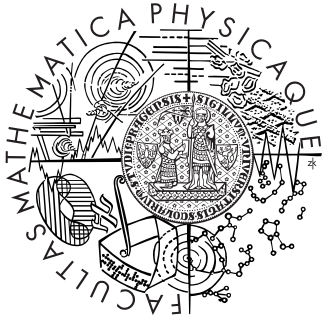
REU Problems of the Czech Group

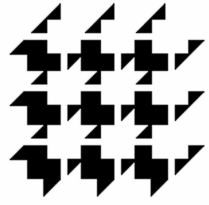
Adam Juraszek, Tomáš Masařík, Jitka Novotná,
Martin Töpfer, Tomáš Toufar, Jan Voborník, Peter Zeman

Mentor: Dr. James Abello



DIMACS

Center for Discrete Mathematics and Theoretical Computer Science
Founded as a National Science Foundation Science and Technology Center

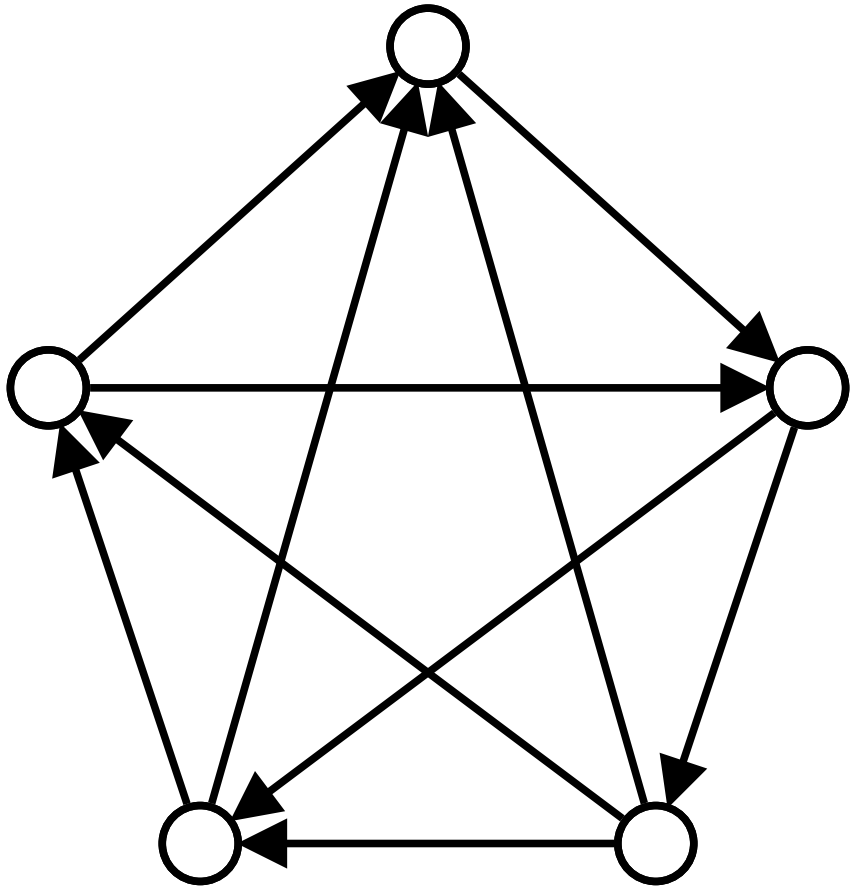


DIMACS REU 2015, Piscataway

Dominating Sets on Colored Tournaments

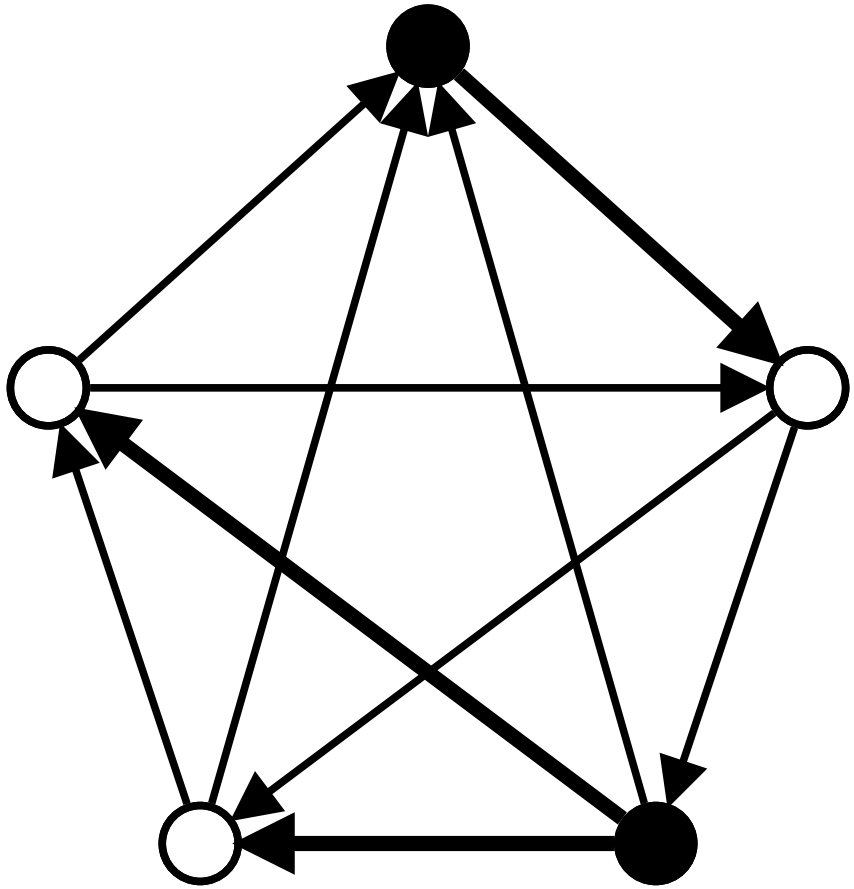
Dominating Set on Tournaments

- A tournament is a complete graph with orientated edges.
- A vertex v is dominated by u if there is an edge $u \to v$.
- 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.



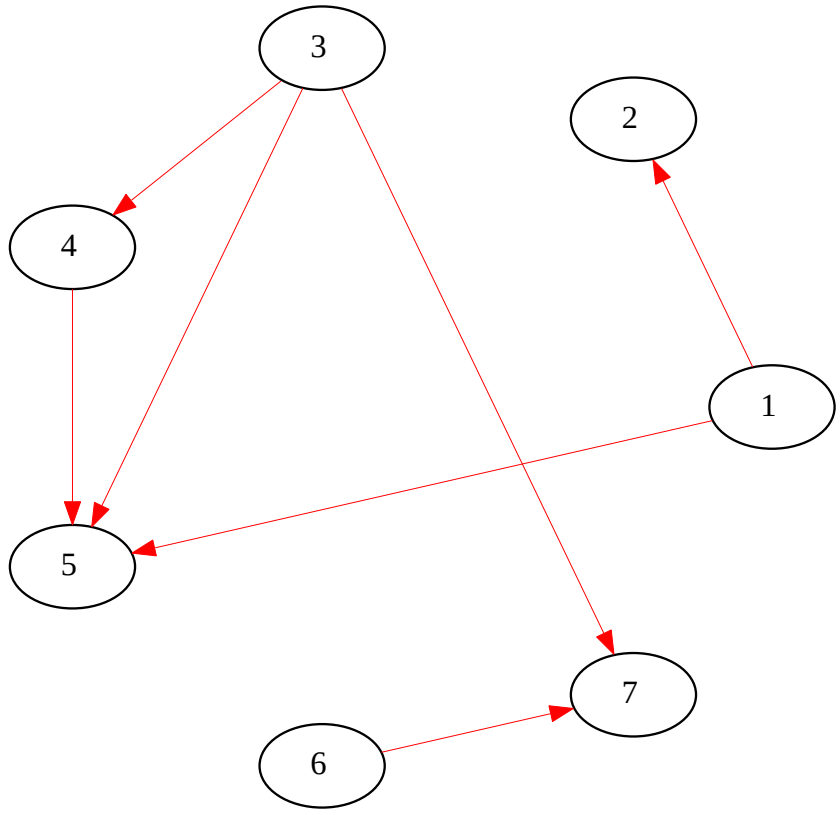
Dominating Set on Tournaments

- A tournament is a complete graph with orientated edges.
- A vertex v is dominated by u if there is an edge $u \to v$.
- 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.



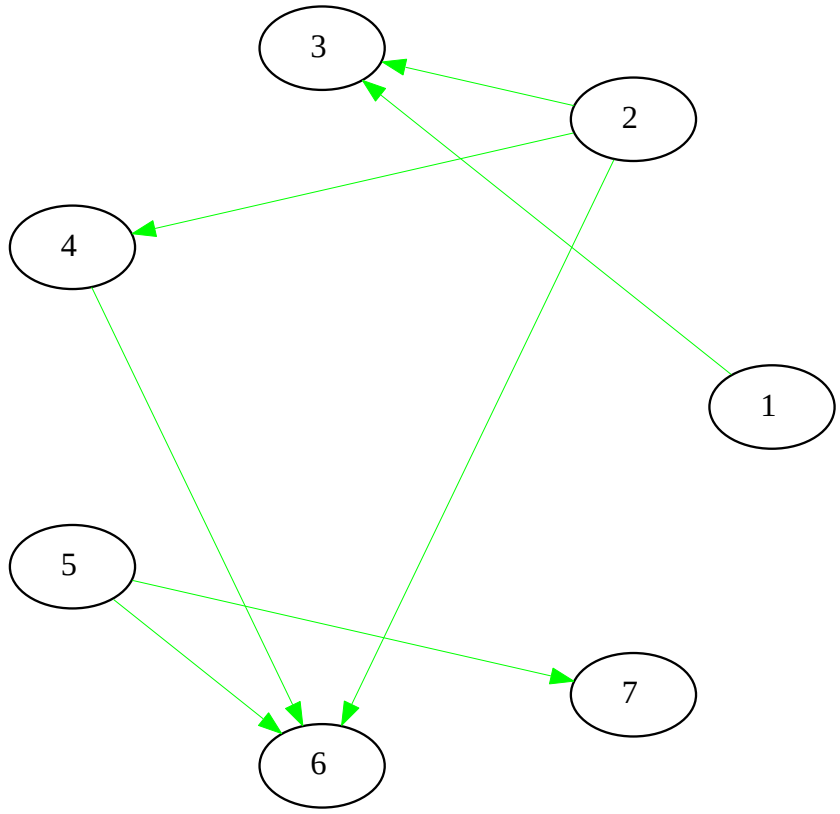
3-colored Transitive Tournaments

- Every edge is red, blue or green.
- Edges of the same color satisfy transitivity.
  $u \to v \& v \to w \implies u \to w$



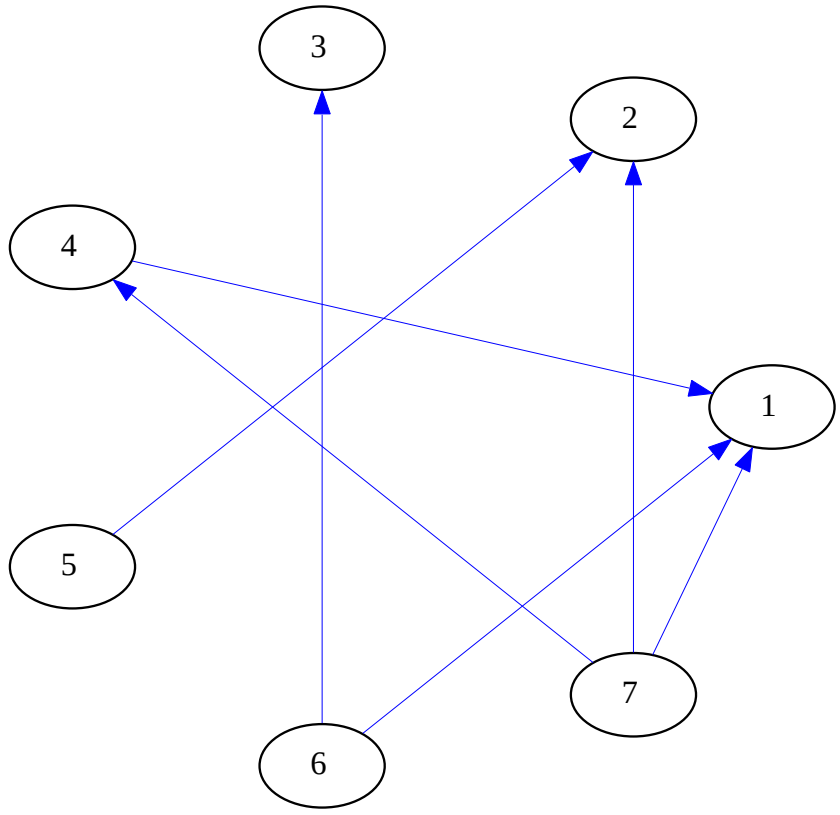
3-colored Transitive Tournaments

- Every edge is red, blue or green.
- Edges of the same color satisfy transitivity.
  $u \to v \& v \to w \implies u \to w$



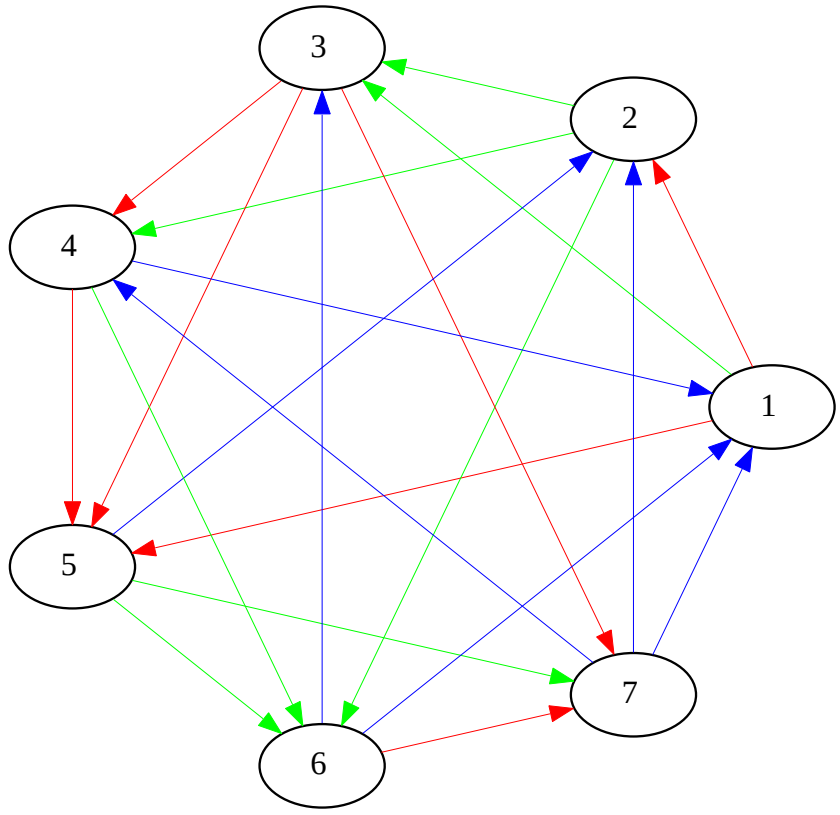
3-colored Transitive Tournaments

- Every edge is red, blue or green.
- Edges of the same color satisfy transitivity.
  $u \to v \& v \to w \implies u \to w$



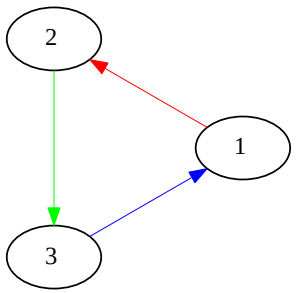
3-colored Transitive Tournaments

- Every edge is red, blue or green.
- Edges of the same color satisfy transitivity.
  $u \to v \& v \to w \implies u \to w$



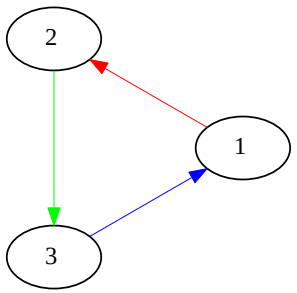
Rainbow Triangles

A rainbow triangle is a triangle with edges oriented in a cycle, and therefore it must use three different colors.



Rainbow Triangles

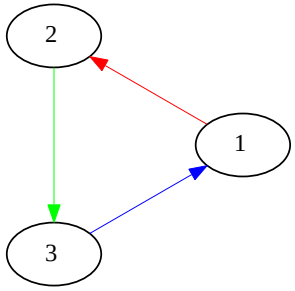
A rainbow triangle is a triangle with edges oriented in a cycle, and therefore it must use three different colors.



- Every transitively 3-colored tournament is either isomorphic to a linear order, or it contains a rainbow triangle.

Rainbow Triangles

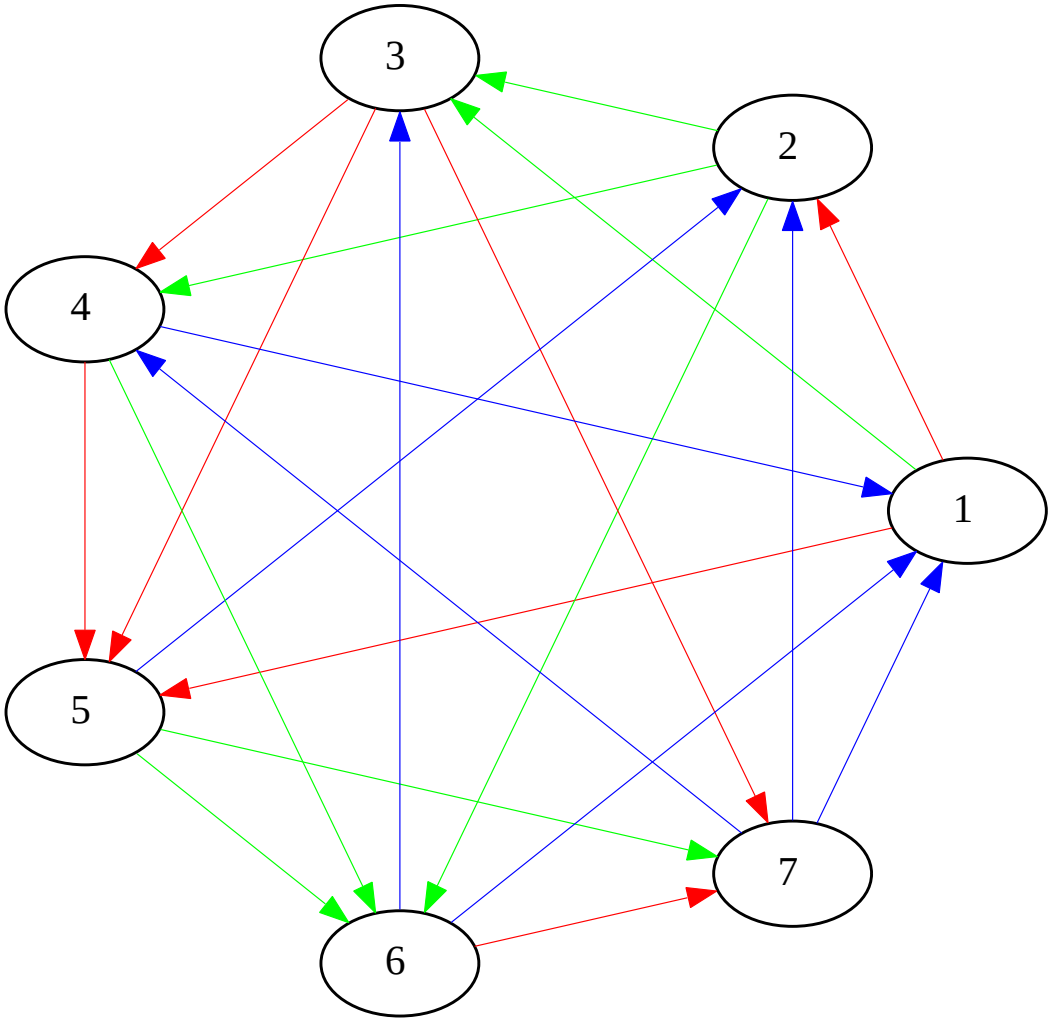
A rainbow triangle is a triangle with edges oriented in a cycle, and therefore it must use three different colors.



- 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.

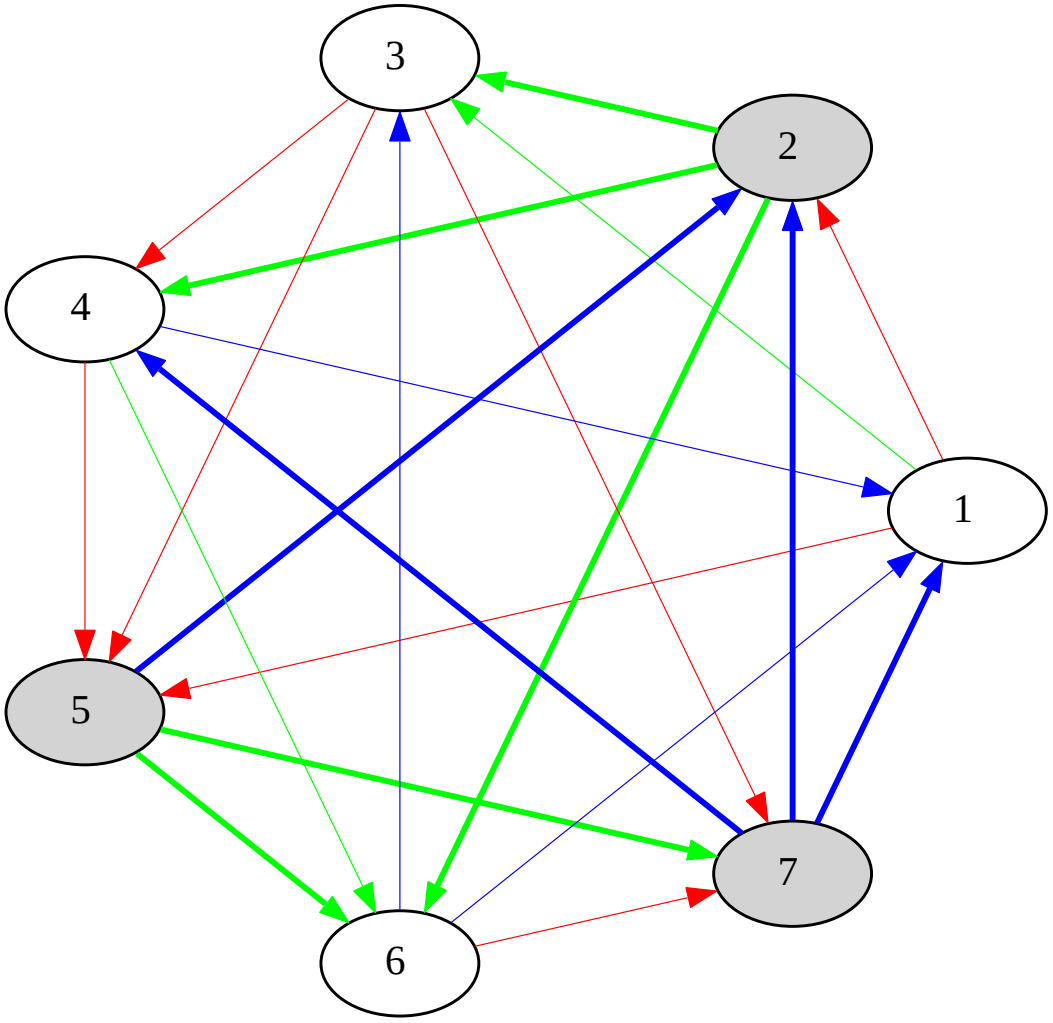
3-colored Transitive Tournaments

Paley on 7 vertices and its dominating set



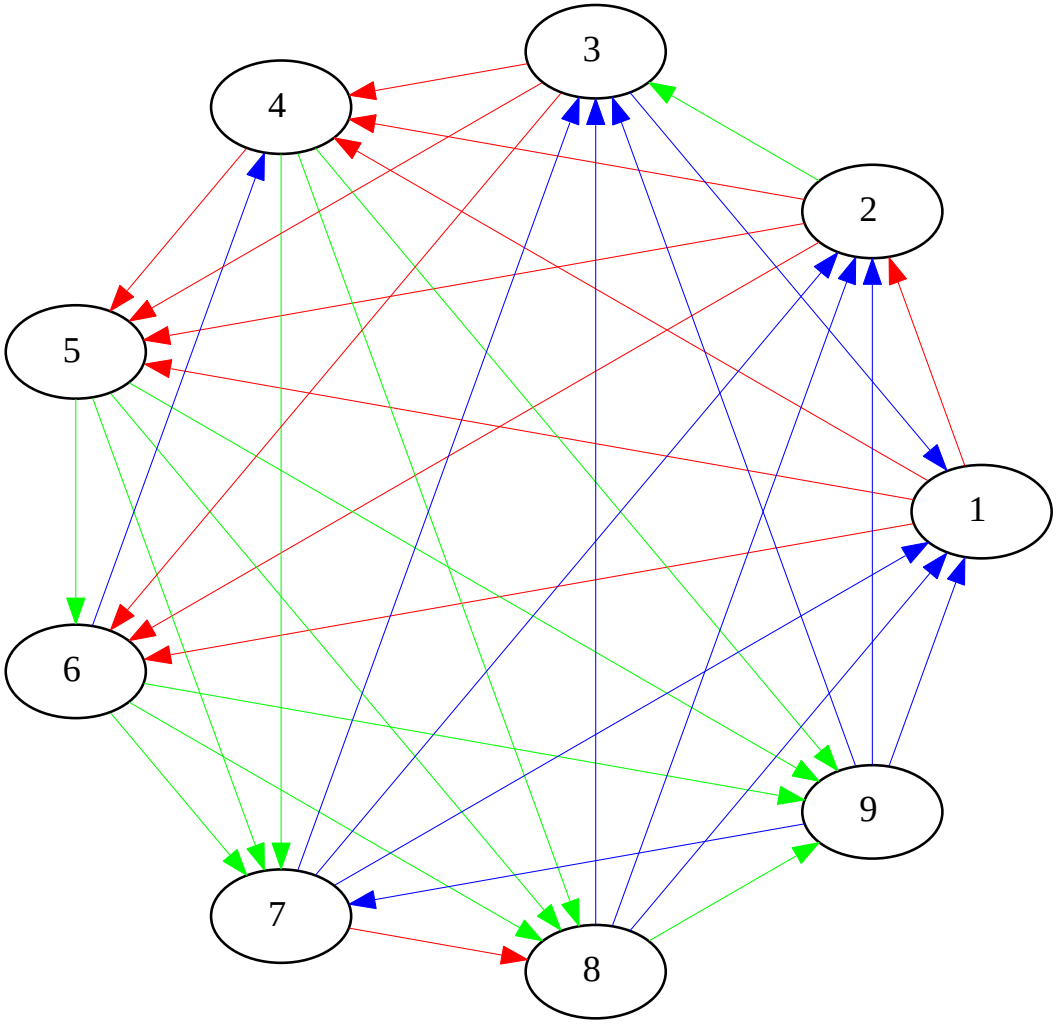
3-colored Transitive Tournaments

Paley on 7 vertices and its dominating set



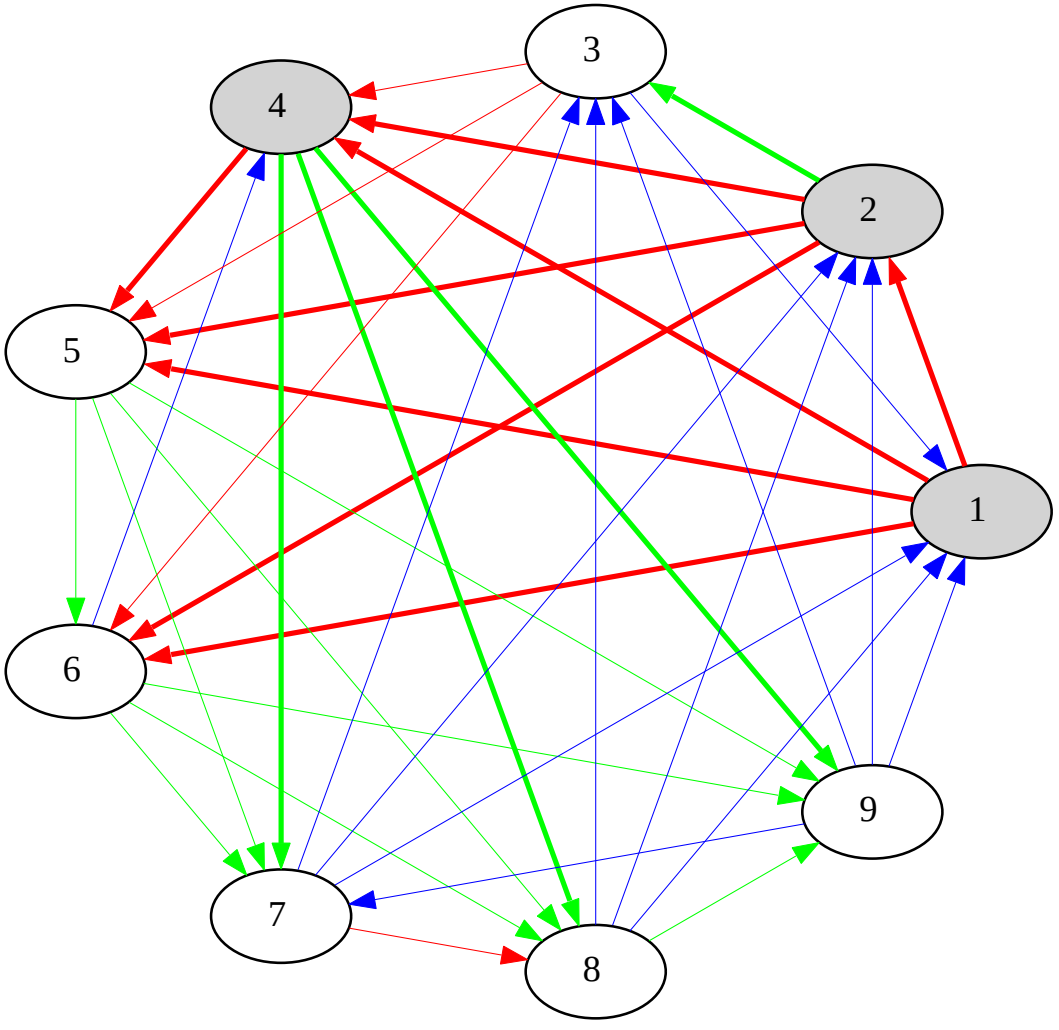
3-colored Transitive Tournaments

K_3 fully substituted into K_3 and its dominating set



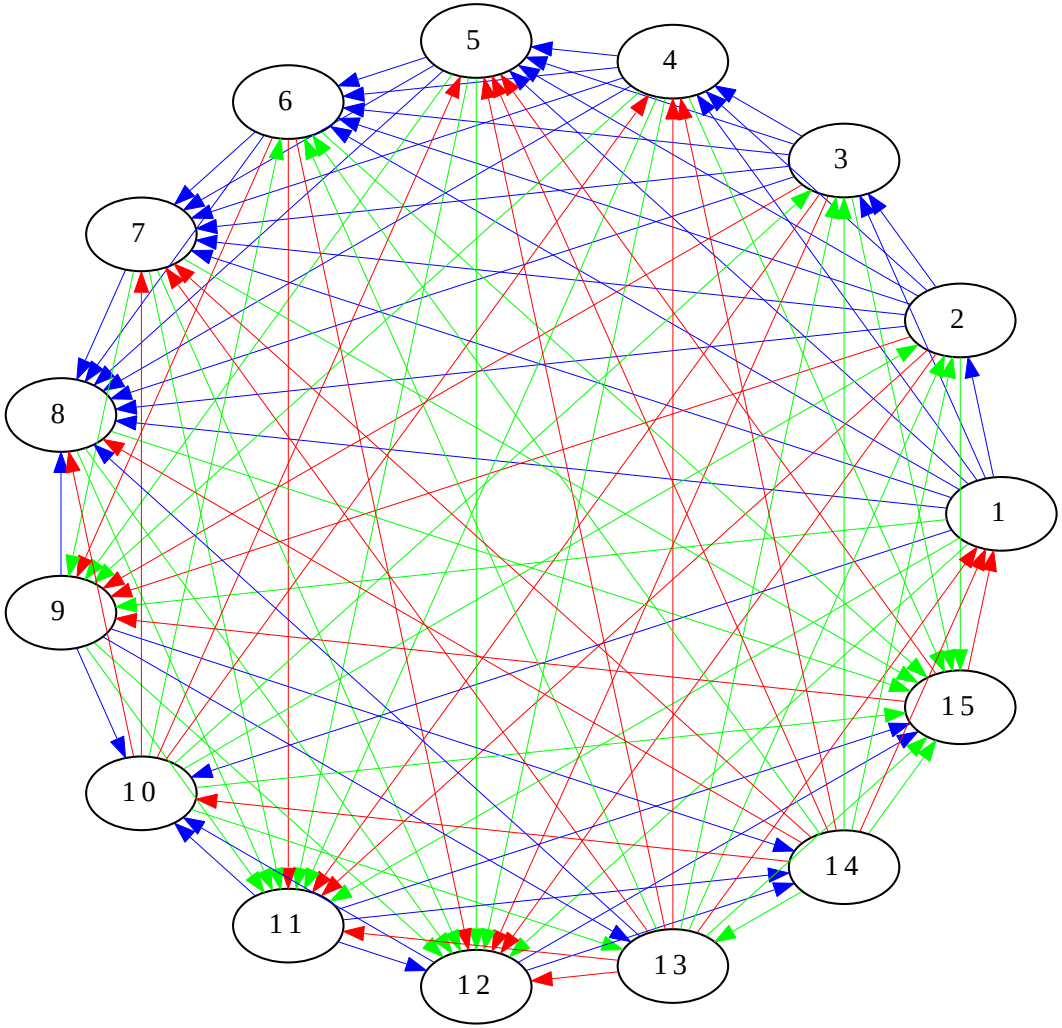
3-colored Transitive Tournaments

K_3 fully substituted into K_3 and its dominating set



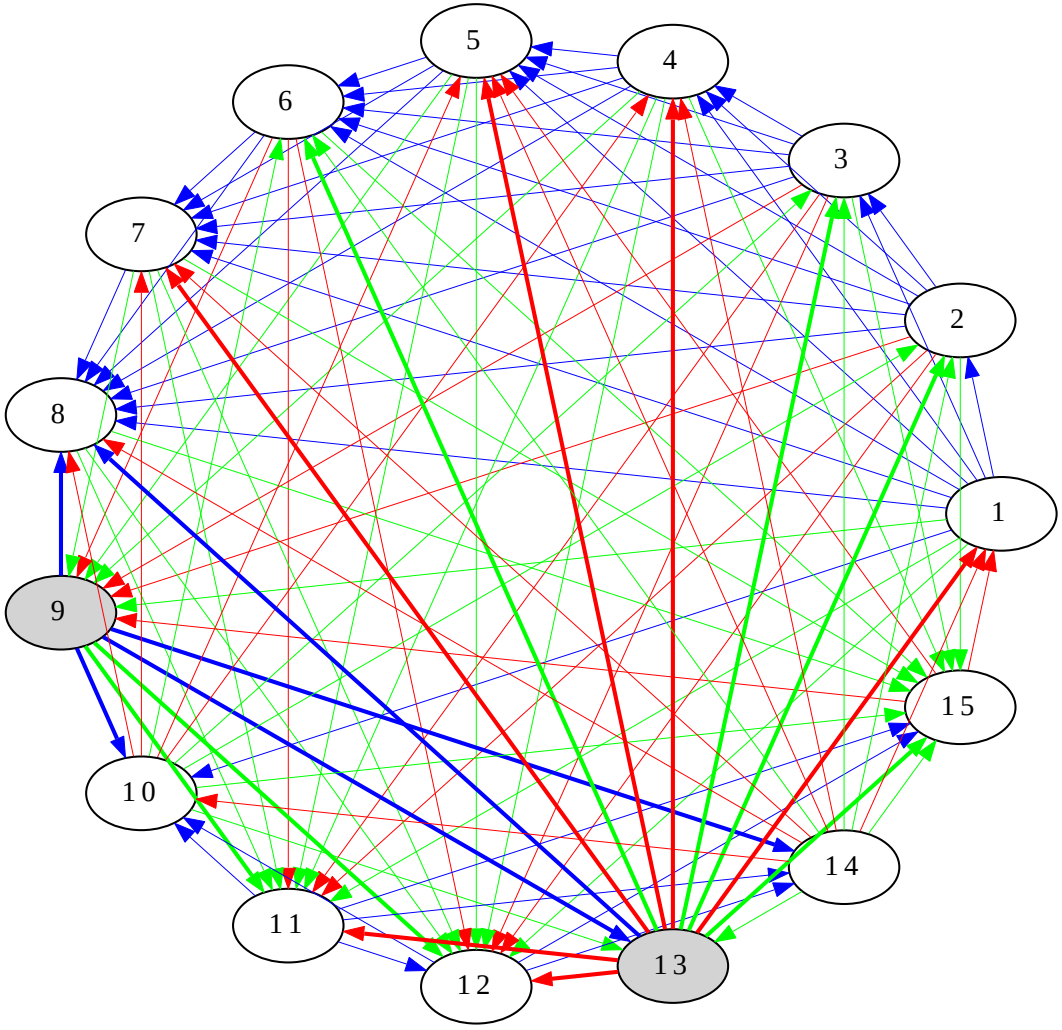
3-colored Transitive Tournaments

A random graph on 15 vertices and its dominating set



3-colored Transitive Tournaments

A random graph on 15 vertices and its dominating set



Counting Arguments

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

Counting Arguments

- 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.

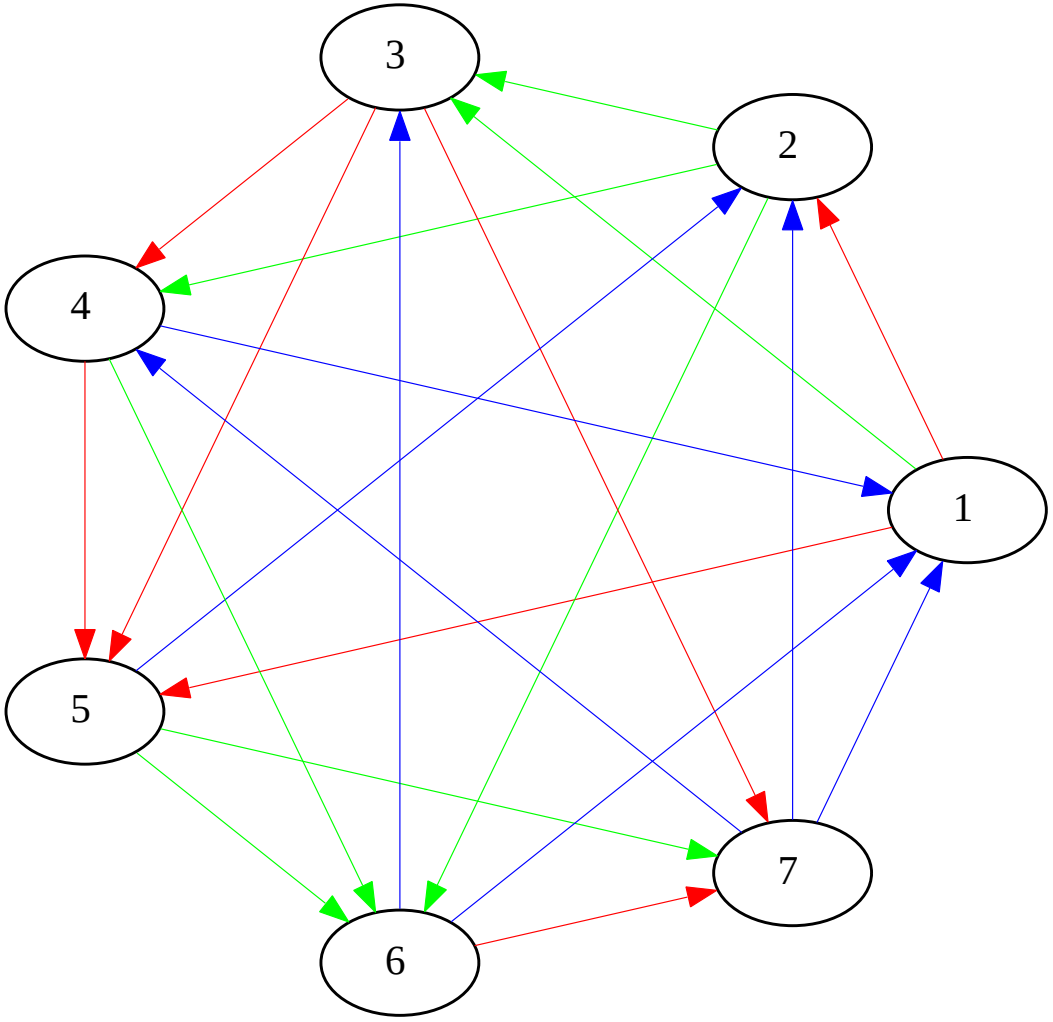
$$N = 2^D - 1^1$$

D	N
1	1
2	3
3	7
4	15
5	31

[1] http://oeis.org/A000225

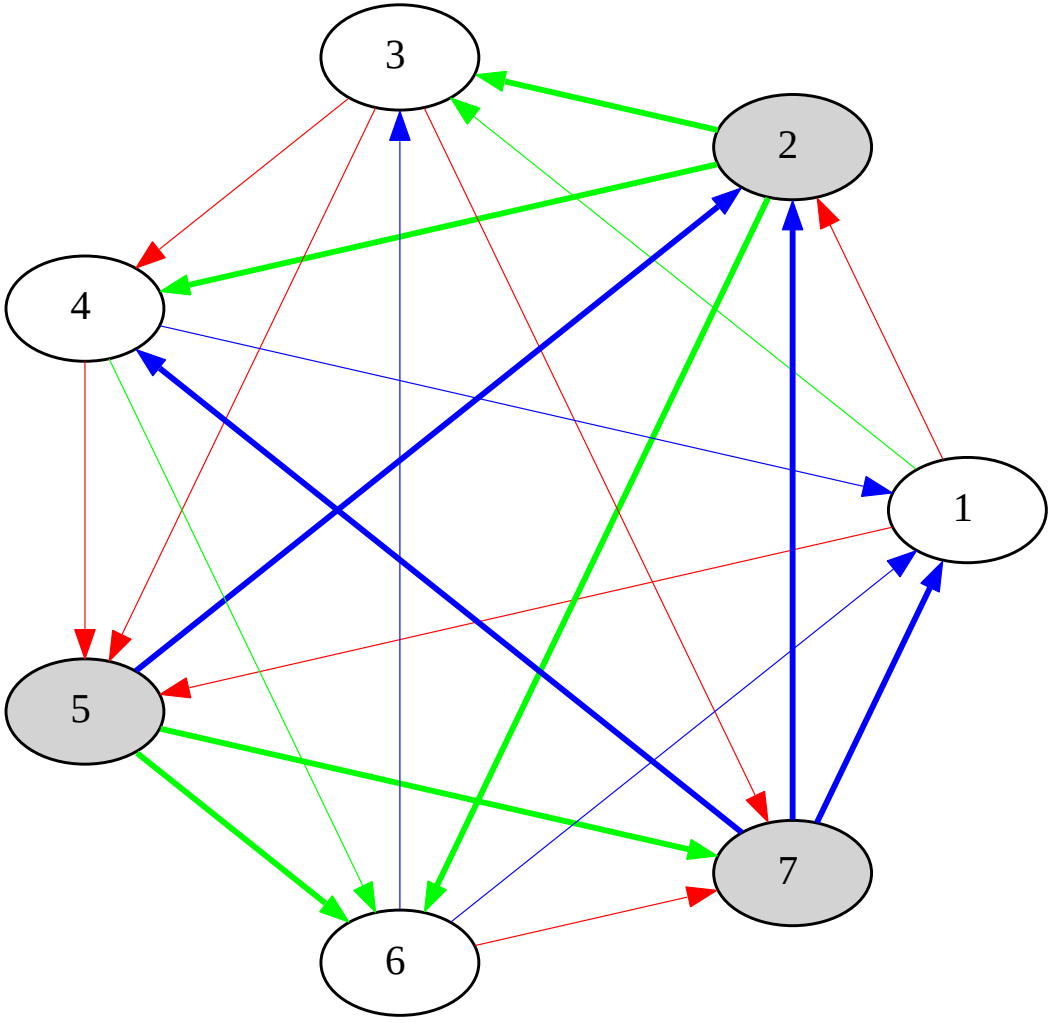
Substitution

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



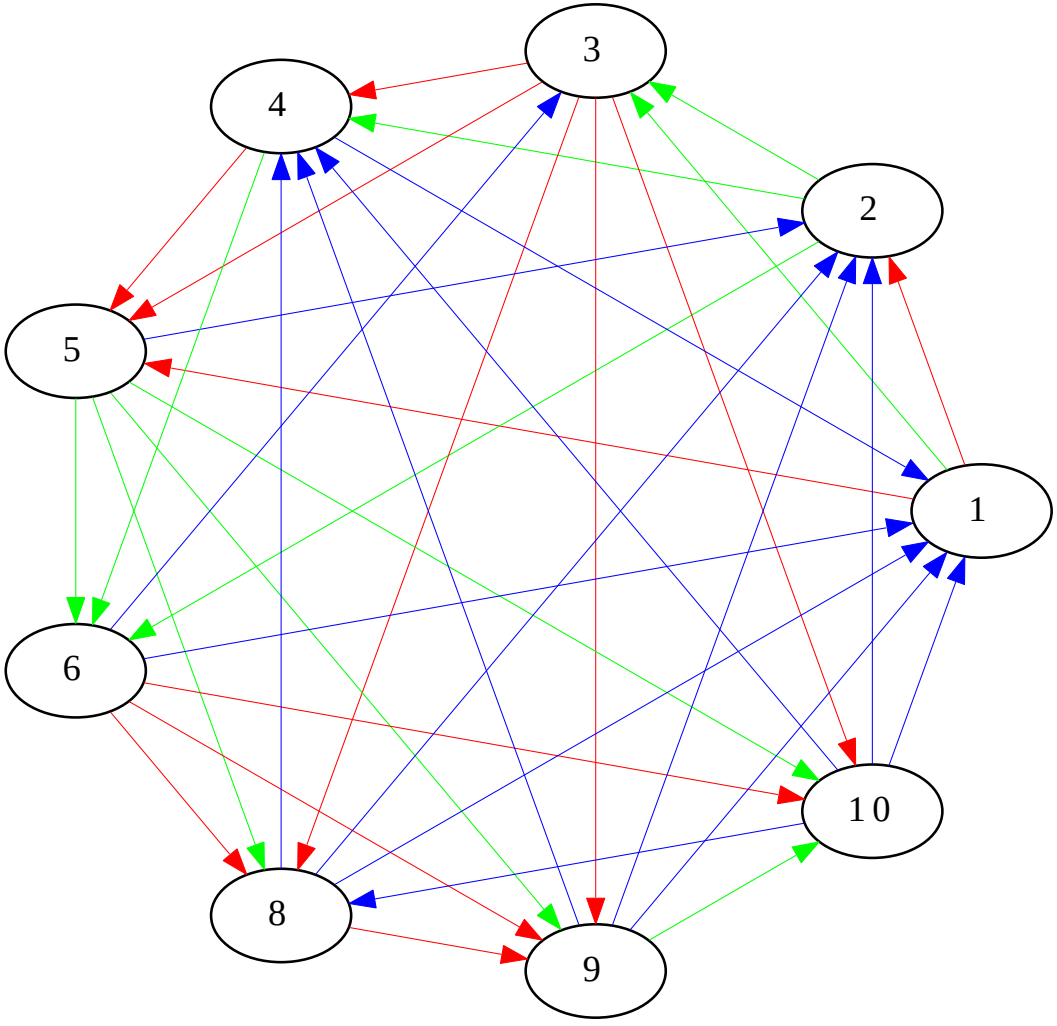
Substitution

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



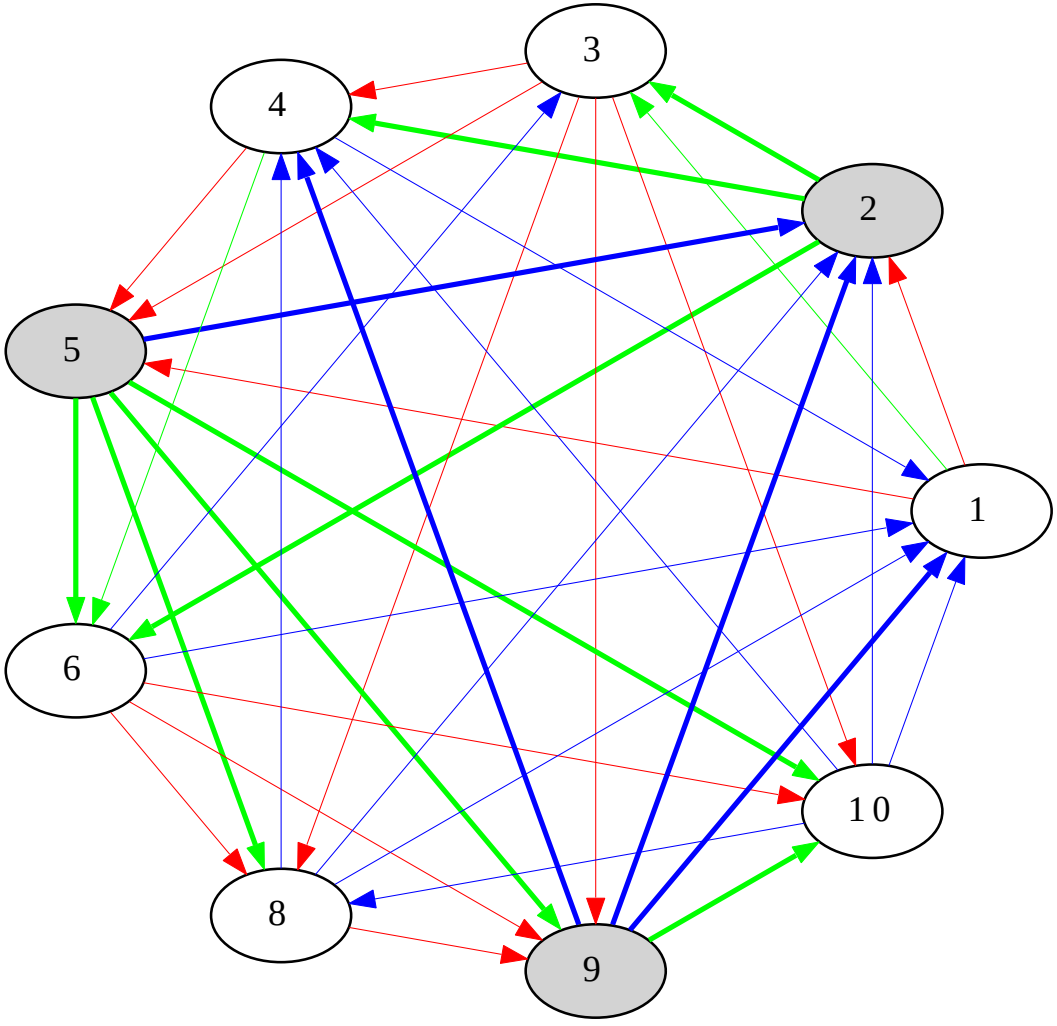
Substitution

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



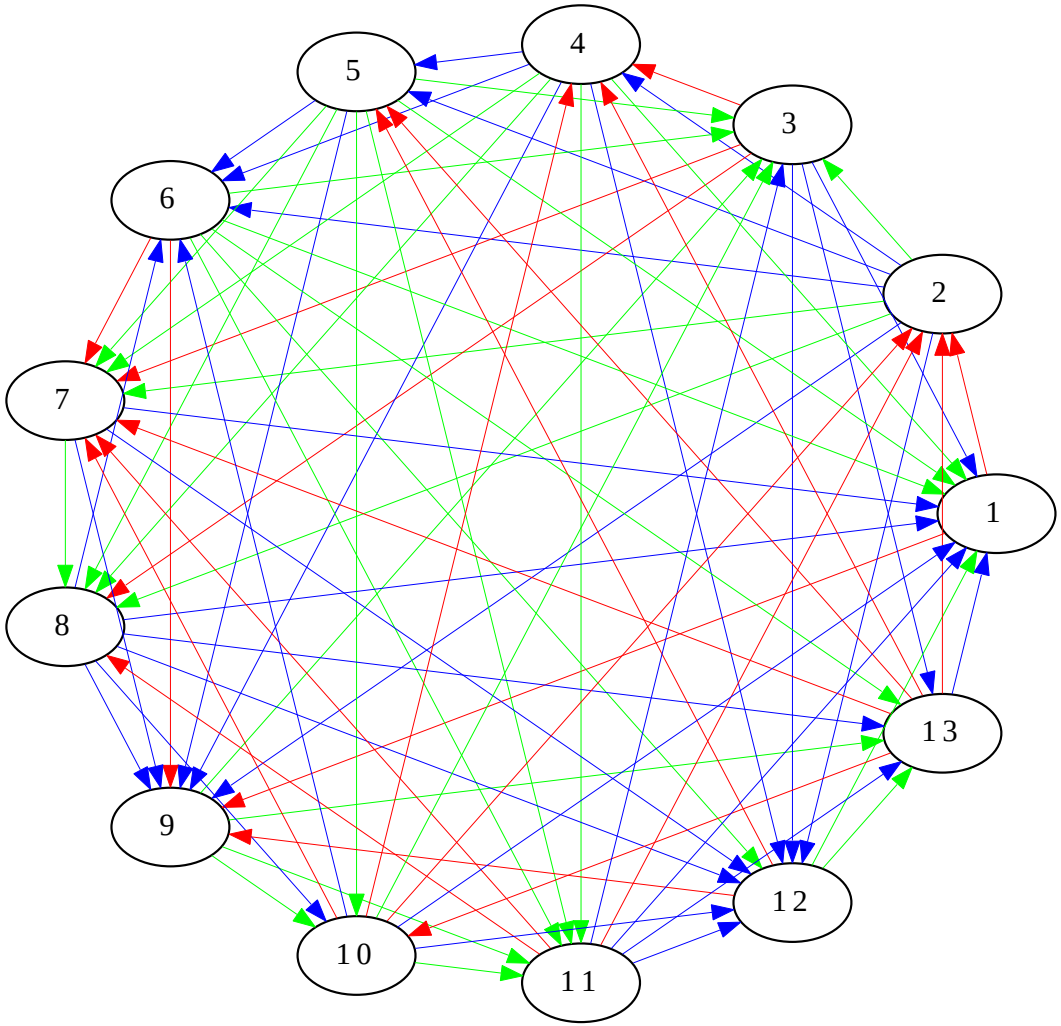
Substitution

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



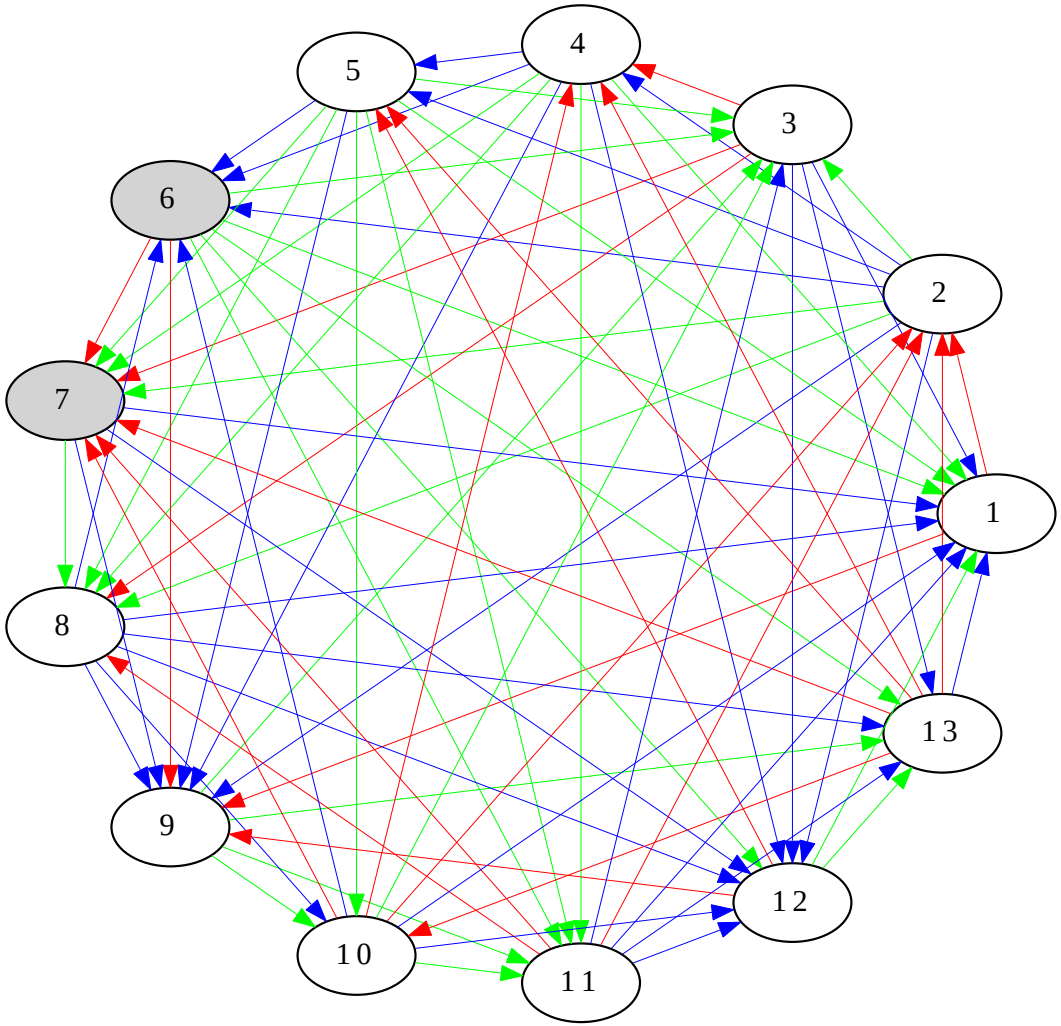
Reduction

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



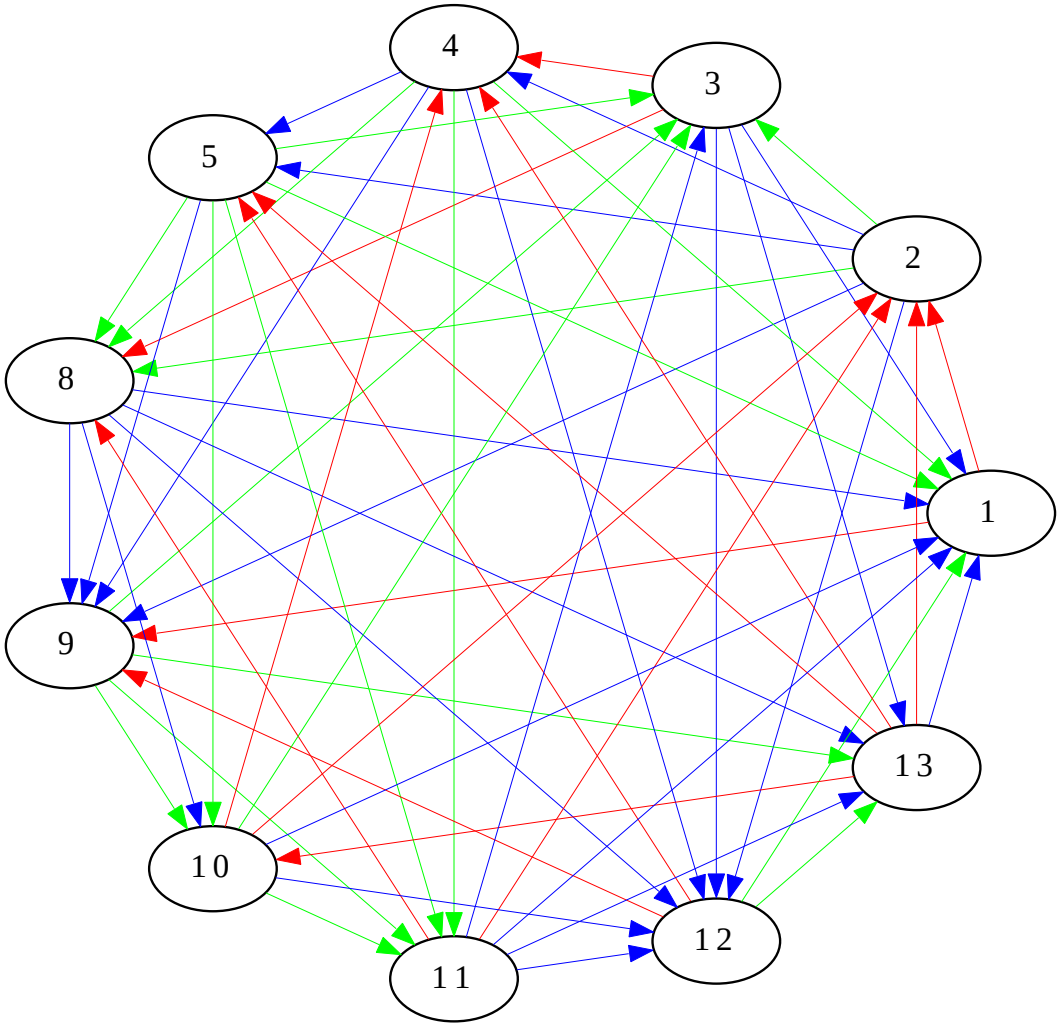
Reduction

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.



Strongly Connected Components

- 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 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 compontent. Only strongly connected tournaments are interesting for our study.

Strongly Connected Components

- 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 compontent. Only strongly connected tournaments are interesting for our study.
- A strongly connected tournament has a Hamiltonian cycle.

Inductive Construction

- Every tournament can be constructed by introduction of a new vertices and orientation/coloring of edges to all other vertices.

Inductive Construction

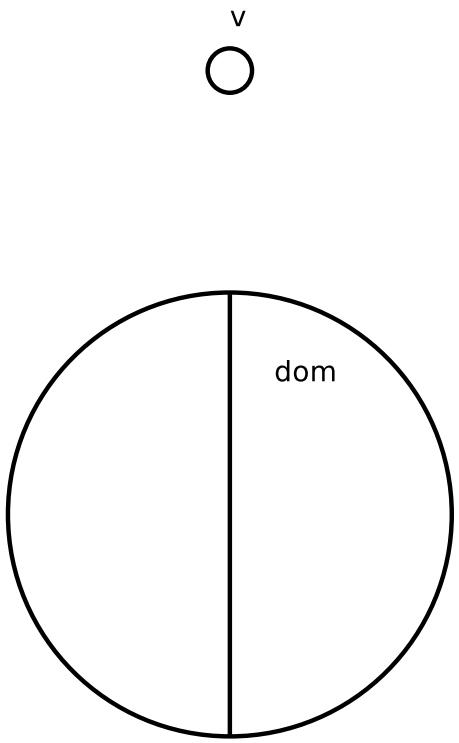
- Every tournament can be constructed by introduction of a new vertices and orientation/coloring of edges to all other vertices.
- Addition of the new vertex can:
  - decrease number of strongly connected components arbitrarily and increase by one,
  - decrease size of minimum dominating set arbitrarily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

Inductive Construction

- 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,
- It is sufficient to study edges incident to the first strongly connected component. There must be edges both incomming and outgoing.

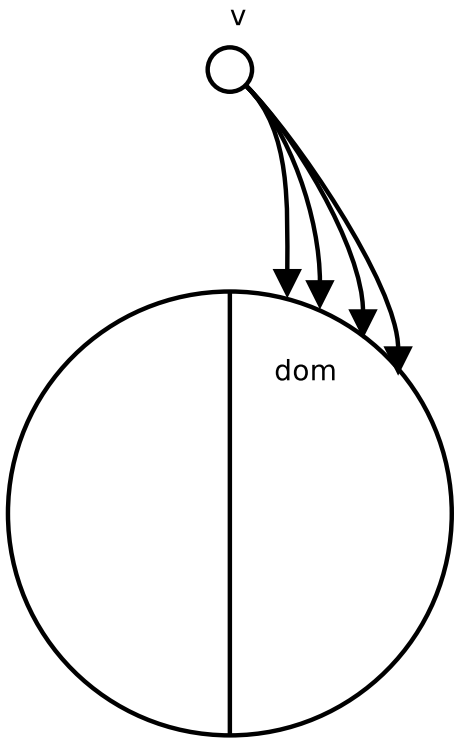
Inductive Construction

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.



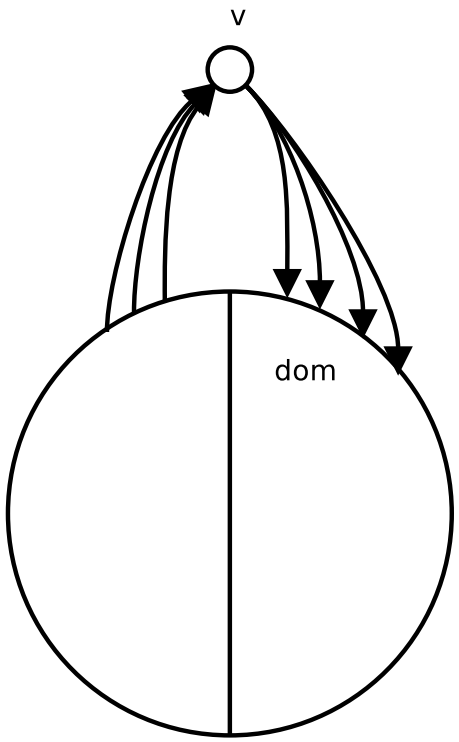
Inductive Construction

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.



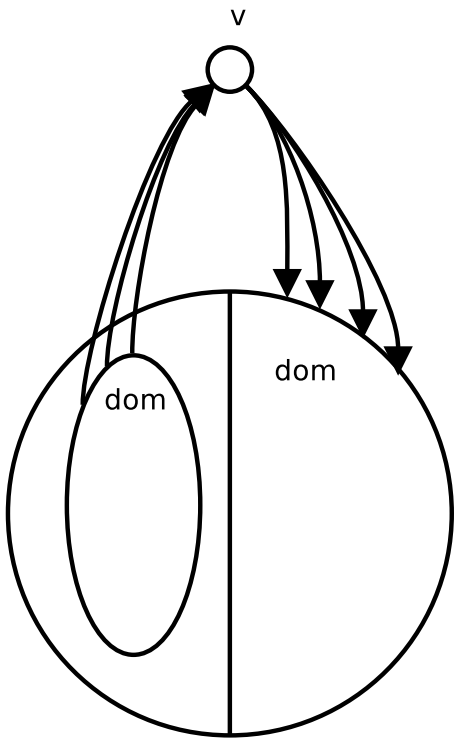
Inductive Construction

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.

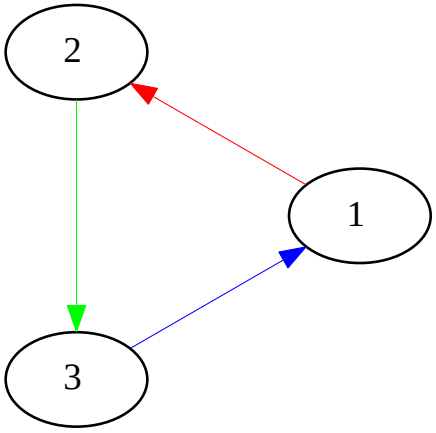


Inductive Construction

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.



2-Majority Graphs



$1 > 2 > 3$

$2 > 3 > 1$

$3 > 1 > 2$

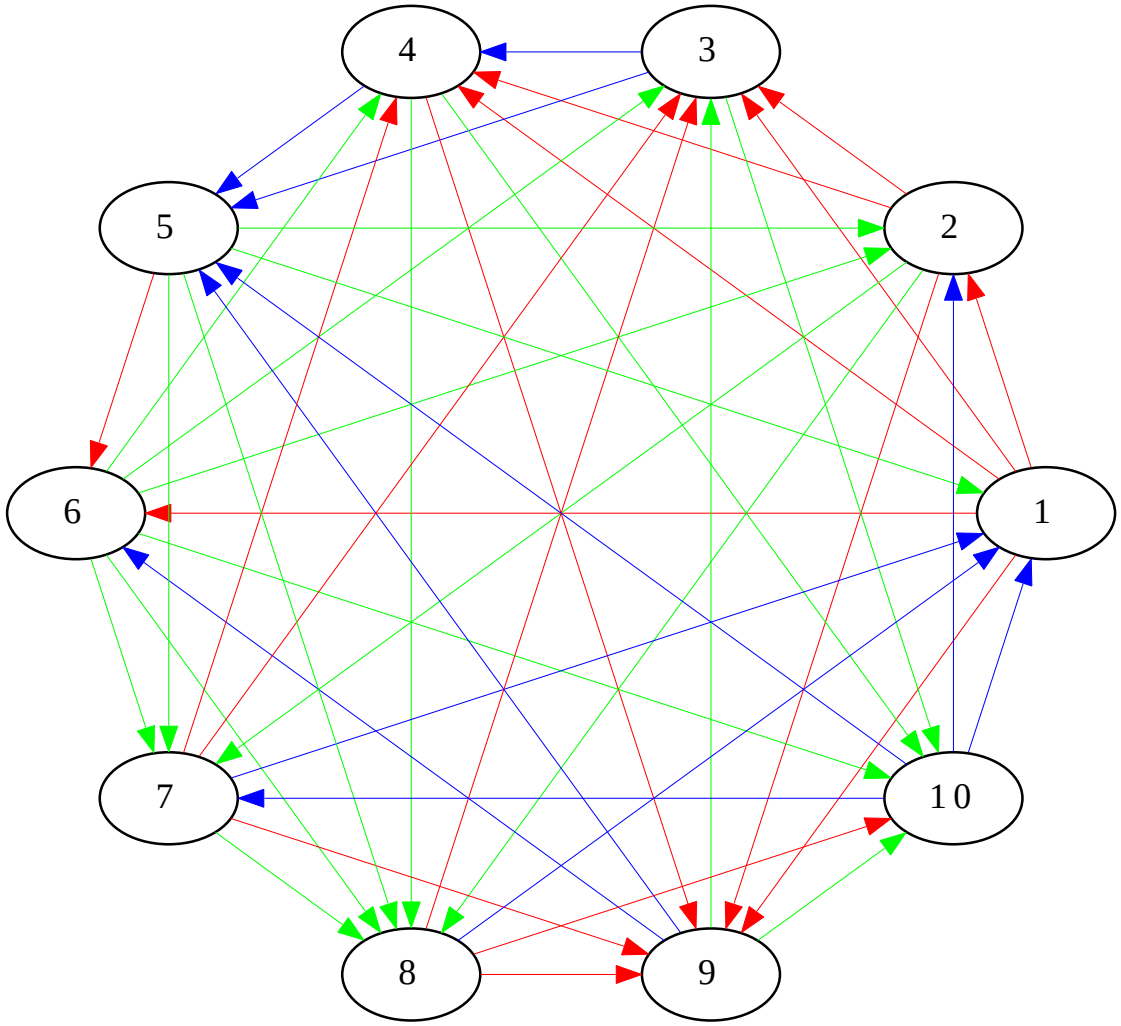
- A 2-majority graph is a graph induced by three linear orders voting for edge direction.
- 2-majority graphs have at most three dominators. If they do not have one dominator, then they have a dominating set of size three on a cyclic triangle.

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).

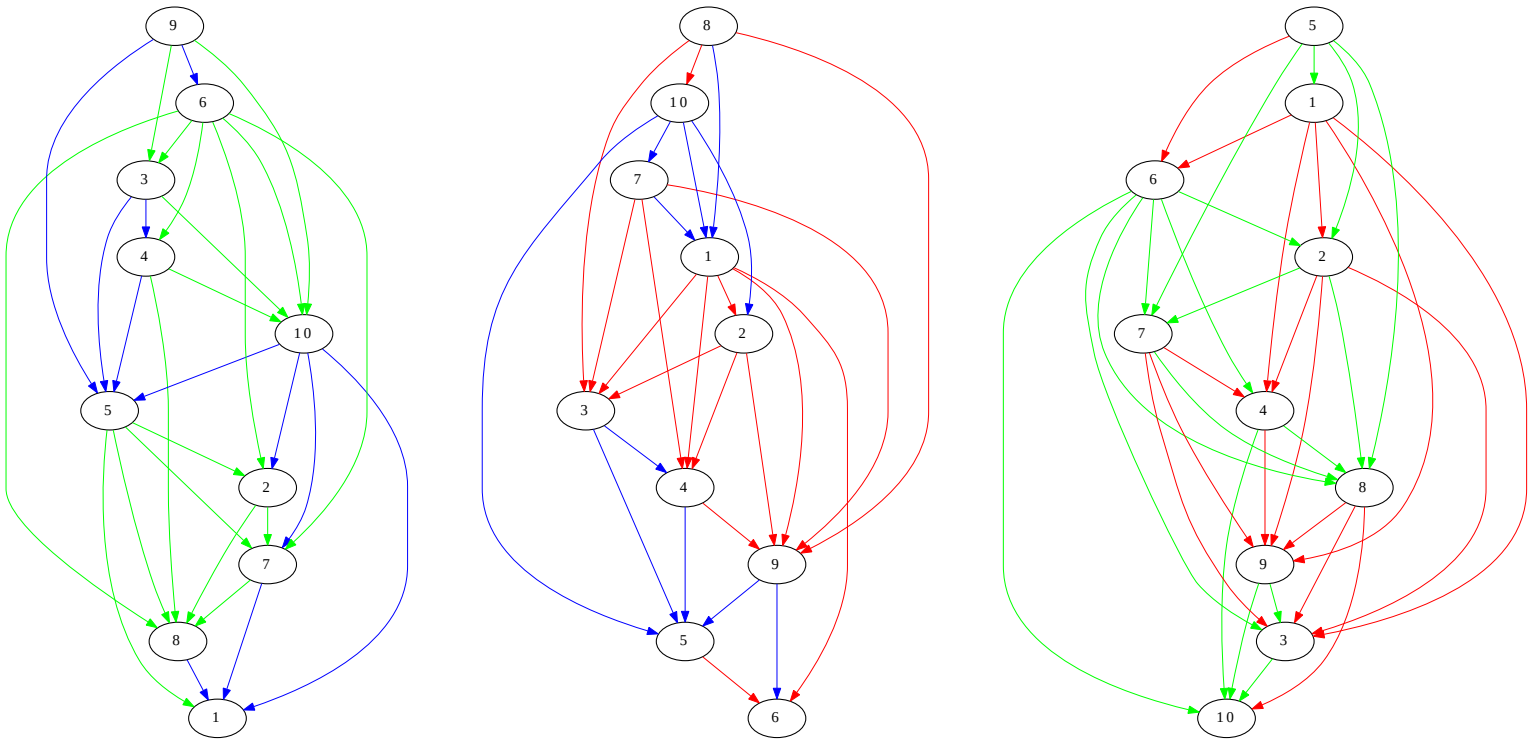
2-Majority Graphs

K_{10} which is acyclic in all three induced subgraphs.



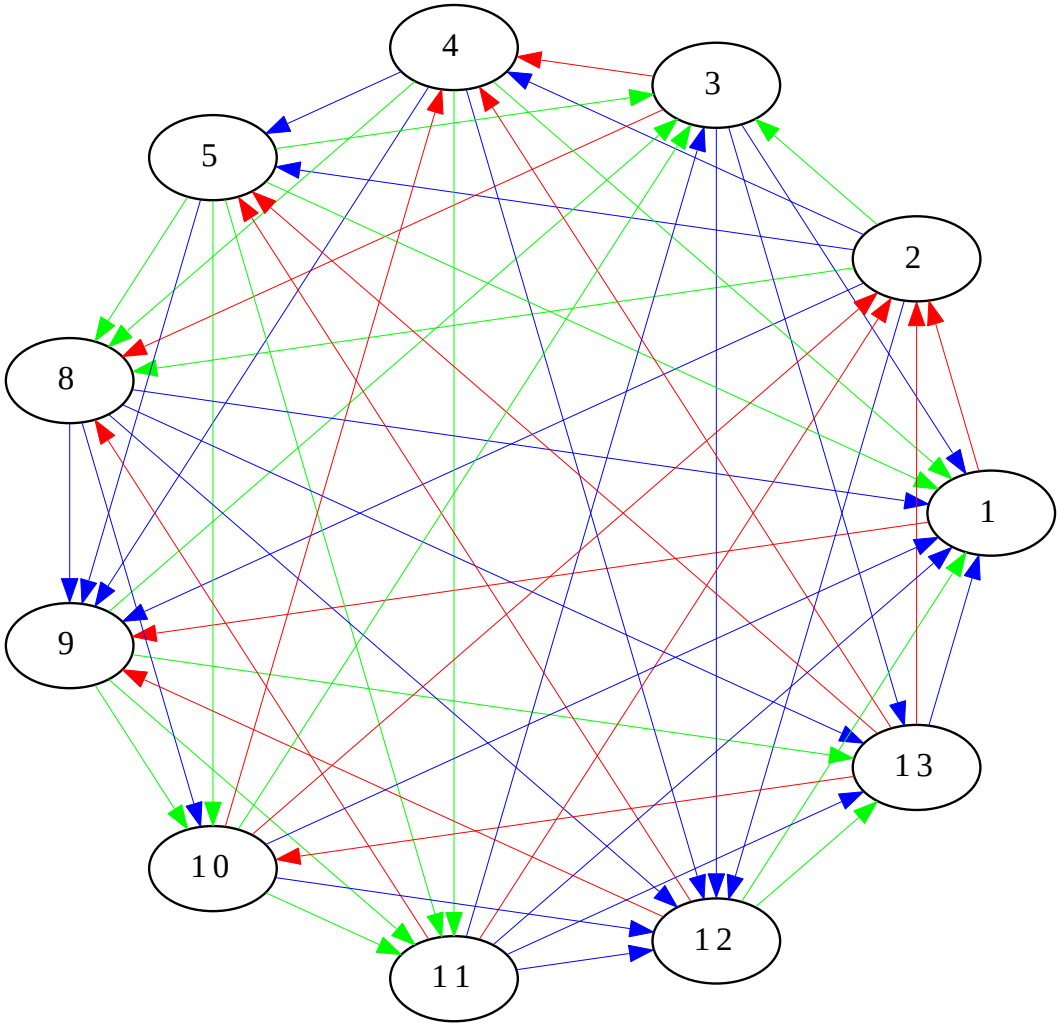
2-Majority Graphs

K_{10} which is acyclic in all three induced subgraphs.



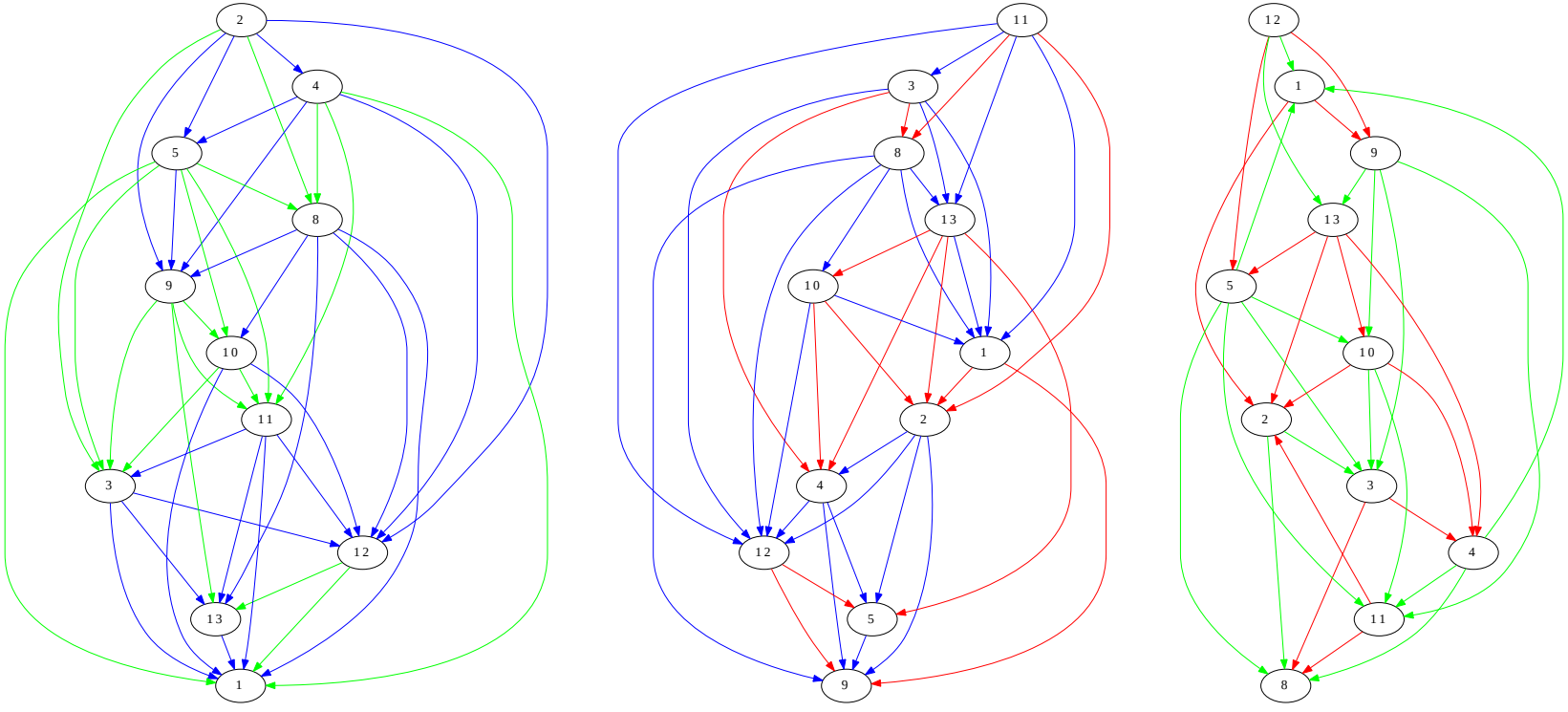
2-Majority Graphs

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.



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.

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 contains 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

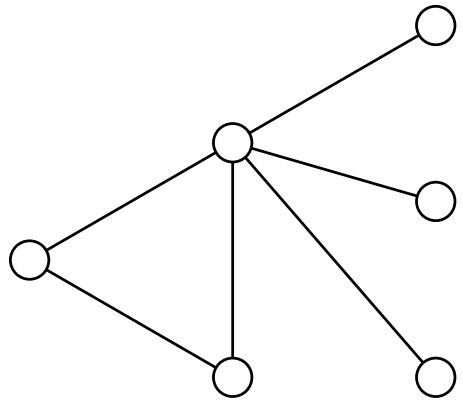
Graph Deletion Problems

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.

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.

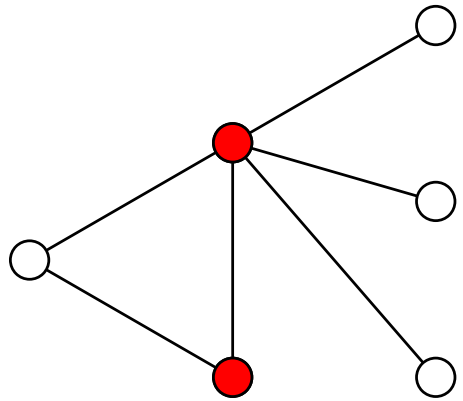
- Vertex Cover - Find a set $W \subseteq V$ such that $G[V \setminus W]$ contains no edge



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.

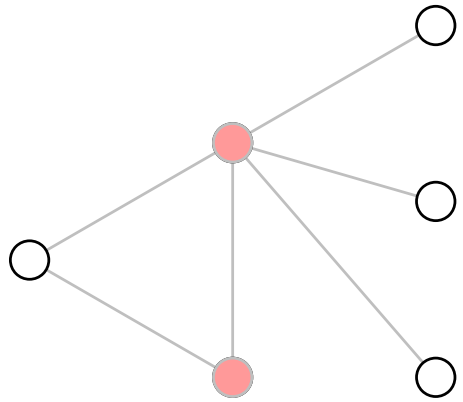
- Vertex Cover - Find a set $W \subseteq V$ such that $G[V \setminus W]$ contains no edge



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



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

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

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

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.

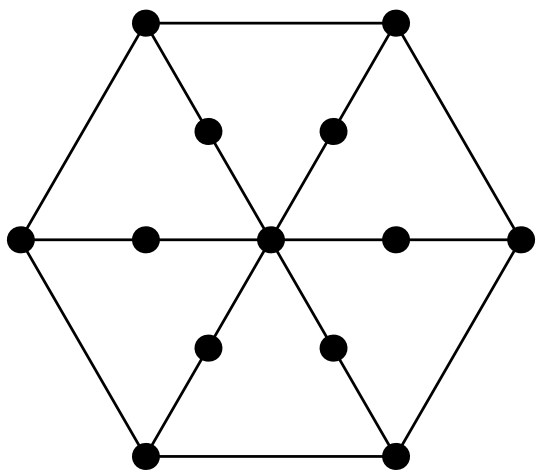
- Vertex Cover - Find a set $W \subseteq V$ such that $G[V \setminus W]$ contains no edge
- Feedback Vertex Set - Find a set $W \subseteq V$ such that $G[V \setminus W]$ is acyclic
- Feedback Arc Set - Find a set $F \subseteq E$ such that $(V, E \setminus F)$ is acyclic
- Odd Cycle Transversal - Find a set $F \subseteq E$ such that $G \setminus F$ is bipartite

The goal is to find the smallest such set.

Fair Versions of Deletion Problems

Sometimes the smallest set is not what we want.

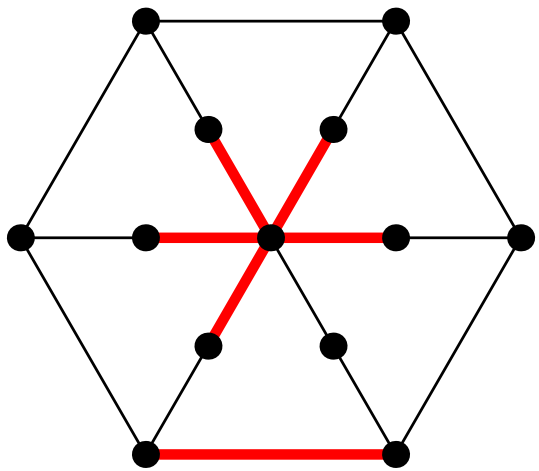
- We have a communication network, we want to remove redundancies
- We do not want to reduce the capability of each vertex by too much (a local condition in contrast to the previous global one)



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)

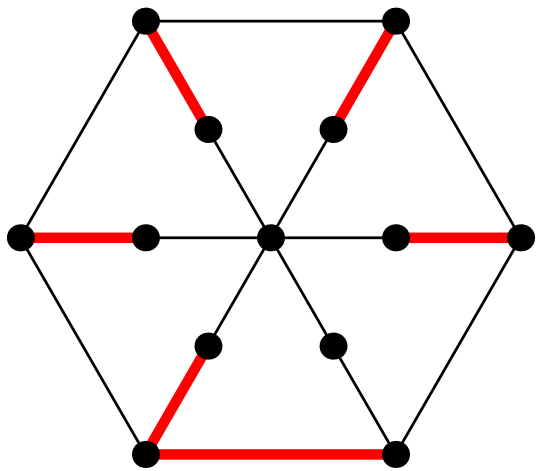


This solution removes a lot of links from single vertex.

Fair Versions of Deletion Problems

Sometimes the smallest set is not what we want.

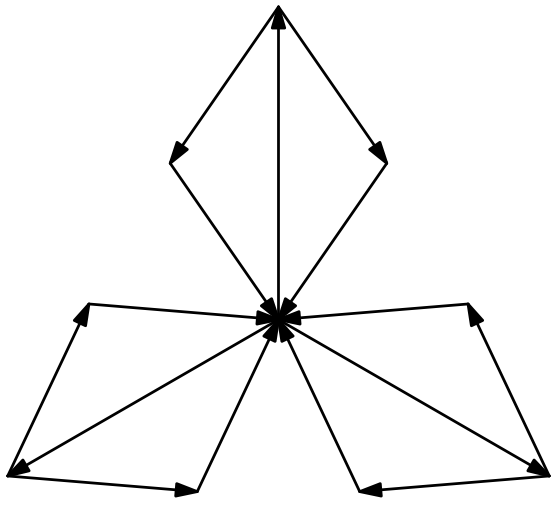
- We have a communication network, we want to remove redundancies
- We do not want to reduce the capability of each vertex by too much (a local condition in contrast to the previous global one)



A much better solution.

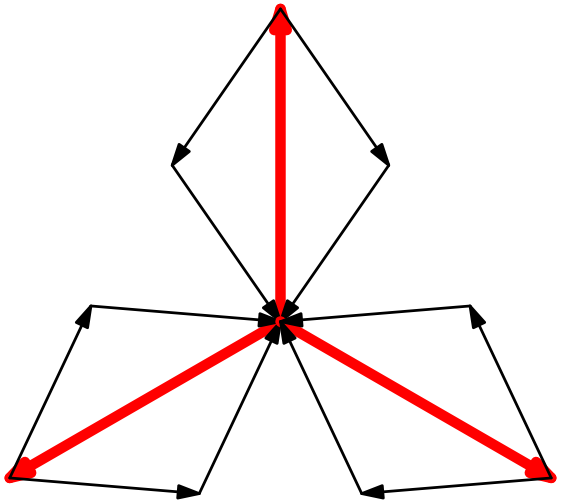
Fair Versions of Deletion Problems

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



Fair Versions of Deletion Problems

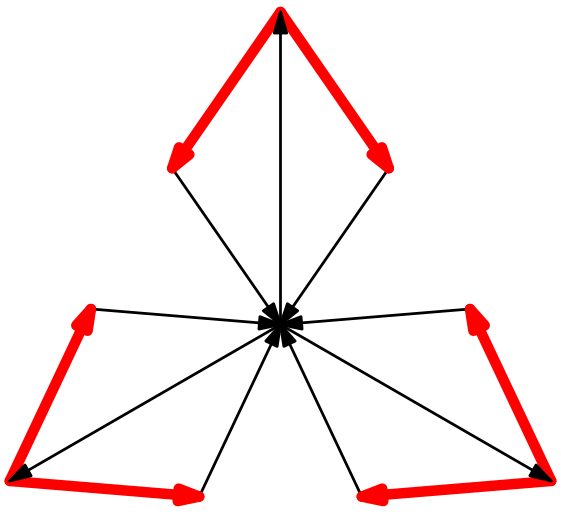
Optimal fair solution might contain much more edges in total than optimal classical solution.



Optimal solution for the classical case

Fair Versions of Deletion Problems

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



Optimal solution for the fair case

Complexity of Graph Deletion Problems

- Classical versions of those problems are NP-complete.

Complexity of Graph Deletion Problems

- Classical versions of those problems are NP-complete.
- In most cases, the fair versions are NP-complete too

Parameterized Complexity

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

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

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...)

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

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

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:

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)

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)

Parameterized Complexity – Classes

- The central class of parameterized complexity is FPT – Fixed parameter tractable

Parameterized Complexity – Classes

- 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 $2^k n^2$)

Parameterized Complexity – Classes

- 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

Parameterized Complexity – Classes

- 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},...)

Parameterized Complexity – Classes

- 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

Parameterized Complexity – Classes

- 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
- 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:

$$vertex\_cover(W) \equiv (\forall u, v \in V \setminus W)(\neg adj(u, v))$$

Our Results – Hardness

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] = 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] = 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

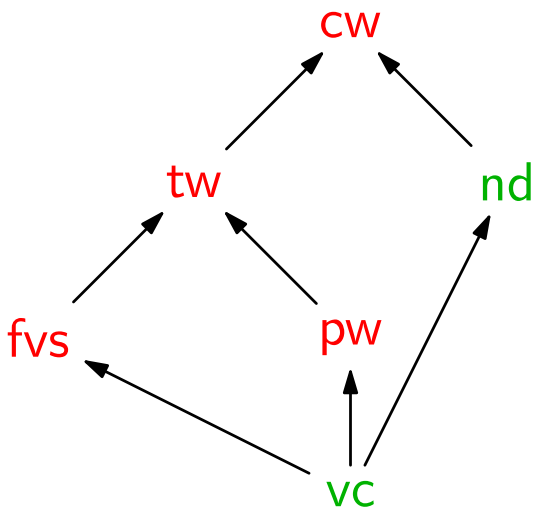
Our Results – Tractability

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:



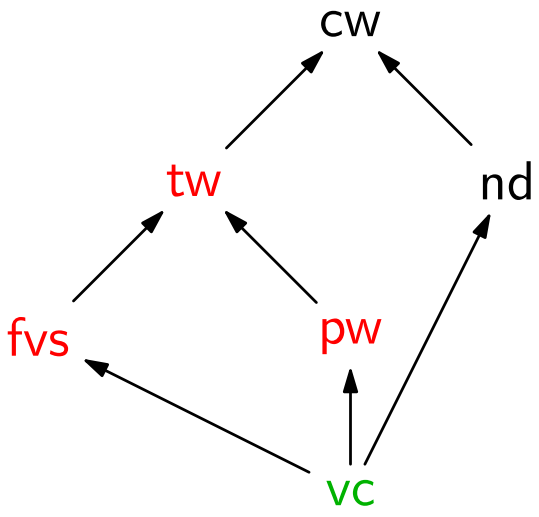
Parameters for which we have proven the metatheorem is hard (not FPT unless $W[1] = FPT$)

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)

Complexity of Specific Problems

- If we have parameter for which is the metatheorem hard, it is still of interest to study specific problems.

Complexity of Specific Problems

- 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

Complexity of Specific Problems

- 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

Open Questions and Future Work

- Can be the bound $f(k)n^{o(\sqrt{k})}$ be improved to $f(k)n^{o(k)}$? (which is essentially tight)

Open Questions and Future Work

- 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 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?

Open Questions and Future Work

- 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?

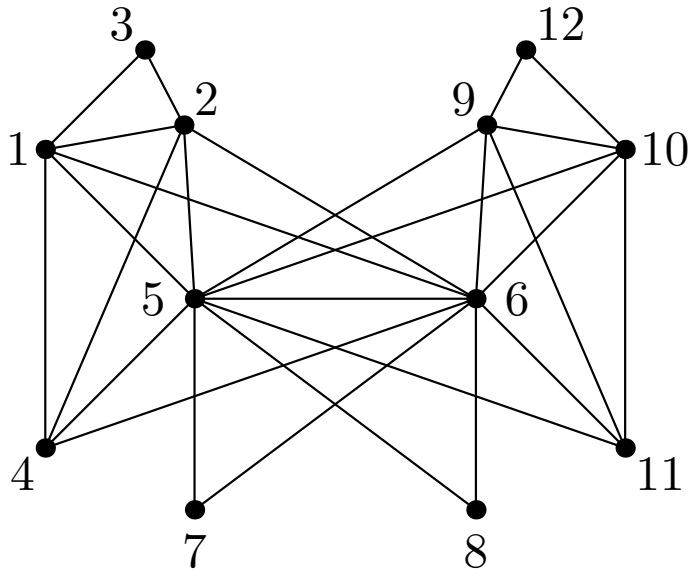
Questions?

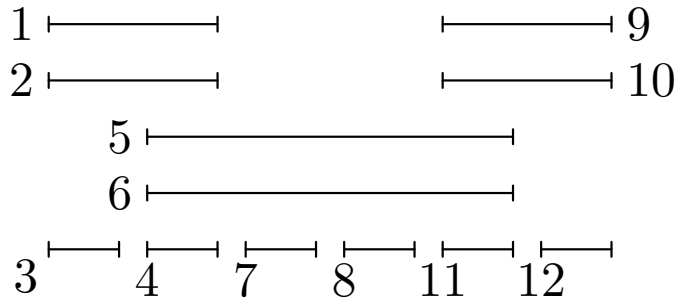
Subclasses of Chordal Graphs

Martin Töpfer, Jan Voborník, Peter Zeman

Interval Graphs (INT)

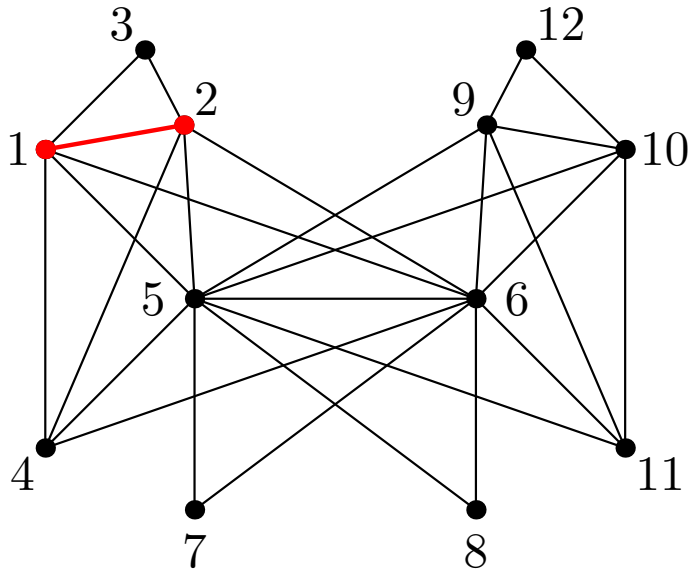
Interval graphs are intersection graphs of intervals of the real line.

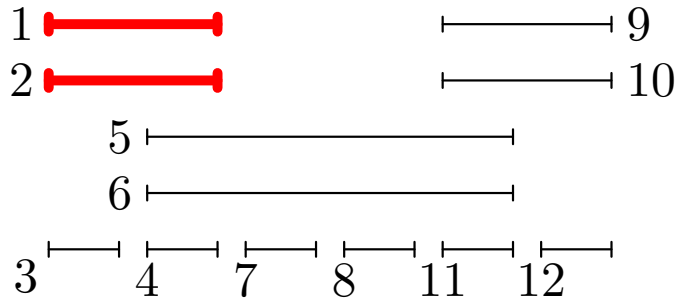




Interval Graphs (INT)

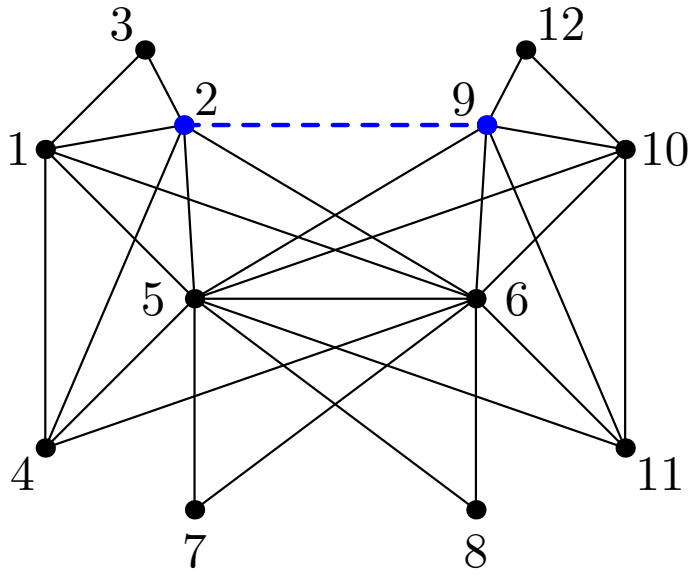
Interval graphs are intersection graphs of intervals of the real line.

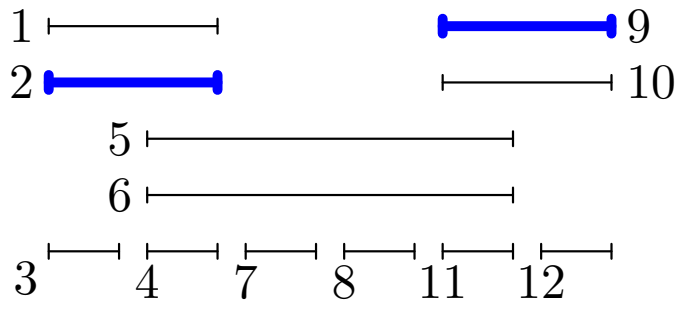




Interval Graphs (INT)

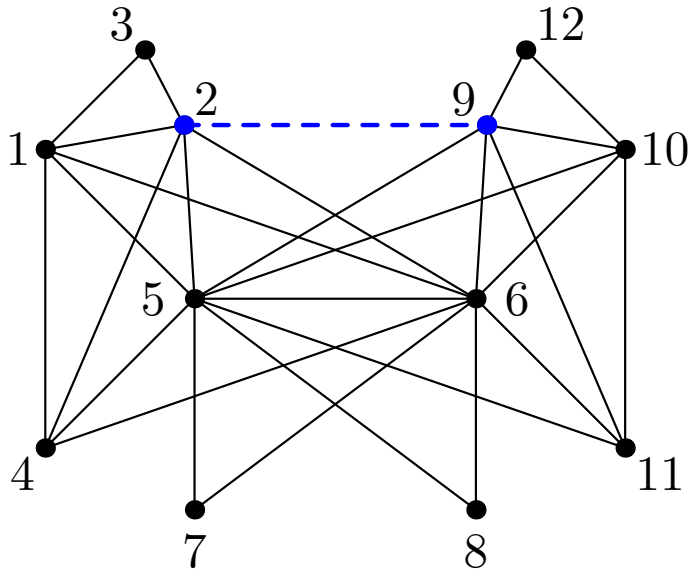
Interval graphs are intersection graphs of intervals of the real line.

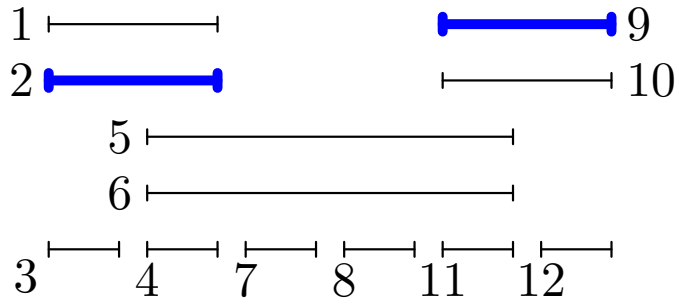




Interval Graphs (INT)

Interval graphs are intersection graphs of intervals of the real line.

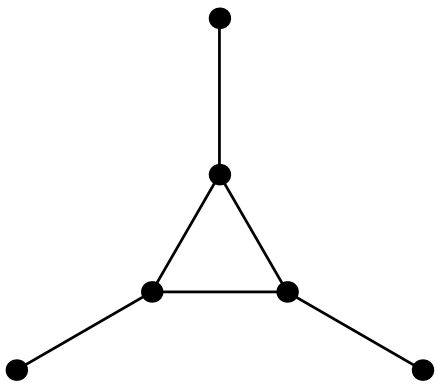




Many computational problems are polynomially solvable for interval graphs (graph isomorphism, maximum clique, k-coloring, maximum independent set, ...).

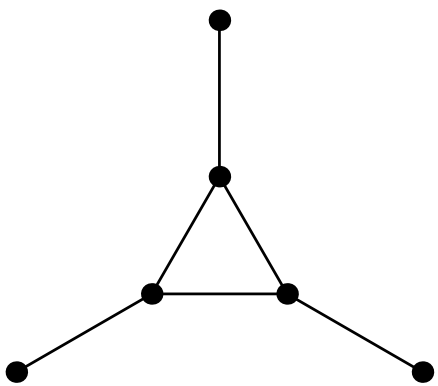
Chordal Graphs (CHOR)

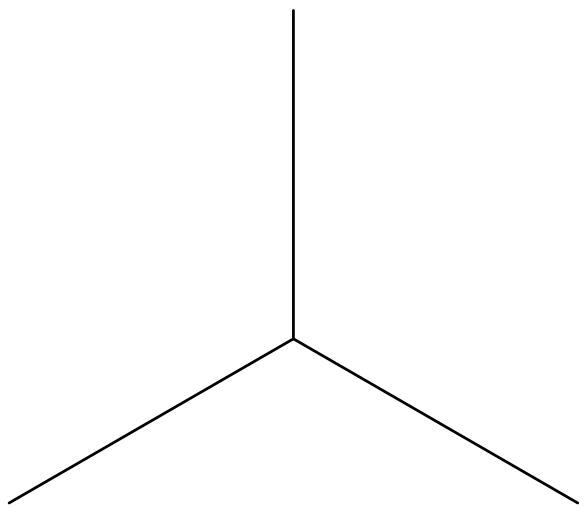
A graph is a chordal graph if and only if there exists a tree T such that it can be represented as an intersection graph of subtrees of the tree T.



Chordal Graphs (CHOR)

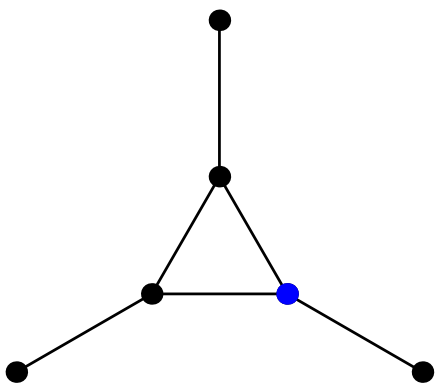
A graph is a chordal graph if and only if there exists a tree T such that it can be represented as an intersection graph of subtrees of the tree T.

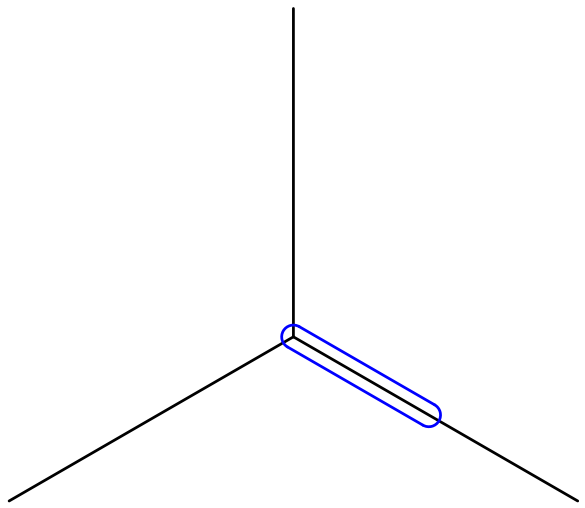




Chordal Graphs (CHOR)

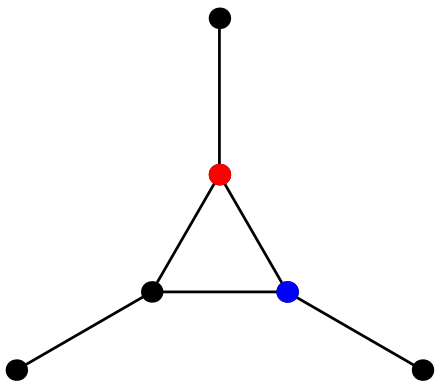
A graph is a chordal graph if and only if there exists a tree T such that it can be represented as an intersection graph of subtrees of the tree T.

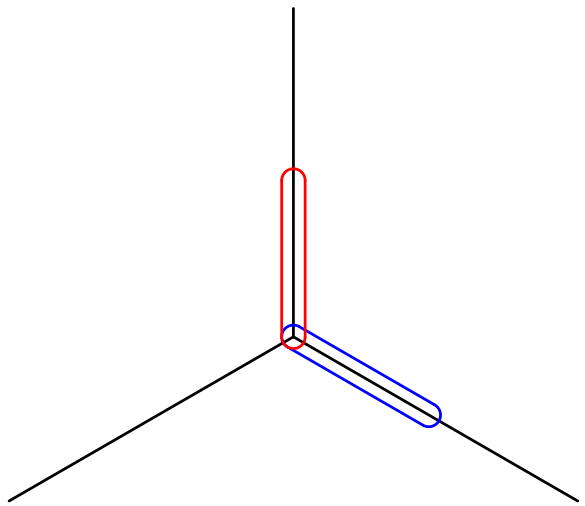




Chordal Graphs (CHOR)

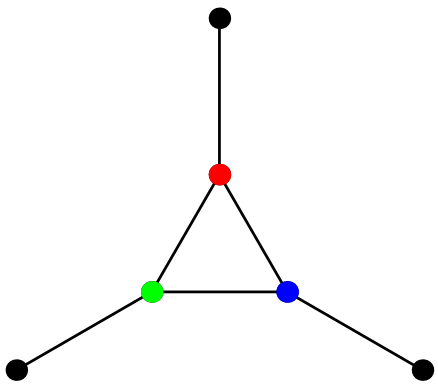
A graph is a chordal graph if and only if there exists a tree T such that it can be represented as an intersection graph of subtrees of the tree T.

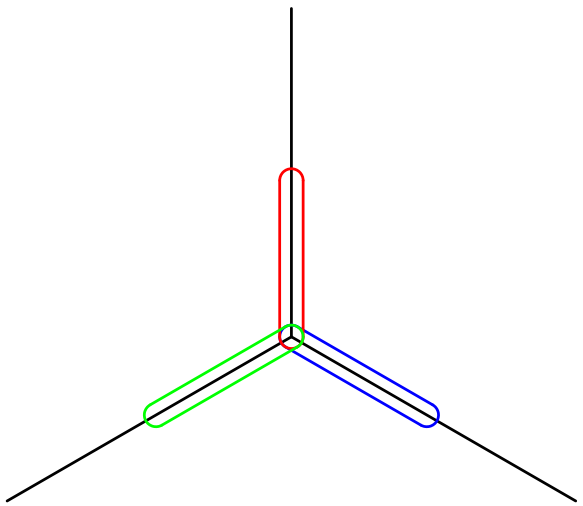




Chordal Graphs (CHOR)

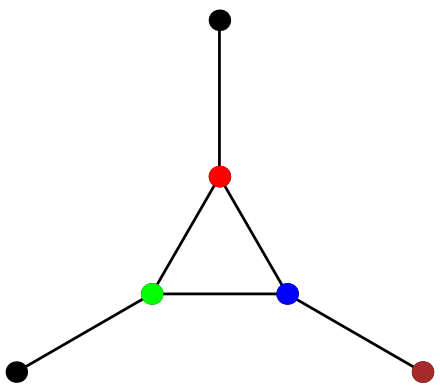
A graph is a chordal graph if and only if there exists a tree T such that it can be represented as an intersection graph of subtrees of the tree T.

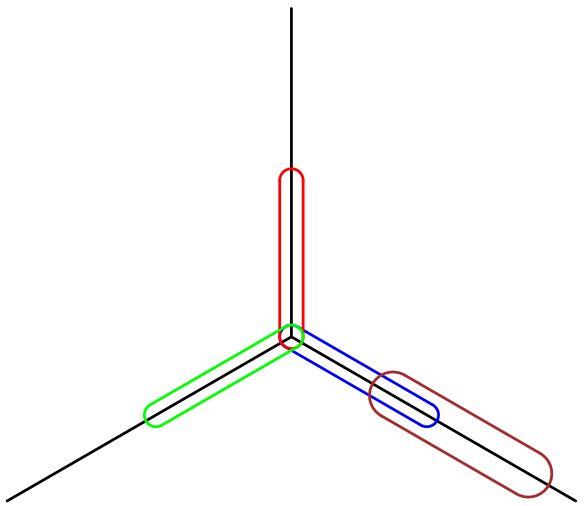




Chordal Graphs (CHOR)

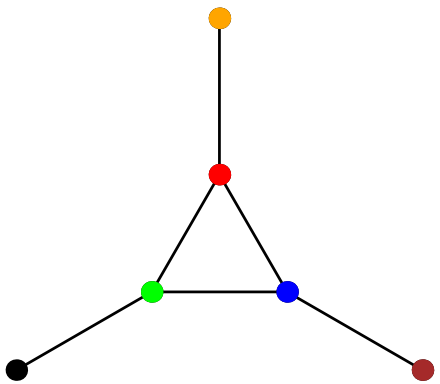
A graph is a chordal graph if and only if there exists a tree T such that it can be represented as an intersection graph of subtrees of the tree T.

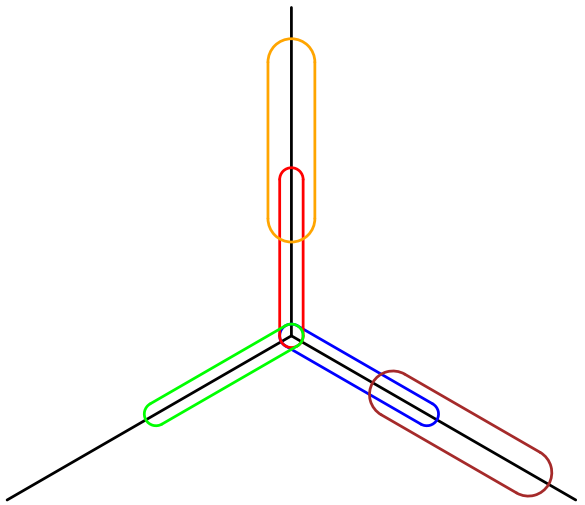




Chordal Graphs (CHOR)

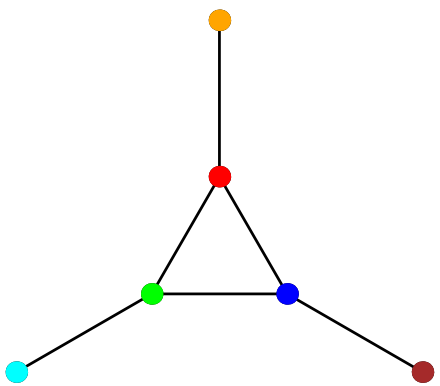
A graph is a chordal graph if and only if there exists a tree T such that it can be represented as an intersection graph of subtrees of the tree T.

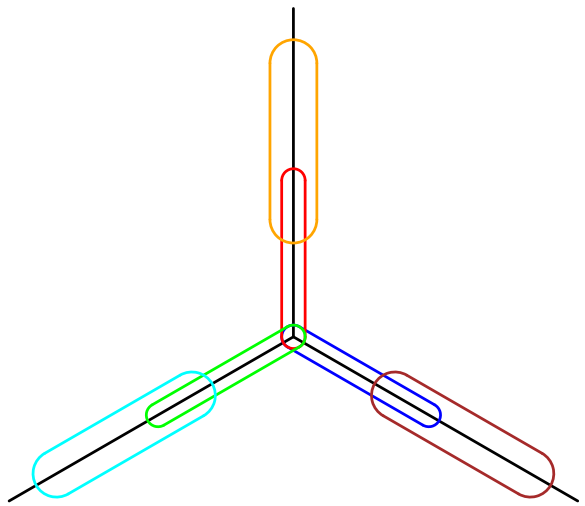




Chordal Graphs (CHOR)

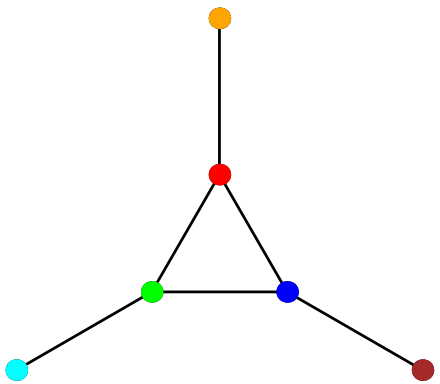
A graph is a chordal graph if and only if there exists a tree T such that it can be represented as an intersection graph of subtrees of the tree T.

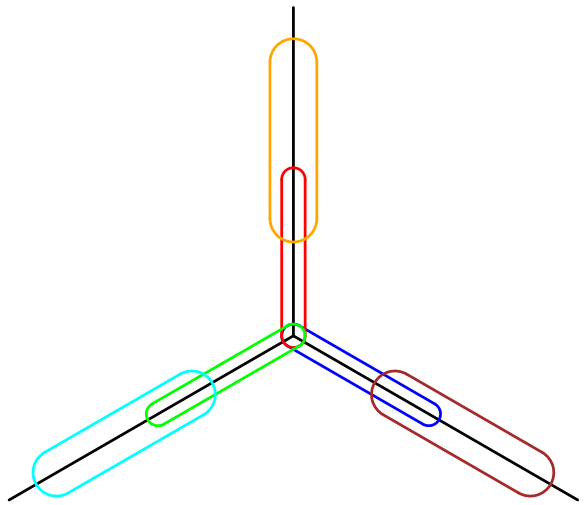




Chordal Graphs (CHOR)

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 INT $\subsetneq$ 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.

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

$$\text{INT} \subseteq T\text{-GRAPH} \subsetneq \text{CHOR}.$$

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

$$\text{INT} \subseteq T\text{-GRAPH} \subsetneq \text{CHOR}.$$

Our main aim was to study structural and computational properties of the classes T-GRAPH.

Recognition and Representation Extension Problems

Problem: Recognition – $\textsc{Recog}(\mathcal{C})$
Input: A graph G.
Question: Does G belong to the class $\mathcal{C}$?

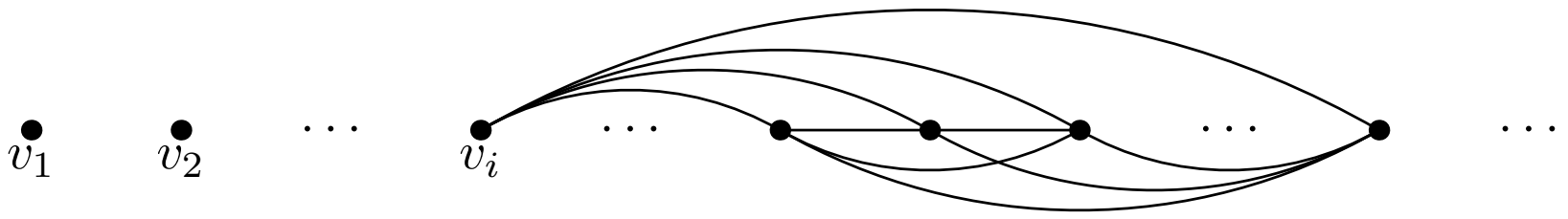
Problem: Perfect Representation Extension – $\textsc{RepExt}(\mathcal{C})$
Input: A graph G and a partial representation $\mathcal{R}'$.
Question: Is there a representation $\mathcal{R}$ of G extending $\mathcal{R}'$?

Previous results: NP-hardness, $W[1]$-hardness.

Our result: An algorithm for $\textsc{RepExt}(T\text{-GRAPH})$ with running time $n^{\mathcal{O}(\|T\|)}$.

Perfect Elimination Scheme (PES)

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.



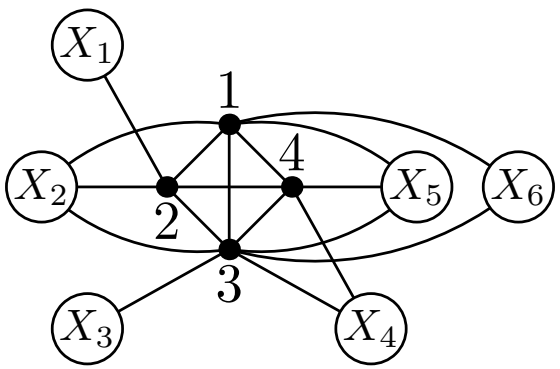
It follows that every chordal graph has at most n maximal cliques.

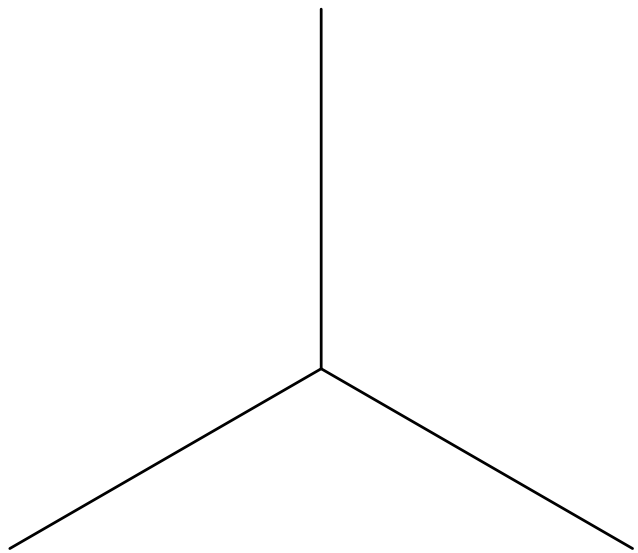
Recog(S_d-GRAPH)

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.



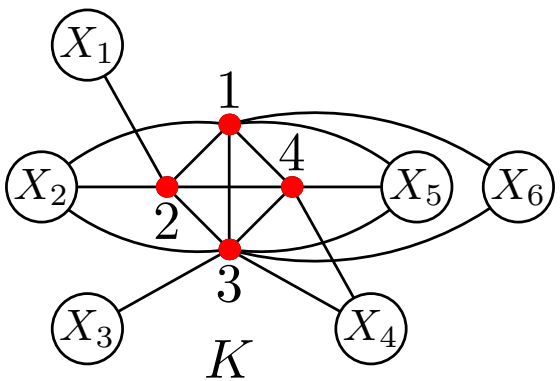


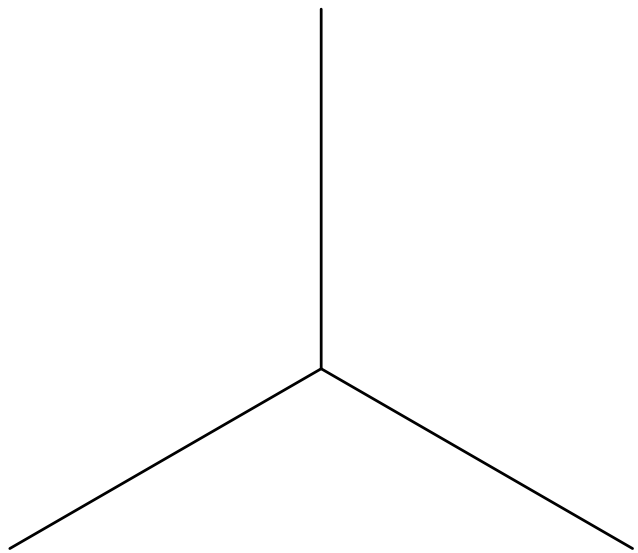
RECOG(S_d-GRAPH)

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.



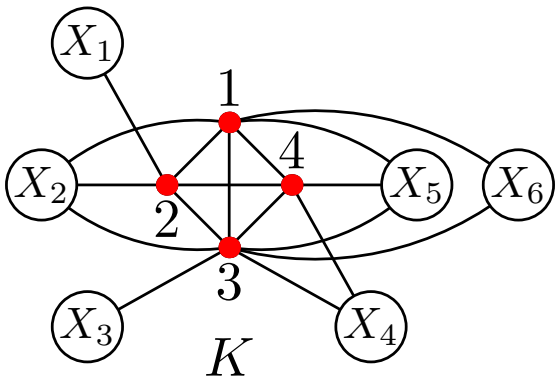


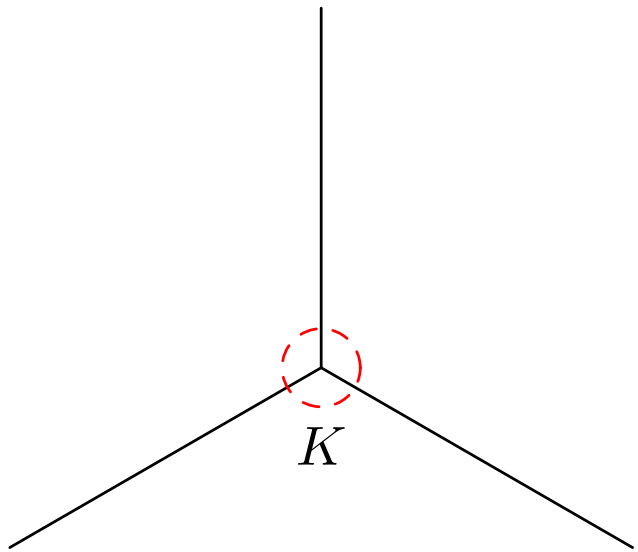
Recog(S_d-GRAPH)

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.



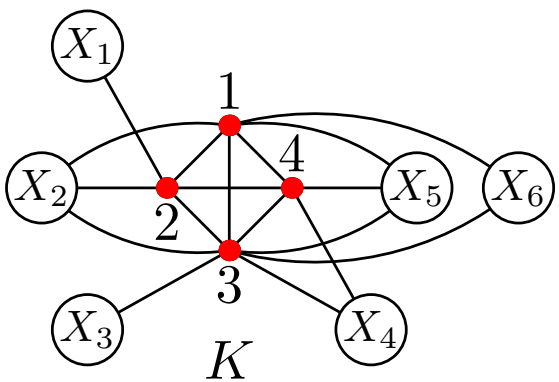


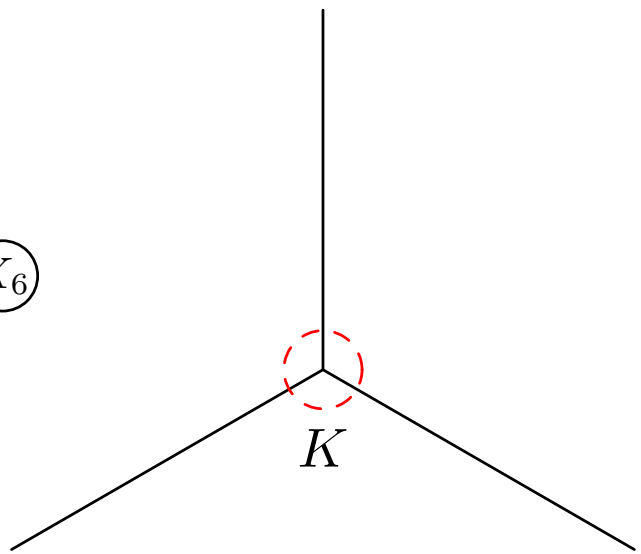
Finding a Correct Placement of Components

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 $V(C')$.



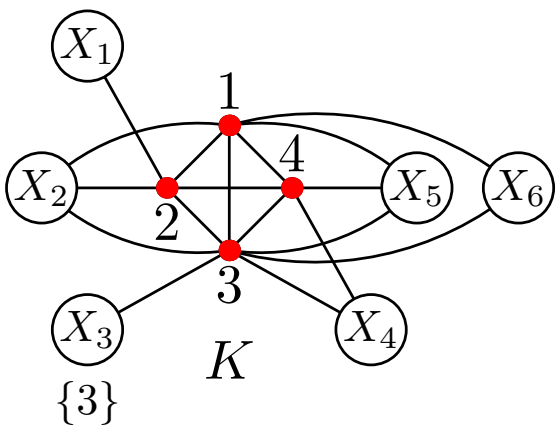


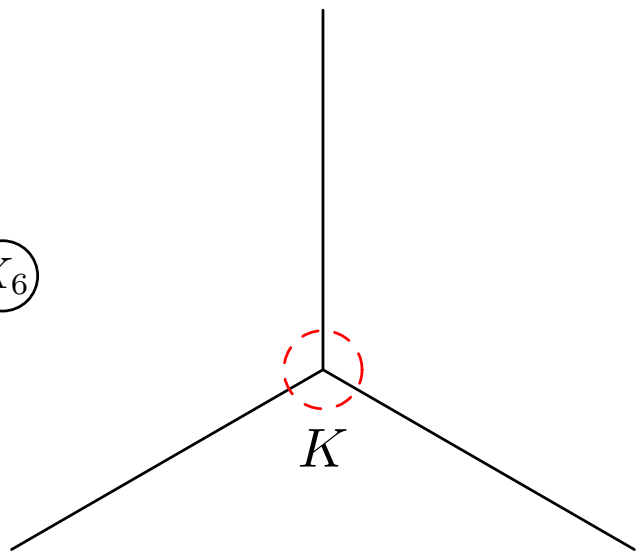
Finding a Correct Placement of Components

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 $V(C')$.



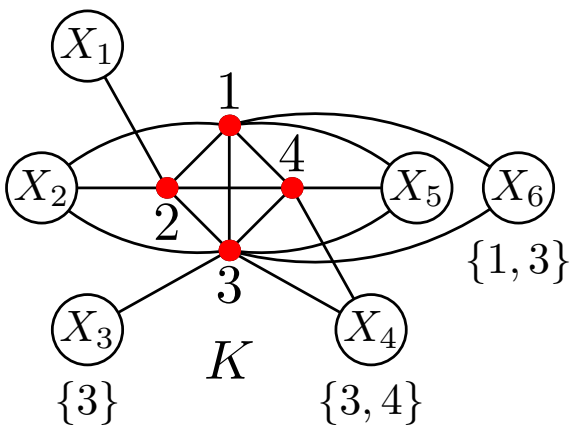


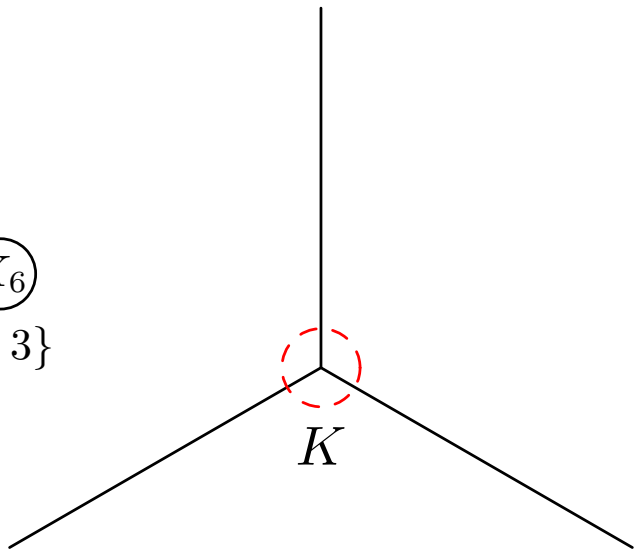
Finding a Correct Placement of Components

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 $V(C')$.



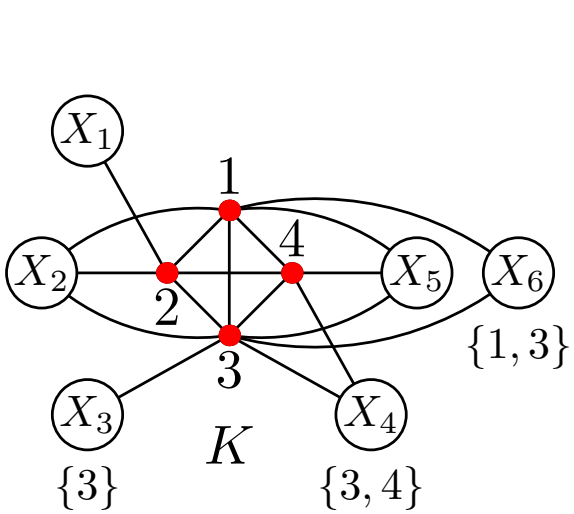


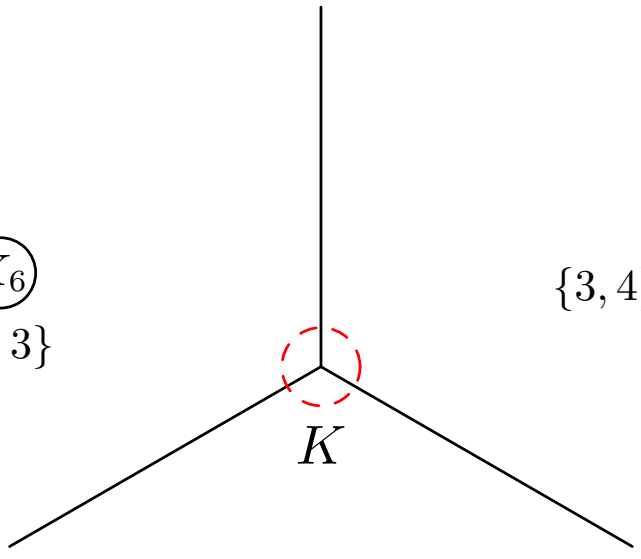
Finding a Correct Placement of Components

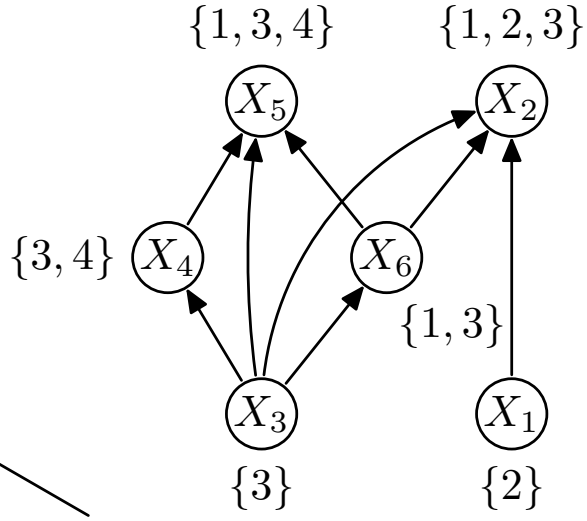
For components C and C', we define that $C \succeq C'$ if and only if

(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 $V(C')$.





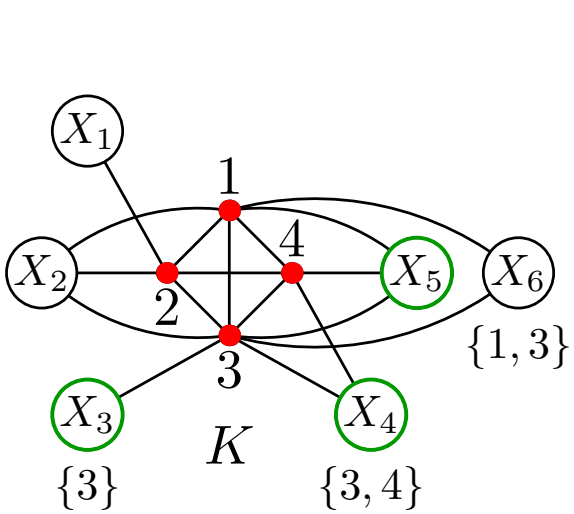


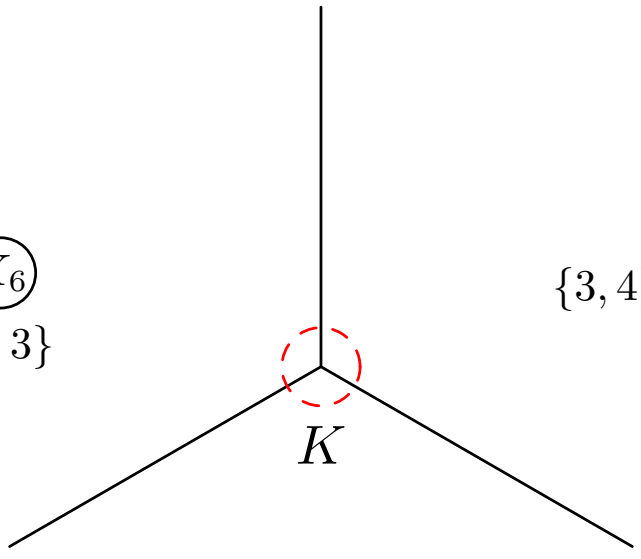
Finding a Correct Placement of Components

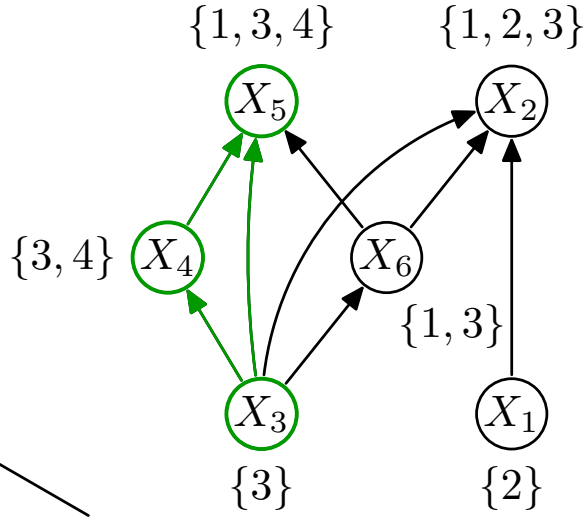
For components C and C', we define that $C \succeq C'$ if and only if

(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 $V(C')$.





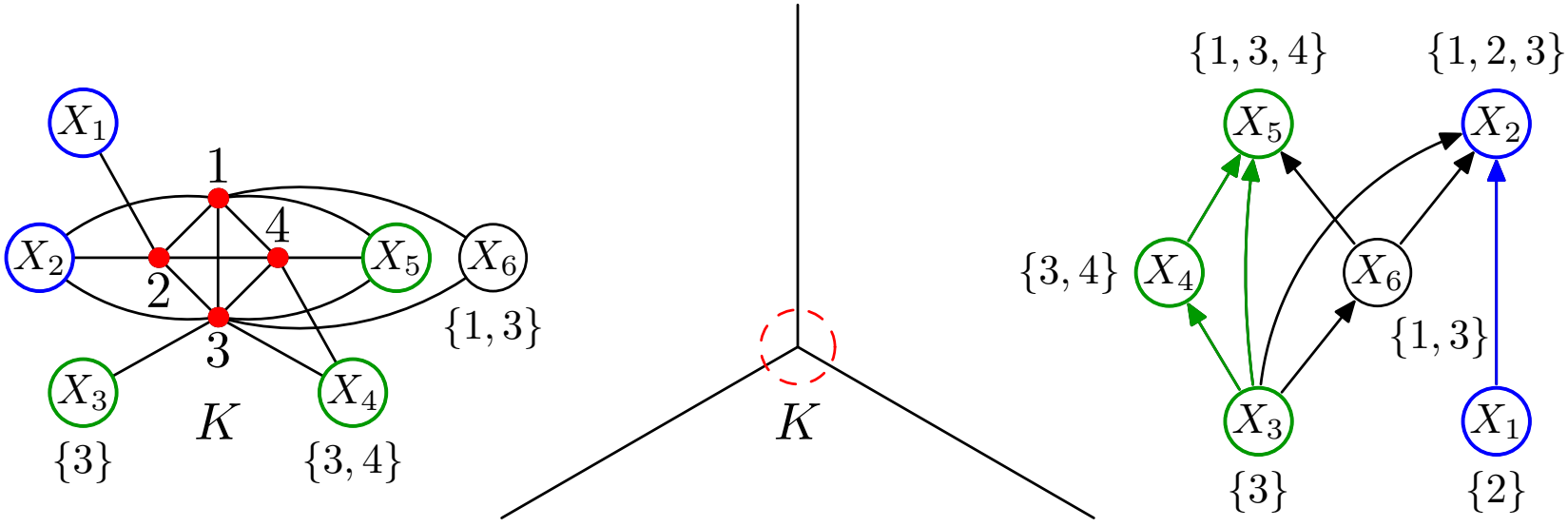


Finding a Correct Placement of Components

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 $V(C')$.

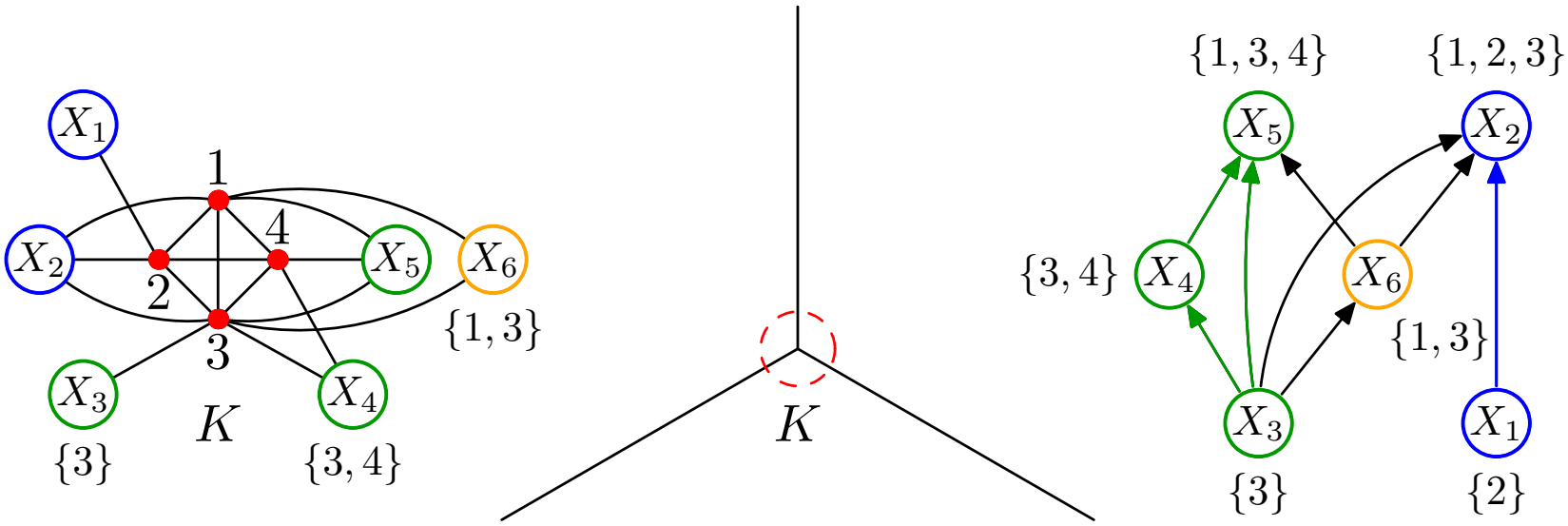


Finding a Correct Placement of Components

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 $V(C')$.

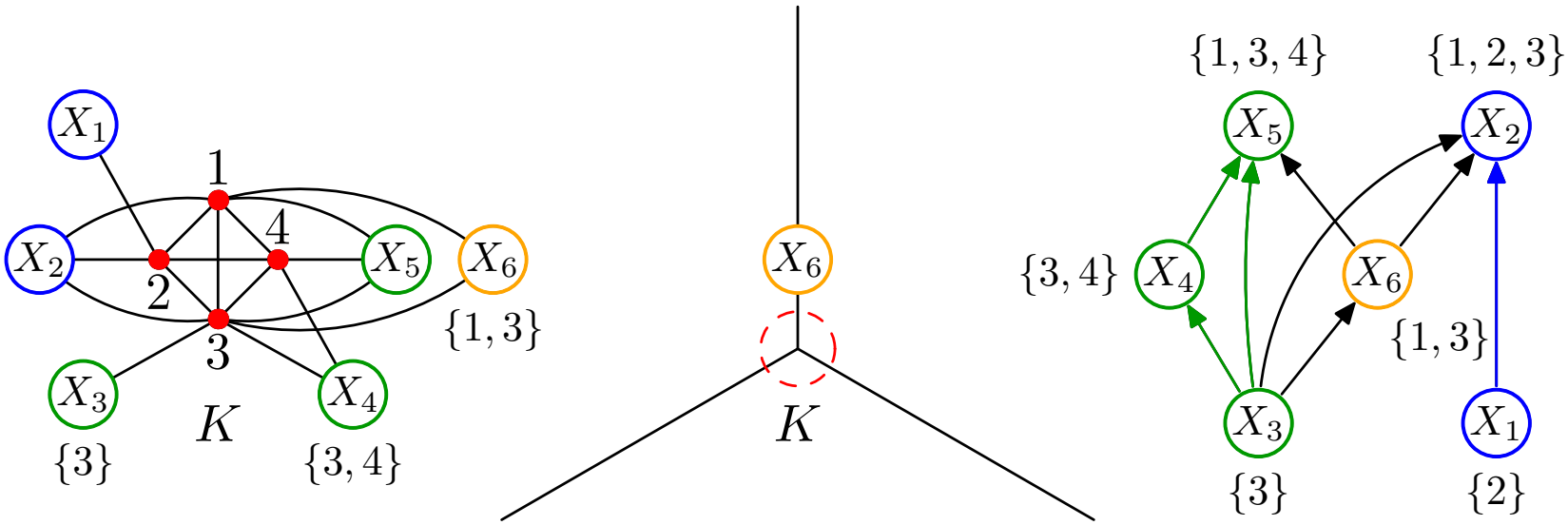


Finding a Correct Placement of Components

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 $V(C')$.

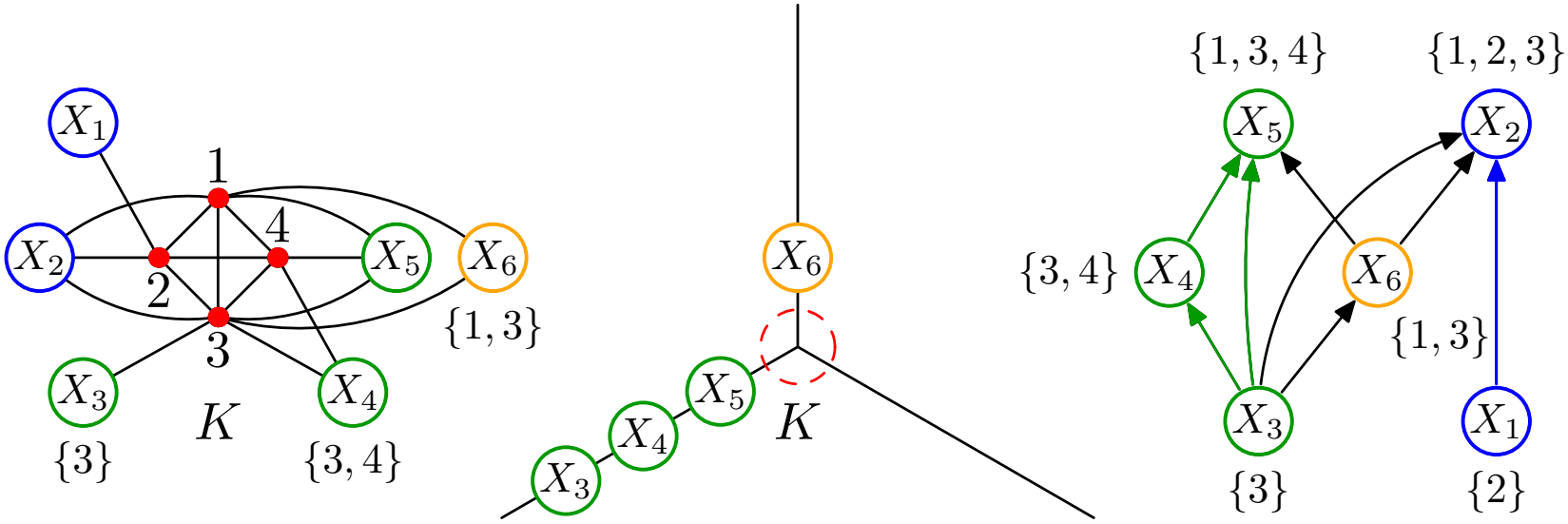


Finding a Correct Placement of Components

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 $V(C')$.

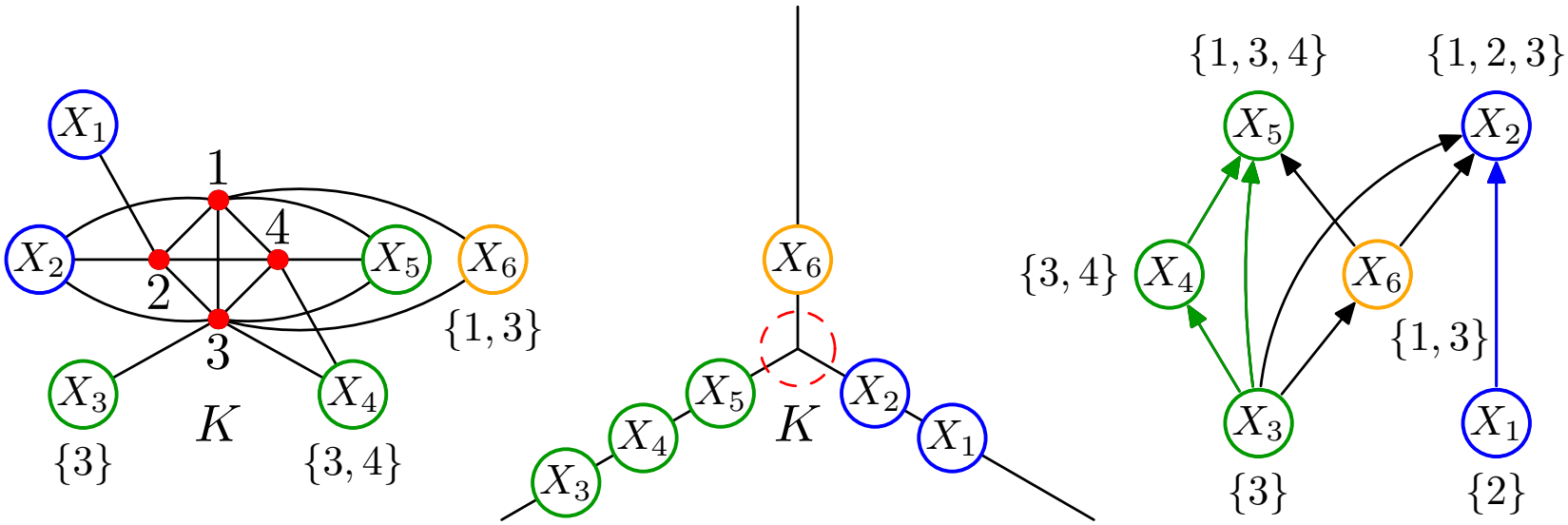


Finding a Correct Placement of Components

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 $V(C')$.



RECOG(T-GRAPH) and REPEXT(T-GRAPH)

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!