Finding disjoint Paths in Graphs

Torsten Tholey

Universität Augsburg

September 2011
Finding Disjoint Paths

**Given:** A graph with sources and targets.
**Problem:** Connect sources and targets by disjoint paths.
The problem

Finding Disjoint Paths

**Given:** A graph with sources and targets.
**Problem:** Connect sources and targets by disjoint paths.
The problem

Several version

- single/multiple sources and targets.
Several version

- single/multiple sources and targets.
Several versions:

- single/multiple sources and targets.
- connecting sets/pairs sources/targets.
The problem

Several version

- single/multiple sources and targets.
- connecting sets/pairs sources/targets.
The problem

Several version

- single/multiple sources and targets.
- connecting sets/pairs sources/targets.
- vertex-/edge-disjoint paths.
Several version

- single/multiple sources and targets.
- connecting sets/pairs sources/targets.
- vertex-/edge-disjoint paths.
The problem

Several versions

- single/multiple sources and targets.
- connecting sets/pairs sources/targets.
- vertex-/edge-disjoint paths.
- in undirected/directed graphs
The problem

Several versions

- single/multiple sources and targets.
- connecting sets/pairs sources/targets.
- vertex-/edge-disjoint paths.
- in undirected/directed graphs
The problem

Applications

- routing in computer/traffic networks
- reliability of networks
- VLSI design
A first problem

**Given:** vertices $s$ and $t$ in a directed graph, $k \in \mathbb{N}$

**Output:** $k$ edge-disjoint paths from $s$ to $t$, if they exist.
A first approach

**Repeat:**
- Return a path $p$.
- Delete $p$ from $G$

**Until** there is no path in $G$
The approach does not work

$G$

$G, p_1$
The approach does not work
The approach does not work.

In $G \setminus \{p_1, p_2\}$ there is no path from $s$ to $t$. 
The approach does not work

Let $G$ be a graph with single vertices $s$, $t$, $a$, $b$, $c$, $d$, $e$, and $f$. Consider two paths $p_1 = s \rightarrow t$ and $p_2 = a \rightarrow b$. In the graph $G \backslash \{p_1, p_2\}$, there is no path from $s$ to $t$.

However, in $G$ there are three edge-disjoint paths $p_i : s \rightarrow t$:
The approach does not work

$G$

$G \{ p_1, p_2 \}$
Conclusion

**Problem:** We make wrong decisions.
Conclusion

**Problem:** We make wrong decisions.

**Solution:** Edges of already constructed paths may be used in reverse direction.
Conclusion

**Problem:** We make wrong decisions.

**Solution:** Edges of already constructed paths may be used in reverse direction. Afterward delete doubly visited edges.
Connecting single vertices

residual graph $G_{p_1,\ldots,p_k}$

**Given:** A graph $G$, edge-disjoint paths $p_1, \ldots, p_k$

$G_{p_1,\ldots,p_k}$: Graph obtained from $G$ be reversing the edges of $p_1, \ldots, p_k$.

$G, p_1, p_2$

$R_{G,p_1,p_2}$
Deletion of edges from a path in the residual graph
Deletion of edges from a path in the residual graph
Deletion of edges from a path in the residual graph
### The algorithm of Ford and Fulkerson

1. **For** $i = 1$ **to** $k$:

2. Construct $R_{G,p_1,...,p_{i-1}}$ ($= G$ if $i = 1$). $O(m + n)$

3. Find a path $p : s \rightarrow t$ in $R_{G,p_1,...,p_{i-1}}$. $O(m + n)$

4. Delete all backward edges from $p$ and the corresponding forward edges from $p$. $O(m + n)$

5. Rename the resulting paths $p_1, \ldots, p_i$. $O(m + n)$

6. **End For**

**Running time:** $O(k(m + n))$. 
### The algorithm of Ford and Fulkerson

1. **For** $i = 1$ **to** $k$:
2. Construct $R_{G, p_1, \ldots, p_{i-1}}$ (= $G$ if $i = 1$). $O(m + n)$
3. Find a path $p : s \rightarrow t$ in $R_{G, p_1, \ldots, p_{i-1}}$. $O(m + n)$
4. Delete all backward edges from $p$ and the corresponding forward edges from $p$. $O(m + n)$
5. Rename the resulting paths $p_1, \ldots, p_i$. $O(m + n)$
6. **End For**

**Running time:** $O(k(m + n))$.

### Correctness

We have to show: If there is no path $p : s \rightarrow t$ in $R_{p_1, \ldots, p_k}$, then $G$ has at most $k$ edge-disjoint paths from $s$ to $t$. 
$s, t$-edge cut

An $s, t$-edge cut is a set $S$ of edges with $s, t$ being not connected in $G \setminus S$. 
Lemma 1

For an $s$-$t$ edge-cut $C$, there are at most $|C|$ edge-disjoint paths from $s$ to $t$.

Proof

Every paths from $s$ to $t$ must visit an edge of $C$. 

Connecting single vertices
Lemma 2

If there is no path $p : s \rightarrow t$ in $R_{p_1, \ldots, p_k}$, then $G = (V, E)$ has at most $k$ edge-disjoint paths from $s$ to $t$. 
Lemma 2

If there is no path $p : s \rightarrow t$ in $R_{p_1, \ldots, p_k}$, then $G = (V, E)$ has at most $k$ edge-disjoint paths from $s$ to $t$.

Proof

Assume there is no path $p : s \rightarrow t$ in $R_{p_1, \ldots, p_k}$. 
Lemma 2

If there is no path $p : s \rightarrow t$ in $R_{p_1, \ldots, p_k}$, then $G = (V, E)$ has at most $k$ edge-disjoint paths from $s$ to $t$.

Proof

- Assume there is no path $p : s \rightarrow t$ in $R_{p_1, \ldots, p_k}$.
- Let $S$ be the vertices reachable from $s$ in $R_{p_1, \ldots, p_k}$.
Lemma 2

If there is no path $p : s \rightarrow t$ in $R_{p_1,\ldots,p_k}$, then $G = (V, E)$ has at most $k$ edge-disjoint paths from $s$ to $t$.

Proof

- Assume there is no path $p : s \rightarrow t$ in $R_{p_1,\ldots,p_k}$.
- Let $S$ be the vertices reachable from $s$ in $R_{p_1,\ldots,p_k}$.

$\Rightarrow F = S \times (V \setminus S)$ is an $s,t$-edge cut in $G$. 
Lemma 2

If there is no path $p : s \rightarrow t$ in $R_{p_1, \ldots, p_k}$, then $G = (V, E)$ has at most $k$ edge-disjoint paths from $s$ to $t$.

Proof

- Assume there is no path $p : s \rightarrow t$ in $R_{p_1, \ldots, p_k}$.
- Let $S$ be the vertices reachable from $s$ in $R_{p_1, \ldots, p_k}$.

$\implies F = S \times (V \setminus S)$ is an $s, t$-edge cut in $G$.

$\implies$ Every path from $s$ to $t$ uses an edge in $F$. 
Lemma 2
If there is no path $p : s \rightarrow t$ in $R_{p_1,...,p_k}$, then $G = (V, E)$ has at most $k$ edge-disjoint paths from $s$ to $t$.

Proof
- Assume there is no path $p : s \rightarrow t$ in $R_{p_1,...,p_k}$.
- Let $S$ be the vertices reachable from $s$ in $R_{p_1,...,p_k}$.
- $F = S \times (V \setminus S)$ is an $s, t$-edge cut in $G$.
- Every path from $s$ to $t$ uses an edge in $F$.
- No path $p : s \rightarrow t$ uses an edge in $(V \setminus S) \times S$ (otherwise there is an backward edge in $S \times (V \setminus S)$ in $R_{p_1,...,p_k}$).
Connecting single vertices

Lemma 2

If there is no path $p : s \rightarrow t$ in $R_{p_1, \ldots, p_k}$, then $G = (V, E)$ has at most $k$ edge-disjoint paths from $s$ to $t$.

Proof

- Assume there is no path $p : s \rightarrow t$ in $R_{p_1, \ldots, p_k}$.
- Let $S$ be the vertices reachable from $s$ in $R_{p_1, \ldots, p_k}$.

$\Rightarrow F = S \times (V \setminus S)$ is an $s, t$-edge cut in $G$.

$\Rightarrow$ Every path from $s$ to $t$ uses an edge in $F$.

- No path $p : s \rightarrow t$ uses an edge in $(V \setminus S) \times S$ (otherwise there is an backward edge in $S \times (V \setminus S)$ in $R_{p_1, \ldots, p_k}$).
- All edges in $F$ are visited by a path in $p_1, \ldots, p_k$. 
Lemma 2

If there is no path $p : s \rightarrow t$ in $R_{p_1,...,p_k}$, then $G = (V, E)$ has at most $k$ edge-disjoint paths from $s$ to $t$.

Proof

- Assume there is no path $p : s \rightarrow t$ in $R_{p_1,...,p_k}$.
- Let $S$ be the vertices reachable from $s$ in $R_{p_1,...,p_k}$.
  \[ F = S \times (V \setminus S) \]  is an $s, t$-edge cut in $G$.
  Every path from $s$ to $t$ uses an edge in $F$.
- No path $p : s \rightarrow t$ uses an edge in $(V \setminus S) \times S$ (otherwise there is an backward edge in $S \times (V \setminus S)$ in $R_{p_1,...,p_k}$).
- All edges in $F$ are visited by a path in $p_1, \ldots, p_k$.
  \[ |F| = k. \]
Lemma 2

If there is no path \( p : s \rightarrow t \) in \( R_{p_1, \ldots, p_k} \), then \( G = (V, E) \) has at most \( k \) edge-disjoint paths from \( s \) to \( t \).

Proof

- Assume there is no path \( p : s \rightarrow t \) in \( R_{p_1, \ldots, p_k} \).
- Let \( S \) be the vertices reachable from \( s \) in \( R_{p_1, \ldots, p_k} \).
  \[ F = S \times (V \setminus S) \] is an \( s, t \)-edge cut in \( G \).
  Every path from \( s \) to \( t \) uses an edge in \( F \).
- No path \( p : s \rightarrow t \) uses an edge in \( (V \setminus S) \times S \) (otherwise there is an backward edge in \( S \times (V \setminus S) \) in \( R_{p_1, \ldots, p_k} \)).
- All edges in \( F \) are visited by a path in \( p_1, \ldots, p_k \).
  \[ |F| = k. \]
  There are at most \( k = |F| \) paths (Lemma 1).
Observation 1

Lemma 1 and the line $|F| = k$ in the last lemma also imply the following theorem.
### Observation 1

Lemma 1 and the line $|F| = k$ in the last lemma also imply the following theorem.

### Menger’s Theorem

There are $k$-edge disjoint paths between two vertices if and only if there are no $s, t$-edge cut of size $k - 1$. 
Observation 1

Lemma 1 and the line $|F| = k$ in the last lemma also imply the following theorem.

Menger’s Theorem

There are $k$-edge disjoint paths between two vertices if and only if there are no $s, t$-edge cut of size $k - 1$.

Menger’s Theorem (vertex version)

There are $k$-vertex disjoint paths between two nonadjacent vertices if and only if there are no $s, t$-vertex cut of size $k - 1$. 

Connecting single vertices
Connecting single vertices

Other versions

If we want to find edge-disjoint paths in an undirected graph

- we replace an edge \( \{u, v\} \) by directed edges \((u, v), (v, u)\).
If we want to find edge-disjoint paths in an undirected graph
- we replace an edge \( \{u, v\} \) by directed edges \((u, v), (v, u)\).
- A solution might use the same edge in different directions.
Connecting single vertices

Other versions

If we want to find edge-disjoint paths in an undirected graph
- we replace an edge \( \{u, v\} \) by directed edges \((u, v), (v, u)\).
- A solution might use the same edge in different directions.
- Doubly visited edges can be removed as backward edges and their counterparts.
Connecting single vertices

Other versions

For finding vertex-disjoint paths in a directed graph $G$, let $G'$ be obtained from $G$ by replacing

- each vertex by vertices $v'$ and $v''$ and an edge $(v, v')$.
- each edge $(u, v)$ by an edge $(u', v')$.

![Diagram showing transformation from $G$ to $G'$](image-url)
Observation 1:
For every path

\[ x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow \cdots \rightarrow x_{k-2} \rightarrow x_{k-1} \rightarrow x_k \]

in \( G \), there is a path in \( G' \)

\[ x_1'' \rightarrow x_2' \rightarrow x_2'' \rightarrow x_3' \rightarrow \cdots \rightarrow x_{k-2}' \rightarrow x_{k-1}' \rightarrow x_{k-1}'' \rightarrow x_k' \]

and vice versa.
Observation 1:
For every path

\[ x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow \cdots \rightarrow x_{k-2} \rightarrow x_{k-1} \rightarrow x_k \]

in \( G \), there is a path in \( G' \)

\[ x_1'' \rightarrow x_2' \rightarrow x_2'' \rightarrow x_3' \rightarrow \cdots \rightarrow x_{k-2}'' \rightarrow x_{k-1}' \rightarrow x_{k-1}'' \rightarrow x_k' \]

and vice versa.

Observation 2:
There are \( k \) internally vertex-disjoint paths \( s \rightarrow t \) in \( G \)

\( \iff \)
There are \( k \) edge-disjoint path \( s'' \rightarrow t' \) in \( G' \).
Connecting single vertices

Other versions

If we want to find vertex-disjoint paths in an undirected graph we replace an vertex \( \{u, v\} \) by directed edges \((u, v), (v, u)\).
Connecting single vertices

Other versions

If we want to find vertex-disjoint paths in an undirected graph

- we replace an vertex \( \{u, v\} \) by directed edges
  \((u, v), (v, u)\).
Connecting single vertices

Theorem
Given two vertices $s$ and $t$ in an undirected or directed graph, $k$ edge or internally vertex-disjoint paths from $s$ to $t$ can be computed in $O(k(m + n))$ time.
Theorem

Given two vertices $s$ and $t$ in an undirected or directed graph, $k$ edge or internally vertex-disjoint paths from $s$ to $t$ can be computed in $O(k(m + n))$ time.

Theorem

Given two vertices $s$ and $t$ in an undirected or directed graph $G$, a maximum number edge or internally vertex-disjoint paths from $s$ to $t$ can be computed in $O(n(m + n))$ time.
Theorem
Given two vertices $s$ and $t$ in an undirected or directed graph, $k$ edge or internally vertex-disjoint paths from $s$ to $t$ can be computed in $O(k(m + n))$ time.

Theorem
Given two vertices $s$ and $t$ in an undirected or directed graph $G$, a maximum number edge or internally vertex-disjoint paths from $s$ to $t$ can be computed in $O(n(m + n))$ time.

Proof
There are at most $n - 1$ such paths since $\deg(s) \leq n - 1$. 
Question?

Can a maximum number of disjoint paths from $s$ and $t$ be computed faster.
Question?
Can a maximum number of disjoint paths from $s$ and $t$ be computed faster.

Definition

Given: Vertices $s$ and $t$ in a directed $G$.
$L(G, s, t)$: The graph obtained from $G$ after

1. removing all edge $(v,w)$ with $\text{dist}_G(v, t) \neq \text{dist}_G(w, t) + 1$
2. removing all vertices not reachable from $s$.
3. removing all vertices from which $t$ is not reachable.
Connecting single vertices

Construction of $L(G, s, t)$
Connecting single vertices

Construction of $L(G, s, t)$
Advantages of a level graph

- we can construct a path $p$ in reverse direction in $O(|p|)$ instead of $O(m + n)$ time:
  
  Starting with $p : t \rightarrow t$ always add a vertex with a distance one larger than the current start point.

- Time for computing the set $T$ of edges of $L(G, s, t)$ from which $t$ is not reachable in $L(G \setminus p, s, t)$: $O(|T|)$

  (During the construction of $p$, remove the edges of $p$ from $L(G, s, t)$, after removing an edge $(u, v)$ update $\text{outdeg}(u)$, and for each vertex with $\text{outdeg}(u) = 0$ remove all edges ending in $u$.)

- Time for computing the set $S$ of edges of $L(G, s, t)$ not reachable from $S$ in $L(G \setminus p, s, t)$: $O(|S|)$. 

Advantages of a level graph

- \( L(G \setminus p, s, t) \) can be constructed in a time linear in the edges of \( L(G, s, t) - L(G \setminus p, s, t) \).

\[ \Rightarrow \]

Applying the algorithm recursively on \( L(G \setminus p, s, t) \) a maximal number of edge-disjoint paths in \( L(G, s, t) \) can be computed in \( O(m + n) \) time.
Connecting single vertices

Update of the level graph

a) \( T = \emptyset \)

\[
\begin{array}{c}
\text{s} \\
\text{a} \\
\end{array}
\begin{array}{c}
\text{c} \\
\text{u} \\
\end{array}
\begin{array}{c}
\text{g} \\
\text{h} \\
\end{array}
\begin{array}{c}
\text{p} \\
\end{array}
\begin{array}{c}
\text{f} \\
\text{v} \\
\text{b} \\
\text{r} \\
\end{array}
\begin{array}{c}
\text{t} \\
\text{m} \\
\end{array}
\]

b) \( T = T \cup \{ g \} \)

\[
\begin{array}{c}
\text{s} \\
\text{a} \\
\text{g} \\
\text{c} \\
\text{u} \\
\end{array}
\begin{array}{c}
\text{h} \\
\text{p} \\
\end{array}
\begin{array}{c}
\text{f} \\
\text{v} \\
\text{b} \\
\text{r} \\
\end{array}
\begin{array}{c}
\text{m} \\
\text{t} \\
\end{array}
\]

Torsten Tholey

Universität Augsburg
Update of the level graph

b) 

\[ T = T \cup \{c, r\} \]
Connecting single vertices

Update of the level graph

c) 

\[ T = T \cup \{ a, b, u \} \]
Update of the level graph

d) $T = T \cup \emptyset$
Connecting single vertices

Update of the level graph

e) Torsten Tholey

Universität Augsburg
## The algorithm of Dinitz

1. \( R := G \);
2. \textbf{while} \( t \) is reachable from \( s \) in \( R \)
3. \( \text{Construct } G' = L(R, s, t), \ j = 0 \)
4. \textbf{while} \( t \) is reachable from \( s \) in \( G' \)
5. \( j = j + 1 \).
6. \( \text{Construct a path } q_j : s \rightarrow t \) in \( G' \) and update \( G' \) as the level graph without \( q_j \)
7. \textbf{end while}
8. Replace \( p_1, \ldots, p_i, q_1, \ldots, q_j \) by edge-disjoint paths \( p'_1, \ldots, p'_{i+j} \) in \( G \).
9. Rename \( p'_1, \ldots, p'_{i+j} \) in \( p_1, \ldots, p_{i+j} \).
10. \( i := i + j \)
11. \( R := R_{G, p_1, \ldots, p_i} \)
12. \textbf{end while}
Analyzing running time

- The outer loop divides the algorithms into rounds.
- We will show the distance between \( s \) and \( t \) in the residual graph \( R \) increases after each round.
  - After \( \sqrt{m} \) rounds the distance is \( O(\sqrt{m}) \).
  - There are at most \( O(\sqrt{m}) \) edge-disjoint paths in \( R \).
  - There are at most \( O(\sqrt{m}) \) additional paths in \( G \).
  - The algorithm stops after \( O(\sqrt{m}) \) additional rounds.
  - The total running time is \( O(\sqrt{m}(m + n)) \).
Lemma

the distance between $s$ and $t$ in $R$ increases after each round.
Lemma
the distance between $s$ and $t$ in $R$ increases after each round.

Proof
- Each path $p : s \rightarrow t$ in $R$ has $dist_R(s, t)$ edges.
- $R$ has no edges $(v, w)$ with $dist_R(v, t) > dist_R(w, t) + 1$. 
Lemma

the distance between $s$ and $t$ in $R$ increases after each round.

Proof

- Each path $p : s \rightarrow t$ in $R$ has $\text{dist}_R(s, t)$ edges.
- $R$ has no edges $(v, w)$ with $\text{dist}_R(v, t) > \text{dist}_R(w, t) + 1$.
- $\text{dist}_R(v, t) = \text{dist}_R(w, t) + 1$ for all edges $(v, w)$ on $p$. 
Lemma

the distance between $s$ and $t$ in $R$ increases after each round.

Proof

- Each path $p : s \rightarrow t$ in $R$ has $\text{dist}_R(s, t)$ edges.
- $R$ has no edges $(v, w)$ with $\text{dist}_R(v, t) > \text{dist}_R(w, t) + 1$.
- $\text{dist}_R(v, t) = \text{dist}_R(w, t) + 1$ for all edges $(v, w)$ on $p$.

$\Rightarrow$ The residual graph $R'$ of the next round has also no edges $(v, w)$ with $\text{dist}_R(v, t) > \text{dist}_R(w, t) + 1$. 
Lemma

the distance between \( s \) and \( t \) in \( R \) increases after each round.

Proof

- Each path \( p : s \rightarrow t \) in \( R \) has \( \text{dist}_R(s, t) \) edges.
- \( R \) has no edges \((v, w)\) with \( \text{dist}_R(v, t) > \text{dist}_R(w, t) + 1 \).
- \( \text{dist}_R(v, t) = \text{dist}_R(w, t) + 1 \) for all edges \((v, w)\) on \( p \).

\( \Rightarrow \) The residual graph \( R' \) of the next round has also no edges \((v, w)\) with \( \text{dist}_R(v, t) > \text{dist}_R(w, t) + 1 \).

\( \Rightarrow \) Each path in \( p' \) in \( R' \) must visit edges \((v, w)\) with

\[ i = \text{dist}_R(v, t) = \text{dist}_R(w, t) + 1 \quad \text{for all} \quad i \geq \text{dist}_R(s, t). \]
Lemma

The distance between $s$ and $t$ in $R$ increases after each round.

Proof

- Each path $p : s \rightarrow t$ in $R$ has $\text{dist}_R(s, t)$ edges.
- $R$ has no edges $(v, w)$ with $\text{dist}_R(v, t) > \text{dist}_R(w, t) + 1$.
- $\text{dist}_R(v, t) = \text{dist}_R(w, t) + 1$ for all edges $(v, w)$ on $p$.

$\Rightarrow$ The residual graph $R'$ of the next round has also no edges $(v, w)$ with $\text{dist}_R(v, t) > \text{dist}_R(w, t) + 1$.

$\Rightarrow$ Each path in $p'$ in $R'$ must visit edges $(v, w)$ with $i = \text{dist}_R(v, t) = \text{dist}_R(w, t) + 1$ for all $i \geq \text{dist}_R(s, t)$.

- Paths exclusively existing of such edges were removed in the previous round.
Lemma

the distance between $s$ and $t$ in $R$ increases after each round.

Proof

- Each path $p : s \rightarrow t$ in $R$ has $\text{dist}_R(s, t)$ edges.
- $R$ has no edges $(v, w)$ with $\text{dist}_R(v, t) > \text{dist}_R(w, t) + 1$.
- $\text{dist}_R(v, t) = \text{dist}_R(w, t) + 1$ for all edges $(v, w)$ on $p$.

$\Rightarrow$ The residual graph $R'$ of the next round has also no edges $(v, w)$ with $\text{dist}_R(v, t) > \text{dist}_R(w, t) + 1$.

$\Rightarrow$ Each path in $p'$ in $R'$ must visit edges $(v, w)$ with $i = \text{dist}_R(v, t) = \text{dist}_R(w, t) + 1$ for all $i \geq \text{dist}_R(s, t)$.

- Paths exclusively existing of such edges were removed in the previous round.

$\Rightarrow$ Each path in $p'$ has length $> \text{dist}_R(s, t)$.
A new problem

**Given:** vertices $s$ and $t$ in a directed weighted graph, $k \in \mathbb{N}$

**Output:** $k$ edge-disjoint paths from $s$ to $t$, such that the sum of the edge weights is minimal.
A simple approach

- Let $g(v, w)$ be the weight of $(v, w)$.
- Assume we have constructed edge-disjoint paths $p_1, \ldots, p_i$ of shortest total length.
- Search for a shortest path in $R_{p_1, \ldots, p_k}$, where we assign weight $-g(v, w)$ to a backward edge $(w, v)$. 
Example:
Example:
Example:
Example:
Example:
Simplification

We assume that the graph is antisymmetric, i.e.,

\[(v, w) \in E \Rightarrow (w, v) \notin E.\]

Otherwise the following replacement is possible.
Simplification (positive edge weights)

- We replace \( g(v, w) \) by
  \[
  g^*(v, w) := g(v, w) + \text{dist}_g(s, v) - \text{dist}_g(s, w).
  \]

- For a path \( p = ((u_i, v_i))_{1 \leq i \leq k} \) from a vertex \( u \) to a vertex \( v \), we have

  \[
  \sum_i g^*((u_i, v_i)) = \sum_i g((u_i, v_i)) + \sum_{1 \leq i \leq k} \text{dist}_g(s, u_i) - \sum_{1 \leq i \leq k} \text{dist}_g(s, v_i)
  = \text{dist}_g(s, u) - \text{dist}_g(s, v) + \sum_i g((u_i, v_i)).
  \]

\(\Rightarrow\) A shortest path \( p : u \rightarrow v \) with respect to \( g \) is also a shortest path with respect to \( g^* \) and vice versa.

\(\Rightarrow\) We only have to compute shortest with respect to \( g^* \).
Advantages of the new edge weights

- All edge weights are positive.
- Edges on a shortest path $p$ have weight 0.
- Edges in $R_p$ have weight 0.
The correctness of the simple approach follows from the following lemma.

**Lemma**

- Let $p_1, \ldots, p_k$ simple edge-disjoint paths from $s$ to $t$ of shortest total length $d$.
- Let $p$ be a shortest path from $s$ to $t$ in $G$.
- Then there are edge-disjoint paths $q_1 \ldots, q_{k-1}$ from $s$ to $t$ in $R_p$ of total length $\leq d$. 
Correctness of the simple approach

The correctness of the simple approach follows from the following lemma.

Lemma

- Let $p_1, \ldots, p_k$ simple edge-disjoint paths from $s$ to $t$ of shortest total length $d$.
- Let $p$ be a shortest path from $s$ to $t$ in $G$.
- Then there are edge-disjoint paths $q_1 \ldots, q_{k-1}$ from $s$ to $t$ in $R_p$ of total length $\leq d$. 
Lemma

- Let $p_1, \ldots, p_k$ simple edge-disjoint paths from $s$ to $t$ of shortest total length $d$.
- Let $p$ be a shortest path from $s$ to $t$ in $G$.
- Then there are edge-disjoint paths $q_1, \ldots, q_k$ from $s$ to $t$ in $R_p$ of total length $\leq d$.  

Torsten Tholey

Universität Augsburg
Lemma

- Let $p_1, \ldots, p_k$ simple edge-disjoint paths from $s$ to $t$ of shortest total length $d$.
- Let $p$ be a shortest path from $s$ to $t$ in $G$.
- Then there are edge-disjoint paths $q_1, \ldots, q_k$ from $s$ to $t$ in $R_p$ of total length $\leq d$.

$I_u$ should mean

- $q_1, \ldots, q_k$ are edge-disjoint paths on $(V_G, E_G \cup E_{R_G,p})$ of length $\leq d$ except that edges of $p[u, t]$ are used $\leq 2$ times.
- The backward edges of $q_1, \ldots, q_k$ are part of $p[s, u]$.
- $q_k[s, u] = p[s, u]$.

$I_s$ holds for $q_i = p_i$ for $i < k$, $q_k = p$. We want that $I_t$ holds.
Proof (Sketch)

- Assume that $I_u$ holds for a vertex $u$ on $p$.
- Then we show that it also holds for a vertex after $u$ on $p$.
- After a finite number of steps $I_t$ holds.
Proof (Sketch)

- Assume that $I_u$ holds for a vertex $u$ on $p$.
- Then we show that it also holds for a vertex after $u$ on $p$.
- After a finite number of steps $I_t$ holds.

Proof (Sketch)

- Assume that $I_u$ holds for a vertex $u$ on $p$.
- Let $(v, w)$ be the next edge on $p[u, t]$ that is part of $q_1, \ldots, q_k$. 
Case 1 \(((v, w)\) does not exist)\n
- Replace \(q_k[u, t]\) by \(p[u, t]\).
Case 1 \((v, w)\) does not exist

- Replace \(q_k[u, t]\) by \(p[u, t]\).
- \(q_k\) remains to be a simple path.
Case 1 ((v, w) does not exist)

- Replace $q_k[u, t]$ by $p[u, t]$.
- $q_k$ remains to be a simple path.
- Assumption: $q_k[u, t]$ was shorter than $p[u, t]$. 

\[s \xrightarrow{q_k} u \xrightarrow{q_k} t \Rightarrow s \xrightarrow{q_k} u \xrightarrow{q_k} t\]
Case 1 \(((v, w)\) does not exist)\

- Replace \(q_k[u, t]\) by \(p[u, t]\).
- \(q_k\) remains to be a simple path.
- **Assumption:** \(q_k[u, t]\) was shorter than \(p[u, t]\)

\[\Rightarrow q_k\] uses a backward edge.

\[s\rightarrow u\rightarrow t\]

\[s\rightarrow u\rightarrow t\]

\[s\rightarrow u\rightarrow t\]
Case 1 ($(v, w)$ does not exist)

- Replace $q_k[u, t]$ by $p[u, t]$.
- $q_k$ remains to be a simple path.
- Assumption: $q_k[u, t]$ was shorter than $p[u, t]$

$\implies q_k$ uses a backward edge.

- The corresponding forward edge appears on $p[s, u] = q_k[s, u]$ ($I_u$).
Case 1 \(((v, w) \text{ does not exist})\)

- Replace \(q_k[u, t]\) by \(p[u, t]\).
- \(q_k\) remains to be a simple path.
- **Assumption:** \(q_k[u, t]\) was shorter than \(p[u, t]\)
  \[\Rightarrow q_k\text{ uses a backward edge.}\]
- The corresponding forward edge appears on \(p[s, u] = q_k[s, u]\ (I_u)\).
- **Contradiction to \(q_k\) being simple.**
Case 1 \(((v, w) \text{ does not exist)}\)

- Replace \(q_k[u, t]\) by \(p[u, t]\).
- \(q_k\) remains to be a simple path.
- Assumption: \(q_k[u, t]\) was shorter than \(p[u, t]\)
  \[\Rightarrow q_k\] uses a backward edge.
- The corresponding forward edge appears on \(p[s, u] = q_k[s, u]\) \((I_u)\).
- Contradiction to \(q_k\) being simple.
  \[\Rightarrow I_t\] holds after the replacement.
Case 2 ($(v, w)$ appears on $q_k$)

- Replace $q_k[u, w]$ to $p[u, w]$
Case 2 \((v, w)\) appears on \(q_k\)

- Replace \(q_k[u, w]\) to \(p[u, w]\)
- Shorten the resulting paths if it is not simple.
Case 2 \((v, w)\) appears on \(q_k\)

- Replace \(q_k[u, w]\) to \(p[u, w]\)
- Shorten the resulting paths if it is not simple.
- \(p[u, w]\) not longer than \(q_k[u, w]\) (similar to Case 1)
Case 2 \((v, w)\) appears on \(q_k\)

- Replace \(q_k[u, w]\) to \(p[u, w]\)
- Shorten the resulting paths if it is not simple.
- \(p[u, w]\) not longer than \(q_k[u, w]\) (similar to Case 1)

\[\Rightarrow I_{w'}\] holds for a vertex after \(u\) on \(p\).

\[
\begin{array}{c}
\text{Case 2 (labelling)} \\
\begin{array}{c}
\text{Replace } q_k[u, w] \text{ to } p[u, w] \\
\text{Shorten the resulting paths if it is not simple.} \\
\text{p[u, w] not longer than } q_k[u, w] \text{ (similar to Case 1)} \\
\Rightarrow I_{w'} \text{ holds for a vertex after } u \text{ on } p.
\end{array}
\end{array}
\]
Case 3 \((v, w)\) appears on \(p_i\) for \(i < k - 1\)

- Replace \(q_k\) by \(q_k[s, u] \circ p[u, v] \circ q_i[v, t]\) and \(q_i\) by \(q_i[s, v] \circ p[v, u] \circ q_k[u, t]\).
Case 3 \( ((v, w) \text{ appears on } p_i \text{ for } i < k - 1) \)

- Replace \( q_k \) by \( q_k[s, u] \circ p[u, v] \circ q_i[v, t] \) and \( q_i \) by \( q_i[s, v] \circ p[v, u] \circ q_k[u, t] \).
- This does not change the length since backward edges have length 0.
Case 3 \((v, w)\) appears on \(p_i\) for \(i < k - 1\)

- Replace \(q_k\) by \(q_k[s, u] \circ p[u, v] \circ q_i[v, t]\) and \(q_i\) by \(q_i[s, v] \circ p[v, u] \circ q_k[u, t]\).

- This does not change the length since backward edges have length 0.

\(\Rightarrow I_w\) holds.
Given: Vertices $s, t_1, \ldots, t_k$ in a graph.
Output: Disjoint paths $p_1 : s \rightarrow t_1, \ldots, p_k : s \rightarrow t_k$.
Solution: Reduction to single source/single target version.
Multiple sources and targets

**single source/multiple target**

**Given:** Vertices $s, t_1, \ldots, t_k$ in a graph.

**Output:** Disjoint paths $p_1 : s \rightarrow t_1, \ldots, p_k : s \rightarrow t_k$.

**Solution:** Reduction to single source/single target version.
Multiple sources and targets

**Multiple source/multiple target**

**Given:** Vertices $s_1, t_1, \ldots, t_1, t_k$ in a graph.

**Output:** $k$ disjoint paths connecting each source with an arbitrary target.

**Solution:** Reduction to single source/single target version.
Multiple sources and targets

Given: Vertices $s_1, t_1, \ldots, t_1, t_k$ in a graph.
Output: $k$ disjoint paths connecting each source with an arbitrary target.
Solution: Reduction to single source/single target version.
The \( k \)-VDPP

The \((k-)\)disjoint path-problem \((k-)\)VDPP

**Given:** Vertex pairs \((s_1, t_1), \ldots, (s_k, t_k)\) in a graph.

**Output:** Disjoint paths \(p_1 : s_1 \rightarrow t_1, \ldots, p_k : s_k \rightarrow t_k\).
The \( k \)-VDPP

The \((k-)\)disjoint path-problem \((k-)\)VDPP

**Given:** Vertex pairs \((s_1, t_1), \ldots, (s_k, t_k)\) in a graph.

**Output:** Disjoint paths \(p_1 : s_1 \to t_1, \ldots, p_k : s_k \to t_k\).
Applications: Simple Computer Networks

Torsten Tholey
Universität Augsburg
Step 1: Remove all small vertex-cuts (Sketch)

- Reduce the problem to one connected component
Step 1: Remove all small vertex-cuts (Sketch)

- Reduce the problem to one connected component
- Afterwards remove all vertex-cuts of size 1 (trivial by using a so-called block cutpoint tree).
Solving the 2-VDPP (Shiloach’s algorithm)

Step 1: Remove all small vertex-cuts (Sketch)

- Reduce the problem to one connected component
- Afterwards remove all vertex-cuts of size 1 (trivial by using a so-called block cutpoint tree).
- Remove all vertex-cuts of size 2 (Sketch):

![Diagram](attachment:image.png)
Solving the 2-VDPP (Shiloach’s algorithm)

Step 1: Remove all small vertex-cuts (Sketch)

- Reduce the problem to one connected component
- Afterwards remove all vertex-cuts of size 1 (trivial by using a so-called block cutpoint tree).
- Remove all vertex-cuts of size 2 (Sketch):
  - Construct disjoint paths $p_1, p_2$ from $\{s_1, s_2\}$ to $\{t_1, t_2\}$. 

![Diagram of 2-vertex cuts](image)
Step 1: Remove all small vertex-cuts (Sketch)

- Reduce the problem to one connected component
- Afterwards remove all vertex-cuts of size 1 (trivial by using a so-called block cutpoint tree).
- Remove all vertex-cuts of size 2 (Sketch):
  
  Construct disjoint paths $p_1, p_2$ from $\{s_1, s_2\}$ to $\{t_1, t_2\}$. If $s_1$ is connected to $t_2$. 

---

The diagram illustrates the process of removing vertex-cuts. The 2-vertex cuts are indicated by dashed lines. The vertices $s_1, s_2, t_1, t_2$ are connected through the intermediate vertices $u_1, v_1, u_2, v_2$.
Solving the 2-VDPP (Shiloach’s algorithm)

Step 1: Remove all small vertex-cuts (Sketch)

- Reduce the problem to one connected component
- Afterwards remove all vertex-cuts of size 1 (trivial by using a so-called block cutpoint tree).
- Remove all vertex-cuts of size 2 (Sketch):
  - Construct disjoint paths $p_1, p_2$ from $\{s_1, s_2\}$ to $\{t_1, t_2\}$.
  - If $s_1$ is connected to $t_2$.
  - Try to find a crossing in one of the 3-connected components between consecutive vertex cuts.
Step 2: Test if $G$ is planar

A graph is *planar* if it can be drawn in the plane without any crossing edges.
Solving the 2-VDPP (Shiloach’s algorithm)

Step 2: Test if $G$ is planar

A graph is *planar* if it can be drawn in the plane without any crossing edges.

```
planar
```

![Graph diagram](attachment:image.png)
Step 2: Test if $G$ is planar

A graph is *planar* if it can be drawn in the plane without any crossing edges.

planar

since it can be also drawn as
A subdivision $H$ of a graph $G = (V, E)$ is a graph $(V', E')$ for which there are injective mappings $\varphi_1 : V' \rightarrow V$ $\varphi_2 : E' \rightarrow \mathcal{P}$ such that

- $\mathcal{P}$ is a set of internally vertex-disjoint paths in $G$.
- $\varphi_2((u, v)) : \varphi_1(u) \rightarrow \varphi_1(v)$.
Theorem of Kuratowski

A graph is planar if does not contain a subgraph being a subdivision of the $K_5$ or $K_{3,3}$. 

$K_5$

$K_{3,3}$
Step 2: Test if $G$ is planar

Use one of the well known $O(m + n)$-time algorithms to compute a subgraph of $G$ being a subdivision of a $K_5$ of $K_{3,3}$ if it exists.
Solving the 2-VDPP (Shiloach’s algorithm)

Step 2: Test if $G$ is planar

Use one of the well known $O(m + n)$-time algorithms to compute a subgraph of $G$ being a subdivision of a $K_5$ of $K_{3,3}$ if it exists.

Step 3: If $G$ is planar

Solve the problem as shown on the following slides.
Solving the 2-VDPP on a planar triconnected graph

Case 1

$s_1$ and $t_1$ are on the boundary of a common face $F$ and each of the two boundary paths contains one of $s_2$ and $t_2$
Solving the 2-VDPP on a planar triconnected graph

Case 1

\( s_1 \) and \( t_1 \) are on the boundary of a common face \( F \) and each of the two boundary paths contains one of \( s_2 \) and \( t_2 \)

- Output that the instance is not solvable.
Case 2

$s_1$ and $t_1$ are on the boundary of a common face $F$ and one of the two boundary paths $p$ does contain neither $s_2$ nor $t_2$. 
Solving the 2-VDPP on a planar triconnected graph

Case 2

$s_1$ and $t_1$ are on the boundary of a common face $F$ and one of the two boundary paths $p$ does contain neither $s_2$ nor $t_2$.

- Let $q_1, q_2, q_3$ disjoint paths from $s_2$ to $t_2$.
- Return $p, q$ with $q \in \{q_1, q_2, q_3\}$ not visiting a vertex of $p$.
- Such a path must exist: otherwise there is a subdivision of a $K_{3,3}$ after adding a new vertex $x$ into $F$:

![Diagram](image-url)
Solving the 2-VDPP on a planar triconnected graph

Case 3

$s_1$ and $t_1$ are not on the boundary of a common face $F$. 
Solving the 2-VDPP on a planar triconnected graph

Case 3

$s_1$ and $t_1$ are not on the boundary of a common face $F$.

Solution

One can always find a solution.
Step 4: If a subdivision of a $K_5$ is found

- If $\{s_1, s_2, t_1, t_2\}$ is separated by a 3-vertex cut $S$ from at least one vertex $v$ of the $K_5$:
  - Remove the connected component containing $v$.
  - Insert edges between all vertex pairs in $S$.
  - Solve the 2-VDPP on the new instance.
Step 5: If a subdivision of a $K_5$ is found

- If $\{s_1, s_2, t_1, t_2\}$ are not separated by a 3-vertex cut $S$ from at least one vertex $v$ of the $K_5$:
  - Use the $K_5$ to connect the vertex pairs correctly.
Step 5: If a subdivision of a $K_5$ is found

- If $\{s_1, s_2, t_1, t_2\}$ are not separated by a 3-vertex cut $S$ from at least one vertex $v$ of the $K_5$:
  - Use the $K_5$ to connect the vertex pairs correctly.
Step 5: If a subdivision of a $K_5$ is found

- If $\{s_1, s_2, t_1, t_2\}$ are not separated by a 3-vertex cut $S$ from at least one vertex $v$ of the $K_5$:
- Use the $K_5$ to connect the vertex pairs correctly.
Step 5: If a subdivision of a $K_5$ is found

- If $\{s_1, s_2, t_1, t_2\}$ are not separated by a 3-vertex cut $S$ from at least one vertex $v$ of the $K_5$:
  - Use the $K_5$ to connect the vertex pairs correctly.
Step 5: If a subdivision of a $K_5$ is found

- If $\{s_1, s_2, t_1, t_2\}$ are not separated by a 3-vertex cut $S$ from at least one vertex $v$ of the $K_5$:
  - Use the $K_5$ to connect the vertex pairs correctly.
Step 5: If a subdivision of a $K_5$ is found

If $\{s_1, s_2, t_1, t_2\}$ are not separated by a 3-vertex cut $S$ from at least one vertex $v$ of the $K_5$:

- Use the $K_5$ to connect the vertex pairs correctly.
Solving the 2-VDPP (Shiloach’s algorithm)

Step 6: If a subdivision of a $K_{3,3}$ is found

- Similar to the case of a $K_5$. 
Known results

Knuth (1974), Lynch (1975)

The DPP is $\mathcal{NP}$-hard.
Knuth (1974), Lynch (1975)

The DPP is \( \mathcal{NP} \)-hard.

Robertson and Seymour (1995)

The \( k \)-DPP on undirected graphs can be solved in polynomial time for all fixed \( k \in \mathbb{N} \).
<table>
<thead>
<tr>
<th>Known results</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Knuth (1974), Lynch (1975)</strong></td>
</tr>
<tr>
<td>The DPP is $\mathcal{NP}$-hard.</td>
</tr>
<tr>
<td><strong>Robertson and Seymour (1995)</strong></td>
</tr>
<tr>
<td>The $k$-DPP on undirected graphs can be solved in polynomial time for all fixed $k \in \mathbb{N}$.</td>
</tr>
<tr>
<td><strong>Fortune, Hopcroft and Wyllie (1980)</strong></td>
</tr>
<tr>
<td>The $k$-DPP on directed graphs is $\mathcal{NP}$-hard for all $k \geq 2$.</td>
</tr>
</tbody>
</table>
A directed acyclic graph (dag) is a directed graph $G(V, E)$ such that there is a mapping $	au : V \rightarrow \{1, \ldots, |V|\}$ with 

$$\tau(v) \leq \tau(w) \text{ for all } (v, w) \in E.$$ 

$	au$ is called a topological numbering of $G$. 
A directed acyclic graph (dag)
is a directed graph $G(V, E)$ such that there is a mapping
$\tau : V \to \{1, \ldots, |V|\}$ with

$$\tau(v) \leq \tau(w) \text{ for all } (v, w) \in E.$$ 

$\tau$ is called a topological numbering of $G$. 

\begin{center}
\begin{tikzpicture}
\node[circle, draw] (1) at (0,2) {1};
\node[circle, draw] (2) at (0,0) {2};
\node[circle, draw] (3) at (1,-1) {3};
\node[circle, draw] (4) at (2,0) {4};
\node[circle, draw] (5) at (3,2) {5};
\node[circle, draw] (6) at (2,0) {6};
\draw (1) -- (2);
\draw (1) -- (3);
\draw (1) -- (4);
\draw (3) -- (2);
\draw (3) -- (4);
\draw (4) -- (6);
\end{tikzpicture}
\end{center}
The k-DPP on directed acyclic graphs
motivation:
The k-DPP on directed acyclic graphs

motivation:
The k-DPP on directed acyclic graphs

motivation:
The k-DPP on directed acyclic graphs

motivation:
The k-DPP on directed acyclic graphs

motivation:
The k-DPP on directed acyclic graphs

motivation:
The k-DPP on directed acyclic graphs
The k-DPP on directed acyclic graphs
The k-DPP on directed acyclic graphs

Torsten Tholey
Universität Augsburg
The k-DPP on directed acyclic graphs
The k-DPP on directed acyclic graphs

$G$

$G'$
The k-DPP on directed acyclic graphs

$G$

$G'$

Torsten Tholey
Universität Augsburg
The k-DPP on directed acyclic graphs

$G$

$G'$
The k-DPP on directed acyclic graphs

$G$

1 -> 3 -> 4
2 -> 3
3 -> 4
3 -> 5
3 -> 6

$G'$

1,2 -> 1,3 -> 1,6
1,2 -> 4,3

1,2 -> 5,1
1,2 -> 5,2
1,2 -> 5,3
1,2 -> 5,4
1,2 -> 6,5

1,2 -> 1,5
1,2 -> 2,1
1,2 -> 2,5
1,2 -> 3,1
1,2 -> 3,5
1,2 -> 4,2
1,2 -> 4,6
1,2 -> 5,6
1,2 -> 6,2
1,2 -> 6,3
1,2 -> 6,4
1,2 -> 1,4

Torsten Tholey
Universität Augsburg
Observation

There are $k$ vertex-disjoint paths $p_i : s_i \rightarrow t_i$ of total length $d$ in $G$ if and only if there is a path $p : (s_1, \ldots, s_k) \rightarrow (t_1, \ldots, t_k)$ of length $d$ in $G'$. 
Observation

- There are $k$ vertex-disjoint paths $p_i : s_i \to t_i$ of total length $d$ in $G$ if and only if there is a path $p : (s_1, \ldots, s_k) \to (t_1, \ldots, t_k)$ of length $d$ in $G'$.
- $G'$ has $O(mn^{k-1})$ edge and $O(n^k)$ vertices and can be constructed in $O(mn^{k-1})$ time.
Observation

- There are $k$ vertex-disjoint paths $p_i : s_i \rightarrow t_i$ of total length $d$ in $G'$ if and only if there is a path $p : (s_1, \ldots, s_k) \rightarrow (t_1, \ldots, t_k)$ of length $d$ in $G'$.
- $G'$ has $O(mn^{k-1})$ edge and $O(n^k)$ vertices and can be constructed in $O(mn^{k-1})$ time.

Theorem

Disjoint paths of shortest total length solving the $k$-VDPP can be computed in $O(mn^{k-1})$ time.
Observation

- There are $k$ vertex-disjoint paths $p_i : s_i \rightarrow t_i$ of total length $d$ in $G$
  if and only if there is a path $p : (s_1, \ldots, s_k) \rightarrow (t_1, \ldots, t_k)$ of length $d$ in $G'$.
- $G'$ has $O(mn^{k-1})$ edge and $O(n^k)$ vertices and can be constructed in $O(mn^{k-1})$ time.

Theorem

Disjoint paths of shortest total length solving the $k$-VDPP can be computed in $O(mn^{k-1})$ time.

Remark

- The result can be extended to weighted graphs with a running time linear in the computation of a shortest weighted path in graph with $O(n^k)$ vertices and $O(mn^{k-1})$ edges.
The algorithm of Lucchesi and Giglio

1. Delete all ingoing edges of $s_i$ and outgoing edges of $t_i$. 
The algorithm of Lucchesi and Giglio

1. Delete all ingoing edges of \( s_i \) and outgoing edges of \( t_i \).
2. Delete all vertices \( v \not\in \{s_1, s_2\} \) with \( \text{indeg}(v) = 0 \) and all \( v \not\in \{t_1, t_2\} \) with \( \text{outdeg}(v) = 0 \).
The algorithm of Lucchesi and Giglio

1. Delete all ingoing edges of $s_i$ and outgoing edges of $t_i$.
2. Delete all vertices $v \not\in \{s_1, s_2\}$ with $\text{indeg}(v) = 0$ and all $v \not\in \{t_1, t_2\}$ with $\text{outdeg}(v) = 0$. 

Diagram: 

- Nodes: $s_1, s_2, t_1, t_2$ 
- Edges: Incoming to $s_1$, outgoing from $s_1$ to $s_2$, incoming to $s_2$, outgoing from $s_2$ to $t_1$, incoming to $t_1$, outgoing from $t_1$ to $t_2$, incoming to $t_2$ 
- Red X: Node $s_1$ with outgoing edge to $s_2$ 

Torsten Tholey
Universität Augsburg
The 2-VDPP: the algorithm of Lucchesi and Giglio

1. Delete all ingoing edges of $s_i$ and outgoing edges of $t_i$.
2. Delete all vertices $v \notin \{s_1, s_2\}$ with $\text{indeg}(v) = 0$ and all $v \notin \{t_1, t_2\}$ with $\text{outdeg}(v) = 0$. 

Torsten Tholey
Universität Augsburg
The algorithm of Lucchesi and Giglio

1. Delete all ingoing edges of $s_i$ and outgoing edges of $t_i$.

2. Delete all vertices $v \not\in \{s_1, s_2\}$ with indeg$(v) = 0$ and all $v \not\in \{t_1, t_2\}$ with outdeg$(v) = 0$.

3. Delete all vertices $v \not\in \{t_1, t_2\}$ with indeg$(v) = 1$ or outdeg$(v) = 1$ by edge contractions.
### The algorithm of Lucchesi and Giglio

1. Delete all ingoing edges of $s_i$ and outgoing edges of $t_i$.

2. Delete all vertices $v \not\in \{s_1, s_2\}$ with $\text{indeg}(v) = 0$ and all $v \not\in \{t_1, t_2\}$ with $\text{outdeg}(v) = 0$.

3. Delete all vertices $v \not\in \{t_1, t_2\}$ with $\text{indeg}(v) = 1$ or $\text{outdeg}(v) = 1$ by edge contractions.

![Diagram](image)
The algorithm of Lucchesi and Giglio

Observation:
For every pair of different vertices $x, y$ there are
- two disjoint paths from $\{x, y\}$ to $\{t_1, t_2\}$,
Observation:
For every pair of different vertices $x, y$ there are
- two disjoint paths from $\{x, y\}$ to $\{t_1, t_2\}$,
Oberservation:
For every pair of different vertices $x, y$ there are
- two disjoint paths from $\{x, y\}$ to $\{t_1, t_2\}$,
- two disjoint paths from $\{s_1, s_2\}$ to $\{x, y\}$.
The algorithm of Lucchesi and Giglio

4 Construct disjoint paths \( p_1 : s_1 \rightarrow t_1 \) and \( p_2 : s_2 \rightarrow t_2 \) ignoring edge direction.
Definition

A vertex with two incoming or two outgoing edges is called a switch.
The 2-VDPP: the algorithm of Lucchesi and Giglio

The algorithm of Lucchesi and Giglio

5. Remove the switches:
   - $u_i$: switch with the smallest topological number on $p_i$.
   - $v_i$: switch with the smallest topological number on $p_i$.
   - Construct disjoint paths in $G$ connecting $\{s_1, s_2\}$ with $\{u_1, u_2\}$.
   - Construct disjoint paths in $G$ connecting $\{v_1, v_2\}$ with $\{t_1, t_2\}$.
   - Use these paths to remove the switch as shown on the slides.
A switch with the smallest topological number on

Fall 1

\[
\begin{align*}
{s_1} & \rightarrow {u_1} & \leftarrow {v_1} & \rightarrow {t_1} \\
{s_2} & \rightarrow {u_2} & \leftarrow {v_2} & \rightarrow {t_2}
\end{align*}
\Rightarrow
\begin{align*}
{s_1} & \rightarrow {u_1} & \leftarrow {v_1} & \rightarrow {t_1} \\
{s_2} & \rightarrow {u'_2} & \leftarrow {v_2} & \rightarrow {t'}
\end{align*}
\]

Fall 2

\[
\begin{align*}
{s_1} & \rightarrow {u_1} & \leftarrow {v_1} & \rightarrow {t_1} \\
{s_2} & \rightarrow {u_2} & \leftarrow {v_2} & \rightarrow {t_2}
\end{align*}
\Rightarrow
\begin{align*}
{s_1} & \rightarrow {u_1} & \leftarrow {v_1} & \rightarrow {t_1} \\
{s_2} & \rightarrow {u_2} & \leftarrow {v_2} & \rightarrow {t_2}
\end{align*}
\]

Fall 3

\[
\begin{align*}
{s_1} & \rightarrow {u_1} & \leftarrow {v_1} & \rightarrow {t_1} \\
{s_2} & \rightarrow {u_2} & \leftarrow {v_2} & \rightarrow {t_2}
\end{align*}
\Rightarrow
\begin{align*}
{s_1} & \rightarrow {u_1} & \leftarrow {v_1} & \rightarrow {t_1} \\
{s_2} & \rightarrow {u'_2} & \leftarrow {v_1} & \rightarrow {t'}
\end{align*}
\]
Conclusion

Observation

- The running time is dominated by the removal of the switches.
- We have to remove at most $n$ switches.
- For each of them four paths have to be constructed in $O(m + n)$ time.

$\Rightarrow$ Lemma: The whole running time is $O(n(m + n))$. 
We use a data structure that allows us:
To compute for fixed $s_1, s_2$ and any pair of vertices $t_1, t_2$:
paths $p_1, p_2$ connecting $\{s_1, s_2\}$ and $\{t_1, t_2\}$ arbitrarily.
to predict in $O(1)$ time which pairs a connected.
Delay the paths replacements until the very end.
The 2-VDPP: A new Approach

Dominator tree

Torsten Tholey
Universität Augsburg
The 2-VDPP: A new Approach

Dominator tree
The 2-VDPP: A new Approach

Dominator tree

Torsten Tholey  Universität Augsburg
The 2-VDPP: A new Approach

Torsten Tholey
Universität Augsburg
The 2-VDPP: A new Approach

Dominator tree

Shortest-Paths tree

Torsten Tholey
Universität Augsburg
The 2-VDPP: A new Approach

Dominator tree

Shortest-Paths tree

Torsten Tholey
Universität Augsburg
The 2-VDPP: A new Approach

Dominator tree

Shortest-Paths tree

Torsten Tholey
Universität Augsburg
The 2-VDPP: A new Approach

Dominator tree

Shortest-Paths tree

Torsten Tholey

Universität Augsburg
The 2-VDPP: A new Approach

Dominator tree

Shortest-Paths tree

Torsten Tholey
Universität Augsburg
The 2-VDPP: A new Approach

Dominator tree

Shortest-Paths tree

Torsten Tholey
Universität Augsburg
The 2-VDPP: A new Approach

Dominator tree

Shortest-Paths tree

Torsten Tholey
Universität Augsburg
Thank You!