A relaxation of the Directed Disjoint Paths problem: a global congestion metric helps

R. Lopes [1,2]   I. Sau [2]

[1]Universidade Federal do Ceará, Brazil
[2]LIRMM, Université de Montpellier, France

August, 2020

XP and FPT

Problem $\mathcal{P}$ of size n with a parameter k:

- $\mathcal{P} \in \mathsf{XP} \implies \mathcal{P}$ can be solved in $\mathcal{O}\left(g(k) \cdot n^{f(k)}\right)$.

Examples

XP and FPT

Problem $\mathcal{P}$ of size n with a parameter k:

- $\mathcal{P} \in \mathsf{XP} \implies \mathcal{P}$ can be solved in $\mathcal{O}\left(g(k) \cdot n^{f(k)}\right)$.

Examples

$$\mathcal{O}\left(n^{2k}\right)$$

XP and FPT

Problem $\mathcal{P}$ of size n with a parameter k:

- $\mathcal{P} \in \mathrm{XP} \implies \mathcal{P}$ can be solved in $\mathcal{O}\left(g(k) \cdot n^{f(k)}\right)$.

Examples

$$\mathcal{O}\left(n^{2k}\right)$$

Poly time for fixed k.

XP and FPT

Problem $\mathcal{P}$ of size n with a parameter k:

- $\mathcal{P} \in \text{XP} \implies \mathcal{P}$ can be solved in $\mathcal{O}\left(g(k) \cdot n^{f(k)}\right)$.
- $\mathcal{P} \in \text{FPT} \implies \mathcal{P}$ can be solved in $\mathcal{O}\left(f(k) \cdot n^{c}\right)$.

Examples

$$\mathcal{O}\left(n^{2k}\right)$$

Poly time for fixed k.

XP and FPT

Problem $\mathcal{P}$ of size n with a parameter k:

- $\mathcal{P} \in \mathrm{XP} \implies \mathcal{P}$ can be solved in $\mathcal{O}\left(g(k) \cdot n^{f(k)}\right)$.
- $\mathcal{P} \in \mathrm{FPT} \implies \mathcal{P}$ can be solved in $\mathcal{O}\left(f(k) \cdot n^{c}\right)$.

Examples

$$\mathcal{O}\left(n^{2k}\right) \qquad\qquad \mathcal{O}\left(2^k \cdot n^2\right)$$

Poly time for fixed k.

XP and FPT

Problem $\mathcal{P}$ of size n with a parameter k:

- $\mathcal{P} \in \mathrm{XP} \implies \mathcal{P}$ can be solved in $\mathcal{O}\left(g(k) \cdot n^{f(k)}\right)$.
- $\mathcal{P} \in \mathrm{FPT} \implies \mathcal{P}$ can be solved in $\mathcal{O}\left(f(k) \cdot n^{c}\right)$.

Examples

$$\mathcal{O}\left(n^{2k}\right)$$

Poly time for fixed k.

$$\mathcal{O}\left(2^{k} \cdot n^{2}\right)$$

Poly exponent independent of k.

XP and FPT

Problem $\mathcal{P}$ of size n with a parameter k:

- $\mathcal{P} \in \text{XP} \implies \mathcal{P}$ can be solved in $\mathcal{O}\left(g(k) \cdot n^{f(k)}\right)$.
- $\mathcal{P} \in \text{FPT} \implies \mathcal{P}$ can be solved in $\mathcal{O}\left(f(k) \cdot n^{c}\right)$.
- $\mathcal{P}$ is W[1]-hard $\implies$ strong evidence that $\mathcal{P} \notin \text{FPT}$.

Examples

$$\mathcal{O}\left(n^{2k}\right)$$

Poly time for fixed k.

$$\mathcal{O}\left(2^{k} \cdot n^{2}\right)$$

Poly exponent independent of k.

XP and FPT

Problem $\mathcal{P}$ of size n with a parameter k:

- $\mathcal{P} \in \text{XP} \implies \mathcal{P}$ can be solved in $\mathcal{O}\left(g(k) \cdot n^{f(k)}\right)$.
- $\mathcal{P} \in \text{FPT} \implies \mathcal{P}$ can be solved in $\mathcal{O}\left(f(k) \cdot n^{c}\right)$.
- $\mathcal{P}$ is W[1]-hard $\implies$ strong evidence that $\mathcal{P} \notin \text{FPT}$.
- k-Clique $\in$ XP and is W[1]-hard.

Examples

$$\mathcal{O}\left(n^{2k}\right)$$

Poly time for fixed k.

$$\mathcal{O}\left(2^{k} \cdot n^{2}\right)$$

Poly exponent independent of k.

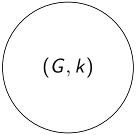
Kernels

Kernelization algorithm:

Kernels

Kernelization algorithm:

Input: Instance (G, k) of $\mathcal{P}$

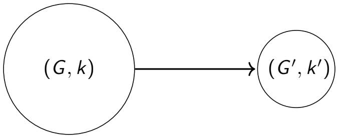


Kernels

Kernelization algorithm:

Input: Instance (G, k) of $\mathcal{P}$

Output: In polynomial time, an instance (G', k') of $\mathcal{P}$ s.t.:



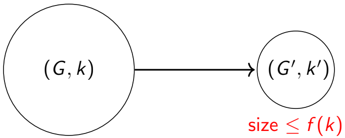
Kernels

Kernelization algorithm:

Input: Instance (G, k) of $\mathcal{P}$

Output: In polynomial time, an instance (G', k') of $\mathcal{P}$ s.t.:

- $|G'| + k' \leq f(k)$.



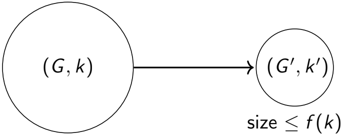
Kernels

Kernelization algorithm:

Input: Instance (G, k) of $\mathcal{P}$

Output: In polynomial time, an instance (G', k') of $\mathcal{P}$ s.t.:

- $|G'| + k' \leq f(k)$.
- $(G, k) \iff (G', k')$.



Kernels

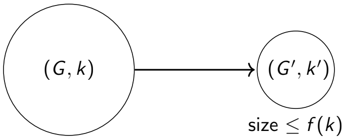
Kernelization algorithm:

Input: Instance (G, k) of $\mathcal{P}$

Output: In polynomial time, an instance (G', k') of $\mathcal{P}$ s.t.:

- $|G'| + k' \leq f(k)$.
- $(G, k) \iff (G', k')$.

- $f(k)$ polynomial $\implies$ Polynomial kernel for $\mathcal{P}$.



Kernels

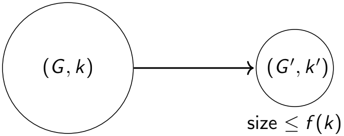
Kernelization algorithm:

Input: Instance (G, k) of $\mathcal{P}$

Output: In polynomial time, an instance (G', k') of $\mathcal{P}$ s.t.:

- $|G'| + k' \leq f(k)$.
- $(G, k) \iff (G', k')$.

- $f(k)$ polynomial $\implies$ Polynomial kernel for $\mathcal{P}$.



Theorem

A parameterized problem $\mathcal{P}$ has a kernel $\iff$ $\mathcal{P}$ is FPT.

Routing problems

In a digraph D:

Input: Set of *requests* $I = \{(s_1, t_1), \ldots, (s_k, t_k)\}$.

Routing problems

In a digraph D:

Input: Set of *requests* $I = \{(s_1, t_1), \ldots, (s_k, t_k)\}$.

Output: Collection of paths $\mathcal{P} = \{P_1, \ldots, P_k\}$ *satisfying* I under some *restriction*.

Routing problems

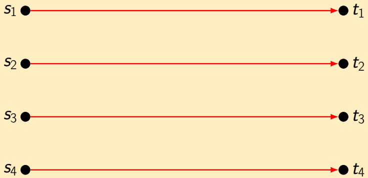
In a digraph D:

Input: Set of *requests* $I = \{(s_1, t_1), \ldots, (s_k, t_k)\}$.

Output: Collection of paths $\mathcal{P} = \{P_1, \ldots, P_k\}$ *satisfying* I under some *restriction*.

Pairwise vertex-disjoint paths.

$\Longrightarrow$ k-Directed Disjoint Paths (k-DDP).



Routing problems

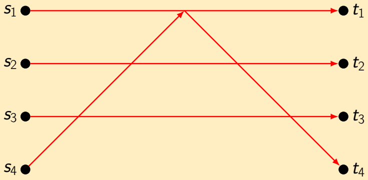
In a digraph D:

Input: Set of *requests* $I = \{(s_1, t_1), \ldots, (s_k, t_k)\}$.

Output: Collection of paths $\mathcal{P} = \{P_1, \ldots, P_k\}$ *satisfying* I under some *restriction*.

Every vertex in $\leq c$ paths.

$\Longrightarrow$ (k, c)-Directed Disjoint Paths with Congestion ((k, c)-DDPC).



$c = 2$ in the example.

Routing problems

In a digraph D:

Input: Set of *requests* $I = \{(s_1, t_1), \dots, (s_k, t_k)\}$.

Output: Collection of paths $\mathcal{P} = \{P_1, \dots, P_k\}$ *satisfying* I under some *restriction*.

Every vertex in $\leq c$ paths.

$\implies$ (k, c)-Directed Disjoint Paths with Congestion ((k, c)-DDPC).

k-DDP

- NP-complete for $k = 2$.

S. Fortune and J.E. Hopcroft and J. Wyllie.
The directed subgraph homeomorphism problem
Theoretical Computer Science, 10, 1980

k-DDP

- NP-complete for $k = 2$.
- XP in DAGs.

S. Fortune and J.E. Hopcroft and J. Wyllie.
The directed subgraph homeomorphism problem
Theoretical Computer Science, 10, 1980

k-DDP

- NP-complete for $k = 2$.
- XP in DAGs.
- W[1]-hard in DAGs.

A. Slivkins
Parameterized Tractability of Edge-Disjoint Paths on Directed Acyclic Graphs
SIAM Journal on Discrete Mathematics 24.1, 2010

k-DDP

- NP-complete for $k = 2$.
- XP in DAGs.
- W[1]-hard in DAGs.
- XP in bounded dtw (param. $k + \mathrm{dtw}(D)$).

T. Johnson and N. Robertson and P.D. Seymour and R. Thomas,
Directed tree-width
Journal of Combinatorial Theory, Series B, Volume 82, Issue 1, 2001, Pages 138-154

k-DDP

- NP-complete for $k = 2$.
- XP in DAGs.
- W[1]-hard in DAGs.
- XP in bounded dtw (param. $k + \mathrm{dtw}(D)$).

(k, c)-DDPC

- W[1]-hard in DAGs.

Extension of Slivkins reduction, and improved upon.

Amiri, S., Kreutzer, S., Marx, D. and Rabinovich, R.
Routing with congestion in acyclic digraphs
Information Processing Letters (151), 2019

k-DDP

- NP-complete for $k = 2$.
- XP in DAGs.
- W[1]-hard in DAGs.
- XP in bounded dtw (param. $k + \mathrm{dtw}(D)$).

(k, c)-DDPC

- W[1]-hard in DAGs.
- XP in DAGs.

Easy reduction to the disjoint case.

Amiri, S., Kreutzer, S., Marx, D. and Rabinovich, R.
Routing with congestion in acyclic digraphs
Information Processing Letters (151), 2019

k-DDP

- NP-complete for $k = 2$.
- XP in DAGs.
- W[1]-hard in DAGs.
- XP in bounded dtw (param. $k + \mathrm{dtw}(D)$).

(k, c)-DDPC

- W[1]-hard in DAGs.
- XP in DAGs.
- XP in bounded bounded dtw (param. $k + \mathrm{dtw}(D)$).

Easy reduction to the disjoint case.

Amiri, S., Kreutzer, S., Marx, D. and Rabinovich, R.
Routing with congestion in acyclic digraphs
Information Processing Letters (151), 2019

k-DDP (*all hold for edge disjoint*)

- NP-complete for $k = 2$.
- XP in DAGs.
- W[1]-hard in DAGs.
- XP in bounded dtw (param. $k + \mathrm{dtw}(D)$).

(k, c)-DDPC (*all hold for edge congestion*)

- W[1]-hard in DAGs.
- XP in DAGs.
- XP in bounded bounded dtw (param. $k + \mathrm{dtw}(D)$).

Easy reduction to the disjoint case.

Amiri, S., Kreutzer, S., Marx, D. and Rabinovich, R.
Routing with congestion in acyclic digraphs
Information Processing Letters (151), 2019

k-DDP (*all hold for edge disjoint*)

- NP-complete for $k = 2$.
- XP in DAGs.
- W[1]-hard in DAGs.
- XP in bounded dtw (param. $k + \mathrm{dtw}(D)$).

(k, c)-DDPC (*all hold for edge congestion*)

- W[1]-hard in DAGs.
- XP in DAGs.
- XP in bounded bounded dtw (param. $k + \mathrm{dtw}(D)$).

Easy reduction to the disjoint case.

Amiri, S., Kreutzer, S., Marx, D. and Rabinovich, R.
Routing with congestion in acyclic digraphs
Information Processing Letters (151), 2019

- k-DDP is FPT in planar digraphs.

Cygan, M., Marx, D., Pilipczuk, M. and Pilipczuk, M.
The planar directed k-vertex-disjoint paths problem is fixed-parameter tractable
In Proc. of the IEEE 54th Annual Symposium on Foundations of Computer Science (FOCS), 2013

- k-DDP is FPT in planar digraphs.
- k-DDPC is XP in $(36k^3 + 2k)$-stronlgy connected digraphs for $c = 2$.

Edwards, K., Muzi, I. and Wollan, P.
Half-integral linkages in highly connected directed graphs
In Proc. of the 25th Annual European Symposium on Algorithms (ESA), 2017

- k-DDP is FPT in planar digraphs.
- k-DDPC is XP in $(36k^3 + 2k)$-stronlgy connected digraphs for $c = 2$.
- DDPC is XP with param $d = k - c$ in DAGs.

Amiri, S., Kreutzer, S., Marx, D. and Rabinovich, R.
Routing with congestion in acyclic digraphs
Information Processing Letters (151), 2019

Summary of positive results

- Congestion c.
- $d = k - c$.

Version \ Digraph	DAG	dtw $\leq w$	Planar	Strong
DDP	XP on k	XP on $k + w$	FPT on k	$- - -$
DDPC	XP on k, XP on d	XP on $k + w$	FPT on k	XP on k for $c = 2$

Summary of positive results

- Congestion c.
- $d = k - c$.

Version \ Digraph	DAG	dtw $\leq w$	Planar	Strong
DDP	XP on k	XP on $k + w$	FPT on k	$- - -$
DDPC	XP on k, XP on d	XP on $k + w$	FPT on k	XP on k for $c = 2$

- What about *general* digraphs?
- Asymmetric version that either outputs a

Summary of positive results

- Congestion c.
- $d = k - c$.

Version \ Digraph	DAG	dtw $\leq w$	Planar	Strong
DDP	XP on k	XP on $k + w$	FPT on k	$- - -$
DDPC	XP on k, XP on d	XP on $k + w$	FPT on k	XP on k for $c = 2$

- What about *general* digraphs?
- Asymmetric version that either outputs a
  - *positive answer* for DDPC; or

Summary of positive results

- Congestion c.
- $d = k - c$.

Version \ Digraph	DAG	dtw $\leq w$	Planar	Strong
DDP	XP on k	XP on $k + w$	FPT on k	$- - -$
DDPC	XP on k, XP on d	XP on $k + w$	FPT on k	XP on k for $c = 2$

- What about *general* digraphs?
- Asymmetric version that either outputs a
  - *positive answer* for DDPC; or
  - *negative answer* for DDP.

Summary of positive results

- Congestion c.
- $d = k - c$.

Version \ Digraph	DAG	dtw $\leq w$	Planar	Strong
DDP	XP on k	XP on $k + w$	FPT on k	$- - -$
DDPC	XP on k, XP on d	XP on $k + w$	FPT on k	XP on k for $c = 2$

- What about *general* digraphs?
- Asymmetric version that either outputs a
  - *positive answer* for DDPC; or
  - *negative answer* for DDP.
  - XP for $c = 3$ (uses Directed Grid Theorem).

K. Kawarabayashi and S. Kreutzer.
The Directed Grid Theorem
Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing

Summary of positive results

- Congestion c.
- $d = k - c$.

Version \ Digraph	DAG	dtw $\leq w$	Planar	Strong
DDP	XP on k	XP on $k + w$	FPT on k	$- - -$
DDPC	XP on k, XP on d	XP on $k + w$	FPT on k	XP on k for $c = 2$

- What about *general* digraphs?
- Asymmetric version that either outputs a
  - *positive answer* for DDPC; or
  - *negative answer* for DDP.
  - XP for $c = 3$ (uses Directed Grid Theorem).

We bring positive news for *general* digraphs.

The Disjoin Enough Paths problem

- Paths be disjoint in a large part of the digraph.

Input: *Requests* $I = \{(s_1, t_1), \ldots, (s_k, t_k)\}$, integers d and s.

The Disjoin Enough Paths problem

- Paths be disjoint in a large part of the digraph.

Input: *Requests* $I = \{(s_1, t_1), \ldots, (s_k, t_k)\}$, integers d and s.

Output: $\mathcal{P} = \{P_1, \ldots, P_k\}$ *satisfying* I s.t.

The Disjoin Enough Paths problem

- Paths be disjoint in a large part of the digraph.

Input: *Requests* $I = \{(s_1, t_1), \ldots, (s_k, t_k)\}$, integers d and s.

Output: $\mathcal{P} = \{P_1, \ldots, P_k\}$ *satisfying* I s.t.

$\geq d$ vertices occurring (*global* congestion metric).

The Disjoin Enough Paths problem

- Paths be disjoint in a large part of the digraph.

Input: *Requests* $I = \{(s_1, t_1), \ldots, (s_k, t_k)\}$, integers d and s.

Output: $\mathcal{P} = \{P_1, \ldots, P_k\}$ *satisfying* I s.t.

$\geq d$ vertices occurring (*global* congestion metric).

in $\leq s$ *paths* of the collection (*local* congestion metric).

The Disjoin Enough Paths problem

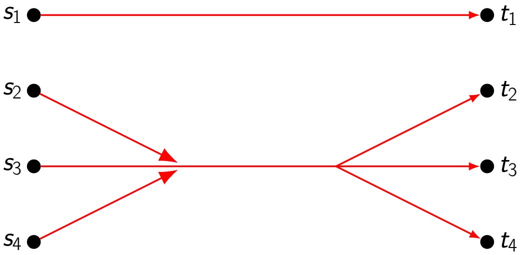
- Paths be disjoint in a large part of the digraph.

Input: *Requests* $I = \{(s_1, t_1), \ldots, (s_k, t_k)\}$, integers d and s.

Output: $\mathcal{P} = \{P_1, \ldots, P_k\}$ *satisfying* I s.t.

$\geq d$ vertices occurring (*global* congestion metric).

in $\leq s$ *paths* of the collection (*local* congestion metric).



The Disjoin Enough Paths problem

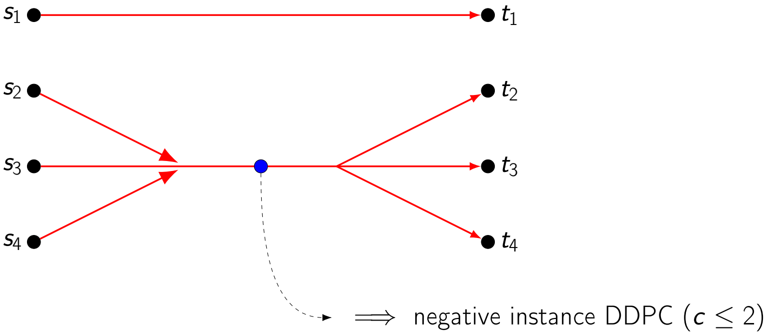
- Paths be disjoint in a large part of the digraph.

Input: *Requests* $I = \{(s_1, t_1), \ldots, (s_k, t_k)\}$, integers d and s.

Output: $\mathcal{P} = \{P_1, \ldots, P_k\}$ *satisfying* I s.t.

$\geq d$ vertices occurring (*global* congestion metric).

in $\leq s$ *paths* of the collection (*local* congestion metric).



The Disjoin Enough Paths problem

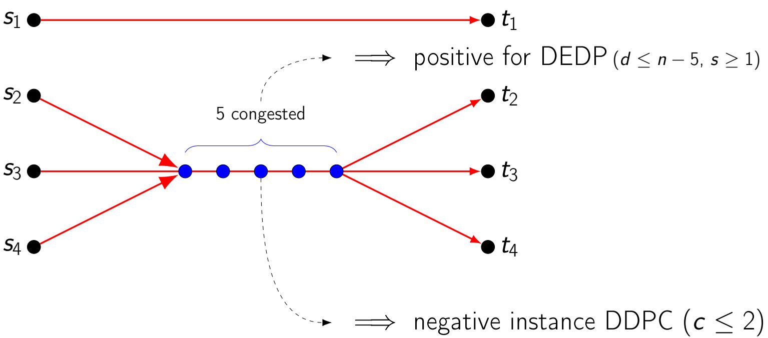
- Paths be disjoint in a large part of the digraph.

Input: *Requests* $I = \{(s_1, t_1), \ldots, (s_k, t_k)\}$, integers d and s.

Output: $\mathcal{P} = \{P_1, \ldots, P_k\}$ *satisfying* I s.t.

$\geq d$ vertices occurring (*global* congestion metric).

in $\leq s$ *paths* of the collection (*local* congestion metric).



Input: *Requests* $I = \{(s_1, t_1), \ldots, (s_k, t_k)\}$, integers d and s.

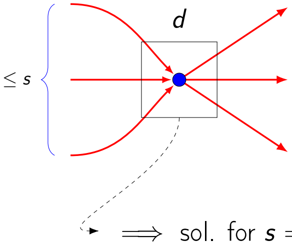
Output: $\mathcal{P} = \{P_1, \ldots, P_k\}$ *satisfying* I s.t.

$\geq d$ vertices occurring in $\leq s$ *paths*.

Input: *Requests* $I = \{(s_1, t_1), \ldots, (s_k, t_k)\}$, integers d and s.

Output: $\mathcal{P} = \{P_1, \ldots, P_k\}$ *satisfying* I s.t.

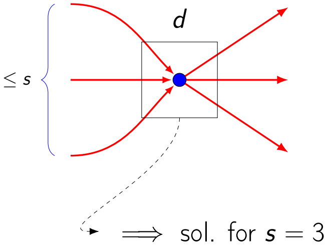
$\geq d$ vertices occurring in $\leq s$ *paths*.



$\Longrightarrow$ sol. for $s = 3$

Input: *Requests* $I = \{(s_1, t_1), \ldots, (s_k, t_k)\}$, integers d and s.

Output: $\mathcal{P} = \{P_1, \ldots, P_k\}$ *satisfying* I s.t.
$\geq d$ vertices occurring in $\leq s$ *paths*.

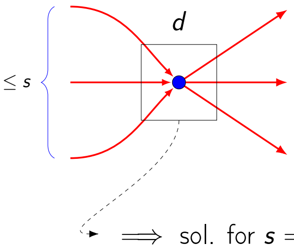


- $d = n, s = 1 \implies$ paths **d**isjoint in $V(D) \implies$ DDP.
- $d = n, s \geq 2 \implies$ congestion s in $V(D) \implies$ DDPC with congestion s.

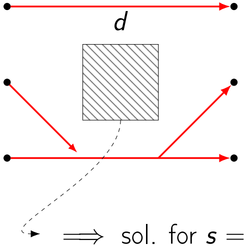
Input: *Requests* $I = \{(s_1, t_1), \ldots, (s_k, t_k)\}$, integers d and s.

Output: $\mathcal{P} = \{P_1, \ldots, P_k\}$ *satisfying* I s.t.

$\geq d$ vertices occurring in $\leq s$ *paths*.



$\Longrightarrow$ sol. for $s = 3$



$\Longrightarrow$ sol. for $s = 0$

- $d = n, s = 1 \implies$ paths **d**isjoint in $V(D) \implies$ DDP.
- $d = n, s \geq 2 \implies$ congestion s in $V(D) \implies$ DDPC with congestion s.
- $s = 0 \implies$ paths avoiding d vertices $\implies$ Steiner Network.

Summary of results

for $0 < \alpha \leq 1$:

Hardness results

- NP-complete for $k \geq 3$, $d = n^{\alpha}$, fixed $s \geq 1$.
- W[1]-hard in DAGs with param. k, $d = n^{\alpha}$, fixed $s \geq 1$.
- W[1]-hard in DAGs with param. d, fixed $s \geq 0$.

Summary of results

for $0 < \alpha \leq 1$:

Hardness results

- NP-complete for $k \geq 3$, $d = n^\alpha$, fixed $s \geq 1$.
- W[1]-hard in DAGs with param. k, $d = n^\alpha$, fixed $s \geq 1$.
- W[1]-hard in DAGs with param. d, fixed $s \geq 0$.

Positive results (all for *general digraphs*)

- XP with params. k and $\mathrm{dtw}(D)$.
- XP with params. d and s.
- FPT with params. k, d and s.

DEDP instances

Instance (D, I, k, d, s) where:

DEDP instances

Instance (D, I, k, d, s) where:

- $D \longrightarrow$ Input digraph
- $I \longrightarrow$ Set of requests $\{(s_1, t_1), \ldots, (s_k, t_k)\}$
- $k \longrightarrow$ $k = |I|$
- $d \longrightarrow$ Global congestion metric
- $s \longrightarrow$ Local congestion metric

DEDP instances

Instance (D, I, k, d, s) where:

$D \longrightarrow$ Input digraph

$I \longrightarrow$ Set of requests $\{(s_1, t_1), \ldots, (s_k, t_k)\}$

$k \longrightarrow$ $k = |I|$

$d \longrightarrow$ Global congestion metric

$s \longrightarrow$ Local congestion metric

To simplify the presentation, we consider that:

Goal: Find collection of paths *satisfying* I s.t. at least d vertices occur in at most s paths of the collection.

DEDP instances

Instance (D, I, k, d, s) where:

$D \longrightarrow$ Input digraph

$I \longrightarrow$ Set of requests $\{(s_1, t_1), \ldots, (s_k, t_k)\}$

$k \longrightarrow k = |I|$

$d \longrightarrow$ Global congestion metric

$s \longrightarrow$ Local congestion metric

To simplify the presentation, we consider that:

- $s = 1$;

Goal: Find collection of paths *satisfying* I s.t. at least d vertices occur in at most s paths of the collection.

DEDP instances

Instance (D, I, k, d, s) where:

- $D \longrightarrow$ Input digraph
- $I \longrightarrow$ Set of requests $\{(s_1, t_1), \ldots, (s_k, t_k)\}$
- $k \longrightarrow$ $k = |I|$
- $d \longrightarrow$ Global congestion metric
- $s \longrightarrow$ Local congestion metric

To simplify the presentation, we consider that:

- $s = 1$;
- all terminals are distinct;

Goal: Find collection of paths *satisfying* I s.t. at least d vertices occur in at most s paths of the collection.

DEDP instances

Instance (D, I, k, d, s) where:

$D \longrightarrow$ Input digraph

$I \longrightarrow$ Set of requests $\{(s_1, t_1), \ldots, (s_k, t_k)\}$

$k \longrightarrow k = |I|$

$d \longrightarrow$ Global congestion metric

$s \longrightarrow$ Local congestion metric

To simplify the presentation, we consider that:

- $s = 1$;
- all terminals are distinct;
- $d^-(s_i) = 0$ and $d^+(t_i) = 0$.

Goal: Find collection of paths *satisfying* I s.t. at least d vertices occur in at most s paths of the collection.

DEDP is NP-complete for fixed $k = 3$

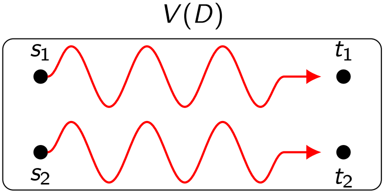
- Reduction from 2-DDP.

DEDP is NP-complete for fixed $k = 3$

- Reduction from 2-DDP.
- $c = n - d \implies \leq c$ vertices in ≥ 2 paths.

DEDP is NP-complete for fixed $k = 3$

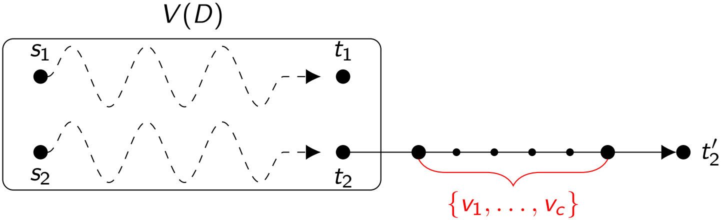
- Reduction from 2-DDP.
- $c = n - d \implies \leq c$ vertices in ≥ 2 paths.



- 2-DDP objective: Find pairwise disjoint paths.

DEDP is NP-complete for fixed $k = 3$

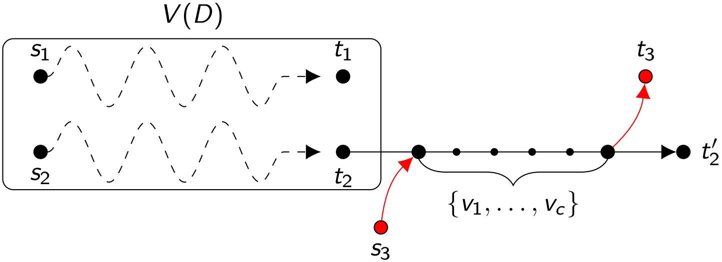
- Reduction from 2-DDP.
- $c = n - d \implies \leq c$ vertices in ≥ 2 paths.



- 2-DDP objective: Find pairwise disjoint paths.
- Add t_2' and c vertices in $t_2 \to t_2'$ path.

DEDP is NP-complete for fixed $k = 3$

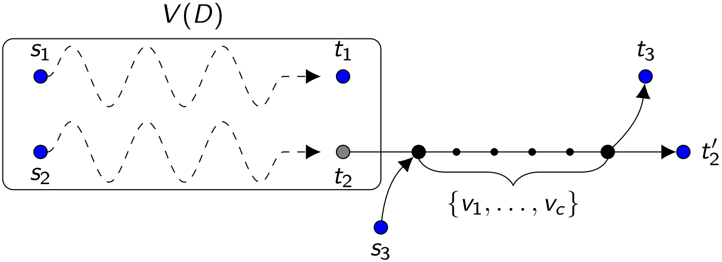
- Reduction from 2-DDP.
- $c = n - d \implies \leq c$ vertices in ≥ 2 paths.



- 2-DDP objective: Find pairwise disjoint paths.
- Add t'_2 and c vertices in $t_2 \to t'_2$ path.
- Add s_3, t_3, edges.

DEDP is NP-complete for fixed $k = 3$

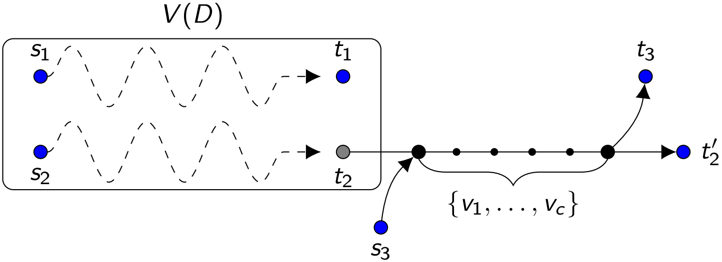
- Reduction from 2-DDP.
- $c = n - d \implies \leq c$ vertices in ≥ 2 paths.



- 2-DDP objective: Find pairwise disjoint paths.
- Add t_2' and c vertices in $t_2 \to t_2'$ path.
- Add s_3, t_3, edges.
- DEDP requests: $\{(s_1, t_1), (s_2, t_2'), (s_3, t_3)\}$.

DEDP is NP-complete for fixed $k = 3$

- Reduction from 2-DDP.
- $c = n - d \implies \leq c$ vertices in ≥ 2 paths.



- 2-DDP objective: Find pairwise disjoint paths.
- Add t_2' and c vertices in $t_2 \to t_2'$ path.
- Add s_3, t_3, edges.
- DEDP requests: $\{(s_1, t_1), (s_2, t_2'), (s_3, t_3)\}$.
- $s_3 \to t_3$ path $\implies$ c congested vertices $\implies$ $V(D) = \{\text{disjoint part}\}$.

Kernelization algorithm overview

Instance (D, I, k, d) ($s = 1$ omitted).

Definition (Congested vertices)

A vertex v is *congested* if v *blocks* ≥ 2 requests. That is, $\exists i, j$ s.t. $i \neq j$ and there is no path from s_i to t_i and no path from s_j to t_j in $D \setminus \{v\}$.

Goal: Find paths satisfying I and $|X| \geq d$ s.t. all $v \in X$ are not congested.

Kernelization algorithm overview

Instance (D, I, k, d) ($s = 1$ omitted).

Definition (Congested vertices)

A vertex v is *congested* if v *blocks* ≥ 2 requests. That is, $\exists i, j$ s.t. $i \neq j$ and there is no path from s_i to t_i and no path from s_j to t_j in $D \setminus \{v\}$.

Goal: Find paths satisfying I and $|X| \geq d$ s.t. all $v \in X$ are not congested.

Step 1: Compute *clean* (free of congested vertices) instance from original.

Kernelization algorithm overview

Instance (D, I, k, d) ($s = 1$ omitted).

Definition (Congested vertices)

A vertex v is *congested* if v *blocks* ≥ 2 requests. That is, $\exists i, j$ s.t. $i \neq j$ and there is no path from s_i to t_i and no path from s_j to t_j in $D \setminus \{v\}$.

Goal: Find paths satisfying I and $|X| \geq d$ s.t. all $v \in X$ are not congested.

Step 1: Compute *clean* (free of congested vertices) instance from original.

Step 2: Reduce *clean* instance with k requests to *clean* instance with $k - 1$ requests (Iteration).

Kernelization algorithm overview

Instance (D, I, k, d) ($s = 1$ omitted).

Definition (Congested vertices)

A vertex v is *congested* if v *blocks* ≥ 2 requests. That is, $\exists i, j$ s.t. $i \neq j$ and there is no path from s_i to t_i and no path from s_j to t_j in $D \setminus \{v\}$.

Goal: Find paths satisfying I and $|X| \geq d$ s.t. all $v \in X$ are not congested.

Step 1: Compute *clean* (free of congested vertices) instance from original.

Step 2: Reduce *clean* instance with k requests to *clean* instance with $k - 1$ requests (Iteration).

Step 3: Solve *large clean* instances when $k = 2$ (Base).

Kernelization algorithm overview

Instance (D, I, k, d) ($s = 1$ omitted).

Definition (Congested vertices)

A vertex v is *congested* if v *blocks* ≥ 2 requests. That is, $\exists i, j$ s.t. $i \neq j$ and there is no path from s_i to t_i and no path from s_j to t_j in $D \setminus \{v\}$.

Goal: Find paths satisfying I and $|X| \geq d$ s.t. all $v \in X$ are not congested.

Step 1: Compute *clean* (free of congested vertices) instance from original.

Step 2: Reduce *clean* instance with k requests to *clean* instance with $k - 1$ requests (Iteration).

Step 3: Solve *large clean* instances when $k = 2$ (Base).

Step 4: Use Steps 2 and 3 to solve *clean* instances with $n \geq f(k, d)$.

Bypassing and clean instances.

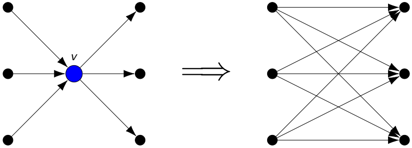
- Instance (D, I, k, d) of DEDP, $I = \{(s_1, t_1), \ldots, (s_k, t_k)\}$.

Bypassing and clean instances.

- Instance (D, I, k, d) of DEDP, $I = \{(s_1, t_1), \ldots, (s_k, t_k)\}$.

Bypassing v

Delete v, add all arcs $N^-(v) \to N^+(v)$.

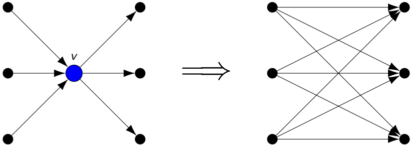


Bypassing and clean instances.

- Instance (D, I, k, d) of DEDP, $I = \{(s_1, t_1), \dots, (s_k, t_k)\}$.

Bypassing v

Delete v, add all arcs $N^-(v) \to N^+(v)$.
Bypass $X \implies$ bypass *all* vertices in X. Denoted by D/X.



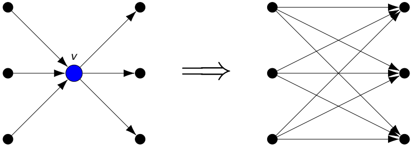
- Can't generate new congested vertices.

Bypassing and clean instances.

- Instance (D, I, k, d) of DEDP, $I = \{(s_1, t_1), \ldots, (s_k, t_k)\}$.

Bypassing v

Delete v, add all arcs $N^-(v) \to N^+(v)$.
Bypass $X \implies$ bypass *all* vertices in X. Denoted by D/X.



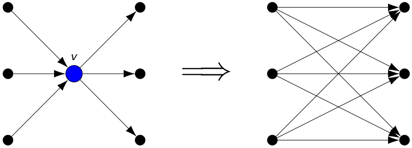
- Can't generate new congested vertices.
- Order is not important.

Bypassing and clean instances.

- Instance (D, I, k, d) of DEDP, $I = \{(s_1, t_1), \ldots, (s_k, t_k)\}$.

Bypassing v

Delete v, add all arcs $N^-(v) \to N^+(v)$.
Bypass $X \implies$ bypass *all* vertices in X. Denoted by D/X.



- Can't generate new congested vertices.
- Order is not important.
- Sol. for $(D/X, I, k, d)$ is sol. for (D, I, k, d).

Step 1

Bypass all *congested* vertices to generate *clean* instance.

Clean instance $k \to k-1$

- $n = |V(D) - \{\text{sources}\} - \{\text{terminals}\}|$.
- Clean instance, $s = 1$, $d^-(s_i) = d^+(t_i) = 0$, all terminals distinct.

Clean instance $k \to k-1$

- $n = |V(D) - \{\text{sources}\} - \{\text{terminals}\}|$.
- Clean instance, $s = 1$, $d^-(s_i) = d^+(t_i) = 0$, all terminals distinct.

- $B_i = \{v \mid (s_i, t_i)$ is *blocked* by $v\}$.

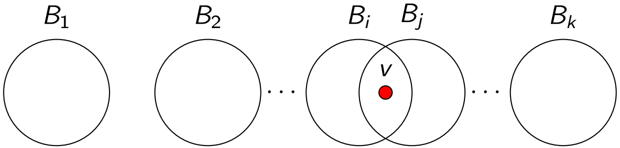
Clean instance $k \to k-1$

- $n = |V(D) - \{\text{sources}\} - \{\text{terminals}\}|$.
- Clean instance, $s = 1$, $d^-(s_i) = d^+(t_i) = 0$, all terminals distinct.

- $B_i = \{v \mid (s_i, t_i) \text{ is } \textit{blocked} \text{ by } v\}$.
- $|B_i| \geq \frac{n}{k} + 1, \forall i$

Clean instance $k \to k-1$

- $n = |V(D) - \{\text{sources}\} - \{\text{terminals}\}|$.
- Clean instance, $s = 1$, $d^-(s_i) = d^+(t_i) = 0$, all terminals distinct.

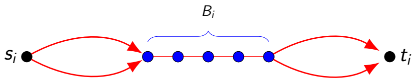


- $B_i = \{v \mid (s_i, t_i) \text{ is } \textit{blocked} \text{ by } v\}$.
- $|B_i| \geq \frac{n}{k} + 1, \forall i$

$\implies \sum_{i \in [k]} |B_i| \geq n + 1 \implies \exists$ *congested* $v \implies$ ctd.

Clean instance $k \to k-1$

- $n = |V(D) - \{\text{sources}\} - \{\text{terminals}\}|$.
- Clean instance, $s = 1$, $d^-(s_i) = d^+(t_i) = 0$, all terminals distinct.



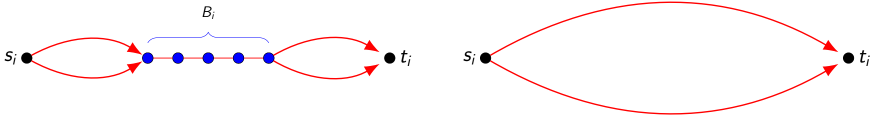
- $B_i = \{v \mid (s_i, t_i) \text{ is } \textit{blocked} \text{ by } v\}$.
- $|B_i| \geq \frac{n}{k} + 1, \forall i$

$$\implies \sum_{i \in [k]} |B_i| \geq n + 1 \implies \exists \textit{ congested } v \implies \text{ctd.}$$

- $\exists |B_i| \leq \frac{n}{k}$.

Clean instance $k \to k-1$

- $n = |V(D) - \{\text{sources}\} - \{\text{terminals}\}|$.
- Clean instance, $s = 1$, $d^-(s_i) = d^+(t_i) = 0$, all terminals distinct.



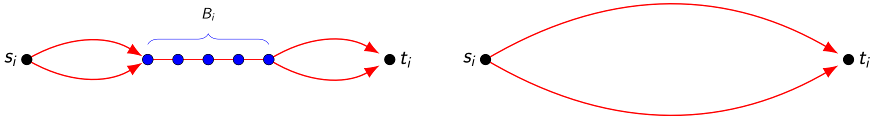
- $B_i = \{v \mid (s_i, t_i)$ is *blocked* by $v\}$.
- $|B_i| \geq \frac{n}{k} + 1, \forall i$

$$\implies \sum_{i \in [k]} |B_i| \geq n + 1 \implies \exists \textit{ congested } v \implies \text{ctd.}$$

- $\exists |B_i| \leq \frac{n}{k}$.
  - Bypass $B_i \implies$ 2 internally disjoint $s_i \to t_i$ paths.

Clean instance $k \to k-1$

- $n = |V(D) - \{\text{sources}\} - \{\text{terminals}\}|$.
- Clean instance, $s = 1$, $d^-(s_i) = d^+(t_i) = 0$, all terminals distinct.



- $B_i = \{v \mid (s_i, t_i)$ is *blocked* by $v\}$.
- $|B_i| \geq \frac{n}{k} + 1, \forall i$

$$\implies \sum_{i \in [k]} |B_i| \geq n + 1 \implies \exists \textit{ congested } v \implies \text{ctd.}$$

- $\exists |B_i| \leq \frac{n}{k}$.
  - Bypass $B_i \implies 2$ internally disjoint $s_i \to t_i$ paths.
  - Take shortest P_i from $s_i \to t_i$.
- $|V(D/B_i)| \geq \frac{n(k-1)}{k} \implies |V(D/(B_i \cup P_i))| \geq \frac{n(k-1)}{2k}$.

Large clean instances

Lemma (Step 2 (Iteration))

(D, I, k, d) clean $\implies \exists P_i$ from $s_i \to t_i$ s.t. $|V(D/(B_i \cup P_i)| \geq \frac{n(k-1)}{2k}$.

Lemma (Step 3 (Base))

(D, I, k, d) clean, $k = 2$, $n \geq 4d \implies$ positive instance, solution in polynomial time.

- For large enough n :
  - Iterate Step 2 until $k = 2$.

Large clean instances

Lemma (Step 2 (Iteration))

(D, I, k, d) *clean* $\implies \exists P_i$ *from* $s_i \to t_i$ *s.t.* $|V(D/(B_i \cup P_i)| \geq \frac{n(k-1)}{2k}$.

Lemma (Step 3 (Base))

(D, I, k, d) *clean,* $k = 2$*,* $n \geq 4d \implies$ *positive instance, solution in polynomial time.*

- For large enough n :
  - Iterate Step 2 until $k = 2$.
  - Solve instance $k = 2$ using Step 3.

Large clean instances

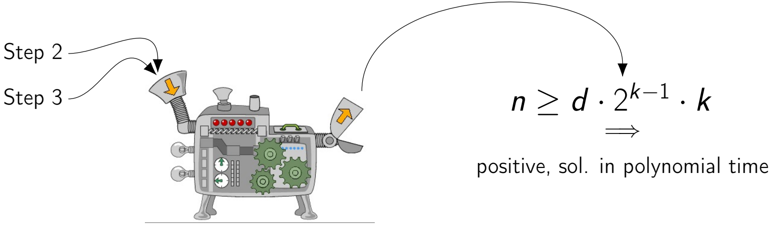
Lemma (Step 2 (Iteration))

(D, I, k, d) *clean* $\implies \exists P_i$ *from* $s_i \to t_i$ *s.t.* $|V(D/(B_i \cup P_i)| \geq \frac{n(k-1)}{2k}$.

Lemma (Step 3 (Base))

(D, I, k, d) *clean,* $k = 2$, $n \geq 4d \implies$ *positive instance, solution in polynomial time.*

- For large enough n :
  - Iterate Step 2 until $k = 2$.
  - Solve instance $k = 2$ using Step 3.



Math Machine

- Real kernel size: $d \cdot 2^{k-s} \cdot \binom{k}{s} + 2k$.

- Real kernel size: $d \cdot 2^{k-s} \cdot \binom{k}{s} + 2k$.
- Open: Poly-kernel?

- Real kernel size: $d \cdot 2^{k-s} \cdot \binom{k}{s} + 2k$.
- Open: Poly-kernel?
- Negative answer for $s = 0$ suffices.

- Real kernel size: $d \cdot 2^{k-s} \cdot \binom{k}{s} + 2k$.
- Open: Poly-kernel?
- Negative answer for $s = 0$ suffices.
- Our result $\implies$ kernel for Steiner Network with params. k and $d = n - c$, where c is the size of the solution.

Disjoint Enough Paths:

Input: *Requests* $I = \{(s_1, t_1), \ldots, (s_k, t_k)\}$, integers d and s.

Output: $\mathcal{P} = \{P_1, \ldots, P_k\}$ *satisfying* I s.t. $\geq d$ vertices occurring in $\leq s$ *paths*.

Parameters	Generalizes
$d = n$, $s = 1$	Disjoint Paths
$d = n$, $s \geq 2$	Disjoint Paths with Congestion
$d \geq 1$, $s = 0$	Steiner Network

k	d	s	dtw	Complexity
fixed ≥ 3	n^α	fixed ≥ 1	—	NP-complete
parameter	n^α	fixed ≥ 1	0	W[1]-hard
input	parameter	fixed ≥ 0	—	W[1]-hard
parameter	—	—	parameter	XP
input	parameter	parameter	—	XP
parameter	parameter	parameter	—	FPT

- Positive results for general digraphs.
- Kernel size: $d \cdot 2^{k-s} \cdot \binom{k}{s} + 2k$.
- Open: Poly-kernel. Negative answer for $s = 0$ suffices.
- Consequence for Steiner Network particularly interesting.