Recovery of disrupted airline operations using k-Maximum Matching in graphs

Julien Bensmail\textsuperscript{1}  Valentin Garnero\textsuperscript{1}  Nicolas Nisse\textsuperscript{1}  
Alexandre Salch\textsuperscript{2}  Valentin Weber\textsuperscript{3}

\textsuperscript{1} Université Côte d’Azur, Inria, CNRS, I3S, France
\textsuperscript{2} Innovation & Research, Amadeus IT Group SA
\textsuperscript{3} Innovation & Research, Amadeus IT Pacific

Xidian University, Xi’an, 12th September 2018
Assignment of slots for landing to aircrafts

Aircrafts arriving at some airport

"Aéroport" de Nice
Assignment of slots for landing to aircrafts

Each aircraft has a set of available and compatible slots depending on the tracks, the schedules, the companies...

J. Bensmail, V. Garnero, N. Nisse, A. Salch and V. Weber

Recovery of disrupted airline operations
Assignment of slots for landing to aircrafts

Initially, one compatible slot is assigned to each aircraft.
Initially, one compatible slot is assigned to each aircraft.
Assignment of slots for landing to aircrafts

Imponderable problems may happen (no refund...)

J. Bensmail, V. Garnero, N. Nisse, A. Salch and V. Weber
Assignment of slots for landing to aircrafts

<table>
<thead>
<tr>
<th>Time</th>
<th>Aircrafts</th>
</tr>
</thead>
<tbody>
<tr>
<td>13:40-14:00</td>
<td></td>
</tr>
<tr>
<td>13:50-14:10</td>
<td></td>
</tr>
<tr>
<td>14:00-14:20</td>
<td></td>
</tr>
<tr>
<td>14:10-14:30</td>
<td></td>
</tr>
<tr>
<td>14:20-14:40</td>
<td></td>
</tr>
<tr>
<td>14:30-14:50</td>
<td></td>
</tr>
<tr>
<td>14:40-15:00</td>
<td></td>
</tr>
<tr>
<td>14:50-15:10</td>
<td></td>
</tr>
<tr>
<td>15:00-15:20</td>
<td></td>
</tr>
</tbody>
</table>

How to return to a normal situation? i.e., maximize # of aircrafts having a slot for landing!!

J. Bensmail, V. Garnero, N. Nisse, A. Salch and V. Weber
Recovery of disrupted airline operations
A simple matching problem in bipartite graphs?

Restart from scratch and compute a maximum matching?
A simple matching problem in bipartite graphs?

Restart from scratch and compute a maximum matching?
No!! Constraints due to the system/to the companies’ policies...
A simple matching problem in bipartite graphs?

Restart from scratch and compute a maximum matching?
No!! Constraints due to the system/to the companies’ policies...
ONLY 2 possible “moves” to satisfy all demands
A simple matching problem in bipartite graphs?

Restart from scratch and compute a maximum matching? No!! Constraints due to the system/to the companies’ policies... ONLY 2 possible “moves” to satisfy all demands
Reminder on Matchings in Graphs

Let $G = (V, E)$ be a graph. A matching $M \subseteq E$ is a set of pairwise disjoint edges.
Reminder on Matchings in Graphs

Let $G = (V, E)$ be a graph. A matching $M \subseteq E$ is a set of pairwise disjoint edges.

Exposed vertex: do not belong to the matching.

Exposed vertex: do not belong to the matching.
Reminder on Matchings in Graphs

Let $G = (V, E)$ be a graph.
A matching $M \subseteq E$ is a set of pairwise disjoint edges

Exposed vertex: do not belong to the matching

$M$-augmenting path : “alternating” with both ends exposed
Reminder on Matchings in Graphs

Let $G = (V, E)$ be a graph.
A matching $M \subseteq E$ is a set of pairwise disjoint edges

Exposed vertex: do not belong to the matching

$M$-augmenting path: “alternating” with both ends exposed

If $\exists$ an $M$-augmenting path $P$:
$P$ can be augmented (switch its edges) $\Rightarrow |M|$ was not maximum
Reminder on Matchings in Graphs

Let $G = (V, E)$ be a graph.
A matching $M \subseteq E$ is a set of pairwise disjoint edges

Exposed vertex: do not belong to the matching

$M$-augmenting path: “alternating” with both ends exposed

If $\exists$ an $M$-augmenting path $P$:
P can be augmented (switch its edges) $\Rightarrow |M|$ was not maximum
Reminder on Matchings in Graphs

Let $G = (V, E)$ be a graph.
A matching $M \subseteq E$ is a set of pairwise disjoint edges.

- Exposed vertex: do not belong to the matching
- $M$-augmenting path: “alternating” with both ends exposed

[Berge 1957]: Let $G$ be a graph

$M$ maximum matching ($|M| = \mu(G)$) iff no $M$-augmenting path.
Let $G = (V, E)$ be a graph.
A matching $M \subseteq E$ is a set of pairwise disjoint edges

**Exposed vertex:** do not belong to the matching

**$M$-augmenting path:** “alternating” with both ends exposed

[Berge 1957]: Let $G$ be a graph

$M$ maximum matching ($|M| = \mu(G)$) iff no $M$-augmenting path.

⇒ the order in which the augmenting paths are augmented is not important
Computing a maximum matching

Maximum Matching in Bipartite Graphs (flow problem)
“easy” [Hungarian method, Kuhn 1955]

Maximum Matching $\mu(G)$
Polynomial [Edmonds 1965]
finding an augmenting path in polynomial time + Berge’s theorem

Augment “greedily” paths of length $\leq 2k - 3$
$(1 - \frac{1}{k})$-Approximation for $\mu(G)$ [Hopcroft,Kraft 1973]

Applications in wireless networks
Let's go back to slots assignment

Let $G$ be a graph, $M$ be a (partial) matching and $k \in \mathbb{N}$ odd

Let $\mu_k(G, M)$ be the maximum size of a matching that can be obtained from $M$ by augmenting only paths of lengths $\leq k$. here $k = 3$
Our problem

Given a graph $G$, a matching $M$ and $k \in \mathbb{N}$ odd

Compute a matching of size $\mu_k(G, M)$ that can be obtained from $M$ by augmenting only paths of lengths $\leq k$.

Goal: algorithm that computes a sequence $(P_1, \cdots, P_r)$ such that:

- $\forall i \leq r$, $P_i$ a path of length $\leq k$ in $G$
- $\forall i \leq r$, after augmenting $P_1, \cdots, P_{i-1}$ starting from $M$, $P_i$ is augmenting
- and $r$ is maximum (w.r.t. these constraints)

$(r_{max} + |M| = \mu_k(G, M))$
Problem: Given a graph $G$, a matching $M$ and $k \in \mathbb{N}$ odd

Compute a matching of size $\mu_k(G, M)$ that can be obtained from $M$ by augmenting only paths of lengths $\leq k$.

**Goal:** algorithm that computes a sequence $(P_1, \cdots, P_r)$ such that:

- $\forall i \leq r$, $P_i$ a path of length $\leq k$ in $G$
- $\forall i \leq r$, after augmenting $P_1, \cdots, P_{i-1}$ starting from $M$ $P_i$ is augmenting
- and $r$ is maximum (w.r.t. these constraints)

**Case $M = \emptyset$.** For any odd $k \geq 0$, $\mu_k(G, \emptyset) = \mu(G)$

Compute a maximum matching, augment its edges one by one.
Problem: Given a graph \( G \), a matching \( M \) and \( k \in \mathbb{N} \) odd

Compute a matching of size \( \mu_k(G, M) \) that can be obtained from \( M \) by augmenting only paths of lengths \( \leq k \).

Goal: algorithm that computes a sequence \((P_1, \cdots, P_r)\) such that:
- \( \forall i \leq r, P_i \) a path of length \( \leq k \) in \( G \)
- \( \forall i \leq r, \) after augmenting \( P_1, \cdots, P_{i-1} \) starting from \( M \) \( P_i \) is augmenting
- and \( r \) is maximum (w.r.t. these constraints)

Case \( M = \emptyset \). For any odd \( k \geq 0 \), \( \mu_k(G, \emptyset) = \mu(G) \)

Compute a maximum matching, augment its edges one by one.

Case \( k = 1 \). For any matching \( M \), \( \mu_1(G, M) = \mu(G \setminus V(M)) + |M| \).

The edges of \( M \) cannot be “modified”

Compute a max. matching in \( G \setminus V(M) \), augment these edges 1 by 1
Our contributions

\[ k = 3 \]

[N., Salch, Weber, EJOR 17+]

Let \( G \) be a graph and \( M \) be a matching

Computing a matching of size \( \mu_3(G, M) \), obtained from \( M \) by augmenting paths of length \( \leq 3 \), is in \( P \).

\[ k \geq 5 \]

[N., Salch, Weber, EJOR 17+]

Let \( G \) be a planar bipartite graph of max. degree 3 and \( M \) be a matching

Computing a matching of size \( \mu_k(G, M) \), obtained from \( M \) by augmenting paths of length at most \( k \geq 5 \), is NP-complete.

\[ G = T \] is a tree.

[Bensmail, Garnero, N., DAM 18+]

Computing \( \mu_k(T, M) \) \( (k \text{ odd}) \) can be done in polynomial-time in trees \( T \)

- with bounded max. degree \( \Delta \) \hspace{1cm} \text{(dynamic prog., FPT in } k + \Delta) \)
- or with vertices of degree \( \geq 3 \) pairwise at distance \( > k \).
Our contributions

\( k = 3 \)  
\[ \text{Let } G \text{ be a graph and } M \text{ be a matching} \]
Computing a matching of size \( \mu_3(G, M) \), obtained from \( M \) by augmenting paths of length \( \leq 3 \), is in \( P \).

\( k \geq 5 \)
\[ \text{Let } G \text{ be a planar bipartite graph of max. degree 3 and } M \text{ be a matching} \]
Computing a matching of size \( \mu_k(G, M) \), obtained from \( M \) by augmenting paths of length at most \( k \geq 5 \), is \text{NP-complete}.

\( G = T \) is a tree.  
\[ \text{Computing } \mu_k(T, M) (k \text{ odd}) \text{ can be done in polynomial-time in trees } T \]
- with bounded max. degree \( \Delta \) (dynamic prog., FPT in \( k + \Delta \))
- or with vertices of degree \( \geq 3 \) pairwise at distance \( > k \).
Our contributions

\[ k = 3 \]

Let \( G \) be a graph and \( M \) be a matching

Computing a matching of size \( \mu_3(G, M) \), obtained from \( M \) by augmenting paths of length \( \leq 3 \), is in \( P \).

\[ k \geq 5 \]

Let \( G \) be a planar bipartite graph of max. degree 3 and \( M \) be a matching

Computing a matching of size \( \mu_k(G, M) \), obtained from \( M \) by augmenting paths of length at most \( k \geq 5 \), is NP-complete.

\( G = T \) is a tree.

Computing \( \mu_k(T, M) \) (\( k \) odd) can be done in polynomial-time in trees \( T \)

- with bounded max. degree \( \Delta \) (dynamic prog., FPT in \( k + \Delta \))
- or with vertices of degree \( \geq 3 \) pairwise at distance \( > k \).
1) Polynomial-time algorithm for computing $\mu_3(G, M)$ in any graph.
Why bounding the length of the augmenting paths makes the problem harder?

ex: \( k = 3 \)

(a) \( \rightarrow \) (b) not optimal, but (a) \( \rightarrow \) (c) \( \rightarrow \) (d) optimal
\[ \Rightarrow \] The result is impacted by the order in which paths are augmented.
Let $G$ be a graph and $M$ be a matching. Let $P_3(G, M)$ be the set of $M$-augmenting paths of length $\leq 3$. (the initial augmenting paths)

**Theorem:** [N., Salch, Weber, EJOR 17+]

A matching of size $\mu_3(G, M)$ can be obtained by augmenting ONLY the paths in $P_3(G, M)$. 
Let $G$ be a graph and $M$ be a matching. Let $P_3(G, M)$ be the set of $M$-augmenting paths of length $\leq 3$. (the initial augmenting paths)

**Theorem:** [N., Salch, Weber, EJOR 17+]

A matching of size $\mu_3(G, M)$ can be obtained by augmenting ONLY the paths in $P_3(G, M)$.

**Consequence:** [N., Salch, Weber, EJOR 17+]

$\mu_3(G, M)$ can be computed in polynomial-time.
Let $G$ be a graph and $M$ be a matching. Let $\mathcal{P}_3(G, M)$ be the set of $M$-augmenting paths of length $\leq 3$.

**Theorem:** [N., Salch, Weber, EJOR 17+]

A matching of size $\mu_3(G, M)$ can be obtained by augmenting ONLY the paths in $\mathcal{P}_3(G, M)$.

**Consequence:** [N., Salch, Weber, EJOR 17+]

$\mu_3(G, M)$ can be computed in polynomial-time.

Example of execution of the algorithm
Let $G$ be a graph and $M$ be a matching. Let $\mathcal{P}_3(G, M)$ be the set of $M$-augmenting paths of length $\leq 3$. (the initial augmenting paths)

**Theorem:** [N., Salch, Weber, EJOR 17+]

A matching of size $\mu_3(G, M)$ can be obtained by augmenting ONLY the paths in $\mathcal{P}_3(G, M)$.

**Consequence:** [N., Salch, Weber, EJOR 17+]

$\mu_3(G, M)$ can be computed in polynomial-time.

Some edges do not belong to any initial 3-augmenting path.
Complexity of computing $\mu_3(G, M)$

Let $G$ be a graph and $M$ be a matching. Let $P_3(G, M)$ be the set of $M$-augmenting paths of length $\leq 3$. (the initial augmenting paths)

**Theorem:**

A matching of size $\mu_3(G, M)$ can be obtained by augmenting ONLY the paths in $P_3(G, M)$.

**Consequence:**

$\mu_3(G, M)$ can be computed in polynomial-time.

Thanks to the theorem, they can be removed
Complexity of computing \( \mu_3(G, M) \)

Let \( G \) be a graph and \( M \) be a matching. Let \( P_3(G, M) \) be the set of \( M \)-augmenting paths of length \( \leq 3 \). (the initial augmenting paths)

Theorem:

[N., Salch, Weber, EJOR 17+]

A matching of size \( \mu_3(G, M) \) can be obtained by augmenting ONLY the paths in \( P_3(G, M) \).

Consequence:

[N., Salch, Weber, EJOR 17+]

\( \mu_3(G, M) \) can be computed in polynomial-time.

Apply a “classical” maximum matching algorithm

Can be obtained by augmenting paths of length \( \leq 3 \) (because the remaining matching has “good” properties)
2) NP-hardness of $\mu_k$ for every odd $k \geq 5$. 
Why bounding the length of the augmenting paths makes the problem harder?

ex: $k = 5$

⇒ The order in which paths are augmented impacts the creation of new augmenting paths that are necessary to reach the optimum
Reminder: 3-SAT and NP-hierarchy

3-SAT Problem $\Phi(\nu_1, \cdots, \nu_\eta)$ boolean formula in 3-Conjunctive Normal Form.

$\exists$ truth-assignment that satisfies $\Phi$

$\Phi(a, b, c, d, e) = (a \lor \bar{b} \lor c) \land (\bar{e} \lor b \lor \bar{c}) \land (\bar{a} \lor \bar{c} \lor e)$

$\Phi(a) = (a \lor a \lor a) \land (\bar{a} \lor \bar{a} \lor \bar{a})$
Reminder: 3-SAT and NP-hierarchy

3-SAT Problem $\Phi(v_1, \ldots, v_\eta)$ boolean formula in 3-Conjunctive Normal Form.

$\exists \text{? truth-assignment that satisfies } \Phi$

$\Phi(a, b, c, d, e) = (a \lor \bar{b} \lor c) \land (\bar{e} \lor b \lor \bar{c}) \land (\bar{a} \lor \bar{c} \lor e)$

$\Phi(1, 1, 0, 0, 0) = \text{TRUE}$

$\Phi(a) = (a \lor a \lor a) \land (\bar{a} \lor \bar{a} \lor \bar{a})$

$\Phi(1) = \Phi(0) = \text{FALSE, then NOT SATISFIABLE}$
Reminder: 3-SAT and NP-hierarchy

### 3-SAT Problem

\( \Phi(v_1, \cdots, v_\eta) \) boolean formula in 3-Conjunctive Normal Form.

\[ \exists \, \text{truth-assignment that satisfies } \Phi \]

\( \Phi(a, b, c, d, e) = (a \lor \overline{b} \lor c) \land (\overline{e} \lor b \lor \overline{c}) \land (\overline{a} \lor \overline{c} \lor e) \]

\( \Phi(1, 1, 0, 0, 0) = \text{true} \)

\( \Phi(a) = (a \lor a \lor a) \land (\overline{a} \lor \overline{a} \lor \overline{a}) \)

\( \Phi(1) = \Phi(0) = \text{false}, \text{ then NOT SATISFIABLE} \)

---

**NP problems whose solutions can be checked in Polynomial Time**

i.e., solvable by polynomial-time non-deterministic Turing machine

**NP-complete = NP \cap NP-hard**

**NP-hard**

Problems "as hard as the hardest problems in NP"

**P problems solvable in Polynomial Time**

**3-SAT**

---

**Cook’s Theorem**

3-SAT is “the” NP-complete problem

\[ \Rightarrow \text{no polynomial-time (} \eta^{O(1)} \text{) algorithm to solve 3-SAT unless } P = NP \]
Complexity of computing $\mu_k(G, M)$

**Theorem:** [N., Salch, Weber, EJOR 17+]

For any odd $k \geq 5$, computing $\mu_k(G, M)$ is NP-complete in the class of planar bipartite graphs with maximum degree 3.

Reduction from planar 3-SAT

Gadget for Variable $i$ positive in Clause $j$. (bold edges are the initial matching $M$)

(example for $k = 7$)
Complexity of computing $\mu_k(G, M)$

**Theorem:**\[N., Salch, Weber, EJOR 17+\]

For any odd $k \geq 5$, computing $\mu_k(G, M)$ is NP-complete in the class of planar bipartite graphs with maximum degree 3.

Example of reduction for the formula (with $m$ clauses):

$$\Phi = (b \lor c \lor d) \land \neg b \lor c \lor \neg e \land \neg c \lor d \lor e \land \neg b \lor \neg d \lor \neg e$$

$\Phi$ satisfiable iff

$$\mu_7(G, M) \geq |M| + 4m$$
3) Computation of $\mu_k(G, M)$ in trees
Question: Complexity of $\mu_k(T, M)$ in trees $T$?

Our results: [Bensmail, Garnero, N., DAM 18+]

Computing $\mu_k(T, M)$ ($k$ odd) can be done in polynomial-time in:

- trees with bounded max. degree $\Delta$ when $k$ fixed parameter (dynamic prog., FPT in $k + \Delta$)
- caterpillars
- $k$-sparse trees
  s.t. vertices of degree $\geq 3$ pairwise at distance $> k$.

Open question

Can $\mu_k(T, M)$ be computed in polynomial-time in general trees? when $k$ is part of the input? when $k$ is a fixed parameter?
Example of a path, $k = 5$.

Consider an alternative path of length $> k$. 
Example of a path, $k = 5$.

Consider an alternative path of length $> k$. Either it is not touched by other augmentation.
Trees: simplify the instance first

Example of a path, $k = 5$.

Consider an alternative path of length $> k$.
Either it is not touched by other augmentation.
Trees: simplify the instance first

Example of a path, $k = 5$.

Consider an alternative path of length $> k$.
Either it is not touched by other augmentation.
Or, it may only increase
Example of a path, $k = 5$. 

Consider an alternative path of length $> k$. Either it is not touched by other augmentation. Or, it may only increase.
Example of a path, $k = 5$.

Consider an alternative path of length $> k$.
Either it is not touched by other augmentation.
Or, it may only increase

$\Rightarrow$ Left and Right instances may be considered independently.
Trees: simplify the instance first

Example of a path, $k = 5$.

Consider an alternative path of length $> k$.
Either it is not touched by other augmentation.
Or, it may only increase
$\Rightarrow$ Left and Right instances may be considered independently.

More generally, in trees

Assume that all leaves are exposed and no alternating path $> k$ with all internal vertices of degree 2 (actually, it is a bit more general)
From above, no alternating path of length $> k$. 

$B =$ number of exposed vertices

Each augmentation “destroys” 2 exposed vertices

$\Rightarrow \leq \left\lfloor \frac{B}{2} \right\rfloor$ augmentations in total

Example with $k = 5$: 

![Diagram illustrating the process with example](image-url)
Trees: case of paths

From above, no alternating path of length \( \geq k \).

\( B \) = number of exposed vertices

Each augmentation “destroys” 2 exposed vertices

\[ \Rightarrow \leq \left\lfloor \frac{B}{2} \right\rfloor \] augmentations in total

Example with \( k = 5 \):

Algorithm: It augment paths from “left to right” when possible.
Trees: case of paths

From above, no alternating path of length $> k$.

$B =$ number of exposed vertices

Each augmentation “destroys” 2 exposed vertices

$\Rightarrow \leq \left\lfloor \frac{B}{2} \right\rfloor$ augmentations in total

Example with $k = 5$:

Algorithm: It augment paths from “left to right” when possible.
Trees: case of paths

From above, no alternating path of length \( > k \).

\( B \) = number of exposed vertices

Each augmentation “destroys” 2 exposed vertices

\[ \Rightarrow \leq \left\lfloor \frac{B}{2} \right\rfloor \] augmentations in total

Example with \( k = 5 \):

Algorithm: It augment paths from “left to right” when possible.
Trees: case of paths

From above, no alternating path of length > \( k \).

\( B \) = number of exposed vertices

Each augmentation “destroys” 2 exposed vertices

\[ \Rightarrow \leq \left\lfloor \frac{B}{2} \right\rfloor \] augmentations in total

Example with \( k = 5 \):

Algorithm: It augment paths from “left to right” when possible.
Trees: case of paths

From above, no alternating path of length $> k$.

$B = \text{number of exposed vertices}$

Each augmentation “destroys” 2 exposed vertices

$\Rightarrow \leq \left\lfloor \frac{B}{2} \right\rfloor$ augmentations in total

Example with $k = 5$:

Algorithm: It augment paths from “left to right” when possible.

Augments exactly $\left\lfloor \frac{B}{2} \right\rfloor$ paths $\Rightarrow$ it is optimal

What matters is the parity of the number of exposed vertices!!

if $B$ is odd, one exposed vertex is “useless”!
Trees: case of subdivided stars

How the paths crossing the branching node (root) are augmented?

*Example with* $k = 9$: 

![Diagram of a tree with subdivided stars]
Trees: case of subdivided stars

How the paths crossing the branching node (root) are augmented?

*Example with* $k = 9$:

"turning around" the root
Trees: case of subdivided stars

How the paths crossing the branching node (root) are augmented?

*Example with* $k = 9$:

”turning around” the root
Trees: case of subdivided stars

How the paths crossing the branching node (root) are augmented?

*Example with $k = 9$: “turning around” the root*
Trees: case of subdivided stars

How the paths crossing the branching node (root) are augmented?

*Example with $k = 9$:*  
"turning around" the root
Trees: case of subdivided stars

How the paths crossing the branching node (root) are augmented?

*Example with $k = 9$:*

"Turning around" the root
Trees: case of subdivided stars

How the paths crossing the branching node (root) are augmented?

*Example with* $k = 9$:

"turning around" the root

Seems equivalent to augment paths not crossing the root!!

Why paths crossing the root may be useful?
Trees: case of subdivided stars

Why paths crossing the root may be useful?

Look at the full star

Branches $B$, $C$, $D$ are even
(even # of exposed nodes)

Branches $A$ and $E$ are odd

$A$, $E$ contains each one useless exposed node
Trees: case of subdivided stars

Why paths crossing the root may be useful?

Look at the full star

Branches $B$, $C$, $D$ are even
(even # of exposed nodes)
Branches $A$ and $E$ are odd

Paths 1, 2, 3 “destroys”:

- 1 exposed node in $A$, $D$
  $\Rightarrow$ Makes $A$ even
  $\Rightarrow$ used a “useless” exposed node in $A$

- 2 exposed nodes in $B$, $C$
  $\Rightarrow$ Makes $D$ odd
  $\Rightarrow$ makes an exposed node useless in $D$
Trees: case of subdivided stars

Why paths crossing the root may be useful?

Look at the full star

Better if the sequence of paths “turning around” the root starts and ends in odd branches
Trees: case of subdivided stars

Why paths crossing the root may be useful?

Look at the full star

Optimal Algorithm:

1. If it exists:
augment a sequence of paths “turning around” the root starting and ending in odd branches (and with even branches in the middle)

(Can be found using a BFS in some auxiliary digraph)

2. Then, deal with each branch independently.
Trees: case of "$k$-sparse trees"

$k$-sparse tree: Vertices with degree $\geq 3$ are at distance $> k$. 
Trees: case of "$k$-sparse trees"

$k$-sparse tree: Vertices with degree $\geq 3$ are at distance $> k$.

Optimal polynomial-time algorithm: [Bensmail, Garnero, N., DAM 18+]

Going bottom-up, considering “independent” subdivided stars.
4) Going further: ⇒ a new problem
Augmenting paths of length exactly $k$
Augmenting paths of length exactly $k$?

**New Problem:** Given a graph $G$, a matching $M$ and $k \in \mathbb{N}$ odd, compute a matching of size $\mu_{=k}(G, M)$ that can be obtained from $M$ by augmenting only paths of lengths $= k$.

**Our results**

- $\forall G$ and matching $M$, deciding if $\mu_{=3}(G, M) \leq q$ is NP-complete
- given a tree $T$, $M$ a matching and $q, k \in \mathbb{N}$ as inputs, deciding if $\mu_{=k}(T, M) \leq q$ is NP-complete
Deciding if $\mu=k(T, M) \leq q$ is NP-complete in trees

Reduction from 3-SAT
Firstly: Learn how to communicate with companies
After many months of discussion, hypotheses are changing...

Actual (?) constraints of the system/of Airline Operation Controllers

- Switch alternating cycles of size 4 and augment paths of length 1
- two classes of slots: the ones of the company and the others
Conclusion

Firstly: Learn how to communicate with companies
After many months of discussion, hypotheses are changing...

Actual (?) constraints of the system/of Airline Operation Controllers
- Switch alternating cycles of size 4 and augment paths of length 1
- two classes of slots: the ones of the company and the others

Further theoretical work
- complexity of $\mu_k(G, M)$ in trees?
- in other graph classes?
Firstly: Learn how to communicate with companies
After many months of discussion, hypotheses are changing...

Actual (?) constraints of the system/of Airline Operation Controllers
- Switch alternating cycles of size 4 and augment paths of length 1
- two classes of slots: the ones of the company and the others

Further theoretical work
- complexity of $\mu_k(G, M)$ in trees?
- in other graph classes?

Thank you, 谢谢, Merci!