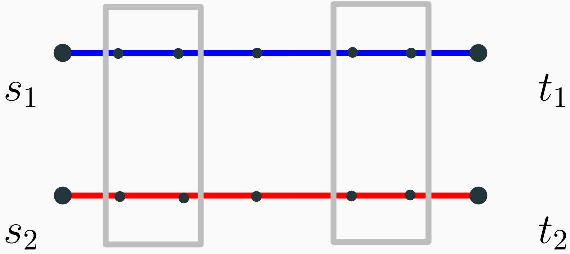
- We can find $\mathcal{H}_1$ and $\mathcal{H}_2$ such that elements of $\mathcal{H}_1$ do not see elements of $\mathcal{H}_2$ outside of C_1.

Proof of the main lemma



- We can find $\mathcal{H}_1$ and $\mathcal{H}_2$ such that elements of $\mathcal{H}_1$ do not see elements of $\mathcal{H}_2$ outside of C_1.
- We can find $\mathcal{Q}_1$ and $\mathcal{Q}_2$ such that elements of $\mathcal{Q}_2$ do not see elements of $\mathcal{Q}_1$ outside of C_2

Proof of the main lemma



- We can find $\mathcal{H}_1$ and $\mathcal{H}_2$ such that elements of $\mathcal{H}_1$ do not see elements of $\mathcal{H}_2$ outside of C_1.
- We can find $\mathcal{Q}_1$ and $\mathcal{Q}_2$ such that elements of $\mathcal{Q}_2$ do not see elements of $\mathcal{Q}_1$ outside of C_2

Taking $\mathcal{H}_1 \cap \mathcal{Q}_1$ and $\mathcal{H}_2 \cap \mathcal{Q}_2$ works for the previous lemma.

Conclusion

Result

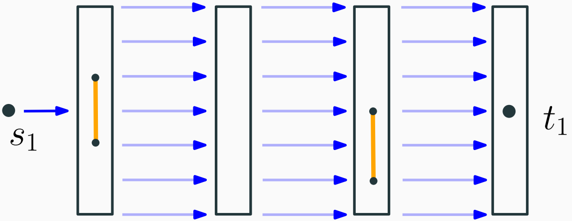
Theorem

The (k,l)-DSP problem admits an algorithm in $n^{f(k,l)}$

Result

Theorem
The (k, l)-DSP problem admits an algorithm in $n^{f(k,l)}$

We can extend it to $|P_i| \leq d(s_i, t_i) + t$ for all $i \in t$ $(n^{g(k,l,t)})$



There is at most t edge which are not between levels/not used with the correct orientation.

Open question

Question
Can we find a polynomial algorithm for the optimal-disjoint paths problem?

Even finding an approximation would be interesting.

Open question

Question
Can we find a polynomial algorithm for the optimal-disjoint paths problem?

Even finding an approximation would be interesting.

Question (Directed version)
Is the directed k-DSP in XP?

- Open for $k = 3$
- Seems much harder!

Thank you!