On two optimization problems:

(1) Permutation Routing on Plane Grids

(2) Find Small Subgraphs of Given Degree

Ignasi Sau

Mascotte Project - INRIA/CNRS-I3S/UNSA - FRANCE
Applied Mathematics IV Department of UPC - SPAIN

Two independent problems

(1) **Permutation Routing on Plane Grids**

Joint work with *Janez Žerovnik*

(2) **Finding Small Subgraphs of Given Minimum Degree**

Joint work with *Omid Amini* and *Saket Saurabh*

Permutation Routing on Plane Grids

Outline

- **Introduction**
  - Statement of the problem
  - Preliminaries
  - Example
- **Permutation routing algorithm for triangular grids**
  - Description
  - Correctness
  - Optimality
- Permutation routing algorithm for hexagonal grids
- (ℓ, k)-routing algorithms
- Conclusions

Permutation routing

- The **permutation routing** problem is a **packet routing** problem.
- Each processor is the **origin of at most one packet** and the **destination of at most one packet**.
- The goal is to **minimize the number of time steps** required to route all packets to their respective destinations.

Statement of the problem

- **Input:**
  - a directed graph $G = (V, E)$ (the *host* graph),
  - a subset $S \subseteq V$ of nodes,
  - and a permutation $\pi : S \rightarrow S$.
    Each node $u \in S$ wants to send a packet to $\pi(u)$.
- **Output:** Find for each pair $(u, \pi(u))$, a path form u to $\pi(u)$ in G.
- **Constraints:**
  - At each step, a packet can either move or stay at a node.
  - No arc can be crossed by two packets at the same step.
  - Cohabitation of multiple packets at the same node is allowed.
- **Goal: minimize the number of time steps** required to route all packets to their respective destinations.

Statement of the problem

- **Input:**
  - a directed graph $G = (V, E)$ (the *host* graph),
  - a subset $S \subseteq V$ of nodes,
  - and a permutation $\pi : S \rightarrow S$.
    Each node $u \in S$ wants to send a packet to $\pi(u)$.
- **Output:** Find for each pair $(u, \pi(u))$, a path form u to $\pi(u)$ in G.
- **Constraints:**
  - At each step, a packet can either move or stay at a node.
  - No arc can be crossed by two packets at the same step.
  - Cohabitation of multiple packets at the same node is allowed.
- **Goal: minimize the number of time steps** required to route all packets to their respective destinations.

Statement of the problem

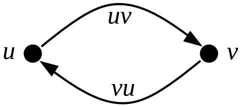
- **Input:**
  - a directed graph $G = (V, E)$ (the *host* graph),
  - a subset $S \subseteq V$ of nodes,
  - and a permutation $\pi : S \rightarrow S$.
    Each node $u \in S$ wants to send a packet to $\pi(u)$.
- **Output:** Find for each pair $(u, \pi(u))$, a path form u to $\pi(u)$ in G.
- **Constraints:**
  - At each step, a packet can either move or stay at a node.
  - No arc can be crossed by two packets at the same step.
  - Cohabitation of multiple packets at the same node is allowed.
- **Goal: minimize the number of time steps** required to route all packets to their respective destinations.

Statement of the problem

- **Input:**
  - a directed graph $G = (V, E)$ (the *host* graph),
  - a subset $S \subseteq V$ of nodes,
  - and a permutation $\pi : S \rightarrow S$.
    Each node $u \in S$ wants to send a packet to $\pi(u)$.
- **Output:** Find for each pair $(u, \pi(u))$, a path form u to $\pi(u)$ in G.
- **Constraints:**
  - At each step, a packet can either move or stay at a node.
  - No arc can be crossed by two packets at the same step.
  - Cohabitation of multiple packets at the same node is allowed.
- **Goal: minimize the number of time steps** required to route all packets to their respective destinations.

Assumptions

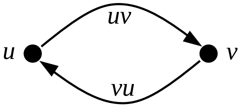
- We consider the **store-and-forward** and **Δ-port** model.
- **Full duplex link**: packets can be sent in the two directions of the link **simultaneously**.



- If the network is **half-duplex** →
  2 factor approximation algorithm from an optimal algorithm for the full-duplex case, by introducing *odd-even* steps.
- Assume **shortest path routing**.

Assumptions

- We consider the **store-and-forward** and **Δ-port** model.
- **Full duplex link**: packets can be sent in the two directions of the link **simultaneously**.



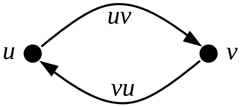
- If the network is **half-duplex** $\rightarrow$

  2 factor approximation algorithm from an optimal algorithm for the full-duplex case, by introducing *odd-even* steps.

- Assume **shortest path routing.**

Assumptions

- We consider the **store-and-forward** and **Δ-port** model.
- **Full duplex link**: packets can be sent in the two directions of the link **simultaneously**.



- If the network is **half-duplex** $\rightarrow$

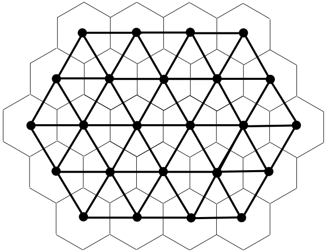
  2 factor approximation algorithm from an optimal algorithm for the full-duplex case, by introducing *odd-even* steps.
- Assume **shortest path routing**.

Network topologies

- There is an **ambiguity** in the notation in the literature:

$$\text{triangular grid} \leftrightarrow \text{hexagonal network},$$
$$\text{hexagonal grid} \leftrightarrow \text{honeycomb network}.$$

- Hexagonal network (△) and hexagonal tessellation (⬡):



- Hexagonal networks are finite subgraphs of the triangular grid.

Previous work

-The permutation routing problem has been studied in:

- Mobile Ad Hoc Networks
- Cube-Connected Cycle Networks
- Wireless and Radio Networks
- All-Optical Networks
- Reconfigurable Meshes...

-But, optimal algorithms:

- 2-circulant graphs, square grids.
- Triangular grids: **Two-terminal routing**
  (only one message to be sent)

-In this talk we describe **optimal permutation routing algorithms for triangular and hexagonal grids.**

Previous work

-The permutation routing problem has been studied in:

- Mobile Ad Hoc Networks
- Cube-Connected Cycle Networks
- Wireless and Radio Networks
- All-Optical Networks
- Reconfigurable Meshes...

-But, optimal algorithms:

- 2-circulant graphs, square grids.
- Triangular grids: **Two-terminal routing**
  (only one message to be sent)

-In this talk we describe **optimal permutation routing algorithms for triangular and hexagonal grids.**

Previous work

-The permutation routing problem has been studied in:

- Mobile Ad Hoc Networks
- Cube-Connected Cycle Networks
- Wireless and Radio Networks
- All-Optical Networks
- Reconfigurable Meshes...

-But, optimal algorithms:

- 2-circulant graphs, square grids.
- Triangular grids: **Two-terminal routing**
  (only one message to be sent)

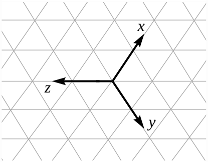
-In this talk we describe **optimal permutation routing algorithms for triangular and hexagonal grids**.

Permutation Routing on Triangular Grids

Notation and preliminary results

Nocetti, Stojmenović and Zhang [IEEE TPDS'02]:

Representation of the relative address of the nodes on a generating system ***i, j, k*** on the directions of the three axis x, y, z.



- This address **is not unique**, but we have that, being (a, b, c) and (a', b', c') the addresses of two $D - S$ pairs,

$$(a, b, c) = (a', b', c') \quad \Leftrightarrow \quad \exists \text{ an integer } d \text{ such that}$$

$$a' = a + d,$$

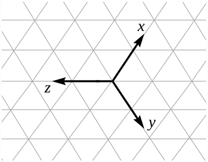
$$b' = b + d,$$

$$c' = c + d.$$

Notation and preliminary results

Nocetti, Stojmenović and Zhang [IEEE TPDS'02]:

Representation of the relative address of the nodes on a generating system ***i, j, k*** on the directions of the three axis x, y, z.



- This address **is not unique**, but we have that, being (a, b, c) and (a', b', c') the addresses of two $D - S$ pairs,

$$(a, b, c) = (a', b', c') \quad \Leftrightarrow \quad \exists \text{ an integer } d \text{ such that}$$

$$a' = a + d,$$
$$b' = b + d,$$
$$c' = c + d.$$

Notation and preliminary results (2)

- A relative address $D - S = (a, b, c)$ is of the ***shortest path form*** if
  - there is a path C from S to D, C=$a\mathbf{i}$+$b\mathbf{j}$+$c\mathbf{k}$,
  - and C has the shortest length over all paths going from S to D.

Theorem (*NSZ'02*)

*An address (a, b, c) is of the **shortest path form** if and only if*

i) ***at least one component is zero*** *(that is, $abc = 0$),*

ii) *and **any two components do not have the same sign** (that is, $ab \leq 0,\ ac \leq 0,\ and\ bc \leq 0$).*

Notation and preliminary results (2)

- A relative address $D - S = (a, b, c)$ is of the ***shortest path form*** if
  - there is a path C from S to D, $C=a\mathbf{i}+b\mathbf{j}+c\mathbf{k}$,
  - and C has the shortest length over all paths going from S to D.

Theorem (*NSZ'02*)

*An address (a, b, c) is of the **shortest path form** if and only if*

i) ***at least one component is zero*** *(that is, $abc = 0$),*

ii) *and **any two components do not have the same sign***
(that is, $ab \leq 0,\ ac \leq 0,$ and $bc \leq 0$).

Notation and preliminary results (3)

Corollary (*NSZ'02*)

Any address has a ***unique*** *shortest path form.*

Corollary (*NSZ'02*)

If $D - S = (a, b, c)$, then the shortest path form is one of those:

$$(0, b-a, c-a),$$

$$(a-b, 0, c-b),$$

$$(a-c, b-c, 0),$$

and thus:

$$|D - S| = \min(|b-a| + |c-a|, |a-b| + |c-b|, |a-c| + |b-c|).$$

Notation and preliminary results (3)

Corollary (*NSZ'02*)

*Any address has a **unique** shortest path form.*

Corollary (*NSZ'02*)

If $D - S = (a, b, c)$*, then the shortest path form is one of those:*

$$(0, b-a, c-a),$$
$$(a-b, 0, c-b),$$
$$(a-c, b-c, 0),$$

and thus:

$$|D-S| = \min(|b-a| + |c-a|, |a-b| + |c-b|, |a-c| + |b-c|).$$

Notation and preliminary results (4)

- Given a packet p and its relative address (a, b, c) *in the shortest path form*,

$$\ell_p := |a| + |b| + |c|,$$

$$\ell_{max} := \max_p(\ell_p)$$

- Trivial **lower bound**:

  Any permutation routing algorithm needs at least ℓ_{max} routing steps.

Notation and preliminary results (4)

- Given a packet p and its relative address (a, b, c) *in the shortest path form*,

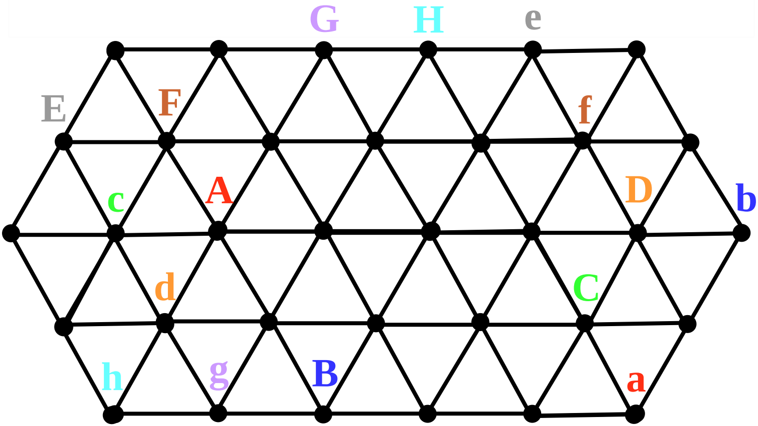
$$\ell_p := |a| + |b| + |c|,$$

$$\ell_{max} := \max_p(\ell_p)$$

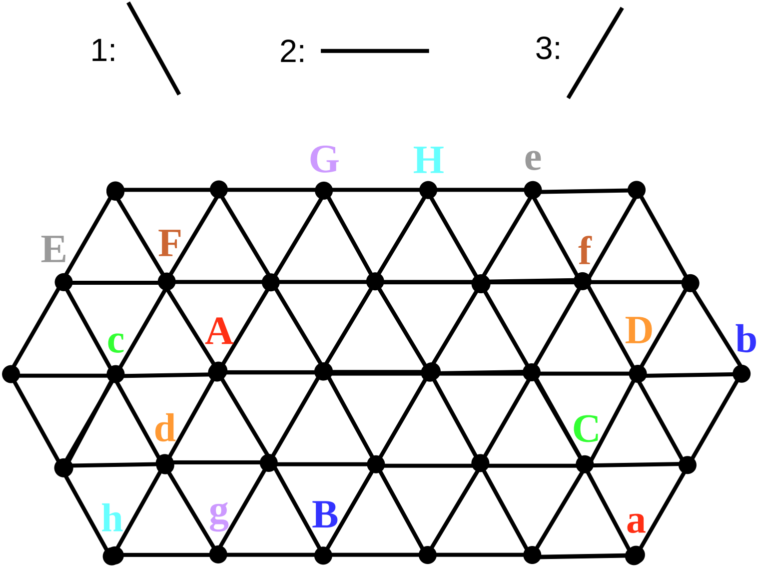
- Trivial **lower bound**:

  Any permutation routing algorithm needs at least ℓ_{max} routing steps.

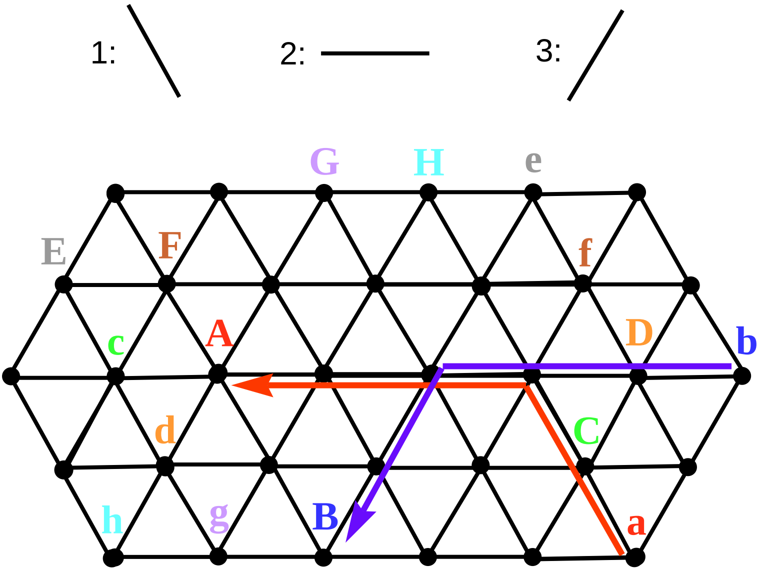
Example of an instance



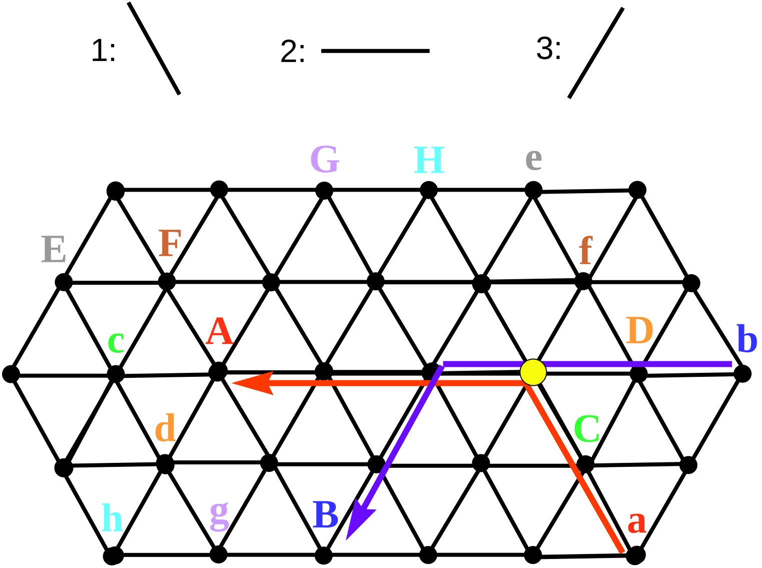
A non-optimal intuitive algorithm



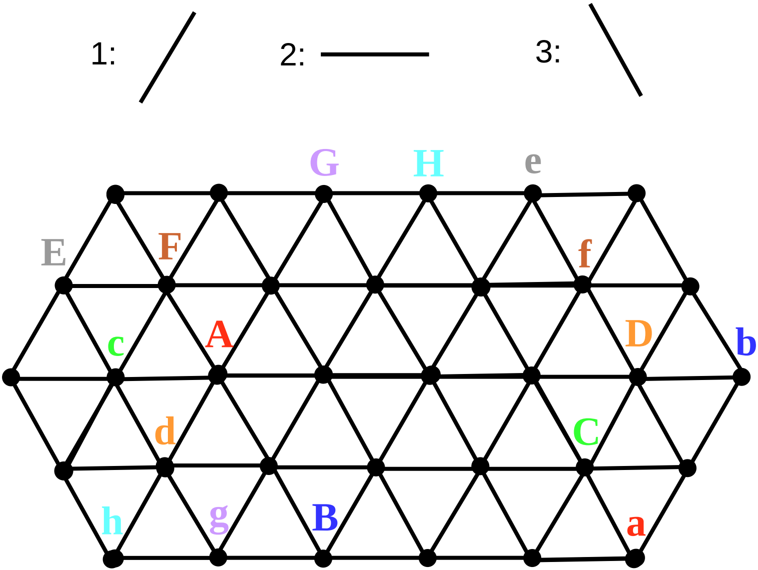
A non-optimal intuitive algorithm (2)



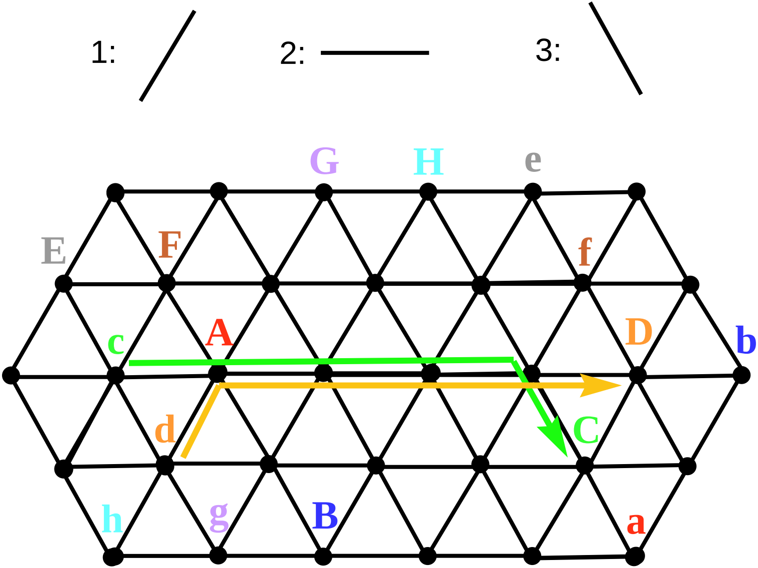
A non-optimal intuitive algorithm (3)



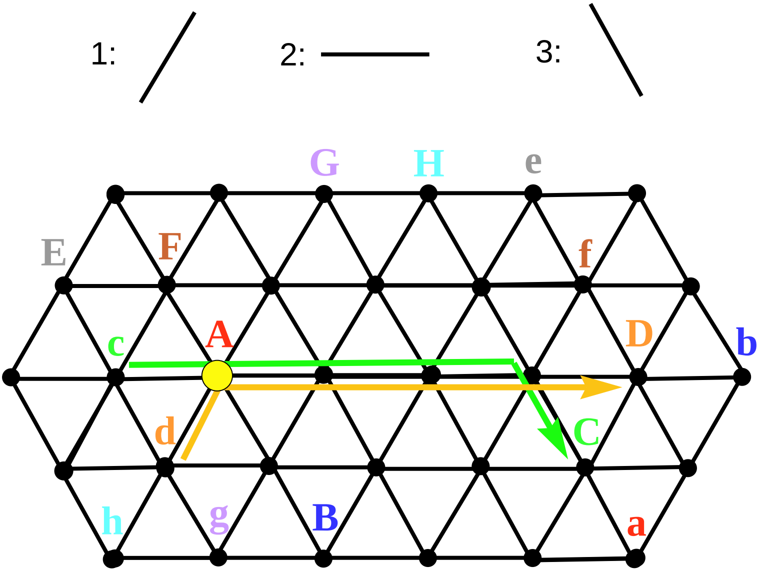
Another non-optimal intuitive algorithm



Another non-optimal intuitive algorithm (2)



Another non-optimal intuitive algorithm (3)



Description of Algorithm $\mathcal{A}$

At each node u of the network:

- ***Preprocessing*:** Initially, if there is a packet at u, compute the relative address $D - S$ of the message in the shortest path form, and add this information to the message.
- ***Reception phase*:** At each step, when a packet is received at u, its relative address is updated.
- ***Transmission phase*:**
  - **a)** If there are packets with **negative components**, send them **immediately** along the direction of this component.
  - **b)** If not, for each outgoing edge order the packets according to **decreasing number of remaining steps**, and send *the* **first** packet of each queue.

Description of Algorithm $\mathcal{A}$

At each node u of the network:

- ***Preprocessing*****:** Initially, if there is a packet at u, compute the relative address $D - S$ of the message in the shortest path form, and add this information to the message.
- ***Reception phase*****:** At each step, when a packet is received at u, its relative address is updated.
- ***Transmission phase*****:**
  - **a)** If there are packets with **negative components**, send them **immediately** along the direction of this component.
  - **b)** If not, for each outgoing edge order the packets according to **decreasing number of remaining steps**, and send *the* **first** packet of each queue.

Description of Algorithm $\mathcal{A}$

At each node u of the network:

- ***Preprocessing***: Initially, if there is a packet at u, compute the relative address $D - S$ of the message in the shortest path form, and add this information to the message.
- ***Reception phase***: At each step, when a packet is received at u, its relative address is updated.
- ***Transmission phase***:
  - a) If there are packets with **negative components**, send them **immediately** along the direction of this component.
  - b) If not, for each outgoing edge order the packets according to **decreasing number of remaining steps**, and send *the* **first** packet of each queue.

Description of Algorithm $\mathcal{A}$

At each node *u* of the network:

- ***Preprocessing*: ** Initially, if there is a packet at *u*, compute the relative address $D - S$ of the message in the shortest path form, and add this information to the message.
- ***Reception phase*: ** At each step, when a packet is received at *u*, its relative address is updated.
- ***Transmission phase*:**

  a) If there are packets with **negative components**, send them **immediately** along the direction of this component.

  b) If not, for each outgoing edge order the packets according to **decreasing number of remaining steps**, and send *the* **first** packet of each queue.

Description of Algorithm $\mathcal{A}$

At each node *u* of the network:

- ***Preprocessing***: Initially, if there is a packet at *u*, compute the relative address $D - S$ of the message in the shortest path form, and add this information to the message.
- ***Reception phase***: At each step, when a packet is received at *u*, its relative address is updated.
- ***Transmission phase***:
  - **a)** If there are packets with **negative components**, send them **immediately** along the direction of this component.
  - **b)** If not, for each outgoing edge order the packets according to **decreasing number of remaining steps**, and send *the* **first** packet of each queue.

Routing of the packets according to $\mathcal{A}$

- Algorithm $\mathcal{A}$ defines for each packet **two directions of movement** (except if a packet has only one non-zero component)
- For instance:
  - if the packet address is of the type $(-, 0, +) \rightarrow$
    this packet goes first in the direction $-x$, and after in $+z$

    $\rightarrow$ We symbolize this rule by the arrow 

  - the routing of the address $(+, -, 0)$ is represented by 

Routing of the packets according to $\mathcal{A}$

- Algorithm $\mathcal{A}$ defines for each packet **two directions of movement** (except if a packet has only one non-zero component)
- For instance:
  - if the packet address is of the type $(-, 0, +) \rightarrow$ this packet goes first in the direction $-x$, and after in $+z$

    $\rightarrow$ We symbolize this rule by the arrow 

  - the routing of the address $(+, -, 0)$ is represented by 

Routing of the packets according to $\mathcal{A}$

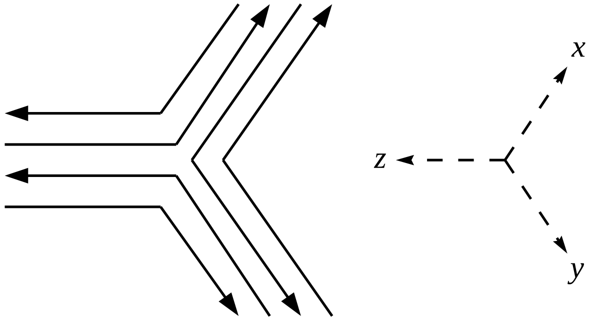
- Algorithm $\mathcal{A}$ defines for each packet **two directions of movement** (except if a packet has only one non-zero component)
- For instance:
  - if the packet address is of the type $(-, 0, +) \rightarrow$
    this packet goes first in the direction $-x$, and after in $+z$

    $\rightarrow$ We symbolize this rule by the arrow 

  - the routing of the address $(+, -, 0)$ is represented by 

Routing the packets (2)

- In this figure all the routing rules are summarized:



Correctness of Algorithm $\mathcal{A}$

At each node u of the network:

- ***Preprocessing***: Initially, if there is a packet at u, compute the relative address $D - S$ of the message in the shortest path form, and add this information to the message.
- ***Reception phase***: At each step, when a packet is received at u, its relative address is updated.
- ***Transmission phase***:

a) If there are packets with **negative components**, send them **immediately** along the direction of this component.

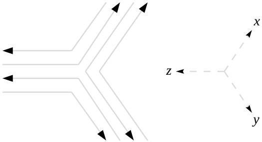
b) If not, for each outgoing edge order the packets according to **decreasing number of remaining steps**, and send *the* **first** packet of each queue.

Correctness (2)

- *Key observation:*

  Packets can only **wait**, possibly, during their **last direction.**

  - this is because if two packets meet when their first direction is not finished yet, they must have the same origin node → contradiction.



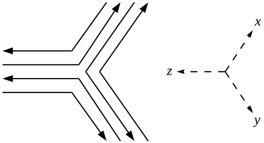
- Thus, in **a)** there can be **at most one packet with negative component at each outgoing edge** → there is no ambiguity.

Correctness (2)

- *Key observation:*

  Packets can only **wait**, possibly, during their **last direction.**

  - this is because if two packets meet when their first direction is not finished yet, they must have the same origin node → contradiction.



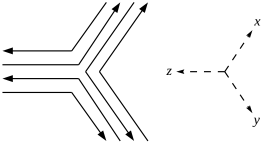
- Thus, in **a)** there can be **at most one packet with negative component at each outgoing edge** → there is no ambiguity.

Correctness (2)

- *Key observation:*

  Packets can only **wait**, possibly, during their **last direction.**

  - this is because if two packets meet when their first direction is not finished yet, they must have the same origin node → contradiction.



- Thus, in **a)** there can be **at most one packet with negative component at each outgoing edge** → there is no ambiguity.

Correctness (3)

At each node u of the network:

- ***Preprocessing***: Initially, if there is a packet at u, compute the relative address $D - S$ of the message in the shortest path form, and add this information to the message.
- ***Reception phase***: At each step, when a packet is received at u, its relative address is updated.
- ***Transmission phase***:

  a) If there are packets with **negative components**, send them **immediately** along the direction of this component.

  b) If not, for each outgoing edge order the packets according to **decreasing number of remaining steps**, and send *the* **first** packet of each queue.

Correctness (4)

- All the packets in in **b)** are moving along their **last direction**
  - their negative component is already finished, otherwise they would be in **a)**
- Thus, since each node is the destination of at most one packet, in **b)** *the* **packet with maximum remaining length at each outgoing edge is unique.**

Correctness (4)

- All the packets in in **b)** are moving along their **last direction**
  - their negative component is already finished, otherwise they would be in **a)**

- Thus, since each node is the destination of at most one packet, in **b)** *the* **packet with maximum remaining length at each outgoing edge is unique.**

Optimality

- Using this algorithm, at each step **all the packets with maximum remaining distance move**

  $\rightarrow$ every step the maximum remaining distance over all packets decreases by one

  $\rightarrow$ the total running time is at most ℓ_{max}, **meeting the lower bound.**

Optimality

- Using this algorithm, at each step **all the packets with maximum remaining distance move**

  $\rightarrow$ every step the maximum remaining distance over all packets decreases by one

  $\rightarrow$ the total running time is at most ℓ_{max}, **meeting the lower bound.**

Analysis

- It is a **distributed** algorithm, because each node needs only information about the packets that are its queues.
- It is an **oblivious** algorithm, since the routing of each packet depends only on the origin and destination nodes.
- It is a **translation invariant** algorithm, since only the relative address $D - S$ is necessary to route the packets.
- The only involved operations are *integer addition and comparison* among the lengths of the addresses of the packets at each node.
- Time complexity: $O(\ell_{max})$

Analysis

- It is a **distributed** algorithm, because each node needs only information about the packets that are its queues.
- It is an **oblivious** algorithm, since the routing of each packet depends only on the origin and destination nodes.
- It is a **translation invariant** algorithm, since only the relative address $D - S$ is necessary to route the packets.
- The only involved operations are *integer addition and comparison* among the lengths of the addresses of the packets at each node.
- Time complexity: $\mathcal{O}(\ell_{max})$

Analysis

- It is a **distributed** algorithm, because each node needs only information about the packets that are its queues.
- It is an **oblivious** algorithm, since the routing of each packet depends only on the origin and destination nodes.
- It is a **translation invariant** algorithm, since only the relative address $D - S$ is necessary to route the packets.
- The only involved operations are *integer addition and comparison* among the lengths of the addresses of the packets at each node.
- Time complexity: $\mathcal{O}(\ell_{max})$

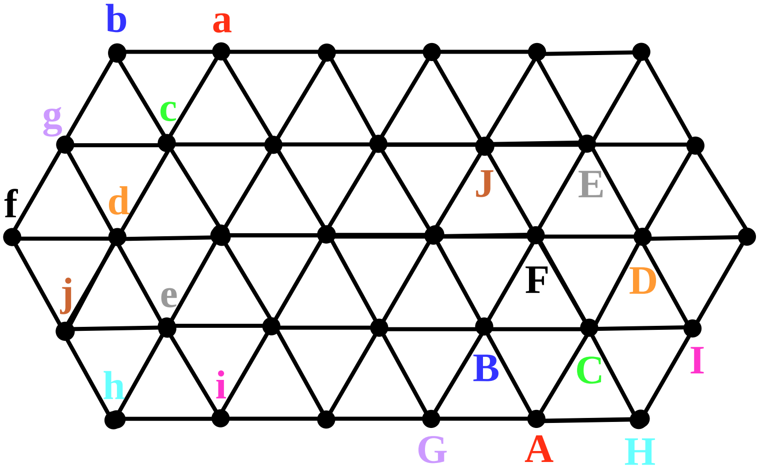
Analysis

- It is a **distributed** algorithm, because each node needs only information about the packets that are its queues.
- It is an **oblivious** algorithm, since the routing of each packet depends only on the origin and destination nodes.
- It is a **translation invariant** algorithm, since only the relative address $D - S$ is necessary to route the packets.
- The only involved operations are *integer addition and comparison* among the lengths of the addresses of the packets at each node.
- Time complexity: $\mathcal{O}(\ell_{max})$

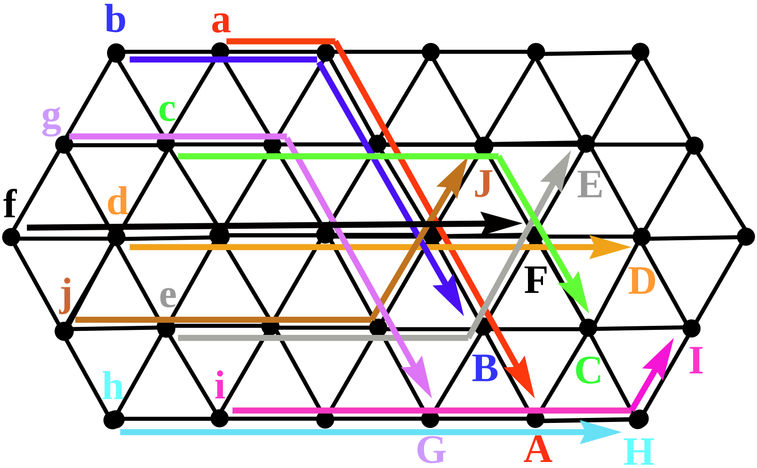
Analysis

- It is a **distributed** algorithm, because each node needs only information about the packets that are its queues.
- It is an **oblivious** algorithm, since the routing of each packet depends only on the origin and destination nodes.
- It is a **translation invariant** algorithm, since only the relative address $D - S$ is necessary to route the packets.
- The only involved operations are *integer addition and comparison* among the lengths of the addresses of the packets at each node.
- Time complexity: $\mathcal{O}(\ell_{max})$

Final example

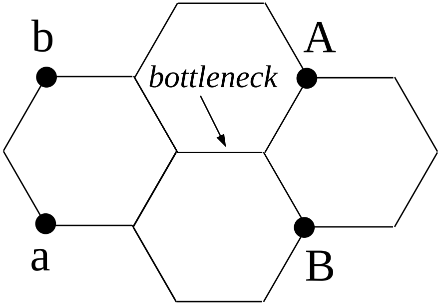


Final example (2)



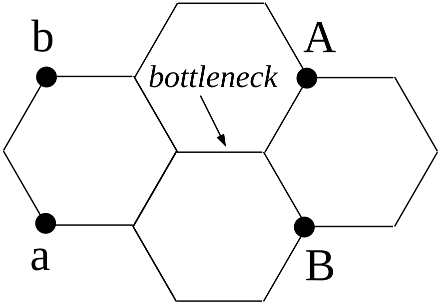
Permutation Routing on Hexagonal Grids

Counterexample



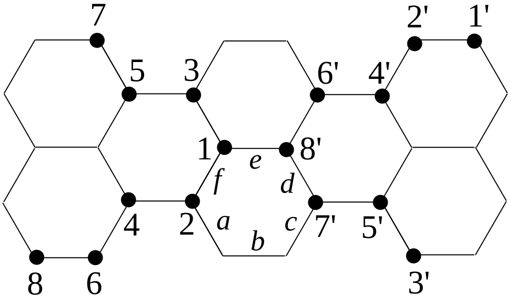
- $\ell_{max} = 4$, but it is not possible to route both packets in less than 5 steps.
- Thus, ℓ_{max} can not always be achieved!

Counterexample



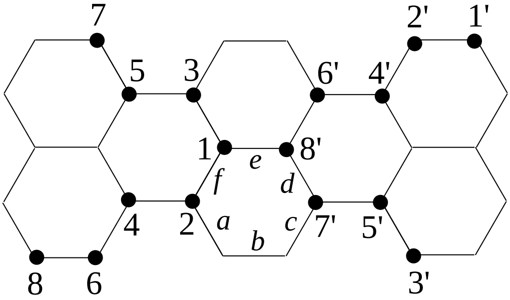
- $\ell_{max} = 4$, but it is not possible to route both packets in less than 5 steps.
- Thus, ℓ_{max} can not always be achieved!

Another counterexample



- Shortest path: 8 steps
- Using edges $\{abcd\}$: 7 steps
- Thus, shortest path routing is not always the best solution!

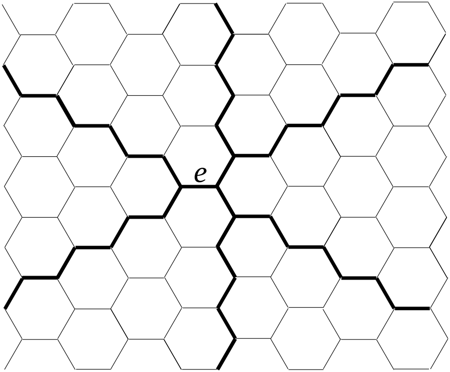
Another counterexample



- Shortest path: 8 steps
- Using edges $\{abcd\}$: 7 steps
- Thus, shortest path routing is not always the best solution!

Hexagonal grid

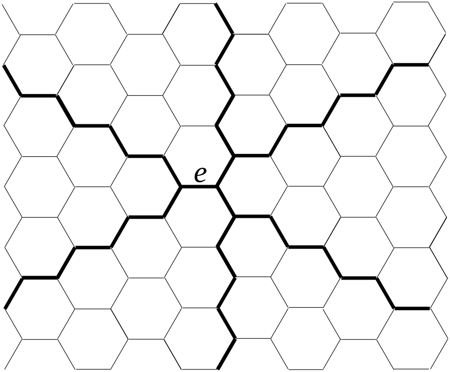
- One can define 3 types of *zigzag chains*:



- Any shortest path uses at most 2 types of zigzag chains

Hexagonal grid

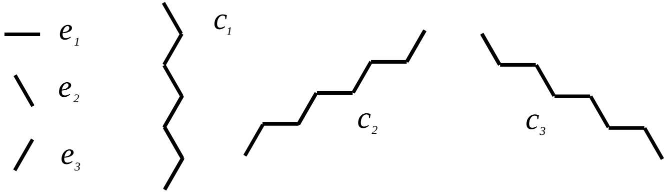
- One can define 3 types of *zigzag chains*:



- Any shortest path uses at most 2 types of zigzag chains

Idea

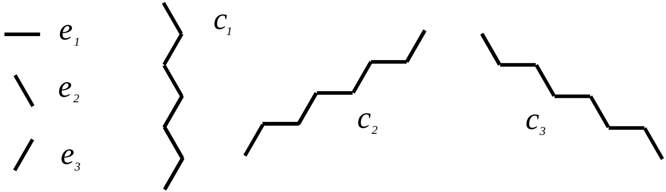
- There are 3 types of edges and 3 types of chains:



- Each edge belongs to exactly 2 different chains, and conversely each chain is made of 2 types of edges.
- Any 2 chains of different type intersect exactly on one edge.
- We can define 2 phases in such a way that at each phase, each type of chain uses only one type of edge.

Idea

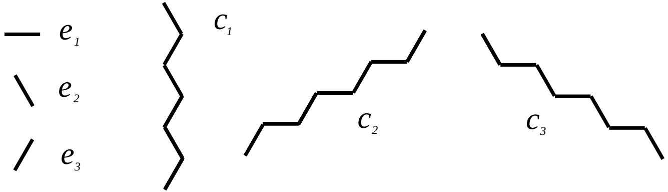
- There are 3 types of edges and 3 types of chains:



- Each edge belongs to exactly 2 different chains, and conversely each chain is made of 2 types of edges.
- Any 2 chains of different type intersect exactly on one edge.
- We can define 2 phases in such a way that at each phase, each type of chain uses only one type of edge.

Idea

- There are 3 types of edges and 3 types of chains:



- Each edge belongs to exactly 2 different chains, and conversely each chain is made of 2 types of edges.
- Any 2 chains of different type intersect exactly on one edge.
- We can define 2 phases in such a way that at each phase, each type of chain uses only one type of edge.

Optimal algorithm

At each node of the network:

1) During the **first step**, move all packets along the direction of their **negative component**. If a packet's address has only a positive component, move it along this direction.

2) From now on, **change alternatively** between Phase 1 and Phase 2.

3) At each step (the same for both phases):

- **a)** If there are packets with negative components, send them immediately along the direction of this component.
- **b)** If not, for each outgoing edge order the packets according to decreasing number of remaining steps, and send the first packet of each queue.

4) At the end of the $(2\ell_{max} - 3)th$ step, move all packets along their unique non-zero component.

Running time

- Every 2 steps (one of Phase 1 and one of Phase 2) the maximum remaining distance over all packets decreases by one.
- During the first and last step all packets decrease their remaining distance by one.
- Thus, the total running time is $2 + 2(\ell_{max} - 2) = 2\ell_{max} - 2$.
- There are examples that need at least $2\ell_{max} - 2$ steps.
- Thus, this algorithm is tight.

Running time

- Every 2 steps (one of Phase 1 and one of Phase 2) the maximum remaining distance over all packets decreases by one.
- During the first and last step all packets decrease their remaining distance by one.
- Thus, the total running time is $2 + 2(\ell_{max} - 2) = 2\ell_{max} - 2$.
- There are examples that need at least $2\ell_{max} - 2$ steps.
- Thus, this algorithm is tight.

Running time

- Every 2 steps (one of Phase 1 and one of Phase 2) the maximum remaining distance over all packets decreases by one.
- During the first and last step all packets decrease their remaining distance by one.
- Thus, the total running time is $2 + 2(\ell_{max} - 2) = 2\ell_{max} - 2$.
- There are examples that need at least $2\ell_{max} - 2$ steps.
- Thus, this algorithm is tight.

(ℓ, k)-Routing

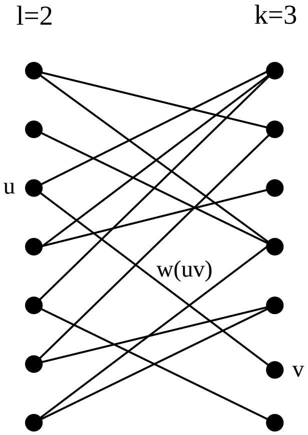
Algorithm (in any grid)

- Each node can send at most ℓ packets and receive at most k packets
- **Idea**: represent the request set as a weighted bipartite graph H:
  - split each vertex of the original graph
  - u and v are adjacent if u wants to send a packet to v
  - for each edge uv, let $w(uv)$ be the length of a shortest path from u to v on the grid

Algorithm (in any grid)

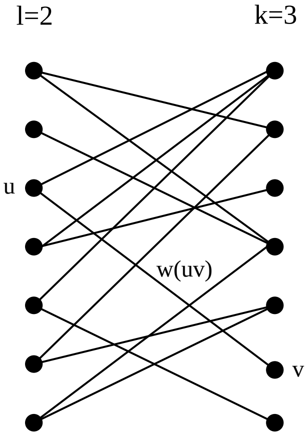
- Each node can send at most ℓ packets and receive at most k packets
- **Idea**: represent the request set as a weighted bipartite graph H:
  - split each vertex of the original graph
  - u and v are adjacent if u wants to send a packet to v
  - for each edge uv, let $w(uv)$ be the length of a shortest path from u to v on the grid

Example



- **Fact**: each matching in H corresponds to an instance of a permutation routing problem $\rightarrow$ it can be solved optimally

Example



- **Fact**: each matching in *H* corresponds to an instance of a permutation routing problem $\rightarrow$ it can be solved optimally

New problem

- **Problem**: find $m := \max\{\ell, k\}$ matchings in H: $M_1, \ldots, M_m$
- Let $c(M_i) := \max\{w(e) | e \in M_i)\}$, $i = 1, \ldots, m$
- **Objective function**:

$$\min \sum_{i=1}^{m} c(M_i)$$

- **Fact**: $\min \sum_{i=1}^{m} c(M_i)$ is the running time of routing a (ℓ, k)-routing instance using this algorithm
- But this problem is **NP-complete...** ☹

New problem

- **Problem**: find $m := \max\{\ell, k\}$ matchings in H: $M_1, \ldots, M_m$
- Let $c(M_i) := \max\{w(e) | e \in M_i\}$, $i = 1, \ldots, m$
- **Objective function**:

$$\min \sum_{i=1}^{m} c(M_i)$$

- **Fact**: $\min \sum_{i=1}^{m} c(M_i)$ is the running time of routing a (ℓ, k)-routing instance using this algorithm
- But this problem is **NP-complete...** ☹

New problem

- **Problem**: find $m := \max\{\ell, k\}$ matchings in H: $M_1, \ldots, M_m$
- Let $c(M_i) := \max\{w(e) | e \in M_i)\}$, $i = 1, \ldots, m$
- **Objective function**:

$$\min \sum_{i=1}^{m} c(M_i)$$

- **Fact**: $\min \sum_{i=1}^{m} c(M_i)$ is the running time of routing a (ℓ, k)-routing instance using this algorithm
- But this problem is **NP-complete...** ☹

Summary and further research

- We have described optimal permutation routing algorithms for triangular and hexagonal grids
- We have also optimal algorithms for the $(1, k)$-routing problem
- It remains to solve the $(\ell - k)$-routing
- Permutation routing on 3-circulant graphs is still a challenging open problem...

Summary and further research

- We have described optimal permutation routing algorithms for triangular and hexagonal grids
- We have also optimal algorithms for the $(1, k)$-routing problem
- It remains to solve the $(\ell - k)$-routing
- Permutation routing on 3-circulant graphs is still a challenging open problem...

Find Small Subgraphs with Given Minimum Degree

Hardness of approximation

- **Class APX (Approximable):**

  an optimization problem is in APX if can be approximated within a constant factor.

  Example: VERTEX COVER

- **Class PTAS (Polynomial-Time Approximation Scheme):**

  an optimization problem is in PTAS if can be approximated within a constant factor $1 + \varepsilon$, for *all* $\varepsilon > 0$
  (the best one can hope for an NP-complete problem).

  Ex.: TRAVELING SALESMAN PROBLEM *in the Euclidean plane*

- We know that

$$\text{PTAS} \subsetneq \text{APX}$$

- Thus, if Π is an optimization problem:

$$\Pi \text{ is APX-hard} \quad \Rightarrow \quad \Pi \notin \text{PTAS}$$

Hardness of approximation

- **Class APX (Approximable):**

  an optimization problem is in APX if can be approximated within a constant factor.

  ***Example**:* VERTEX COVER

- **Class PTAS (Polynomial-Time Approximation Scheme):**

  an optimization problem is in PTAS if can be approximated within a constant factor $1 + \varepsilon$, for *all* $\varepsilon > 0$
  (the best one can hope for an NP-complete problem).

  ***Ex.**:* TRAVELING SALESMAN PROBLEM *in the Euclidean plane*

- We know that

$$\text{PTAS} \subsetneq \text{APX}$$

- Thus, if Π is an optimization problem:

$$\Pi \text{ is APX-hard} \quad \Rightarrow \quad \Pi \notin \text{PTAS}$$

Hardness of approximation

- **Class APX (Approximable):**

  an optimization problem is in APX if can be approximated within a constant factor.

  ***Example**:* VERTEX COVER

- **Class PTAS (Polynomial-Time Approximation Scheme):**

  an optimization problem is in PTAS if can be approximated within a constant factor $1 + \varepsilon$, for *all* $\varepsilon > 0$
  (the best one can hope for an NP-complete problem).

  ***Ex.**:* TRAVELING SALESMAN PROBLEM *in the Euclidean plane*

- We know that

$$\text{PTAS} \subsetneq \text{APX}$$

- Thus, if Π is an optimization problem:

$$\Pi \text{ is APX-hard} \quad \Rightarrow \quad \Pi \notin \text{PTAS}$$

Definition of the problem

- **MINIMUM SUBGRAPH OF MINIMUM DEGREE** $\geq d$ **(MSMD$_d$)**:

  Let d be a natural number, and $G = (V, E)$ a given graph.

  Find a subset of vertices $S \subseteq V$ of minimum size, such that $G[S]$ has minimum degree $\geq d$.

- For $d = 2$ it is the *girth* problem (find the length of the shortest cycle), which is polynomial

- We have proved that for $d \geq 3$, MSMD$_d \notin$ APX

Definition of the problem

- **MINIMUM SUBGRAPH OF MINIMUM DEGREE $\geq d$ (MSMD$_d$)**:

  Let d be a natural number, and $G = (V, E)$ a given graph.

  Find a subset of vertices $S \subseteq V$ of minimum size, such that $G[S]$ has minimum degree $\geq d$.

- For $d = 2$ it is the *girth* problem (find the length of the shortest cycle), which is polynomial

- We have proved that for $d \geq 3$, MSMD$_d \notin$ APX

Definition of the problem

- **Minimum Subgraph of Minimum Degree** $\geq d$ **(MSMD$_d$)**:

  Let d be a natural number, and $G = (V, E)$ a given graph.

  Find a subset of vertices $S \subseteq V$ of minimum size, such that $G[S]$ has minimum degree $\geq d$.

- For $d = 2$ it is the *girth* problem (find the length of the shortest cycle), which is polynomial

- We have proved that for $d \geq 3$, MSMD$_d$ $\notin$ APX

Idea of the proof for $d = 3$

(1) We will see first that $\text{MSMD}_3 \notin \text{PTAS}$.

(2) After we will see that $\text{MSMD}_3 \notin \text{APX}$.

(1) *MSMD*$_3$ is not in *PTAS*

- Reduction from VERTEX COVER:

  Instance H of VERTEX COVER $\rightarrow$ Instance G of MSMD$_3$

- We will see that

$$\text{PTAS for } G \Rightarrow \text{PTAS for } H$$

- And so,

$$\nexists \text{ PTAS for MSMD}_3$$

- We can suppose $|E(H)| = 3 \cdot 2^m$

(1) *$MSMD_3$* is not in *PTAS*

- Reduction from VERTEX COVER:

  Instance H of VERTEX COVER $\rightarrow$ Instance G of $\mathbf{MSMD_3}$

- We will see that

$$\text{PTAS for } G \Rightarrow \text{PTAS for } H$$

- And so,

$$\nexists \text{ PTAS for } \text{MSMD}_3$$

- We can suppose $|E(H)| = 3 \cdot 2^m$

(1) $MSMD_3$ is not in $PTAS$

- Reduction from VERTEX COVER:

  Instance H of VERTEX COVER $\rightarrow$ Instance G of MSMD$_3$

- We will see that

$$\text{PTAS for } G \ \Rightarrow \ \text{PTAS for } H$$

- And so,

$$\nexists \text{ PTAS for MSMD}_3$$

- We can suppose $|E(H)| = 3 \cdot 2^m$

(1) *MSMD*$_3$ is not in *PTAS*

- Reduction from VERTEX COVER:

  Instance H of VERTEX COVER $\rightarrow$ Instance G of MSMD$_3$

- We will see that

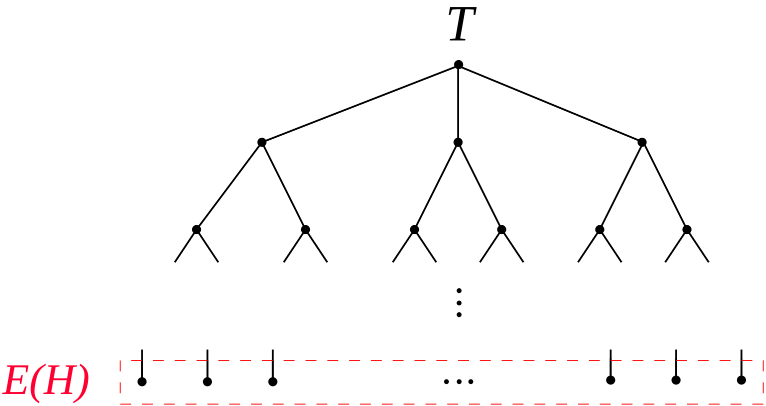
$$\text{PTAS for } G \;\Rightarrow\; \text{PTAS for } H$$

- And so,

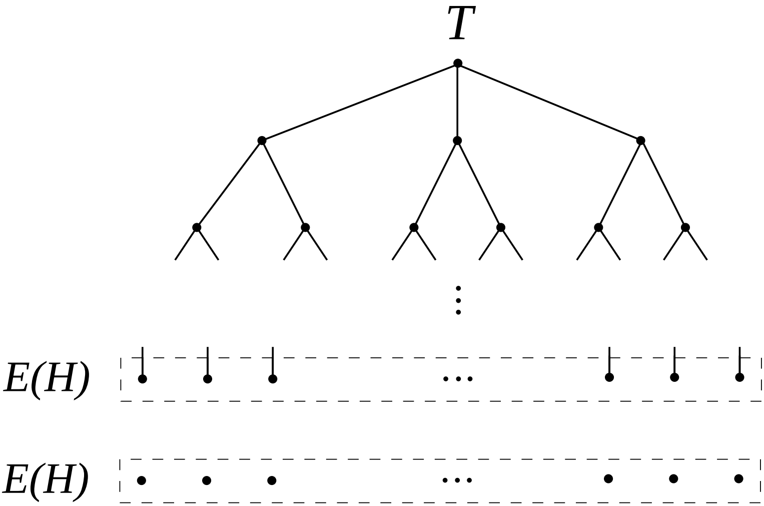
$$\nexists \text{ PTAS for MSMD}_3$$

- We can suppose $|E(H)| = 3 \cdot 2^m$

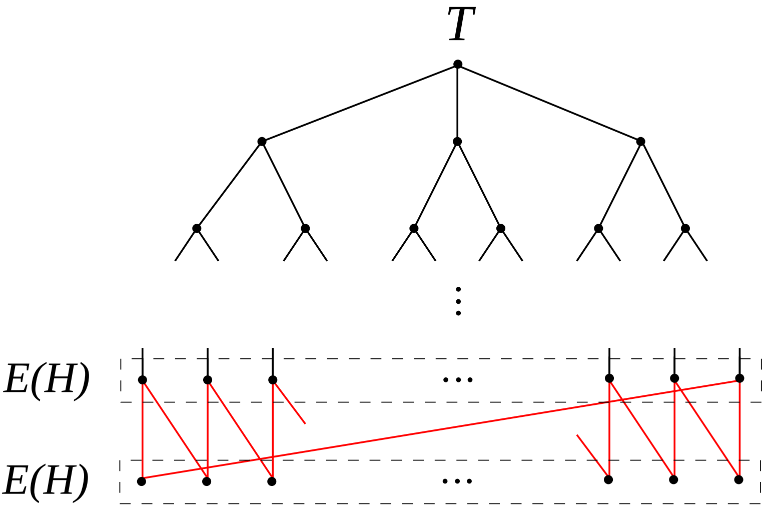
We build a complete ternary tree with $|E(H)| = 3 \cdot 2^m$ leaves:



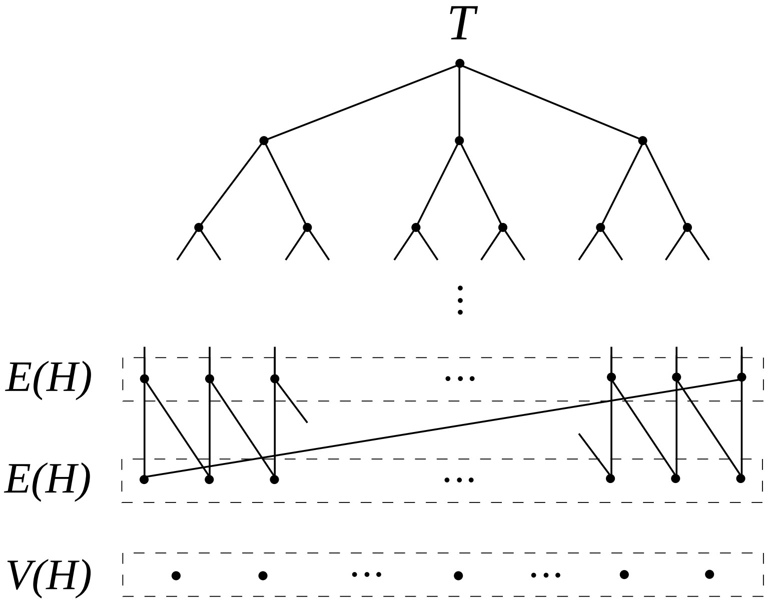
We add a copy of the set of leaves $E(H)$:



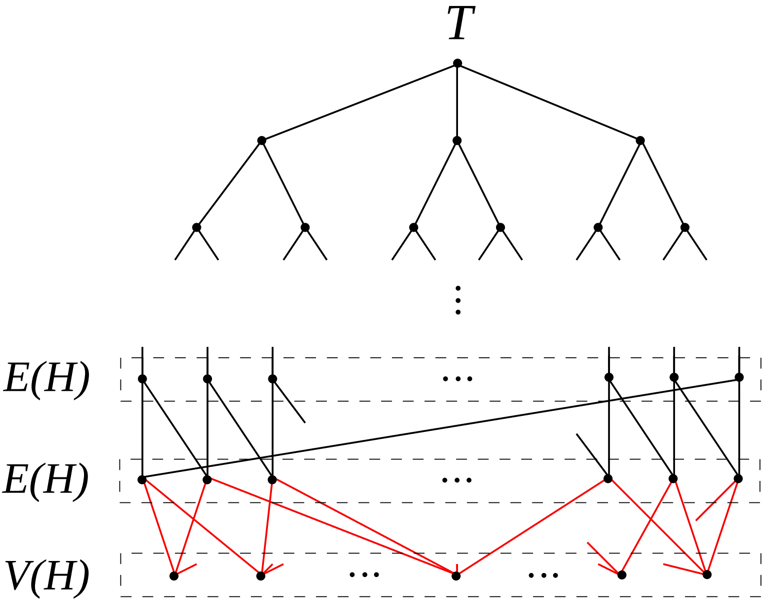
We join both sets with a Hamiltonian cycle (for technical reasons):



We add all the vertices of H:



We build the adjacency graph between $E(H)$ and $V(H)$:



(1) MSMD_3 is not in PTAS

- If we touch a vertex of $G \setminus V(H)$, we have to touch all the vertices of $G \setminus V(H)$

- Thus, MSMD_3 in G is equivalent to minimize the number of selected vertices in $V(H)$

  $\rightarrow$ this is **exactly** VERTEX COVER in H !!

- Thus,

$$OPT_{\mathrm{MSMD}_3}(G) = OPT_{\mathrm{VC}}(H) + |V(G \setminus V(H))| = OPT_{\mathrm{VC}}(H) + 9 \cdot 2^m$$

- This clearly proves that

$$\text{PTAS for } \mathrm{MSMD}_3 \Rightarrow \text{PTAS for VERTEX COVER}$$

(1) MSMD_3 is not in PTAS

- If we touch a vertex of $G \setminus V(H)$, we have to touch all the vertices of $G \setminus V(H)$
- Thus, MSMD_3 in G is equivalent to minimize the number of selected vertices in $V(H)$

  $\rightarrow$ this is **exactly** VERTEX COVER in H !!

- Thus,

$$OPT_{\mathrm{MSMD}_3}(G) = OPT_{\mathrm{VC}}(H) + |V(G \setminus V(H))| = OPT_{\mathrm{VC}}(H) + 9 \cdot 2^m$$

- This clearly proves that

$$\text{PTAS for } \mathrm{MSMD}_3 \Rightarrow \text{PTAS for VERTEX COVER}$$

(1) MSMD_3 is not in PTAS

- If we touch a vertex of $G \setminus V(H)$, we have to touch all the vertices of $G \setminus V(H)$
- Thus, MSMD_3 in G is equivalent to minimize the number of selected vertices in $V(H)$

  $\rightarrow$ this is **exactly** VERTEX COVER in H !!
- Thus,

$$OPT_{\mathrm{MSMD}_3}(G) = OPT_{\mathrm{VC}}(H) + |V(G \setminus V(H))| = OPT_{\mathrm{VC}}(H) + 9 \cdot 2^m$$

- This clearly proves that

$$\text{PTAS for } \mathrm{MSMD}_3 \Rightarrow \text{PTAS for VERTEX COVER}$$

(1) MSMD_3 is not in PTAS

- If we touch a vertex of $G \setminus V(H)$, we have to touch all the vertices of $G \setminus V(H)$
- Thus, MSMD_3 in G is equivalent to minimize the number of selected vertices in $V(H)$

  $\rightarrow$ this is **exactly** VERTEX COVER in H !!
- Thus,

$$OPT_{\text{MSMD}_3}(G) = OPT_{\text{VC}}(H) + |V(G \setminus V(H))| = OPT_{\text{VC}}(H) + 9 \cdot 2^m$$

- This clearly proves that

$$\text{PTAS for MSMD}_3 \;\; \Rightarrow \;\; \text{PTAS for VERTEX COVER}$$

(2) MSMD_3 is not in APX

- Let $\alpha > 1$ be the factor of innaproximability of MSMD_3
- We use a technique called **amplification of the error**:
  - We build a sequence of graphs G_k, such that MSMD_3 is hard to approximate in G_k within a factor α^k
  - This proves that the problem is not in APX

    (for any constant C, $\exists\ k > 0$ such that $\alpha^k > C$)
- Let $G_1 = G$.
  We explain the construction of G_2: first take our graph G and...

(2) MSMD$_3$ is not in APX

- Let $\alpha > 1$ be the factor of innaproximability of MSMD$_3$
- We use a technique called **amplification of the error**:
  - We build a sequence of graphs G_k, such that MSMD$_3$ is hard to approximate in G_k within a factor α^k
  - This proves that the problem is not in APX

    (for any constant C, $\exists\ k > 0$ such that $\alpha^k > C$)

- Let $G_1 = G$.

  We explain the construction of G_2: first take our graph G and...

(2) MSMD_3 is not in APX

- Let $\alpha > 1$ be the factor of innaproximability of MSMD_3
- We use a technique called **amplification of the error**:
  - We build a sequence of graphs G_k, such that MSMD_3 is hard to approximate in G_k within a factor α^k
  - This proves that the problem is not in APX

    (for any constant C, $\exists\ k > 0$ such that $\alpha^k > C$)

- Let $G_1 = G$.
  We explain the construction of G_2: first take our graph G and...

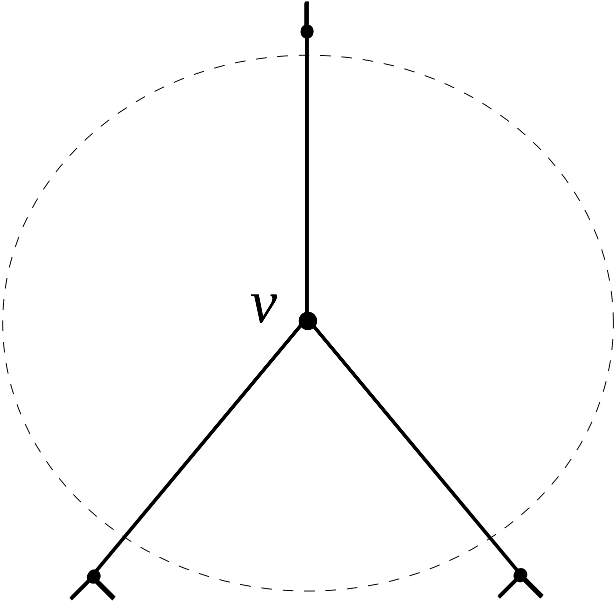
(2) MSMD_3 is not in APX

- Let $\alpha > 1$ be the factor of innaproximability of MSMD_3
- We use a technique called **amplification of the error**:
  - We build a sequence of graphs G_k, such that MSMD_3 is hard to approximate in G_k within a factor α^k
  - This proves that the problem is not in APX

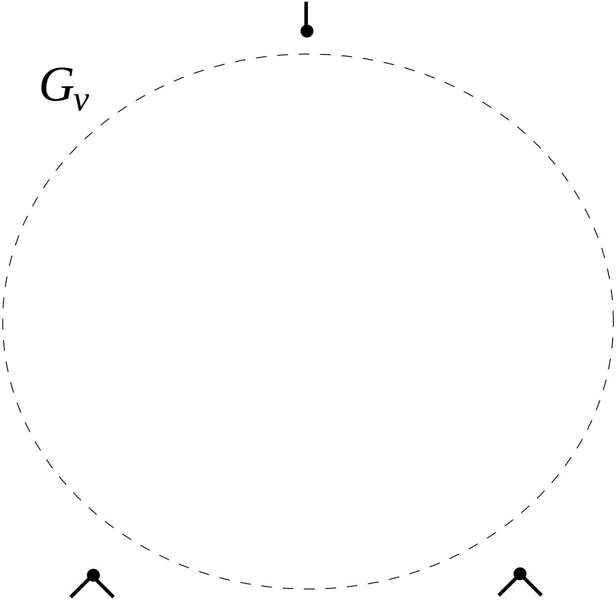
    (for any constant C, $\exists\ k > 0$ such that $\alpha^k > C$)

- Let $G_1 = G$.
  We explain the construction of G_2: first take our graph G and...

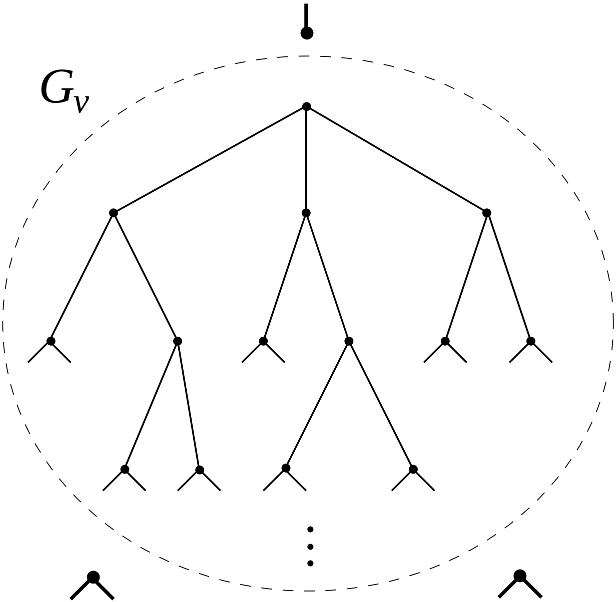
For any vertex v (note its degree by d_v):



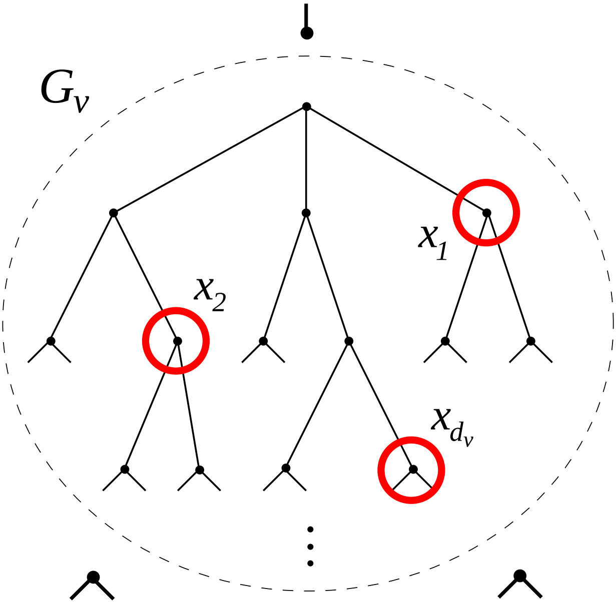
We will replace the vertex v with a graph G_v, built as follows:



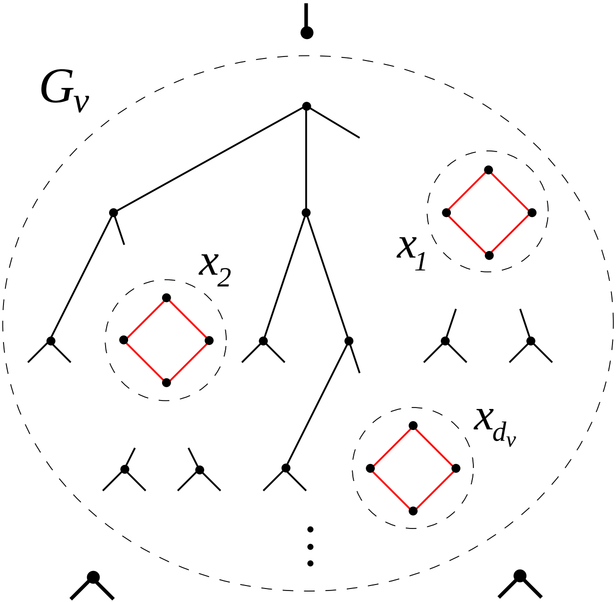
We begin by placing a copy of G (described before):



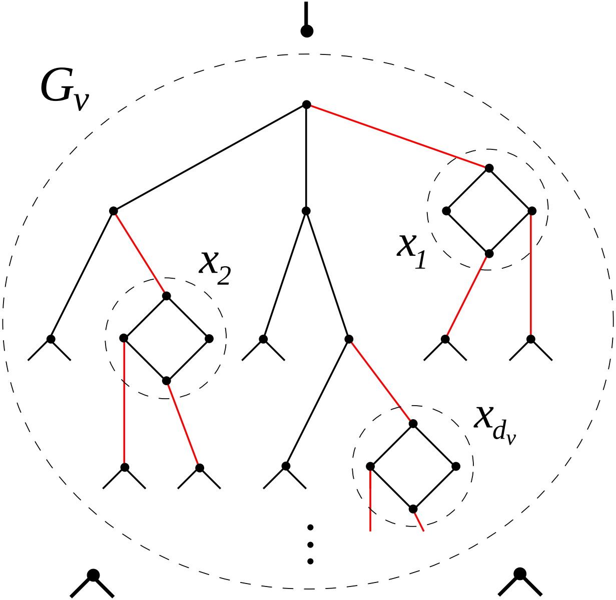
We select d_v vertices of degree 3 in $T \subset G$:



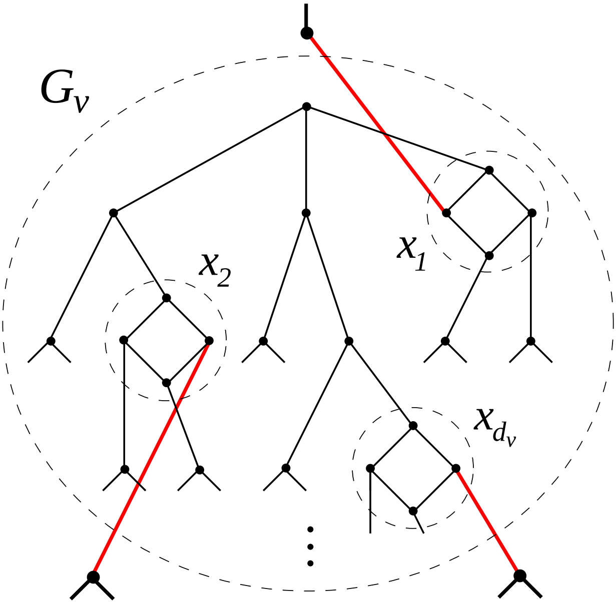
We replace each of these vertices x_i with a C_4:



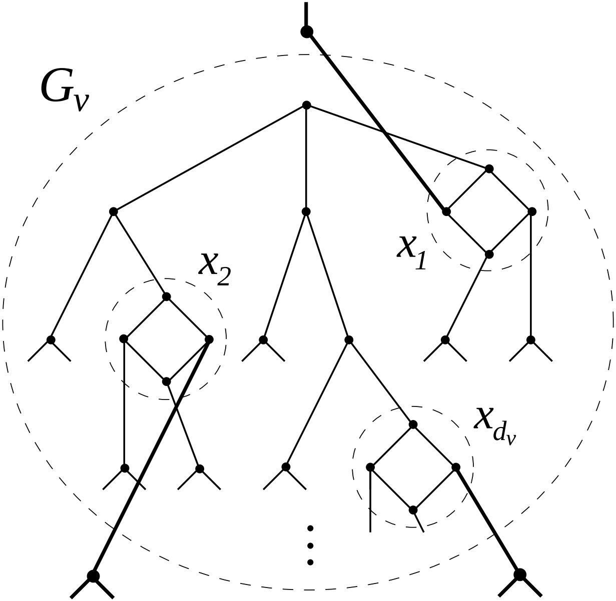
In each C_4, we join 3 of the vertices to the neighbors of x_i:



We join the d_v vertices of degree 2 to the d_v neighbors of v:



This construction for all $v \in G$ defines G_2:



(2) $MSMD_3$ is not in APX

- Once a vertex in one G_v is chosen $\rightarrow$ $MSMD_3$ in G_v (which is hard up to a constant α)

- But minimize the number of v's for which we touch G_v $\rightarrow$ $MSMD_3$ in G (which is also hard up to a constant α)

- Thus, in G_2 the problem is hard to approximate up to a factor $\alpha \cdot \alpha = \alpha^2$

- Inductively we prove that in G_k the problem is hard to approximate up to a factor α^k

(2) $MSMD_3$ is not in APX

- Once a vertex in one G_v is chosen $\rightarrow$ $MSMD_3$ in G_v

  (which is hard up to a constant α)

- But minimize the number of v's for which we touch G_v $\rightarrow$ $MSMD_3$ in G (which is also hard up to a constant α)

- Thus, in G_2 the problem is hard to approximate up to a factor $\alpha \cdot \alpha = \alpha^2$

- Inductively we prove that in G_k the problem is hard to approximate up to a factor α^k

(2) MSMD_3 is not in APX

- Once a vertex in one G_v is chosen $\rightarrow$ MSMD_3 in G_v

  (which is hard up to a constant α)

- But minimize the number of v's for which we touch G_v $\rightarrow$ MSMD_3 in G (which is also hard up to a constant α)

- Thus, in G_2 the problem is hard to approximate up to a factor $\alpha \cdot \alpha = \alpha^2$

- Inductively we prove that in G_k the problem is hard to approximate up to a factor α^k

Conclusions

- We have proved that MSMD_d, $d \geq 3$, is not in APX
- We have also proved that MSMD_d, $d \geq 3$, is W[1]-hard

  (and thus the problem is not FPT tractable in general graphs)
- We have FPT algorithms for minor free graphs

  (for instance: planar graphs, graphs of bounded local treewidth, graphs of bounded genus,...)
- We *almost* have PTAS for minor free graphs
- **Open problem**: find approximation algorithms for general graphs

Conclusions

- We have proved that MSMD_d, $d \geq 3$, is not in APX
- We have also proved that MSMD_d, $d \geq 3$, is W[1]-hard

  (and thus the problem is not FPT tractable in general graphs)
- We have FPT algorithms for minor free graphs

  (for instance: planar graphs, graphs of bounded local treewidth, graphs of bounded genus,...)
- We *almost* have PTAS for minor free graphs
- **Open problem**: find approximation algorithms for general graphs

Conclusions

- We have proved that MSMD_d, $d \geq 3$, is not in APX
- We have also proved that MSMD_d, $d \geq 3$, is W[1]-hard

  (and thus the problem is not FPT tractable in general graphs)
- We have FPT algorithms for minor free graphs

  (for instance: planar graphs, graphs of bounded local treewidth, graphs of bounded genus,...)
- We *almost* have PTAS for minor free graphs
- **Open problem**: find approximation algorithms for general graphs

Conclusions

- We have proved that MSMD_d, $d \geq 3$, is not in APX
- We have also proved that MSMD_d, $d \geq 3$, is W[1]-hard

  (and thus the problem is not FPT tractable in general graphs)
- We have FPT algorithms for minor free graphs

  (for instance: planar graphs, graphs of bounded local treewidth, graphs of bounded genus,...)
- We *almost* have PTAS for minor free graphs
- **Open problem**: find approximation algorithms for general graphs

Thanks!