REU Problems of the Czech Group

Adam Juraszek, Tomáš Masařík, Jitka Novotná, Martin Töpfer, Tomáš Toufar, Jan Voborník, Peter Zeman

Mentor: Dr. James Abello

DIMACS REU 2015, Piscataway
Dominating Sets on Colored Tournaments
A tournament is a complete graph with orientated edges.

A vertex $v$ is dominated by $u$ if there is an edge $u \rightarrow v$.

A dominating set is a set of vertices called dominators such that every vertex of the graph is a dominator or is dominated by at least one.
A **tournament** is a complete graph with orientated edges.

A vertex $v$ is **dominated by** $u$ if there is an edge $u \rightarrow v$.

A **dominating set** is a set of vertices called dominators such that every vertex of the graph is a dominator or is dominated by at least one.
Every edge is red, blue or green.

Edges of the same color satisfy transitivity.

\[ u \rightarrow v \& v \rightarrow w \implies u \rightarrow w \]
3-colored Transitive Tournaments

- Every edge is red, blue or green.
- Edges of the same color satisfy transitivity.
  \[ u \rightarrow v \land v \rightarrow w \implies u \rightarrow w \]
Every edge is red, blue or green.

Edges of the same color satisfy transitivity.

\[ u \rightarrow v \land v \rightarrow w \implies u \rightarrow w \]
3-colored Transitive Tournaments

- Every edge is red, blue or green.
- Edges of the same color satisfy transitivity.

\[ u \rightarrow v \land v \rightarrow w \implies u \rightarrow w \]
A rainbow triangle is a triangle with edges oriented in a cycle, and therefore it must use three different colors.
A rainbow triangle is a triangle with edges oriented in a cycle, and therefore it must use three different colors.

Every transitively 3-colored tournament is either isomorphic to a linear order, or it contains a rainbow triangle.
A rainbow triangle is a triangle with edges oriented in a cycle, and therefore it must use three different colors.

- Every transitively 3-colored tournament is either isomorphic to a linear order, or it contains a rainbow triangle.
- Every transitively 3-colored tournament with minimum dominating set of size three has two edge-disjoint rainbow triangles.
3-colored Transitive Tournaments

Paley on 7 vertices and its dominating set
3-colored Transitive Tournaments

Paley on 7 vertices and its dominating set
$K_3$ fully substituted into $K_3$ and its dominating set
$K_3$ fully substituted into $K_3$ and its dominating set
3-colored Transitive Tournaments

A random graph on 15 vertices and its dominating set
A random graph on 15 vertices and its dominating set
There exists a vertex in a tournaments which dominates $\left\lfloor \frac{N}{2} \right\rfloor$ vertices.

Induced subgraph of a tournament is a tournament.
There exists a vertex in a tournaments which dominates $\lceil \frac{N}{2} \rceil$ vertices.

Induced subgraph of a tournament is a tournament.

Therefore, there exists a lower bound on number of vertices for tournament to have minimum dominating set of size $D$.

$$N = 2^D - 1$$

<table>
<thead>
<tr>
<th>$D$</th>
<th>$N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>31</td>
</tr>
</tbody>
</table>

\(^1\text{http://oeis.org/A000225}\)
Substitution

Replaces a vertex with a subgraph ($K_3$ in our case).
Substitution

Replaces a vertex with a subgraph ($K_3$ in our case).
Replaces a vertex with a subgraph ($K_3$ in our case).
Substitution

Replaces a vertex with a subgraph (\(K_3\) in our case).
Reduction

Removes all vertices whose removal do not change the size of a minimum dominating set.
Reduction

Removes all vertices whose removal do not change the size of a minimum dominating set.
Reduction

Removes all vertices whose removal do not change the size of a minimum dominating set.
Every graph can be divided into strongly connected components which form an acyclic graph. In case of tournaments, this graph is a path.
Strongly Connected Components

- Every graph can be divided into \textit{strongly connected components} which form an \textit{acyclic graph}. In case of tournaments, this graph is a \textit{path}.
- All minimum dominating sets are in the first strongly connected component. Only strongly connected tournaments are interesting for our study.
Every graph can be divided into **strongly connected components** which form an **acyclic graph**. In case of tournaments, this graph is a **path**.

All minimum dominating sets are in the first strongly connected component. Only strongly connected tournaments are interesting for our study.

A strongly connected tournament has a **Hamiltonian cycle**.
Every tournament can be constructed by introduction of a new vertices and orientation/coloring of edges to all other vertices.
Every tournament can be constructed by introduction of a new vertices and orientation/coloring of edges to all other vertices.

Addition of the new vertex can:

- decrease number of strongly connected components arbitrarily and increase by one,
- decrease size of minimum dominating set arbitrarily and increase by one,
Every tournament can be constructed by introduction of a new vertex and orientation/coloring of edges to all other vertices. Addition of the new vertex can:

- decrease number of strongly connected components arbitrarily and increase by one,
- decrease size of minimum dominating set arbitrarily and increase by one,

It is sufficient to study edges incident to the first strongly connected component. There must be edges both incoming and outgoing.
In order to increase size of the minimum dominating set $D$ to $D + 1$, the new vertex must dominate all vertices which are in any minimum dominating set and must be dominated by a subtournament having minimum dominating set of size $D$. 

![Diagram: A vertex $v$ is shown surrounded by a circle labeled 'dom'. The vertex is designed to dominate all other vertices within the circle.]
Inductive Construction

In order to increase size of the minimum dominating set $D$ to $D + 1$, the new vertex must dominate all vertices which are in any minimum dominating set and must be dominated by a subtournament having minimum dominating set of size $D$. 
In order to increase size of the minimum dominating set $D$ to $D + 1$, the new vertex must dominate all vertices which are in any minimum dominating set and must be dominated by a subtournament having minimum dominating set of size $D$. 
Inductive Construction

In order to increase size of the minimum dominating set $D$ to $D + 1$, the new vertex must dominate all vertices which are in any minimum dominating set and must be dominated by a subtournament having minimum dominating set of size $D$. 
A **2-majority graph** is a graph induced by three linear orders voting for edge direction.

2-majority graphs have at most three dominators. If they do not have one dominator, then they have a dominating set of size three on a cyclic triangle.
2-Majority Graphs

- A **2-majority graph** is a graph induced by three linear orders voting for edge direction.
- 2-majority graphs have at most three dominators. If they do not have one dominator, then they have a dominating set of size three on a cyclic triangle.
- Any **partially ordered set** (poset) can be extended into a linear order.
- Graphs induced by two of three colors in tournaments are often acyclic (form a poset).
$K_{10}$ which is acyclic in all three induced subgraphs.
$K_{10}$ which is acyclic in all three induced subgraphs.
$K_{11}$ which contains cycles in at least one induced subgraph.
2-Majority Graphs

$K_{11}$ which contains cycles in at least one induced subgraph.
Constraint model in MiniZinc for generating tournaments with prescribed properties:
- number of vertices,
- transitive coloring,
- must have a dominating set of a certain size,
- must be strongly connected.
Tournament Generation and Tools

- Constraint model in MiniZinc for generating tournaments with prescribed properties:
  - number of vertices,
  - transitive coloring,
  - must have a dominating set of a certain size,
  - must be strongly connected.

- Python scripts for processing tournaments:
  - visualization of tournaments,
  - calculation and visualization of dominating sets,
  - substitution and reduction of tournaments,
  - testing if all tournaments satisfy a certain property.
Future Work

- Describe tournaments which contain cycles in subgraphs induced by two colors. Can they be 2-majority graphs?
- Study transitively 4-colored tournaments and their connection to $k$-majority graphs.
Questions?
Complexity of Fair Deletion Problems
Some important algorithmic problems in graph theory can be stated as **deletion problems** – i.e. find a set such that graph satisfies certain property after deletion of such a set.
Some important algorithmic problems in graph theory can be stated as deletion problems – i.e. find a set such that graph satisfies certain property after deletion of such a set.

- **Vertex Cover** - Find a set $W \subseteq V$ such that $G[V \setminus W]$ contains no edge.
Some important algorithmic problems in graph theory can be stated as deletion problems – i.e. find a set such that graph satisfies certain property after deletion of such a set.

- **Vertex Cover** - Find a set $W \subseteq V$ such that $G[V \setminus W]$ contains no edge
Some important algorithmic problems in graph theory can be stated as **deletion problems** – i.e. find a set such that graph satisfies certain property after deletion of such a set.

- **Vertex Cover** - Find a set $W \subseteq V$ such that $G[V \setminus W]$ contains no edge
Some important algorithmic problems in graph theory can be stated as deletion problems – i.e. find a set such that graph satisfies certain property after deletion of such a set.

- **Vertex Cover** - Find a set $W \subseteq V$ such that $G[V \setminus W]$ contains no edge
- **Feedback Vertex Set** - Find a set $W \subseteq V$ such that $G[V \setminus W]$ is acyclic
Some important algorithmic problems in graph theory can be stated as **deletion problems** – i.e. find a set such that graph satisfies certain property after deletion of such a set.

- **Vertex Cover** - Find a set $W \subseteq V$ such that $G[V \setminus W]$ contains no edge
- **Feedback Vertex Set** - Find a set $W \subseteq V$ such that $G[V \setminus W]$ is acyclic
- **Feedback Arc Set** - Find a set $F \subseteq E$ such that $(V, E \setminus F)$ is acyclic
Graph Deletion Problems

Some important algorithmic problems in graph theory can be stated as deletion problems – i.e. find a set such that graph satisfies certain property after deletion of such a set.

- **Vertex Cover** - Find a set $W \subseteq V$ such that $G[V \setminus W]$ contains no edge
- **Feedback Vertex Set** - Find a set $W \subseteq V$ such that $G[V \setminus W]$ is acyclic
- **Feedback Arc Set** - Find a set $F \subseteq E$ such that $(V, E \setminus F)$ is acyclic
- **Odd Cycle Transversal** - Find a set $F \subseteq E$ such that $G \setminus F$ is bipartite
Some important algorithmic problems in graph theory can be stated as deletion problems – i.e. find a set such that graph satisfies certain property after deletion of such a set.

- **Vertex Cover** - Find a set $W \subseteq V$ such that $G[V \setminus W]$ contains no edge

- **Feedback Vertex Set** - Find a set $W \subseteq V$ such that $G[V \setminus W]$ is acyclic

- **Feedback Arc Set** - Find a set $F \subseteq E$ such that $(V, E \setminus F)$ is acyclic

- **Odd Cycle Transversal** - Find a set $F \subseteq E$ such that $G \setminus F$ is bipartite

The goal is to find the smallest such set.
Sometimes the smallest set is not what we want.

- We have a communication network, we want to remove redundancies
- We do not want to reduce the capability of each vertex by too much (a local condition in contrast to the previous global one)
Fair Versions of Deletion Problems

Sometimes the smallest set is not what we want.

- We have a communication network, we want to remove redundancies
- We do not want to reduce the capability of each vertex by too much (a local condition in contrast to the previous global one)

This solution removes a lot of links from single vertex.
Sometimes the smallest set is not what we want.

- We have a communication network, we want to remove redundancies
- We do not want to reduce the capability of each vertex by too much (a \textit{local} condition in contrast to the previous global one)

A much better solution.
Optimal fair solution might contain much more edges in total than optimal classical solution.
Fair Versions of Deletion Problems

Optimal fair solution might contain much more edges in total than optimal classical solution.

Optimal solution for the classical case
Optimal fair solution might contain much more edges in total than optimal classical solution.

Optimal solution for the fair case
Classical versions of those problems are NP-complete.
Classical versions of those problems are \textit{NP-complete}. In most cases, the fair versions are \textit{NP-complete} too.
Parameterized Complexity

- One way how to deal with NP-complete problems is parameterized complexity
One way how to deal with NP-complete problems is parameterized complexity.

Idea: Perhaps we don’t encounter the worst case instances that often.
One way how to deal with NP-complete problems is parameterized complexity.

Idea: Perhaps we don’t encounter the worst case instances that often.

Our typical instances might have some structure we can exploit (bounded degree, planarity, sparsity...).
One way how to deal with NP-complete problems is parameterized complexity.

Idea: Perhaps we don’t encounter the worst case instances that often.

Our typical instances might have some structure we can exploit (bounded degree, planarity, sparsity...).

If we add parameter describing the structure, we get a finer classification of complexity.
Parameterized Complexity

- One way how to deal with NP-complete problems is \textit{parameterized complexity}
- Idea: Perhaps we don’t encounter the worst case instances that often
- Our typical instances might have some structure we can exploit (bounded degree, planarity, sparsity...)
- If we add \textit{parameter} describing the structure, we get a finer classification of complexity
- This often leads to fast algorithms when the parameter is small
One way how to deal with NP-complete problems is parameterized complexity.

Idea: Perhaps we don’t encounter the worst case instances that often.

Our typical instances might have some structure we can exploit (bounded degree, planarity, sparsity...).

If we add parameter describing the structure, we get a finer classification of complexity.

This often leads to fast algorithms when the parameter is small.

Classical results:
One way how to deal with NP-complete problems is parameterized complexity.

Idea: Perhaps we don’t encounter the worst case instances that often.

Our typical instances might have some structure we can exploit (bounded degree, planarity, sparsity...).

If we add parameter describing the structure, we get a finer classification of complexity.

This often leads to fast algorithms when the parameter is small.

Classical results:

- $2^k n$ time algorithm for VERTEX COVER (where $k$ is the parameter – the size of vertex cover).
Parameterized Complexity

- One way how to deal with NP-complete problems is parameterized complexity.
- Idea: Perhaps we don’t encounter the worst case instances that often.
- Our typical instances might have some structure we can exploit (bounded degree, planarity, sparsity...).
- If we add parameter describing the structure, we get a finer classification of complexity.
- This often leads to fast algorithms when the parameter is small.
- Classical results:
  - $2^k n$ time algorithm for Vertex Cover (where $k$ is the parameter – the size of vertex cover).
  - $f(k)n$ time algorithm for MSO$_2$ model checking for graphs of treewidth $\leq k$ (Courcelle’s theorem).
The central class of parameterized complexity is **FPT – Fixed parameter tractable**
The central class of parameterized complexity is **FPT – Fixed parameter tractable**

It is the class of all problems that can be solved in time \( f(k)n^{O(1)} \) (for example \( 2^k n^2 \))
The central class of parameterized complexity is FPT – Fixed parameter tractable. It is the class of all problems that can be solved in time $f(k)n^{O(1)}$ (for example $2^k n^2$). Another important class is $W[1]$ – problems that are $W[1]$-complete are believed to be outside of FPT.
The central class of parameterized complexity is FPT – Fixed parameter tractable. It is the class of all problems that can be solved in time $f(k)n^{O(1)}$ (for example $2^k n^2$).

Another important class is $W[1]$ – problems that are $W[1]$-complete are believed to be outside of FPT.

The last important is $XP$ – the class of problems that can be solved in time $O(n^{f(k)})$ (for example $n^k$, $n^{k^2}$, ...).
The central class of parameterized complexity is FPT – Fixed parameter tractable

It is the class of all problems that can be solved in time $f(k)n^{O(1)}$ (for example $2^k n^2$)

Another important class is $W[1]$ – problems that are $W[1]$-complete are believed to be outside of FPT

The last important is XP – the class of problems that can be solved in time $O(n^{f(k)})$ (for example $n^k$, $n^{k^2}$, ...)

Problems in XP still have a polynomial algorithm for every fixed $k$, however the degree can grow rapidly with $k$
The central class of parameterized complexity is FPT – Fixed parameter tractable.

It is the class of all problems that can be solved in time $f(k)n^{O(1)}$ (for example $2^kn^2$).

Another important class is $W[1]$ – problems that are $W[1]$-complete are believed to be outside of FPT.

The last important is XP – the class of problems that can be solved in time $O(n^{f(k)})$ (for example $n^k, n^{k^2}, \ldots$).

Problems in XP still have a polynomial algorithm for every fixed $k$, however the degree can grow rapidly with $k$.

Think of FPT as analogue to P (solvable quickly) and of $W[1]$-complete as NP-complete (likely not solvable quickly).
Previous Results

Every MSO$_2$-definable fair deletion problem is solvable in time $f(k)n^{O(k)}$ on graphs of treewidth at most $k$ (XP algorithm) (Kolman, Lidický, Sereni)

Example of MSO formula:

\[
vertex\_cover(W) \equiv (\forall u, v \in V \setminus W)(\neg adj(u, v))
\]
Hardness results:

- We have proven that the metatheorem is not FPT unless $W[1] = FPT$. 
Our Results – Hardness

Hardness results:

- We have proven that the metatheorem is not FPT unless $W[1] = FPT$.

- Moreover, if it can be solved in time $f(k)n^{o(\sqrt{k})}$, then Exponential time hypothesis fails.
Our Results – Hardness

Hardness results:

- We have proven that the metatheorem is not FPT unless $W[1] = \text{FPT}$.
- Moreover, if it can be solved in time $f(k)n^{o(\sqrt{k})}$, then Exponential time hypothesis fails.
- Both results also extend to parameterization by pathwidth and feedback vertex set.
Our Results – Hardness

Hardness results:

- We have proven that the metatheorem is not FPT unless $W[1] = \text{FPT}$.
- Moreover, if it can be solved in time $f(k)n^{o(\sqrt{k})}$, then Exponential time hypothesis fails.
- Both results also extend to parameterization by pathwidth and feedback vertex set.
- Both results continue to hold if we take vertex version of metatheorem with MSO$_1$ formula.
Our Results – Tractability

Tractability:

- We have proven that vertex version of metatheorem is FPT for parameterization by neighborhood diversity
Our Results – Tractability

Tractability:

- We have proven that vertex version of metatheorem is FPT for parameterization by neighborhood diversity.
- Both vertex and edge version are FPT for parameterization by size of vertex cover.
Summary of Parameters

Vertex case:

- Parameters for which we have proven the metatheorem is hard (not FPT unless $W[1] = FPT$)
- Parameters for which it was known that metatheorem is hard
- Parameters for which we have proven the metatheorem is easy (FPT)
Summary of Parameters

Edge case:

Parameters for which we have proven the metatheorem is hard (not FPT unless $W[1] = FPT$)
Parameters for which it was known that metatheorem is hard
Parameters for which we have proven the metatheorem is easy (FPT)
If we have parameter for which the metatheorem is hard, it is still of interest to study specific problems.
If we have parameter for which is the metatheorem hard, it is still of interest to study specific problems.

Example: \textsc{Odd Cycle Transversal} parameterized by neighborhood diversity
If we have parameter for which is the metatheorem hard, it is still of interest to study specific problems.

Example: **Odd Cycle Transversal** parameterized by neighborhood diversity

We have reduced the **Fair Odd Cycle Transversal** parameterized by neighborhood diversity to **Integer Linear Programming** parameterized by dimension, thus giving FPT algorithm
Can be the bound $f(k)n^{o(\sqrt{k})}$ be improved to $f(k)n^{o(k)}$? (which is essentially tight)
Can be the bound $f(k)n^{o(\sqrt{k})}$ be improved to $f(k)n^{o(k)}$? (which is essentially tight)

What edge fair problems are still easy for parameterization by neighborhood diversity?
Open Questions and Future Work

- Can the bound $f(k)n^{o(\sqrt{k})}$ be improved to $f(k)n^{o(k)}$? (which is essentially tight)
- What edge fair problems are still easy for parameterization by neighborhood diversity?
- **Fair Odd Cycle Transversal** can be described by ILP – can classical **Odd Cycle Transversal** be reduced to minimizing quasiconvex function?
Can be the bound $f(k)n^{o(\sqrt{k})}$ be improved to $f(k)n^{o(k)}$? (which is essentially tight)

What edge fair problems are still easy for parameterization by neighborhood diversity?

Fair Odd Cycle Transversal can be described by ILP – can classical Odd Cycle Transversal be reduced to minimizing quasiconvex function?

Is Fair Feedback Arc Set parameterized by neighborhood diversity hard?
Questions?
Subclasses of Chordal Graphs

Martin Töpfer, Jan Voborník, Peter Zeman
Interval graphs are intersection graphs of **intervals of the real line**.
Interval graphs are intersection graphs of intervals of the real line.
Interval graphs are intersection graphs of intervals of the real line.
Interval graphs are intersection graphs of intervals of the real line.

Many computational problems are polynomially solvable for interval graphs (graph isomorphism, maximum clique, \(k\)-coloring, maximum independent set, ...).
A graph is a **chordal graph** if and only if there exists a tree $T$ such that it can be represented as an **intersection graph of subtrees** of the tree $T$. 
A graph is a **chordal graph** if and only if there exists a tree $T$ such that it can be represented as an **intersection graph of subtrees** of the tree $T$. 
A graph is a **chordal graph** if and only if there exists a tree $T$ such that it can be represented as an **intersection graph of subtrees** of the tree $T$. 
A graph is a **chordal graph** if and only if there exists a tree $T$ such that it can be represented as an **intersection graph of subtrees** of the tree $T$. 
A graph is a **chordal graph** if and only if there exists a tree $T$ such that it can be represented as an **intersection graph of subtrees of** the tree $T$. 
A graph is a **chordal graph** if and only if there exists a tree $T$ such that it can be represented as an **intersection graph of subtrees** of the tree $T$. 
A graph is a **chordal graph** if and only if there exists a tree $T$ such that it can be represented as an **intersection graph of subtrees** of the tree $T$. 
A graph is a **chordal graph** if and only if there exists a tree $T$ such that it can be represented as an **intersection graph of subtrees** of the tree $T$. 
A graph is a **chordal graph** if and only if there exists a tree $T$ such that it can be represented as an **intersection graph of subtrees** of the tree $T$.

It is easy to see that $\text{INT} \subsetneq \text{CHOR}$. 
For a fixed tree $T$, the class $T$-GRAPH is the class of all graphs which can be represented as an intersection graph of subtrees of the tree $T$. 
The Class $T$-GRAPH

For a fixed tree $T$, the class $T$-GRAPH is the class of all graphs which can be represented as an intersection graph of subtrees of the tree $T$.

We obtain an infinite hierarchy between INT and CHOR, since

\[ \text{INT } \subseteq \ T$-GRAPH \nsubseteq \ CHOR. \]
The Class $T$-GRAPH

For a fixed tree $T$, the class $T$-GRAPH is the class of all graphs which can be represented as an intersection graph of subtrees of the tree $T$.

We obtain an infinite hierarchy between INT and CHOR, since

\[ \text{INT} \subseteq T\text{-GRAPH} \not\subsetneq \text{CHOR}. \]

Our main aim was to study structural and computational properties of the classes $T$-GRAPH.
Recognition and Representation Extension Problems

**Problem:** Recognition – $\text{RECOG}(C)$

**Input:** A graph $G$.

**Question:** Does $G$ belong to the class $C$?

**Problem:** Perfect Representation Extension – $\text{REPEXT}(C)$

**Input:** A graph $G$ and a partial representation $R'$.

**Question:** Is there a representation $R$ of $G$ extending $R'$?


Our result: An algorithm for $\text{REPEXT}(T\text{-GRAPH})$ with running time $n^{O(\|T\|)}$. 
A graph $G$ is a chordal graph if and only if there exists a linear ordering of its vertices $v_1, \ldots, v_n$ such that for every vertex $v_i$, its neighbors on the right form a clique.

It follows that every chordal graph has at most $n$ maximal cliques.
Every $G \in S_d$-GRAPH has a representation with a maximal clique in the branching point.

We can try to place each maximal clique on the branching point.

The rest of the graph has to be an interval graph. We need to correctly place the remaining connected components on the branches.
Every $G \in S_d$-GRAPH has a representation with a maximal clique in the branching point.

We can try to place each maximal clique on the branching point.

The rest of the graph has to be an interval graph. We need to correctly place the remaining connected components on the branches.
Every $G \in S_d$-GRAPH has a representation with a maximal clique in the branching point.

We can try to place each maximal clique on the branching point.

The rest of the graph has to be an interval graph. We need to correctly place the remaining connected components on the branches.
For components $C$ and $C'$, we define that $C \succeq C'$ if and only if

(i) $N_K(C) \supseteq N_K(C')$,

(ii) There is no vertex in $K$ adjacent to a proper subset of $V(C)$ and a proper subset of $V(C')$. 

\[
\begin{align*}
X_1 & \quad X_2 & \quad X_3 & \quad X_4 & \quad X_5 & \quad X_6 \\
1 & \quad 2 & \quad 3 & \quad 4 & \quad K
\end{align*}
\]
For components $C$ and $C'$, we define that $C \succeq C'$ if and only if

(i) $N_K(C) \supseteq N_K(C')$,

(ii) There is no vertex in $K$ adjacent to a proper subset of $V(C)$ and a proper subset of $V(C')$. 

\begin{figure}
\centering
\includegraphics[width=\textwidth]{diagram}
\caption{Diagram illustrating the adjacency and subset relationships for components in $K$.}
\end{figure}
Finding a Correct Placement of Components

For components \( C \) and \( C' \), we define that \( C \succeq C' \) if and only if

(i) \( N_K(C) \supseteq N_K(C') \),

(ii) There is no vertex in \( K \) adjacent to a proper subset of \( V(C) \) and a proper subset of \( V(C') \).
Finding a Correct Placement of Components

For components $C$ and $C'$, we define that $C \succeq C'$ if and only if

(i) $N_K(C) \supseteq N_K(C')$,

(ii) There is no vertex in $K$ adjacent to a proper subset of $V(C)$ and a proper subset of $V(C')$. 
For components $C$ and $C'$, we define that $C \succeq C'$ if and only if

(i) $N_K(C) \supseteq N_K(C')$,

(ii) There is no vertex in $K$ adjacent to a proper subset of $V(C)$ and a proper subset of $V(C')$. 
For components \( C \) and \( C' \), we define that \( C \succeq C' \) if and only if

(i) \( N_K(C) \supseteq N_K(C') \),

(ii) There is no vertex in \( K \) adjacent to a proper subset of \( V(C) \) and a proper subset of \( V(C') \).
For components $C$ and $C'$, we define that $C \succeq C'$ if and only if
(i) $N_K(C) \supseteq N_K(C')$,
(ii) There is no vertex in $K$ adjacent to a proper subset of $V(C)$ and a proper subset of $V(C')$. 
For components $C$ and $C'$, we define that $C \succeq C'$ if and only if

(i) $N_K(C) \supseteq N_K(C')$,

(ii) There is no vertex in $K$ adjacent to a proper subset of $V(C)$ and a proper subset of $V(C')$. 

\[ \begin{array}{c}
\text{X}_2 \\
\text{X}_5 \\
\text{X}_3 \\
\text{X}_4 \\
\text{X}_6 \end{array} \quad \begin{array}{c}
\text{1} \\
\text{2} \\
\text{3} \\
\text{4} \\
\text{5} \end{array} \quad \begin{array}{c}
\text{\{3\}} \\
\text{\{3,4\}} \\
\text{\{1,3\}} \\
\text{\{1,3,4\}} \\
\text{\{1,2,3\}} \end{array} \]
Finding a Correct Placement of Components

For components $C$ and $C'$, we define that $C \succeq C'$ if and only if

(i) $N_K(C) \supseteq N_K(C')$,

(ii) There is no vertex in $K$ adjacent to a proper subset of $V(C)$ and a proper subset of $V(C')$. 

\begin{itemize}
    \item $X_1$
    \item $X_2$
    \item $X_3$
    \item $X_4$
    \item $X_5$
    \item $X_6$
    \item $1$
    \item $2$
    \item $3$
    \item $4$
    \item $K$
    \item $\{3\}$
    \item $\{3, 4\}$
    \item $\{1, 3\}$
    \item $1$
    \item $2$
    \item $3$
    \item $4$
    \item $K$
    \item $\{1, 3\}$
    \item $\{3, 4\}$
    \item $\{1, 3, 4\}$
    \item $\{1, 2, 3\}$
    \item $\{3\}$
    \item $\{2\}$
\end{itemize}
Finding a Correct Placement of Components

For components $C$ and $C'$, we define that $C \succeq C'$ if and only if

(i) $N_K(C) \supseteq N_K(C')$,

(ii) There is no vertex in $K$ adjacent to a proper subset of $V(C)$ and a proper subset of $V(C')$. 

\[
\begin{align*}
X_1 & \quad 1 \\
X_2 & \quad 2 \\
X_3 & \quad 3 \\
X_4 & \quad 4 \\
X_5 & \quad 5 \\
X_6 & \quad 6
\end{align*}
\]
For general $T$, we assign a maximal clique to every branching point.

Then we determine which components have to be placed on the branches that are bounded by two branching points. The tree then breaks into stars.

The remaining components can be placed with a dynamic programming approach.
Thank you!