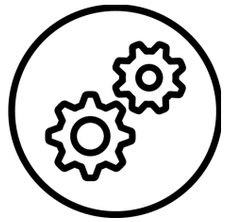
Algorithms for Planar F-Deletion and Satisfiability

Neeldhara Misra

Department of Computer Science and Automation,
Indian Institute of Science

Algorithms for Planar F-Deletion and Satisfiability

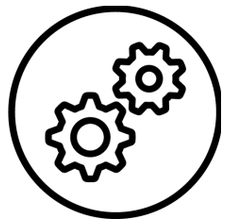
Algorithms for Planar F-Deletion and Satisfiability



The General Setup

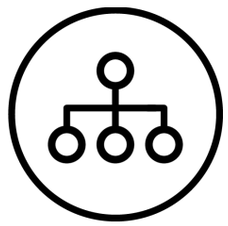
A brief introduction to parameterized algorithmics

Algorithms for Planar F-Deletion and Satisfiability



The General Setup

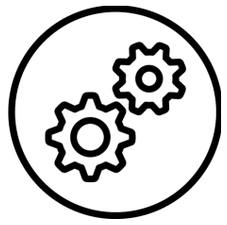
A brief introduction to parameterized algorithmics



One Size Fits All?

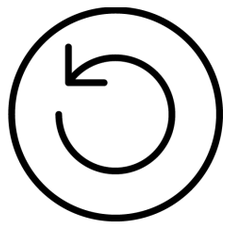
A meta-algorithm for a class of graph-theoretic problems

Algorithms for Planar F-Deletion and Satisfiability



The General Setup

A brief introduction to parameterized algorithmics



Beyond Graphs

Efficient SAT algorithms using backdoors

Algorithms for Planar F-Deletion and Satisfiability



Preference Aggregation

Strategic behavior in the context of voting

Algorithms for Planar F-Deletion and Satisfiability

The General Setup

A brief introduction to parameterized algorithmics

One Size Fits All?

A meta-algorithm for a class of graph-theoretic problems

Beyond Graphs

Efficient SAT algorithms using backdoors

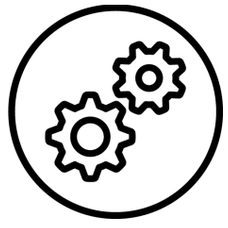
Preference Aggregation

Strategic behavior in the context of voting

Algorithms for Planar F-Deletion and Satisfiability

The General Setup

A brief introduction to parameterized algorithmics



The General Setup

A brief introduction to parameterized algorithmics

The Classical Perspective

The Classical Perspective

Evaluate running times of algorithms as a function of the size of input alone.

The Classical Perspective

Evaluate running times of algorithms as a function of the size of input alone.

The Parameterized Perspective

The Classical Perspective

Evaluate running times of algorithms as a function of the size of input alone.

The Parameterized Perspective

Evaluate running times of algorithms as a function of the size of input **and** additional structural parameters.

Vertex Cover

Does there exist a subset S of at most k vertices that intersects all edges?

Vertex Cover

Does there exist a subset S of at most k vertices that intersects all edges?

Brute Force

Vertex Cover

Does there exist a subset S of at most k vertices that intersects all edges?

Brute Force

$$n^k$$

Vertex Cover

Does there exist a subset S of at most k vertices that intersects all edges?

Pruned Search Tree

Brute Force

n^k

Vertex Cover

Does there exist a subset S of at most k vertices that intersects all edges?

Pruned Search Tree

$2^k n^{O(1)}$

Brute Force

n^k

Vertex Cover

Does there exist a subset S of at most k vertices that intersects all edges?

Pruned Search Tree

$2^k n^{O(1)}$

Brute Force

n^k

(Implemented by CBRG, ETH Zurich)

Exploiting Structure

Exploiting Structure

When will the input of a problem coming from “real life” have no more structure than its size?

Exploiting Structure

When will the input of a problem coming from “real life” have no more structure than its size?

Never!

- Fundamentals of Parameterized Complexity, Downey and Fellows, 1999

Exploiting Structure

For real-world computations, we always know more about the problem.

Exploiting Structure

For real-world computations, we always know more about the problem.

Small Degree

Exploiting Structure

For real-world computations, we always know more about the problem.

Small Degree

Planar graphs

Exploiting Structure

For real-world computations, we always know more about the problem.

Small Degree

Planar graphs

Small Nesting depth

Exploiting Structure

For real-world computations, we always know more about the problem.

Small Degree

Planar graphs

Small Nesting depth

Graphs without cycles

Exploiting Structure

Acknowledge the presence of additional structure.

Exploiting Structure

Acknowledge the presence of additional structure.

This will be a **secondary measurement**, a parameter.

Exploiting Structure

Acknowledge the presence of additional structure.

This will be a **secondary measurement**, a parameter.

Design algorithms that restrict the
combinatorial explosion to a **function of the parameter.**

Fixed-parameter Tractable Algorithms

$$f(k)p(n)$$

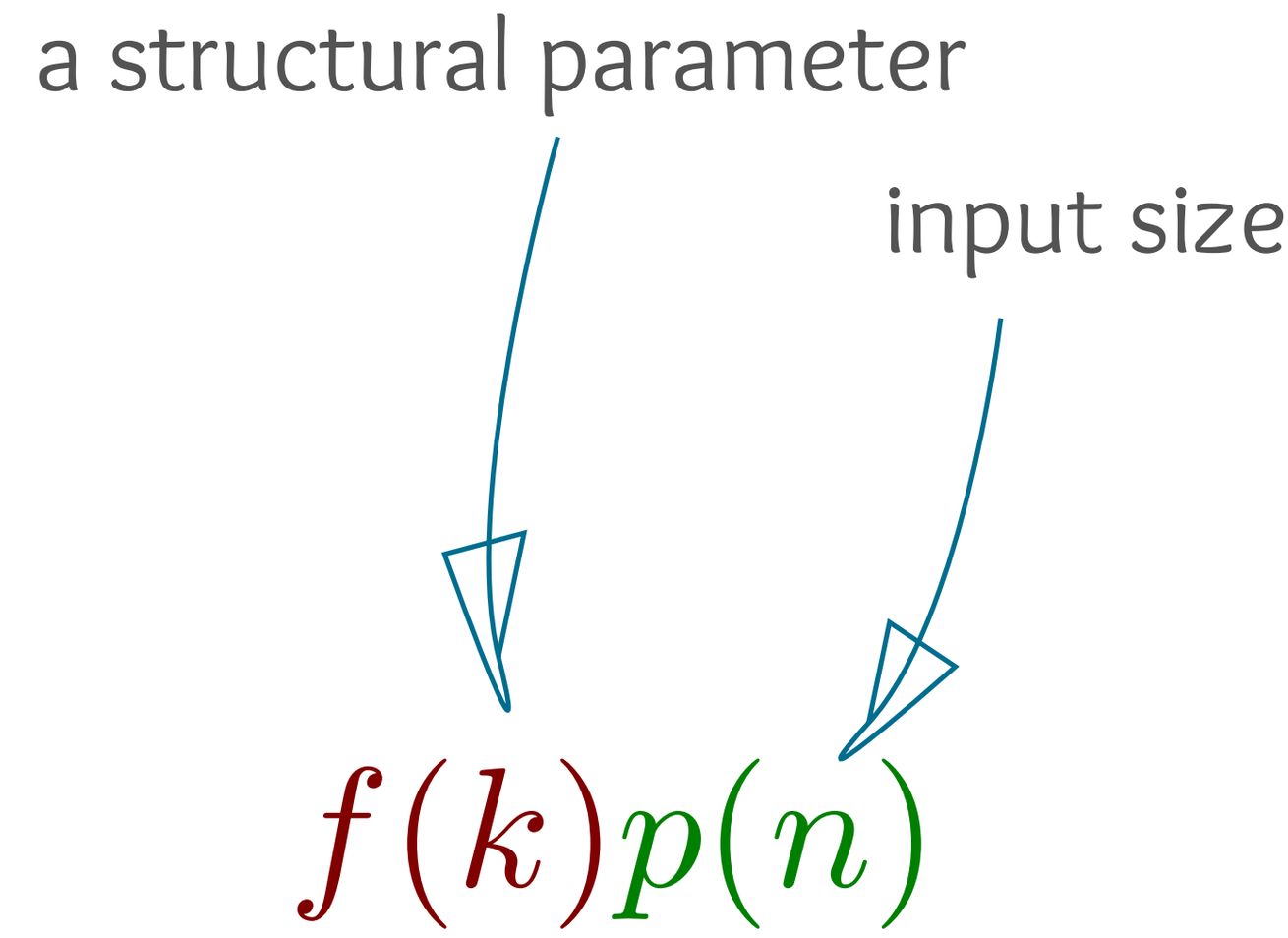
Fixed-parameter Tractable Algorithms

a structural parameter



$$f(k)p(n)$$

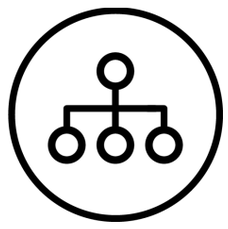
Fixed-parameter Tractable Algorithms



Algorithms for Planar F-Deletion and Satisfiability

One Size Fits All?

A meta-algorithm for a class of graph-theoretic problems



One Size Fits All?

A meta-algorithm for a class of graph-theoretic problems

One Size Fits All?

A meta-algorithm for a class of graph-theoretic problems

Define a class of graph modification problems.

One Size Fits All?

A meta-algorithm for a class of graph-theoretic problems

Define a class of graph modification problems.

Implications of the Graph Minor Theorem.

One Size Fits All?

A meta-algorithm for a class of graph-theoretic problems

Define a class of graph modification problems.

Implications of the Graph Minor Theorem.

Vertex Cover, Revisited.

One Size Fits All?

A meta-algorithm for a class of graph-theoretic problems

Define a class of graph modification problems.

Implications of the Graph Minor Theorem.

Vertex Cover, Revisited.

Beyond Vertex Cover: Feedback Vertex Set
(on graphs of constant degree).

One Size Fits All?

A meta-algorithm for a class of graph-theoretic problems

Define a class of graph modification problems.

Implications of the Graph Minor Theorem.

Vertex Cover, Revisited.

Beyond Vertex Cover: Feedback Vertex Set
(on graphs of constant degree).

Planar F-Deletion.

One Size Fits All?

A meta-algorithm for a class of graph-theoretic problems

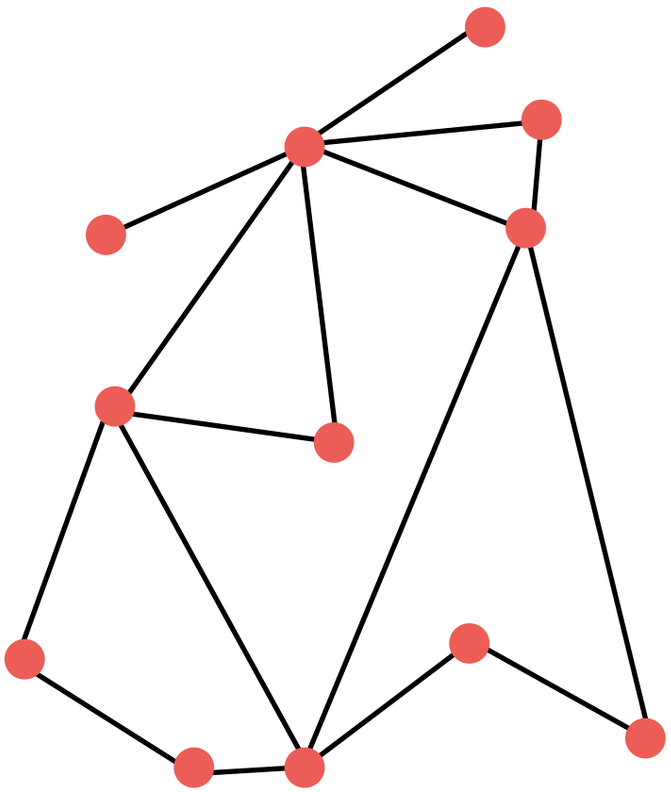
Define a class of graph modification problems.

Implications of the Graph Minor Theorem.

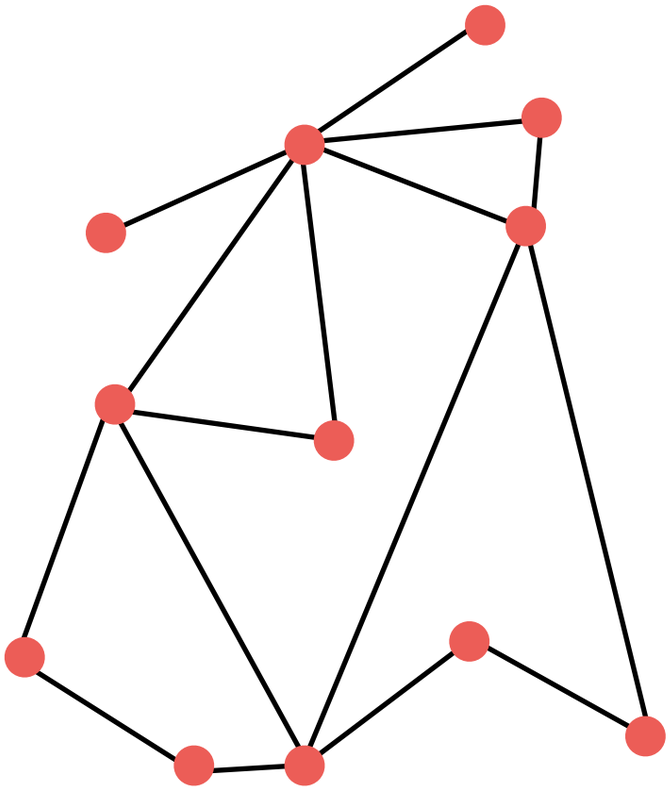
Vertex Cover, Revisited.

Beyond Vertex Cover: Feedback Vertex Set
(on graphs of constant degree).

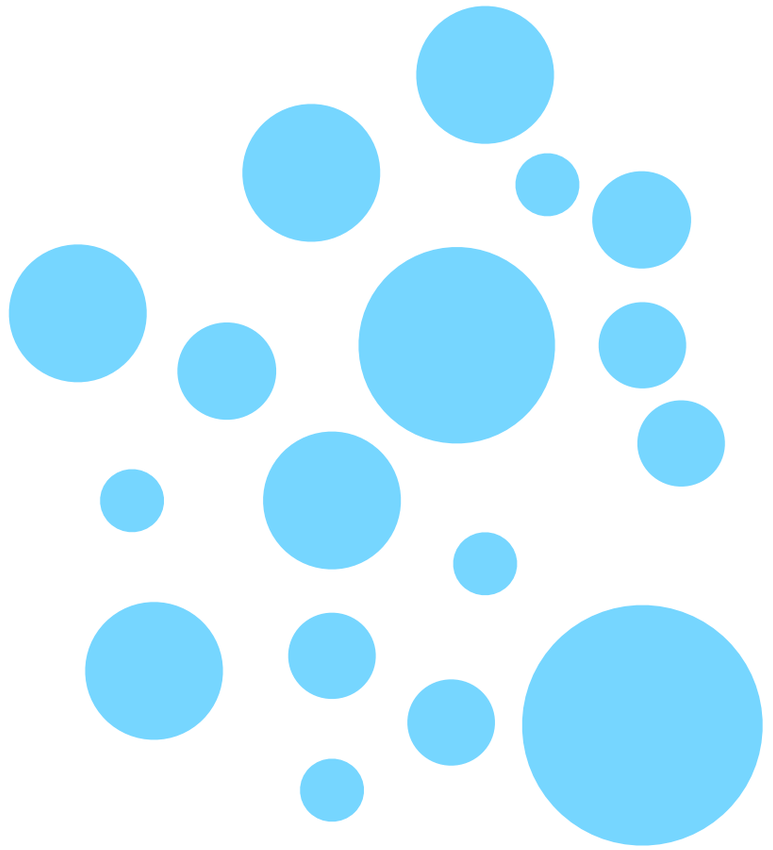
Planar F-Deletion.



Input Graph, G

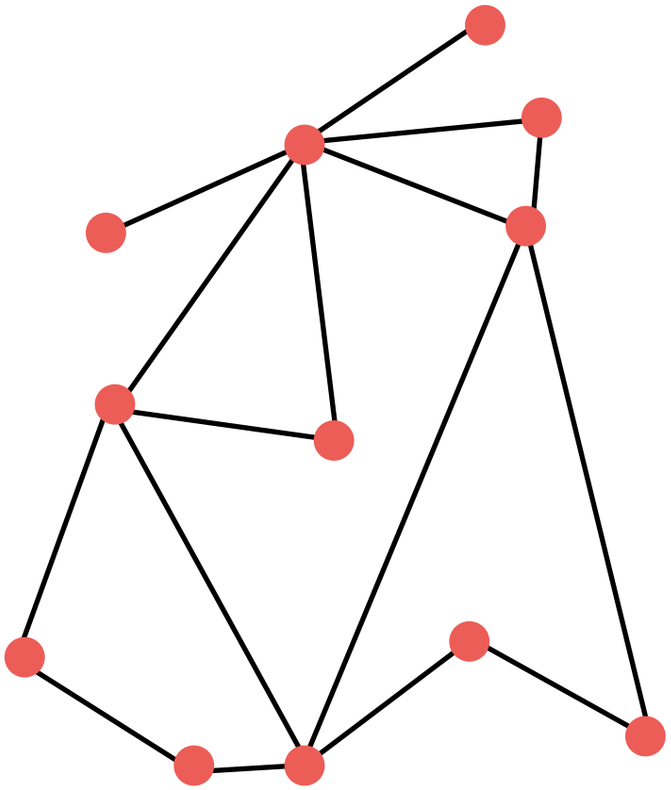


Input Graph, G

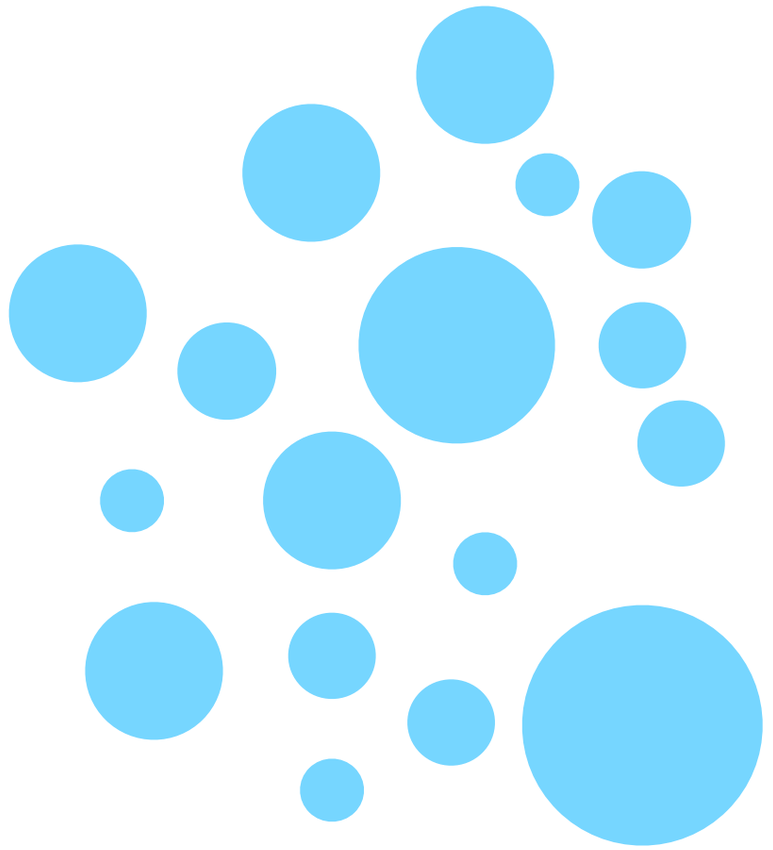


Graph Class or “property”, π

Goal: Transform G so that it belongs to π with "minimum damages".

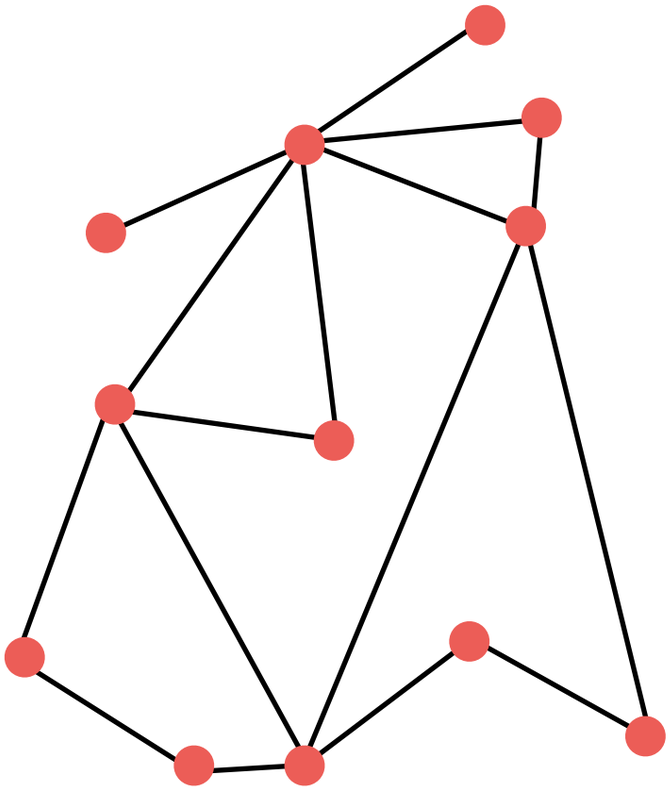


Input Graph, G

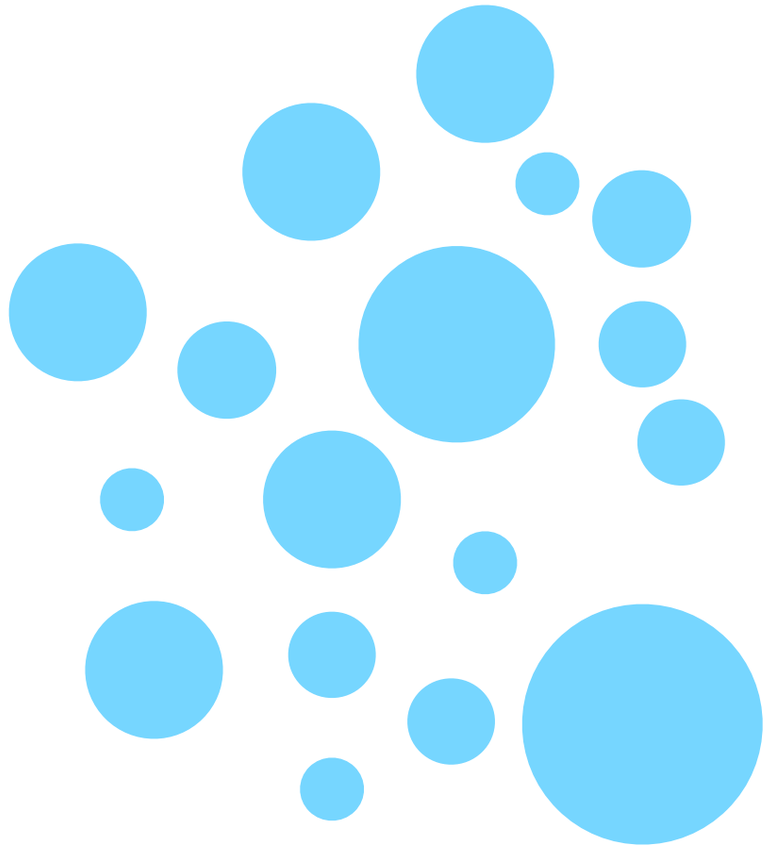


Graph Class or "property", π

Goal: Transform G so that it belongs to π with "minimum damages".



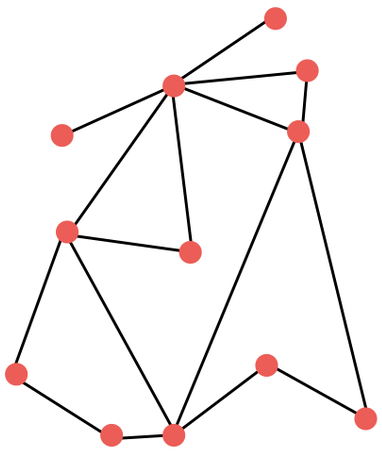
Input Graph, G



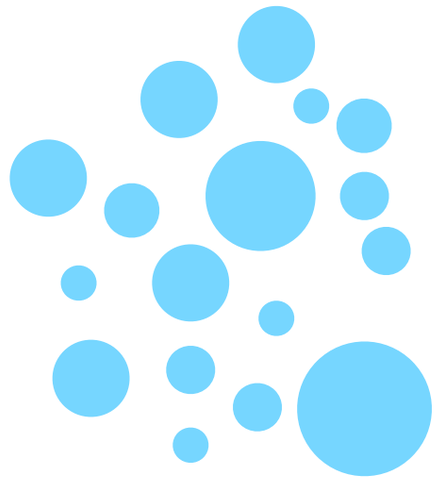
Graph Class or "property", π

Goal: Transform G so that it belongs to **π** with "minimum damages".

Input Graph, G



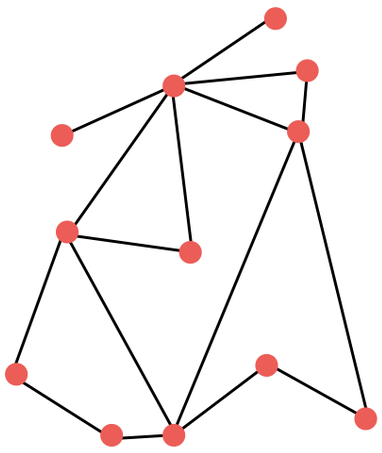
Graph Class or "property", **π**



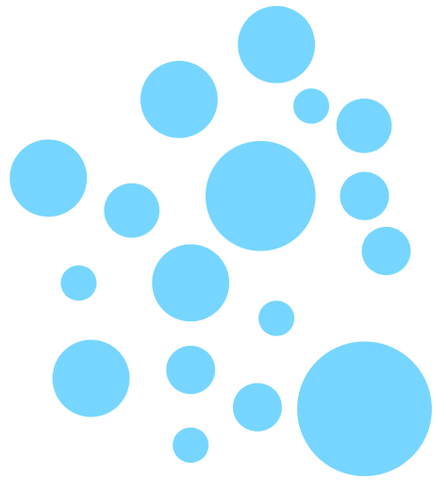
Goal: Transform G so that it belongs to **π** with “minimum damages”.

vertex deletions

Input Graph, G



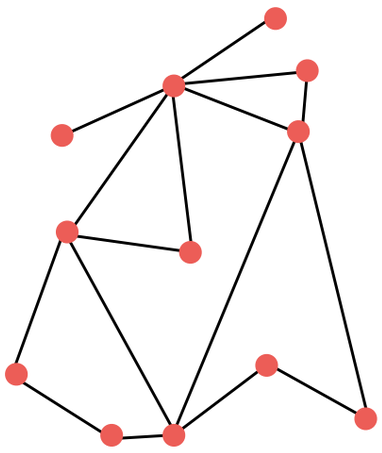
Graph Class or “property”, **π**



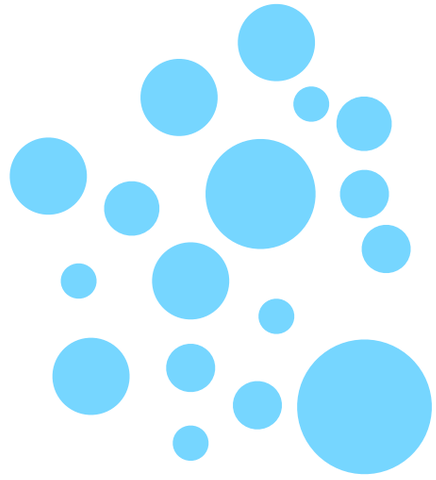
Goal: Transform G so that it belongs to **π** with "minimum damages".

vertex deletions edge deletions

Input Graph, G



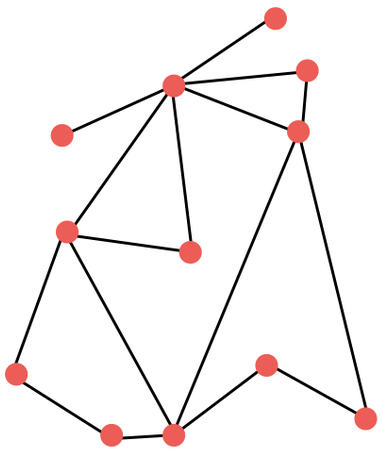
Graph Class or "property", **π**



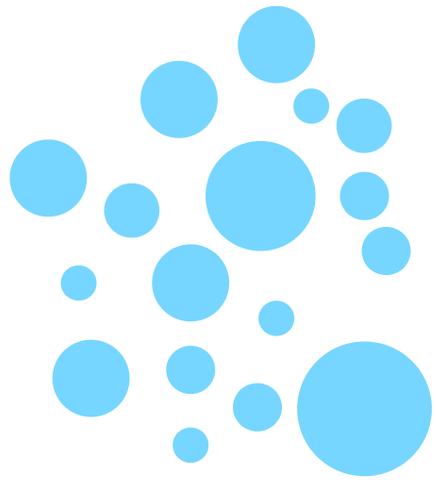
Goal: Transform G so that it belongs to **π** with "minimum damages".

vertex deletions edge deletions edge additions

Input Graph, G



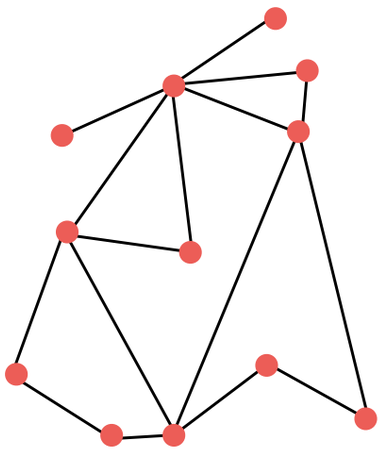
Graph Class or "property", **π**



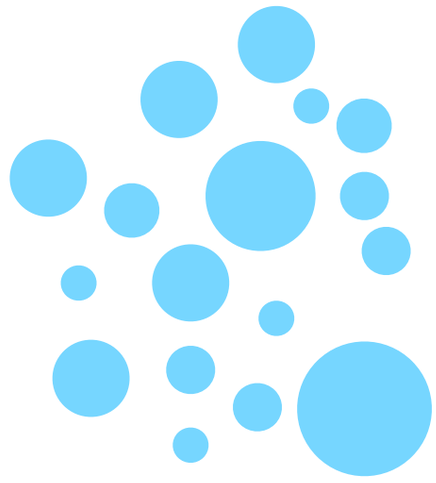
Goal: Transform G so that it belongs to **π** with "minimum damages".

vertex deletions edge deletions edge additions edge contractions

Input Graph, G



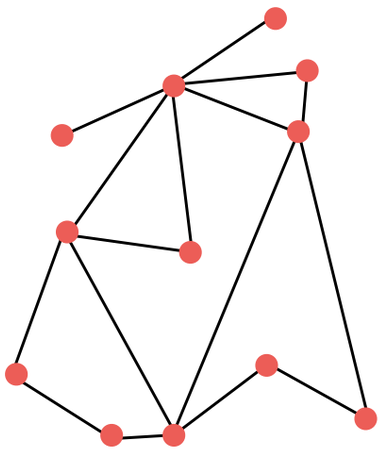
Graph Class or "property", **π**



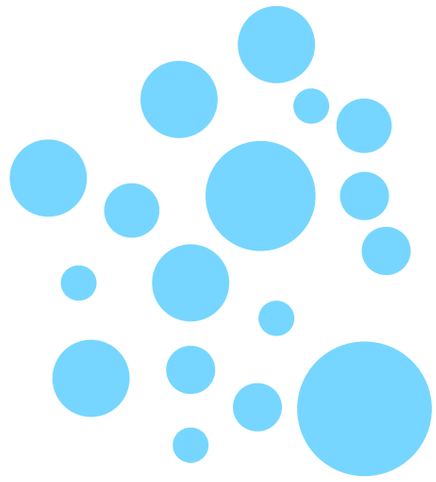
Goal: Transform G so that it belongs to **π** with "minimum damages".

vertex deletions edge deletions edge additions edge contractions edge editing

Input Graph, G



Graph Class or "property", **π**

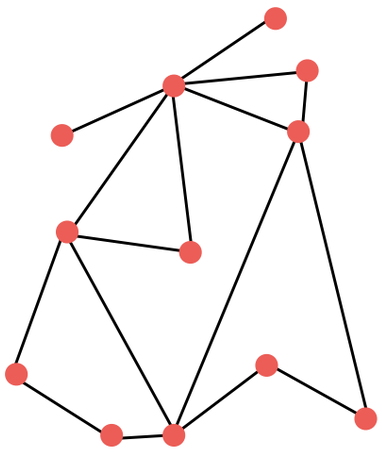


Goal: Transform G so that it belongs to π with "minimum damages".

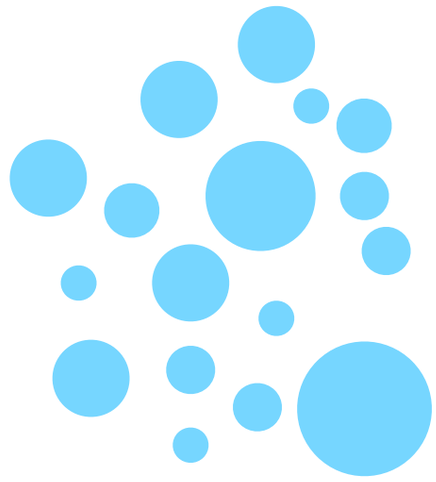
vertex deletions edge deletions edge additions edge contractions edge editing

Vertex Cover

Input Graph, G



Edgeless Graphs

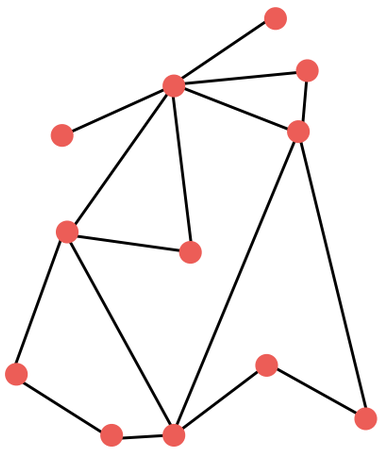


Goal: Transform G so that it belongs to π with "minimum damages".

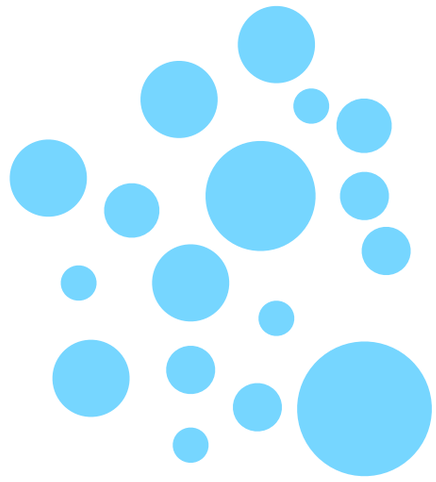
vertex deletions edge deletions edge additions edge contractions edge editing

Feedback Vertex Set

Input Graph, G



Acyclic Graphs

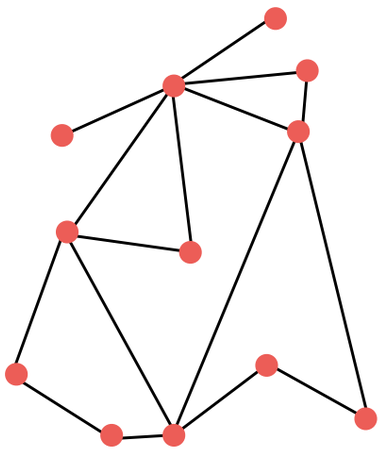


Goal: Transform G so that it belongs to π with "minimum damages".

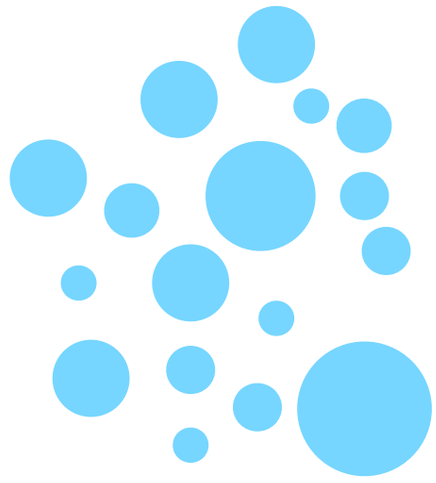
vertex deletions edge deletions edge additions edge contractions edge editing

Minimum Fill-In

Input Graph, G



Chordal Graphs

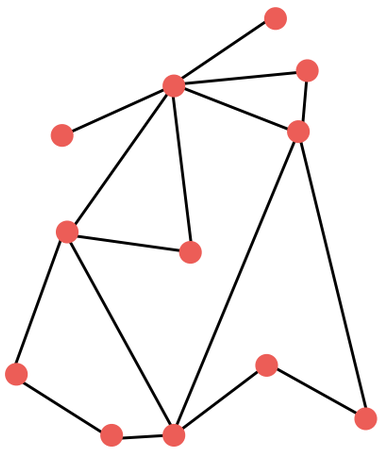


Goal: Transform G so that it belongs to π with "minimum damages".

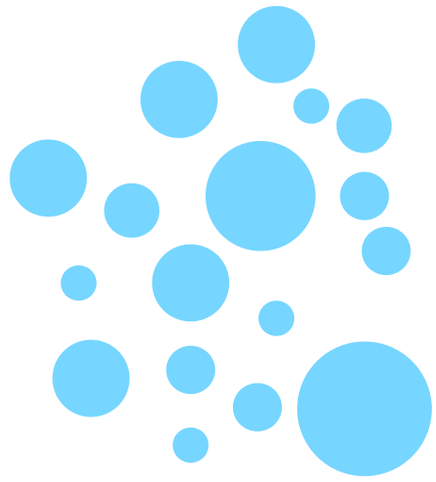
vertex deletions edge deletions edge additions edge contractions edge editing

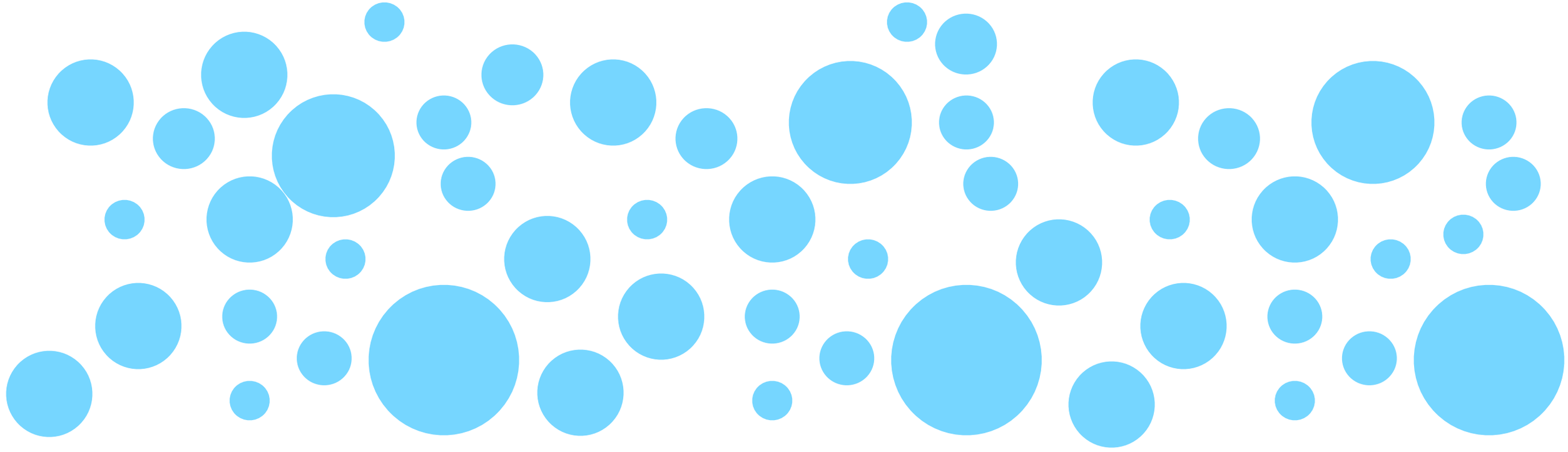
Cluster Editing

Input Graph, G



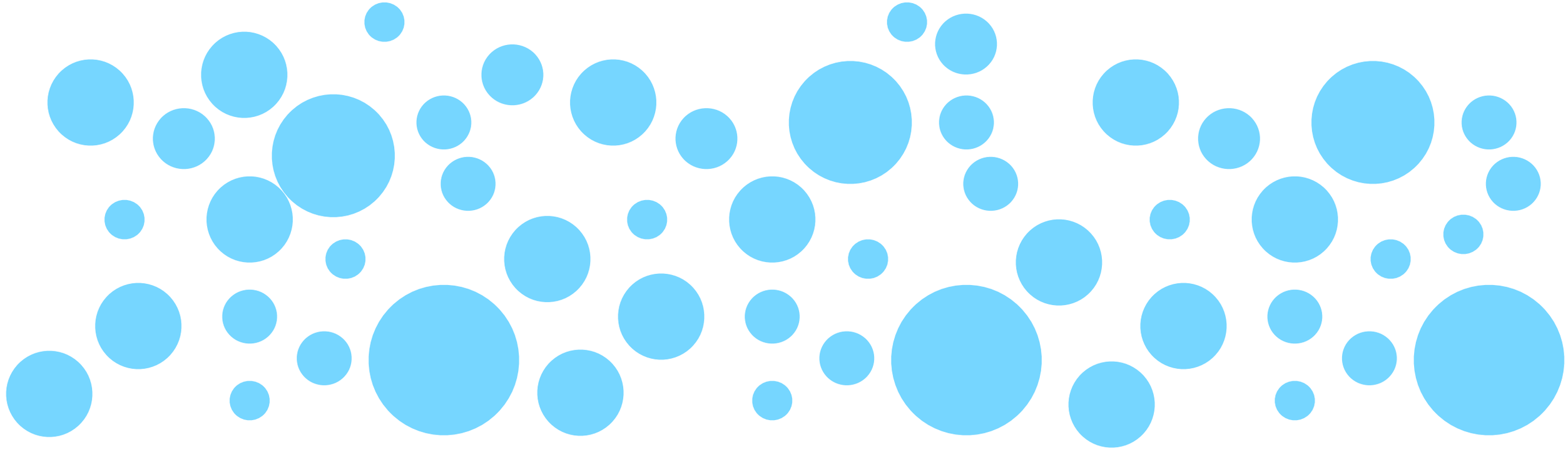
Cluster Graphs





Graph Class or “property”, π

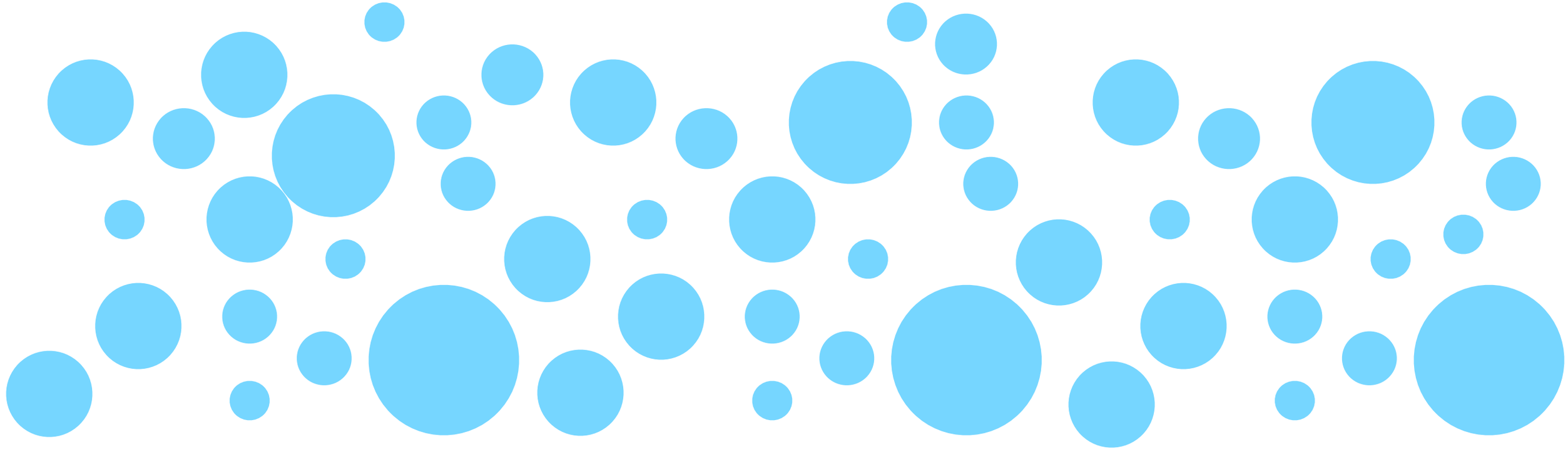
Non-Trivial π admits infinitely many graphs, and also excludes infinitely many graphs.



Graph Class or “property”, π

Non-Trivial **π** admits infinitely many graphs, and also excludes infinitely many graphs.

Monotone If G belongs to **π**, then all **subgraphs** of G also belong to **π**.

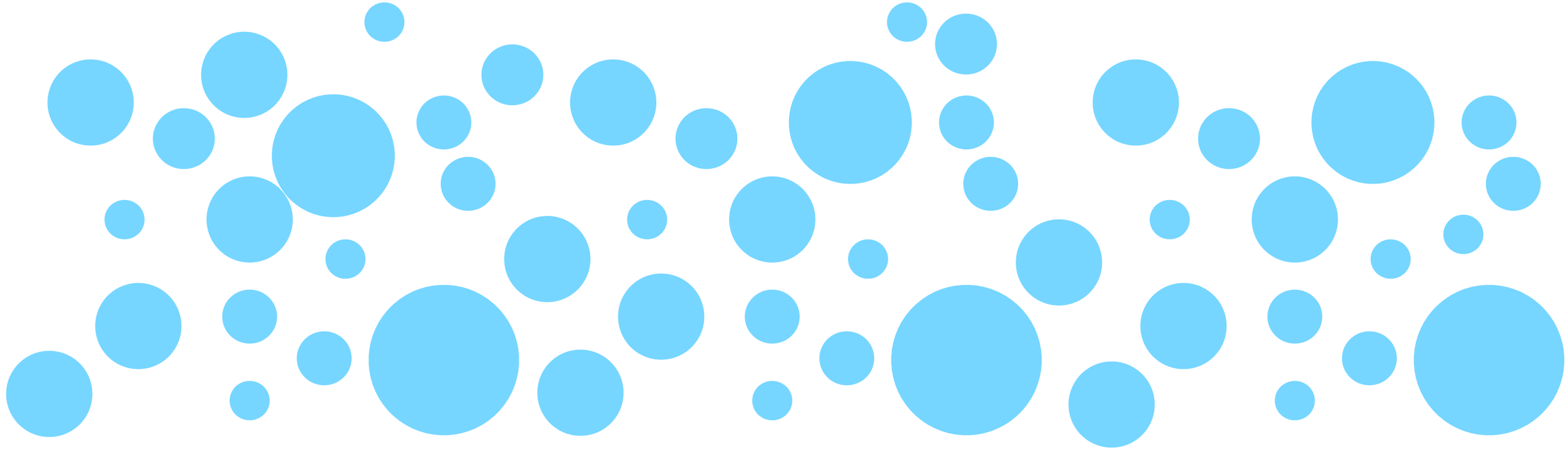


Graph Class or “property”, **π**

Non-Trivial **π** admits infinitely many graphs, and also excludes infinitely many graphs.

Monotone If G belongs to **π**, then all **subgraphs** of G also belong to **π.**

Hereditary If G belongs to **π**, then all **induced subgraphs** of G also belong to **π.**



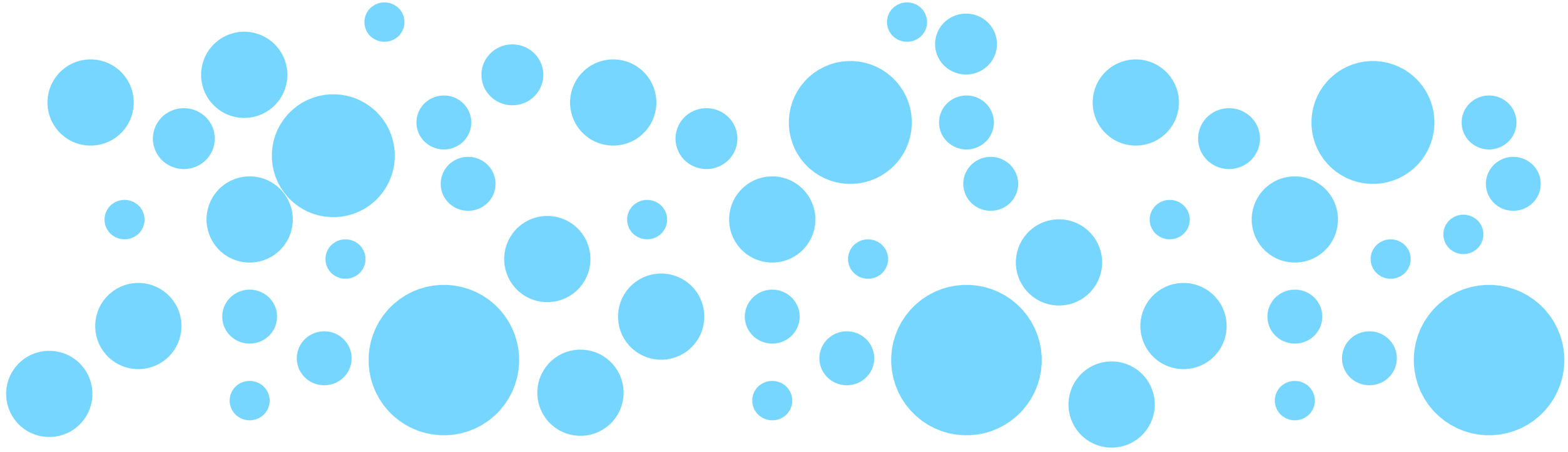
Graph Class or “property”, **π**

Non-Trivial π admits infinitely many graphs, and also excludes infinitely many graphs.

Monotone If G belongs to π, then all **subgraphs** of G also belong to π.

(Bipartite, Planar)

Hereditary If G belongs to π, then all **induced subgraphs** of G also belong to π.



Graph Class or “property”, π

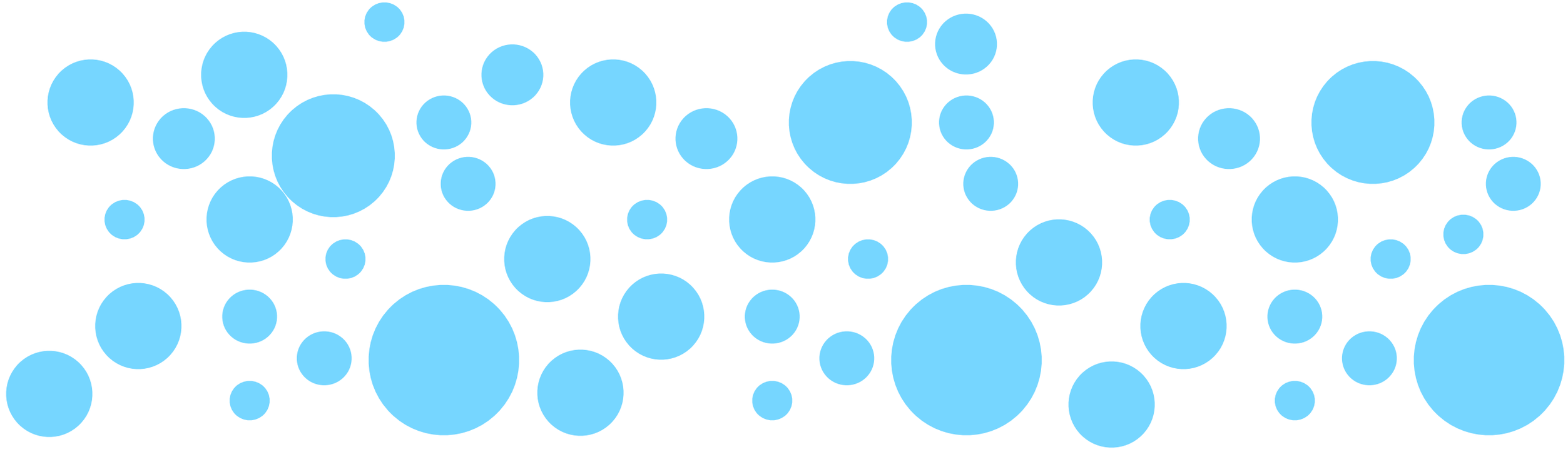
Non-Trivial **π** admits infinitely many graphs, and also excludes infinitely many graphs.

Monotone If G belongs to **π**, then all **subgraphs** of G also belong to **π**.

(Bipartite, Planar)

Hereditary If G belongs to **π**, then all **induced subgraphs** of G also belong to **π**.

(Complete graphs, Forests)



Graph Class or “property”, **π**

Non-Trivial π admits infinitely many graphs, and also excludes infinitely many graphs.

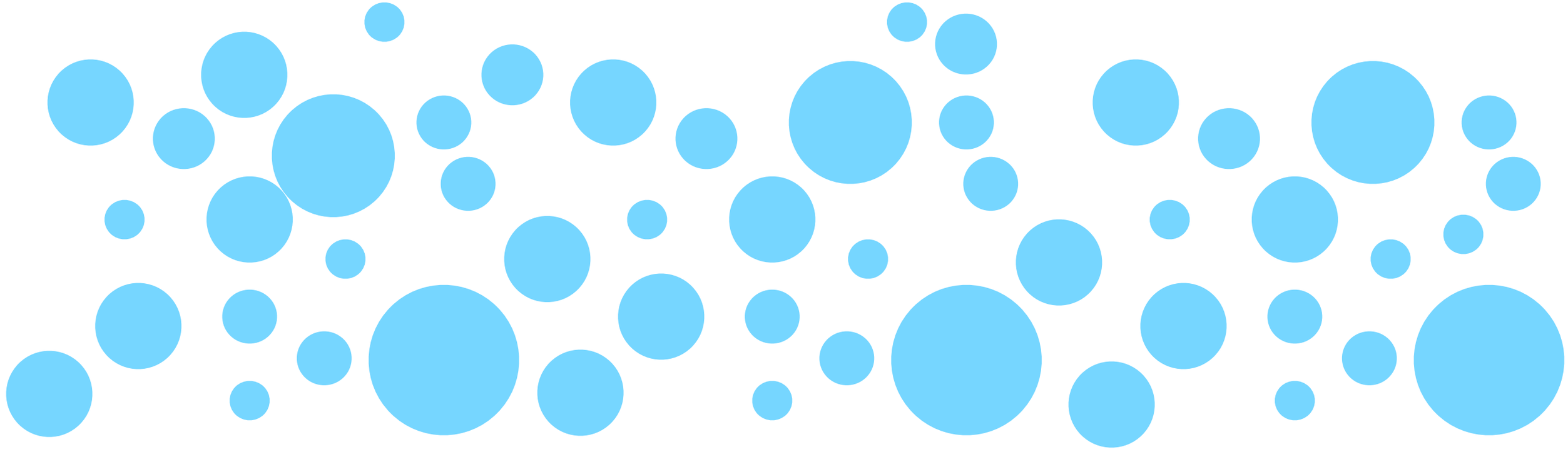
Monotone If G belongs to π, then all **subgraphs** of G also belong to π.

(Bipartite, Planar)

Hereditary If G belongs to π, then all **induced subgraphs** of G also belong to π.

(Complete graphs, Forests)

Connectivity, Biconnectivity, Trees, Stars, Eulerian, etc.



Graph Class or “property”, π

Lewis & Yannakakis [1980]

The vertex-deletion problem is **NP-complete** for non-trivial, hereditary properties.

Lewis & Yannakakis [1980]

The vertex-deletion problem is **NP-complete** for non-trivial, hereditary properties.

Yannakakis [1979]

The *connected* vertex-deletion problem is **NP-complete** for non-trivial properties that hold on connected induced subgraphs.

Lewis & Yannakakis [1980]

The vertex-deletion problem is **NP-complete** for non-trivial, hereditary properties.

(Edgeless, Acyclic, etc.)

Yannakakis [1979]

The *connected* vertex-deletion problem is **NP-complete** for non-trivial properties that hold on connected induced subgraphs.

Lewis & Yannakakis [1980]

The vertex-deletion problem is **NP-complete** for non-trivial, hereditary properties.

(Edgeless, Acyclic, etc.)

Yannakakis [1979]

The *connected* vertex-deletion problem is **NP-complete** for non-trivial properties that hold on connected induced subgraphs.

(Trees, Stars, etc.)

One Size Fits All?

A meta-algorithm for a class of graph-theoretic problems

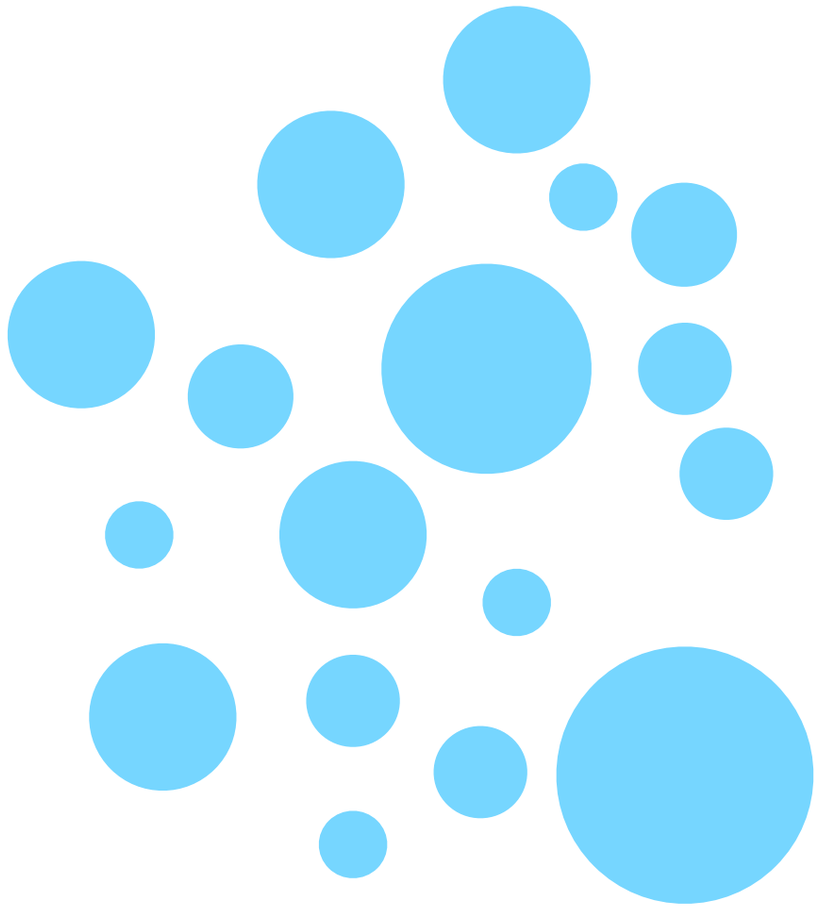
Define a class of graph modification problems.

Implications of the Graph Minor Theorem.

Vertex Cover, Revisited.

Beyond Vertex Cover: Feedback Vertex Set
(on graphs of constant degree).

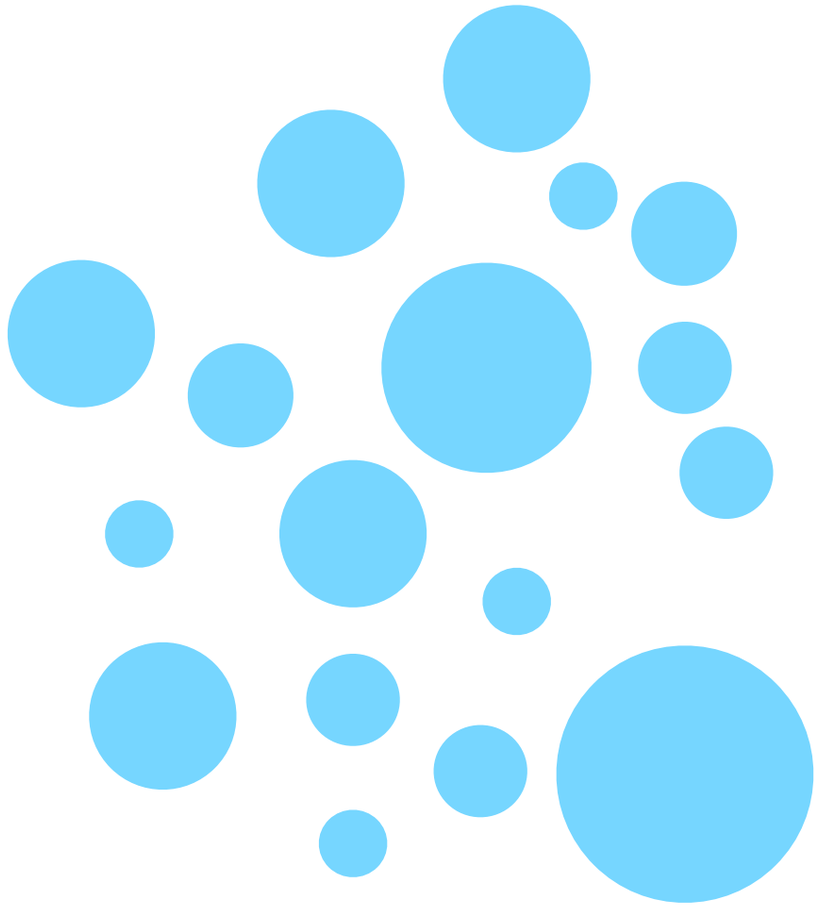
Planar F-Deletion.



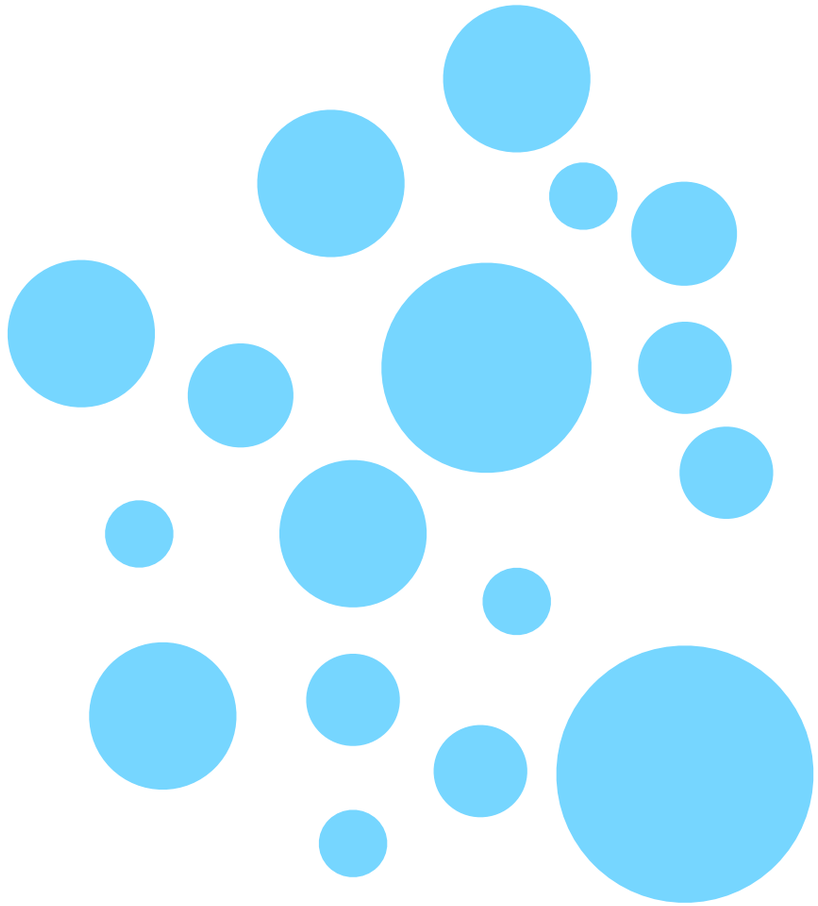
Graph Class or “property”, **π**

Forbidden Subgraph Characterization

A graph G belongs to π if, and only if,
G does not contain any graph from Forb(π)
as an **induced subgraph**.



Graph Class or “property”, π



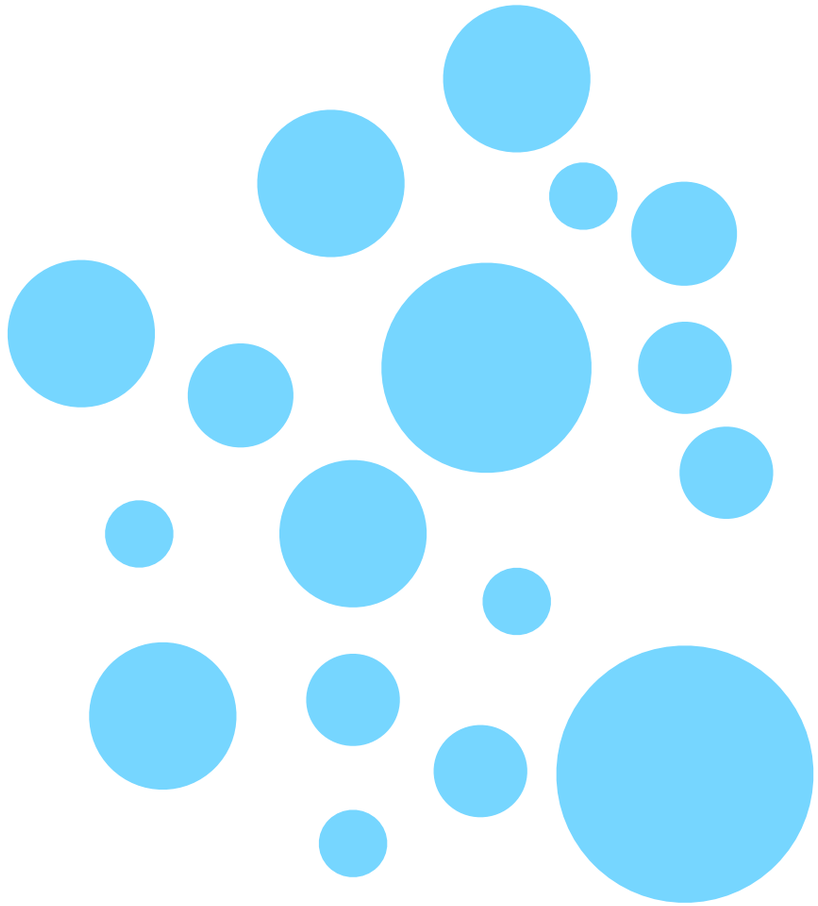
Graph Class or “property”, π

Forbidden Subgraph Characterization

A graph G belongs to π if, and only if,
G does not contain any graph from Forb(π)
as an **induced subgraph**.

Forbidden Minor Characterization

A graph G belongs to π if, and only if,
G does not contain any graph from Forb(π)
as a **minor**.



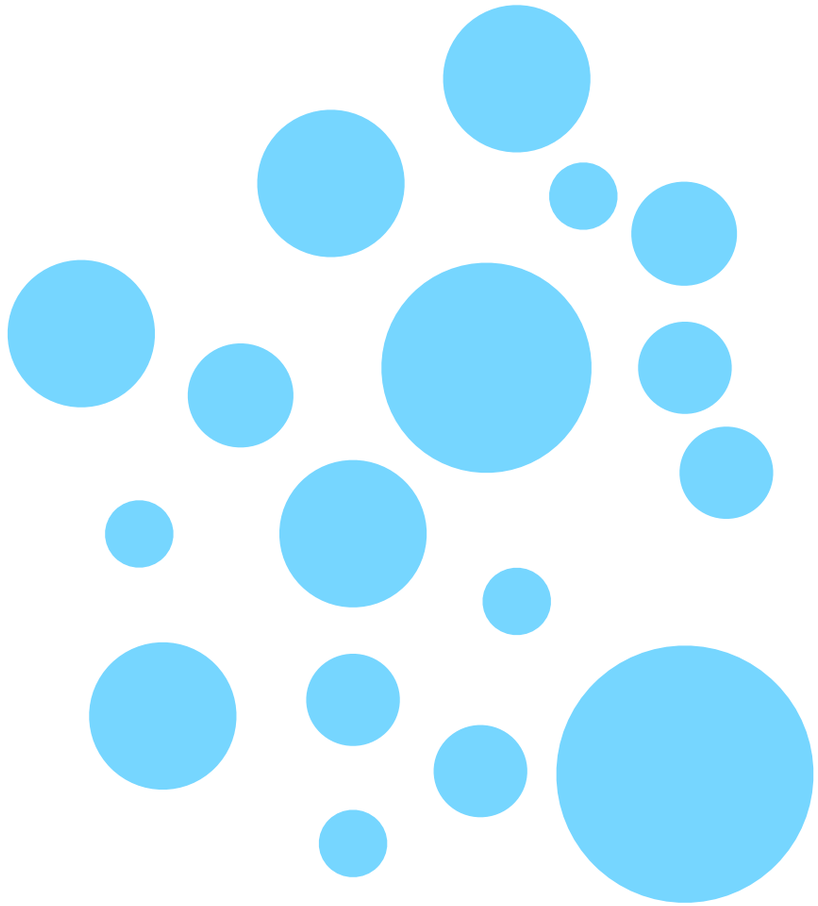
Graph Class or “property”, π

Forbidden Subgraph Characterization

A graph admits a **forbidden subgraph characterization**
if, and only if,
it is **closed under taking induced subgraphs (hereditary)**,
that is, the property is preserved under vertex deletions.

Forbidden Minor Characterization

A graph G belongs to **π** if, and only if,
G does not contain any graph from Forb(**π**)
as a **minor**.



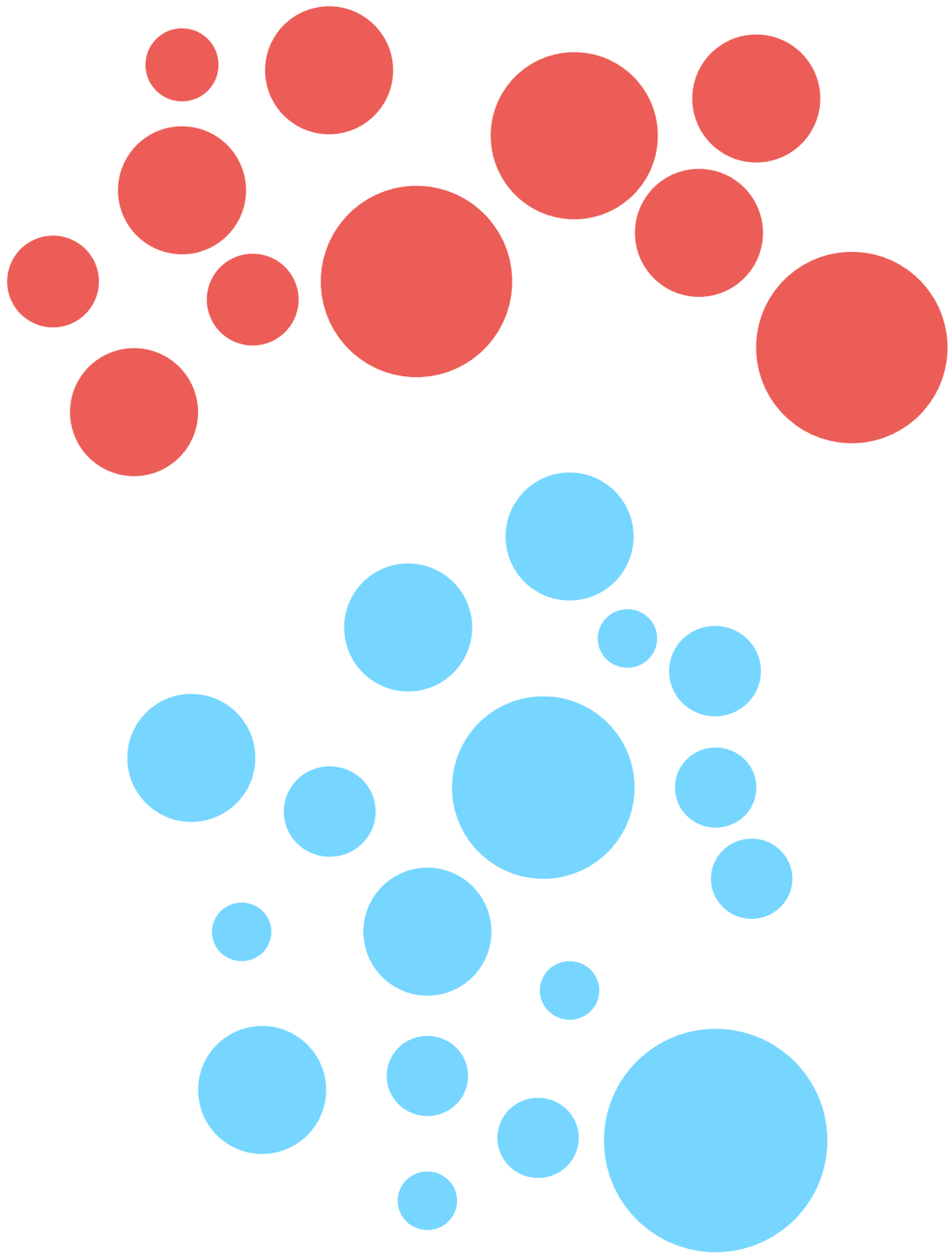
Graph Class or “property”, **π**

Forbidden Subgraph Characterization

A graph admits a **forbidden subgraph characterization**
if, and only if,
it is **closed under taking induced subgraphs (hereditary)**,
that is, the property is preserved under vertex deletions.

Forbidden Minor Characterization

A graph admits a **forbidden minor characterization**
if, and only if,
it is **closed under taking minors**,
that is, the property is preserved under vertex deletions,
edge deletions and edge contractions.



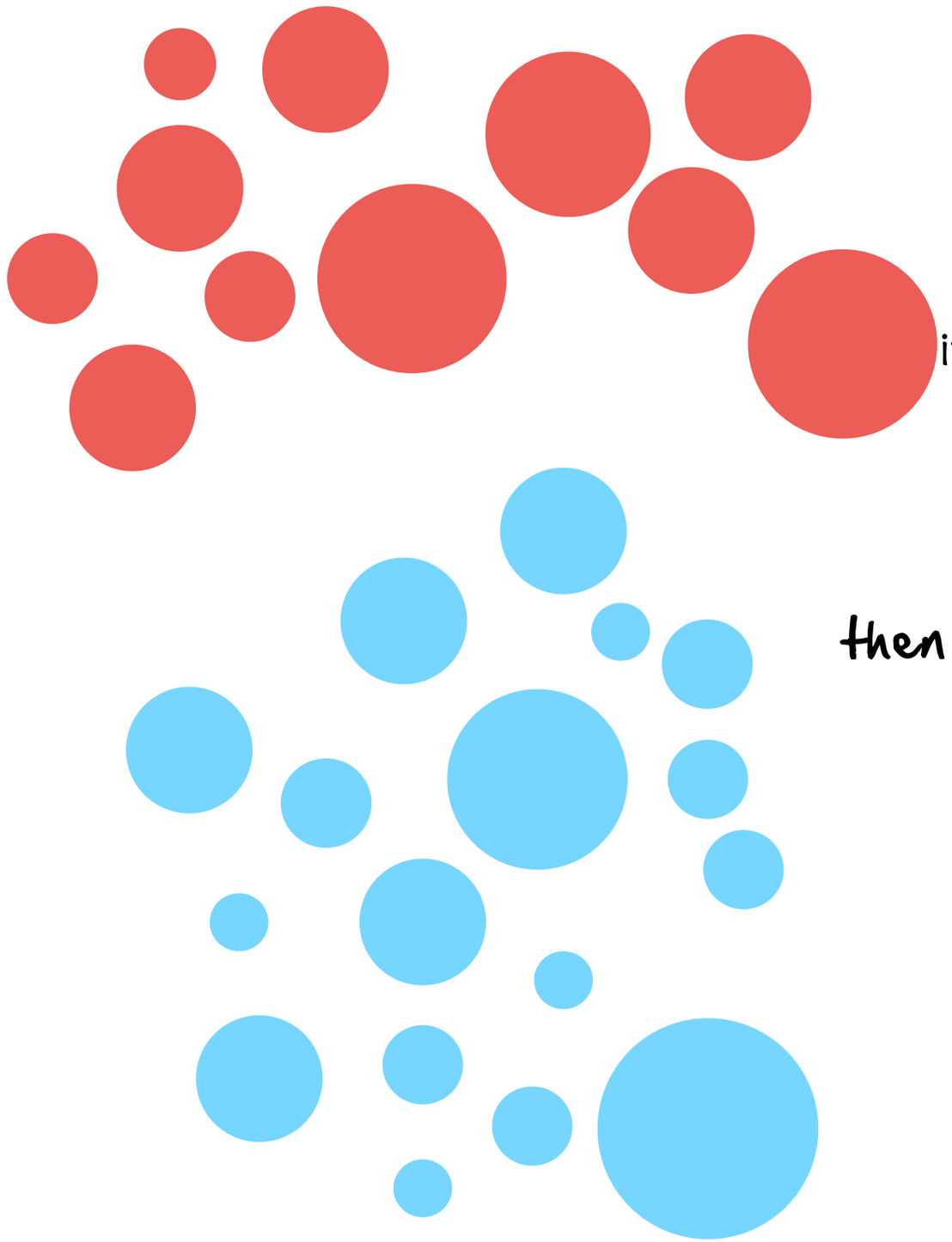
Graph Class or “property”, **π**

Forbidden Subgraph Characterization

A graph admits a **forbidden subgraph characterization**
if, and only if,
it is **closed under taking induced subgraphs (hereditary)**,
that is, the property is preserved under vertex deletions.

Forbidden Minor Characterization

A graph admits a **forbidden minor characterization**
if, and only if,
it is **closed under taking minors**,
that is, the property is preserved under vertex deletions,
edge deletions and edge contractions.



Graph Class or “property”, **π**

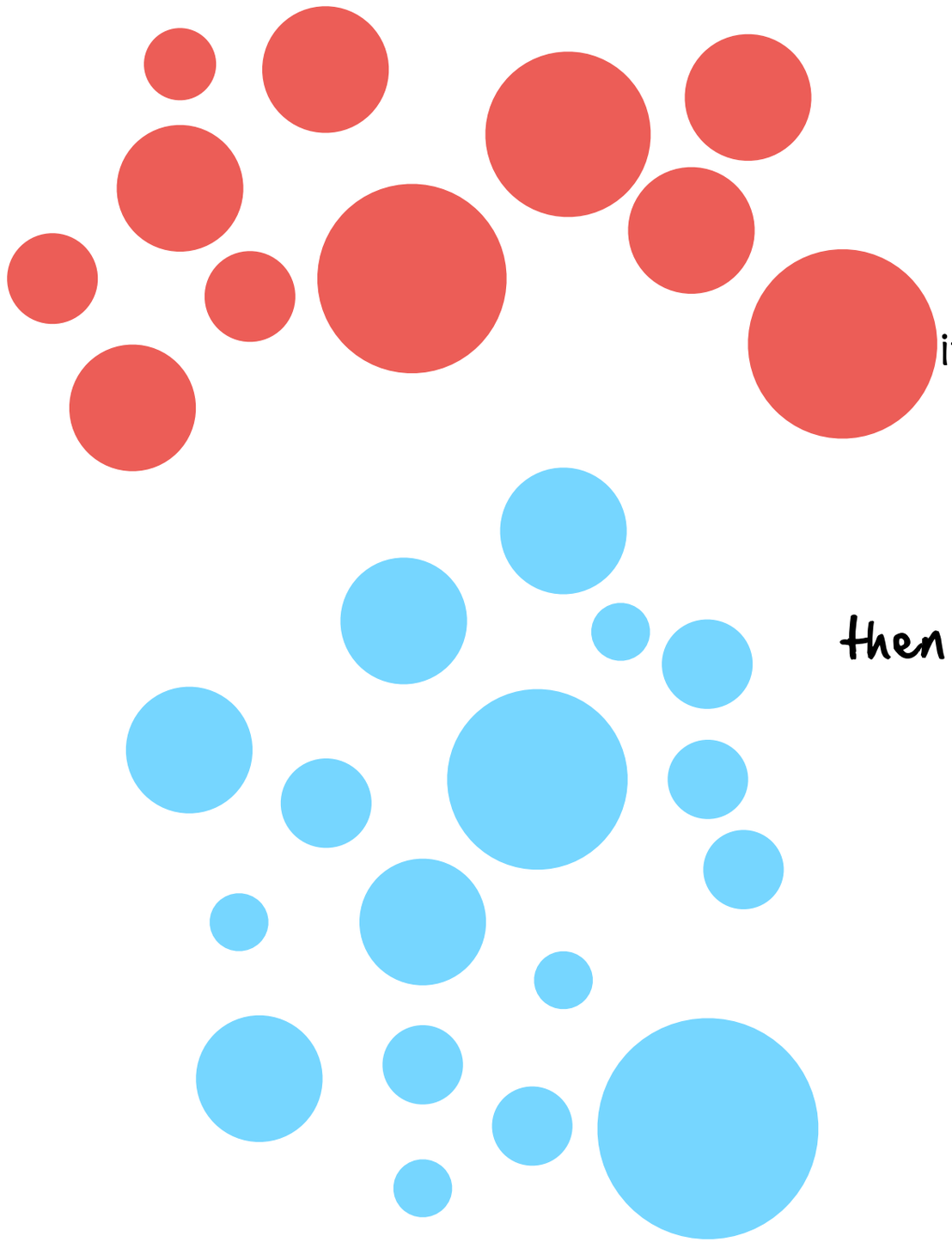
Forbidden Subgraph Characterization

A graph admits a **forbidden subgraph characterization**
if, and only if,
it is **closed under taking induced subgraphs (hereditary)**,
that is, the property is preserved under vertex deletions.

If there are finitely many forbidden graphs,
then the corresponding graph modification problem is FPT.

Forbidden Minor Characterization

A graph admits a **forbidden minor characterization**
if, and only if,
it is **closed under taking minors**,
that is, the property is preserved under vertex deletions,
edge deletions and edge contractions.



Graph Class or “property”, π

Forbidden Subgraph Characterization

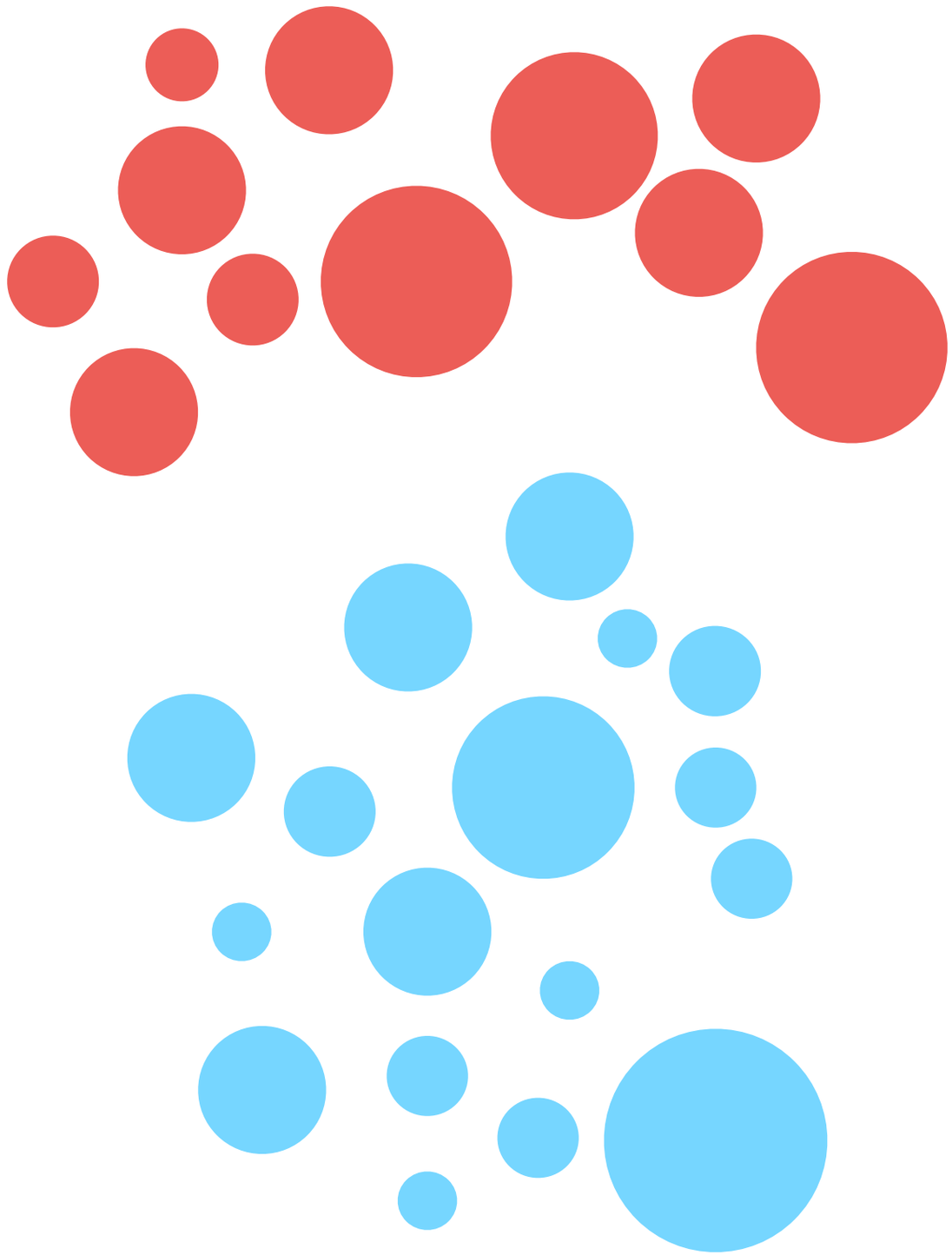
A graph admits a **forbidden subgraph characterization**
if, and only if,
it is **closed under taking induced subgraphs (hereditary)**,
that is, the property is preserved under vertex deletions.

If there are finitely many forbidden graphs,
then the corresponding graph modification problem is FPT.

Forbidden Minor Characterization

A graph admits a **forbidden minor characterization**
if, and only if,
it is **closed under taking minors**,
that is, the property is preserved under vertex deletions,
edge deletions and edge contractions.

If there are finitely many forbidden minors...?

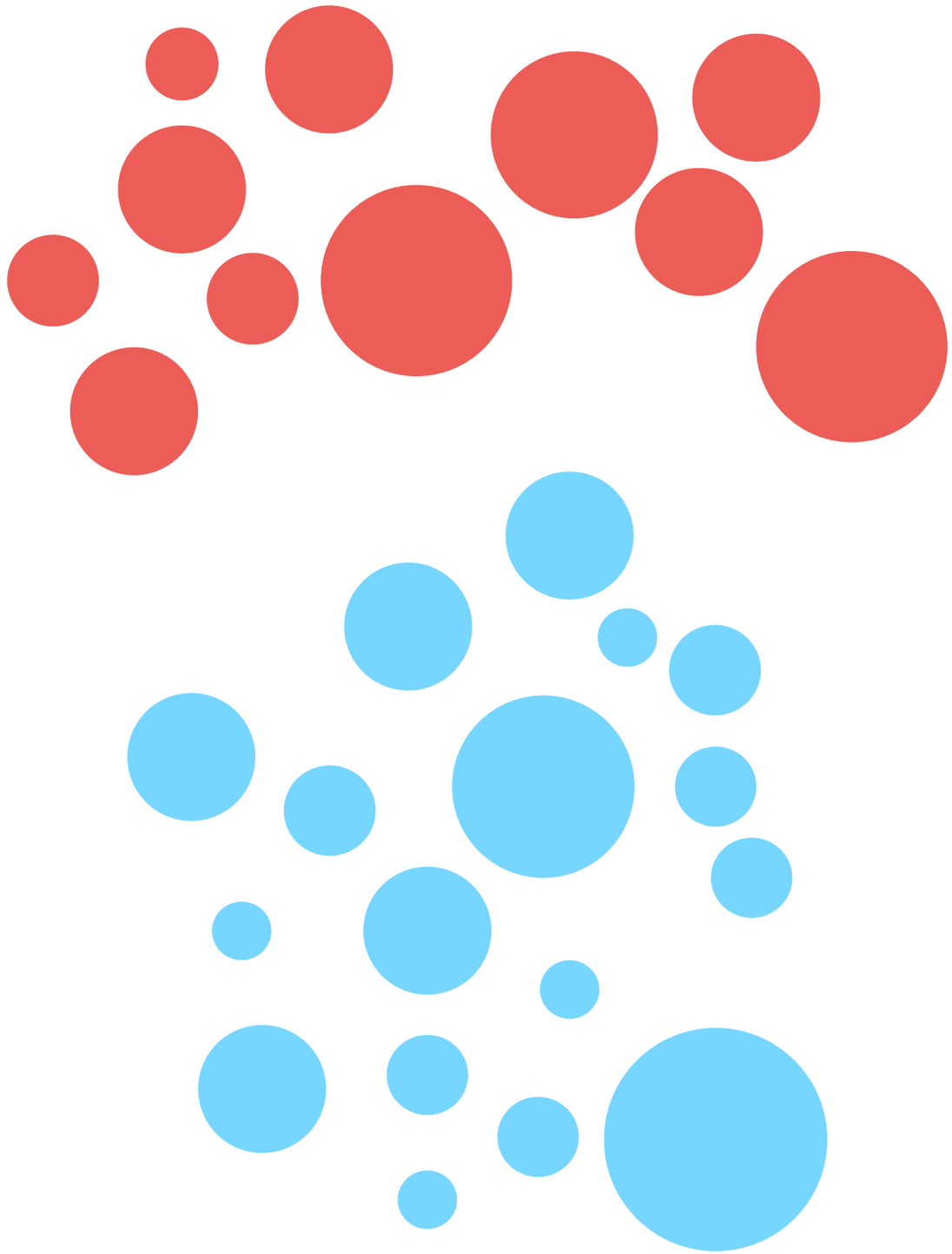


Graph Class or “property”, **π**

Forbidden Minor Characterization

A graph G belongs to **π** if, and only if,
G does not contain any graph from Forb(**π**)
as a **minor**.

A graph admits a **forbidden minor characterization**
if, and only if,
it is **closed under taking minors**.



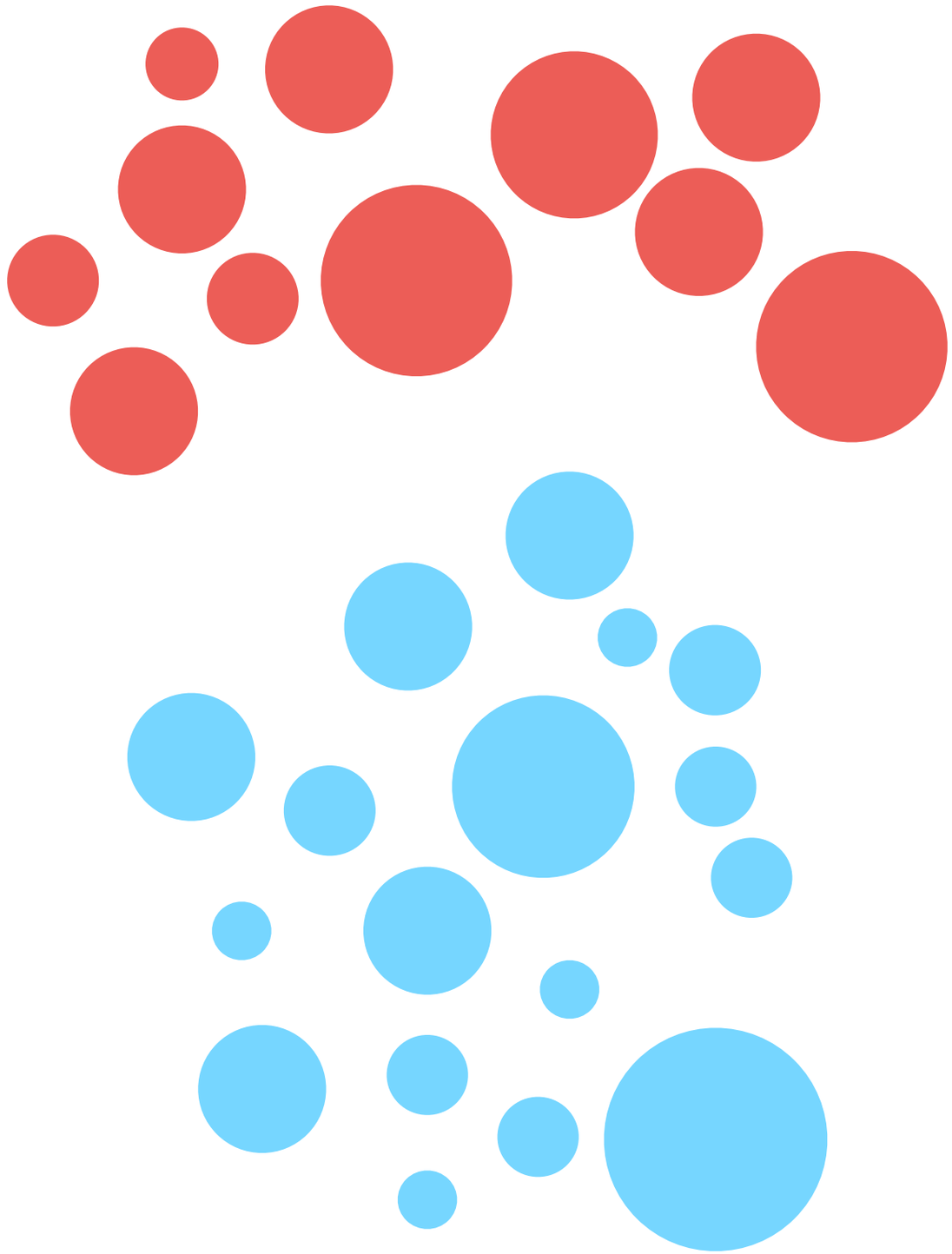
Graph Class or “property”, **π**

Forbidden Minor Characterization

A graph G belongs to **π** if, and only if,
G does not contain any graph from Forb(**π**)
as a **minor**.

A graph admits a **forbidden minor characterization**
if, and only if,
it is **closed under taking minors**.

In fact, there is always a finite forbidden set!
(Robertson-Seymour; Graph Minors Theorem)



Forbidden Minor Characterization

A graph G belongs to **π** if, and only if,
G does not contain any graph from Forb(**π**)
as a **minor**.

A graph admits a **forbidden minor characterization**
if, and only if,
it is **closed under taking minors**.

In fact, there is always a finite forbidden set!
(Robertson-Seymour; Graph Minors Theorem)

Suppose the class **π + (modifications)** is
closed under taking minors…

Graph Class or “property”, **π**



Graph Class or “property”, **π** + (modifications)

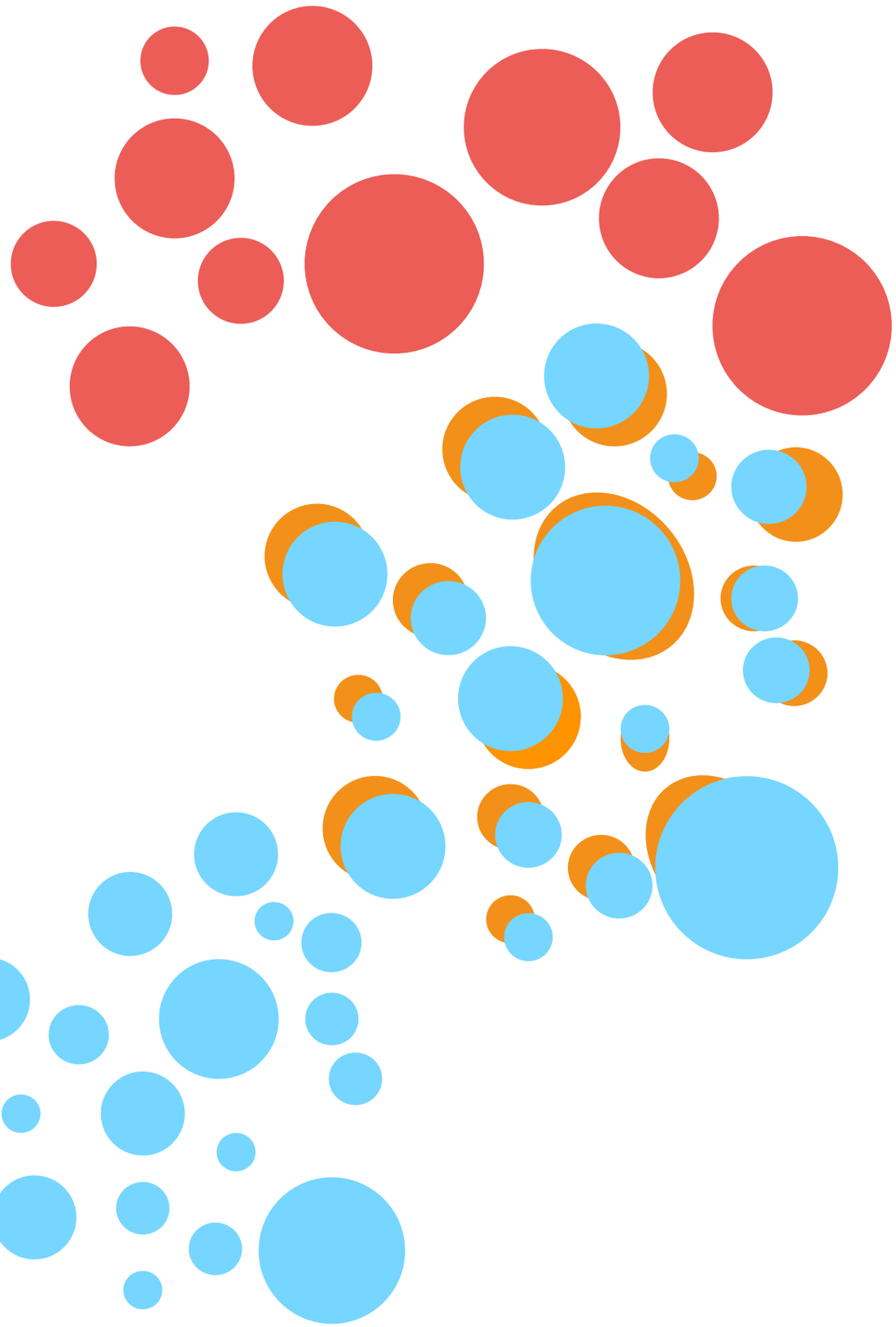
Forbidden Minor Characterization

A graph G belongs to **π** if, and only if,
G does not contain any graph from Forb(**π**)
as a **minor**.

A graph admits a **forbidden minor characterization**
if, and only if,
it is **closed under taking minors**.

In fact, there is always a finite forbidden set!
(Robertson-Seymour; Graph Minors Theorem)

Suppose the class **π + (modifications)** is
closed under taking minors…



Graph Class or “property”, π + (modifications)

Forbidden Minor Characterization

A graph G belongs to π if, and only if,
G does not contain any graph from Forb(π)
as a **minor**.

A graph admits a **forbidden minor characterization**
if, and only if,
it is **closed under taking minors**.

In fact, there is always a finite forbidden set!
(Robertson-Seymour; Graph Minors Theorem)

Suppose the class **π + (modifications)** is
closed under taking minors…

Now: the graph modification problem has boiled down
to a membership testing problem, which can be
done in cubic time, given the finite forbidden set.

Forbidden Subgraph Characterization

A graph admits a **forbidden subgraph characterization**
if, and only if,
it is **closed under taking induced subgraphs (hereditary)**,
that is, the property is preserved under vertex deletions.

Forbidden Minor Characterization

A graph admits a **forbidden minor characterization**
if, and only if,
it is **closed under taking minors**,
that is, the property is preserved under vertex deletions,
edge deletions and edge contractions.

Forbidden Subgraph Characterization

A graph admits a **forbidden subgraph characterization**
if, and only if,
it is **closed under taking induced subgraphs (hereditary)**,
that is, the property is preserved under vertex deletions.

Can the node deletion question for hereditary properties always be answered with the "graph minors hammer"?

Forbidden Minor Characterization

A graph admits a **forbidden minor characterization**
if, and only if,
it is **closed under taking minors**,
that is, the property is preserved under vertex deletions,
edge deletions and edge contractions.

One Size Fits All?

A meta-algorithm for a class of graph-theoretic problems

Define a class of graph modification problems.

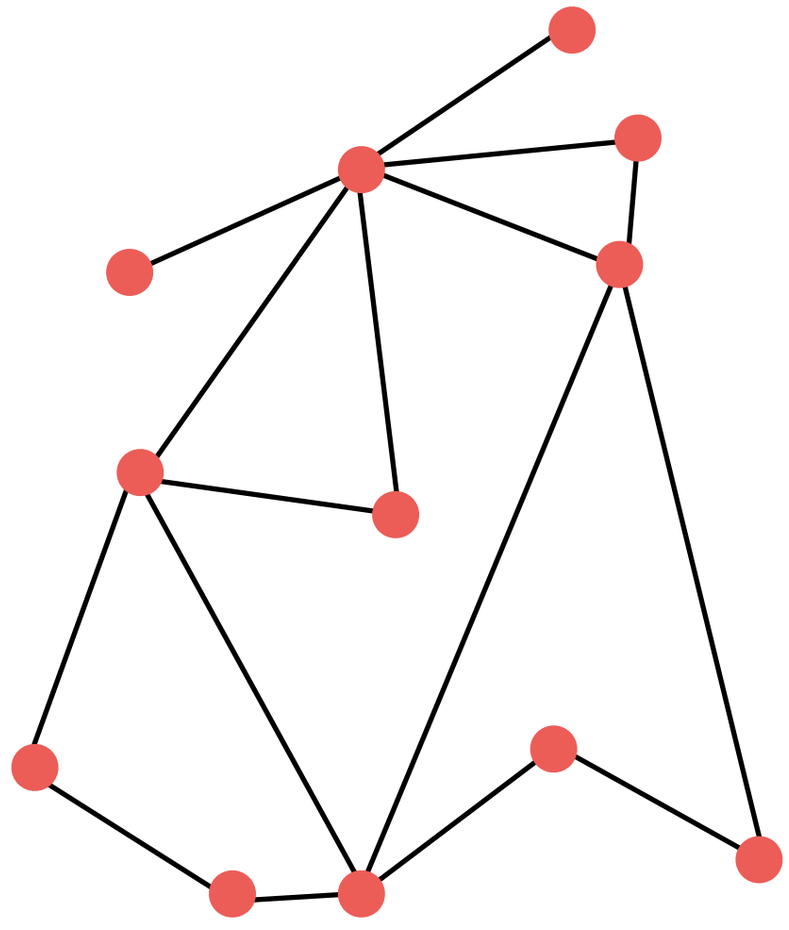
Implications of the Graph Minor Theorem.

Vertex Cover, Revisited.

Beyond Vertex Cover: Feedback Vertex Set
(on graphs of constant degree).

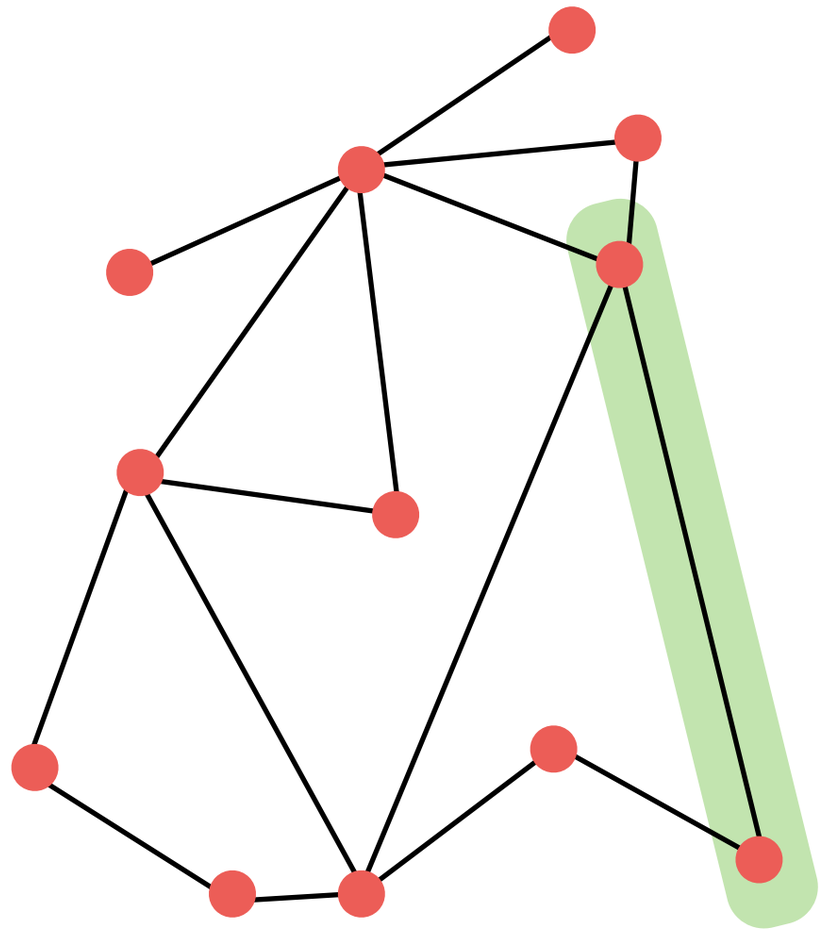
Planar F-Deletion.

Can G be made edgeless by the removal of at most k vertices?



A simple randomized algorithm with an error probability of 2^{-k}.

Can G be made edgeless by the removal of at most k vertices?



A simple randomized algorithm with an error probability of 2^{-k}.

One Size Fits All?

A meta-algorithm for a class of graph-theoretic problems

Define a class of graph modification problems.

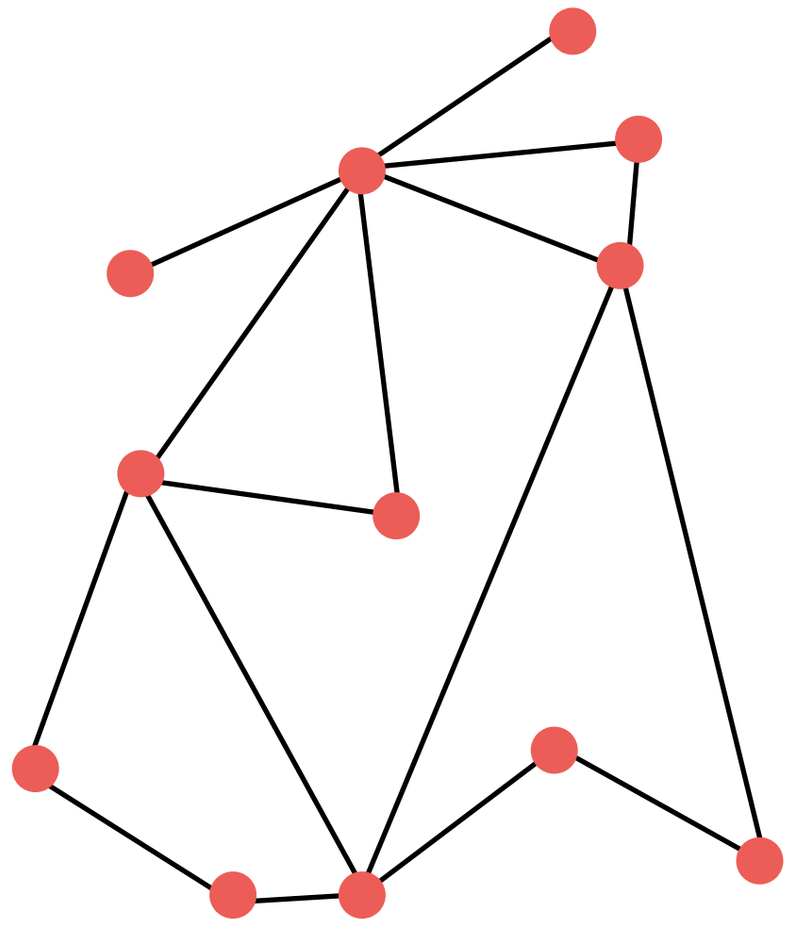
Implications of the Graph Minor Theorem.

Vertex Cover, Revisited.

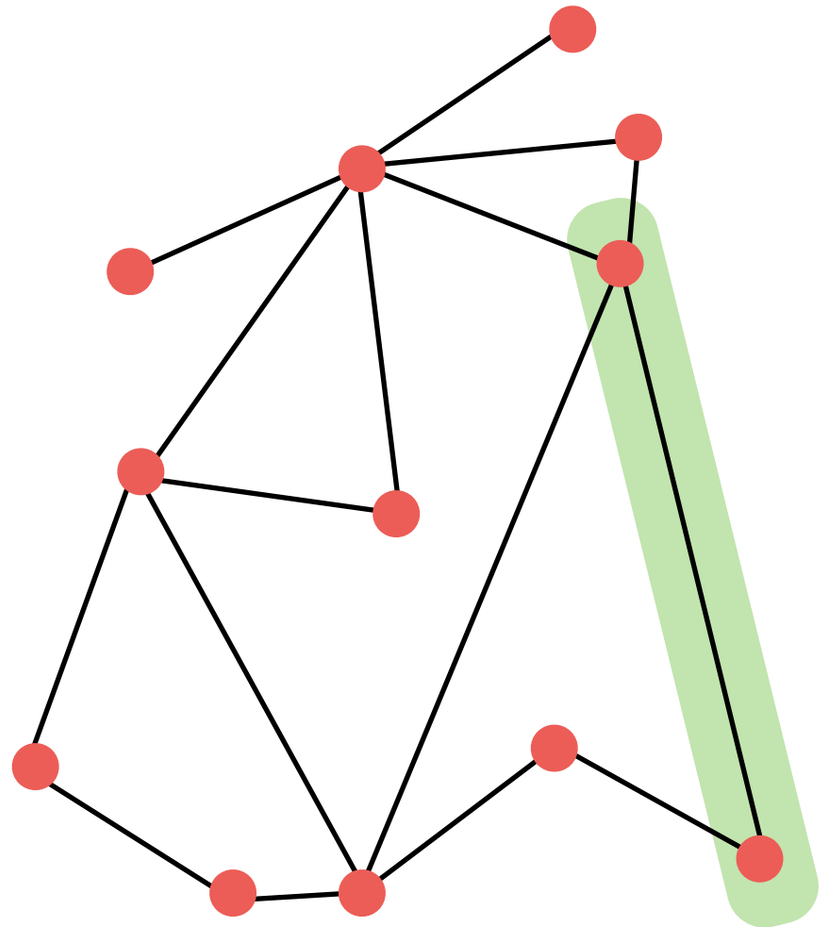
Beyond Vertex Cover: Feedback Vertex Set
(on graphs of constant degree).

Planar F-Deletion.

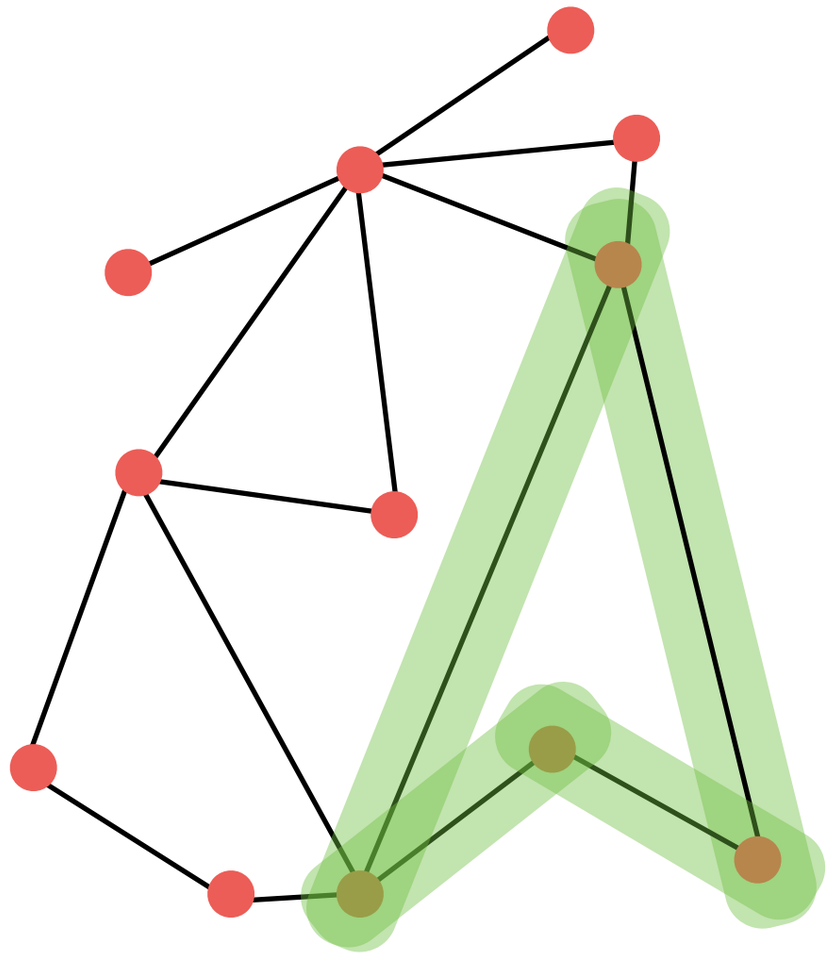
Can G be made acyclic by the removal of at most k vertices?



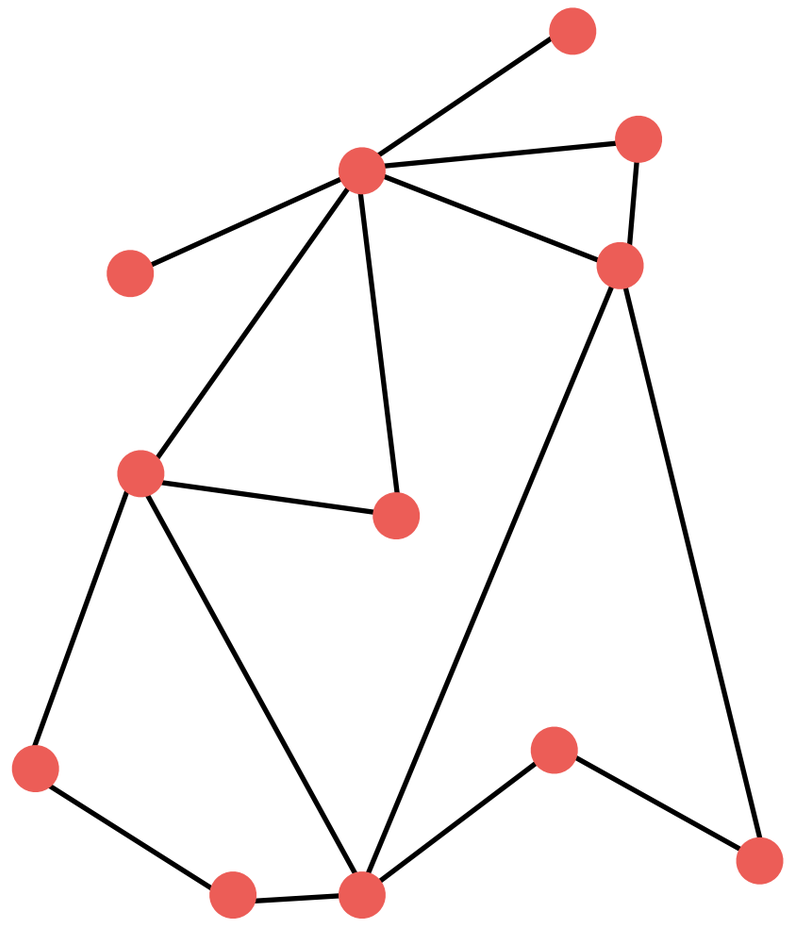
Can G be made acyclic by the removal of at most k vertices?



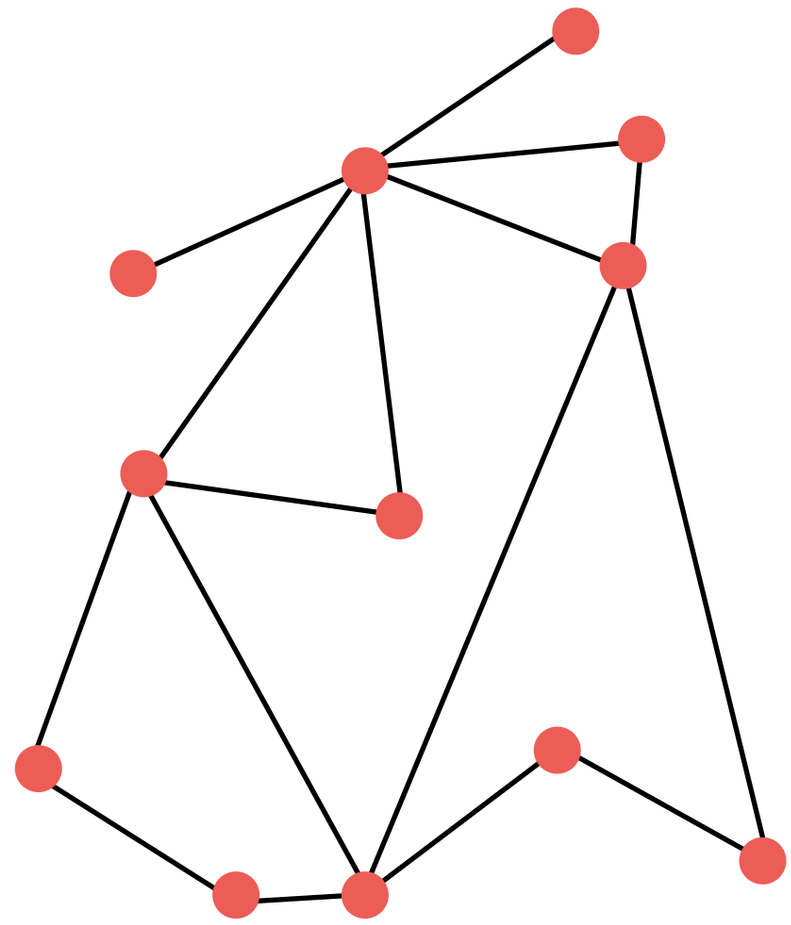
Can G be made acyclic by the removal of at most k vertices?



Can G be made acyclic by the removal of at most k vertices?

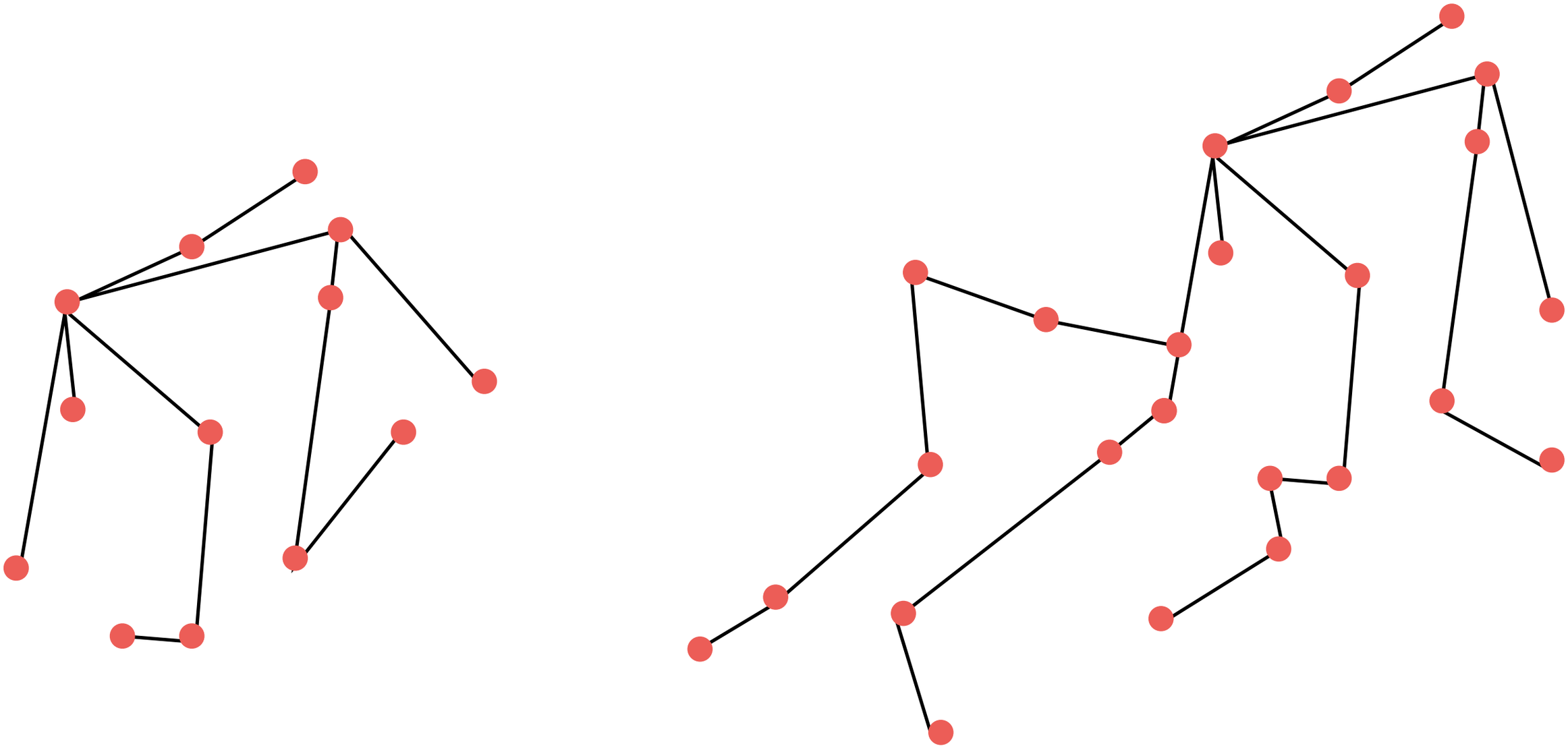


Can G be made acyclic by the removal of at most k vertices?



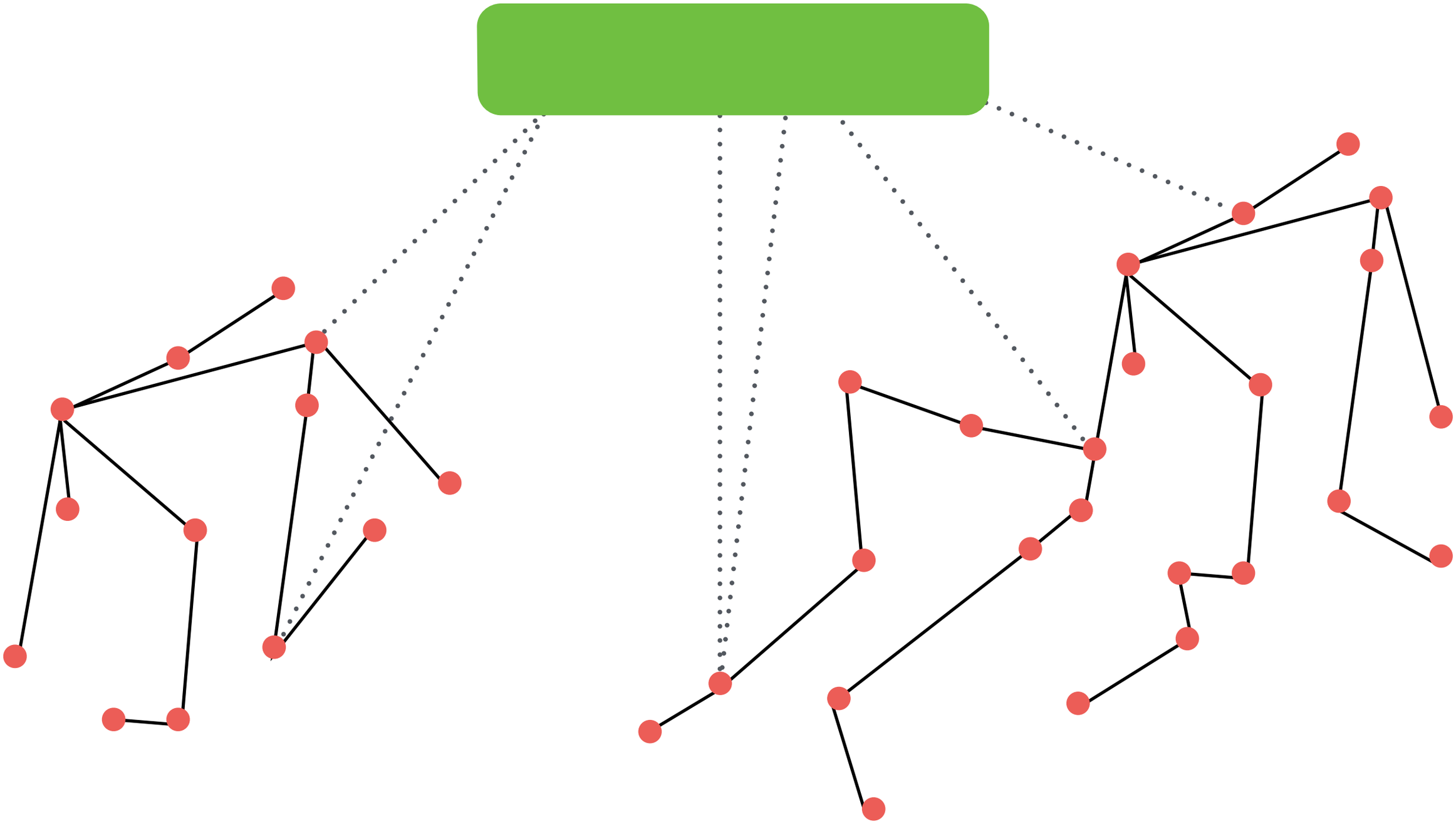
Let us consider the *structure* of a YES-instance.
Let us also restrict ourselves to graphs of constant maximum degree, say five.

The k vertices corresponding to the FVS

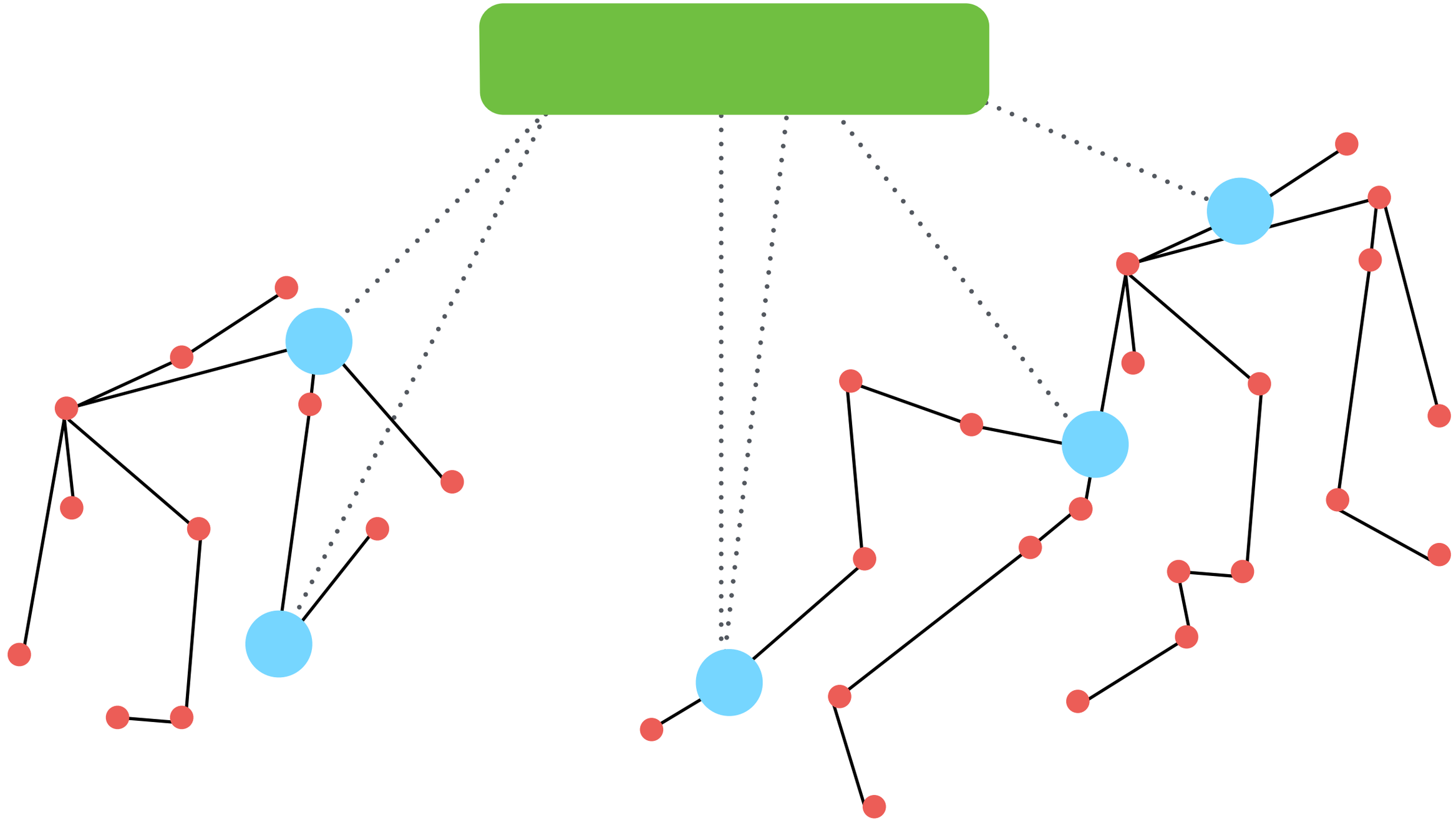


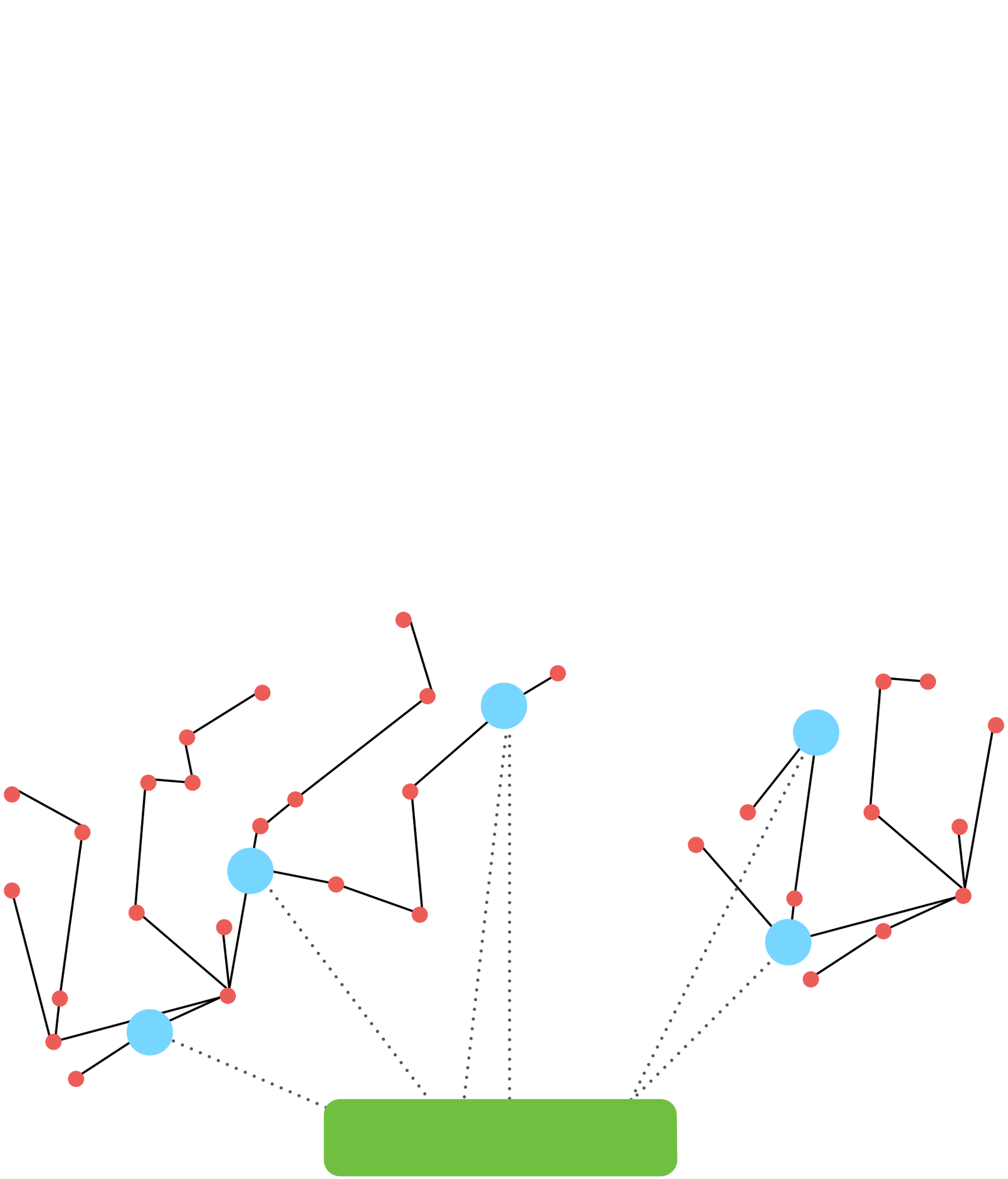
(n-k) vertices and at most (n-k-1) edges “outside”.

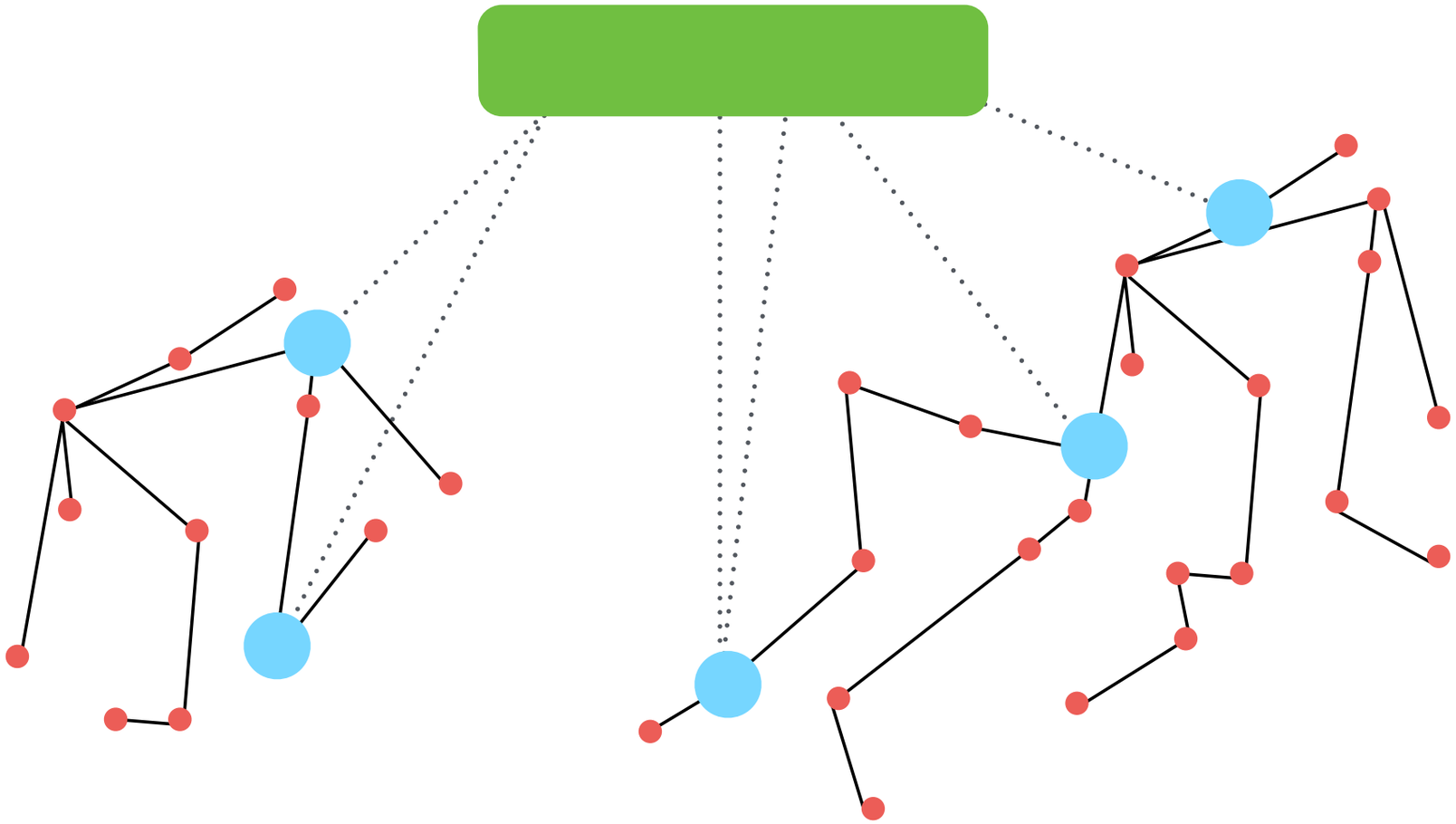
The k vertices corresponding to the FVS



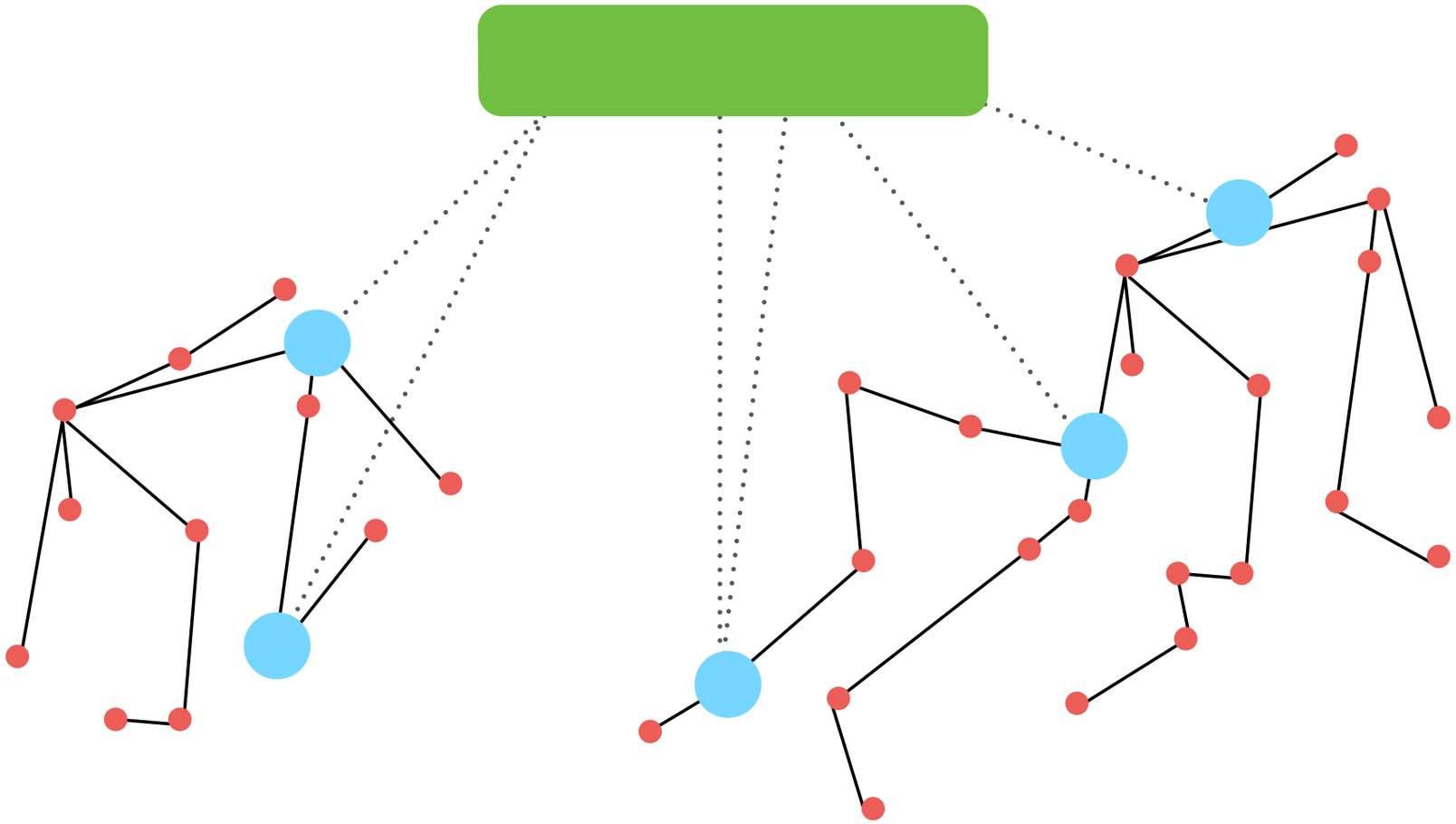
$(n-k)$ vertices and at most $(n-k-1)$ edges “outside”.





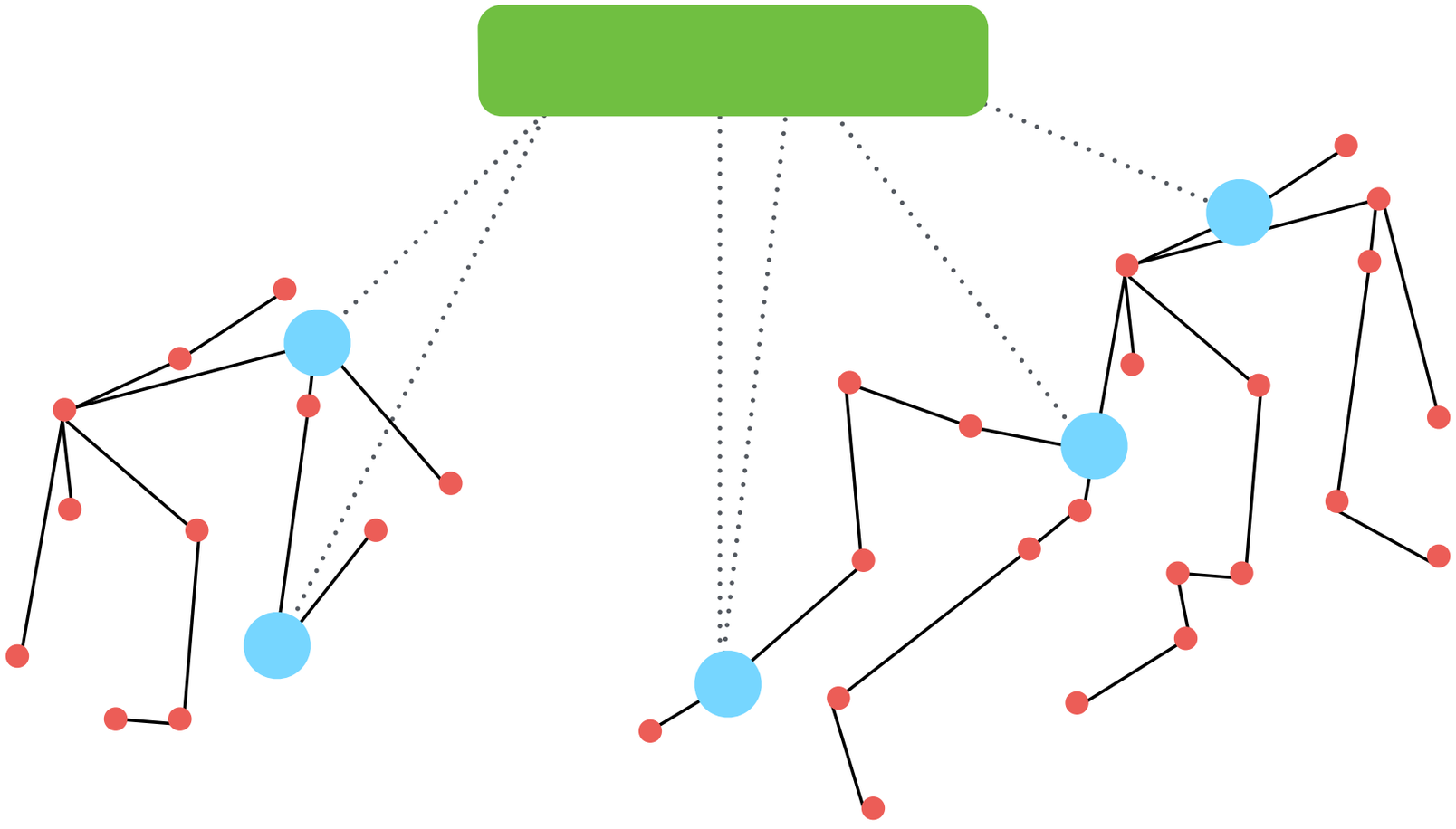


t “points of contact” between S and G\S…



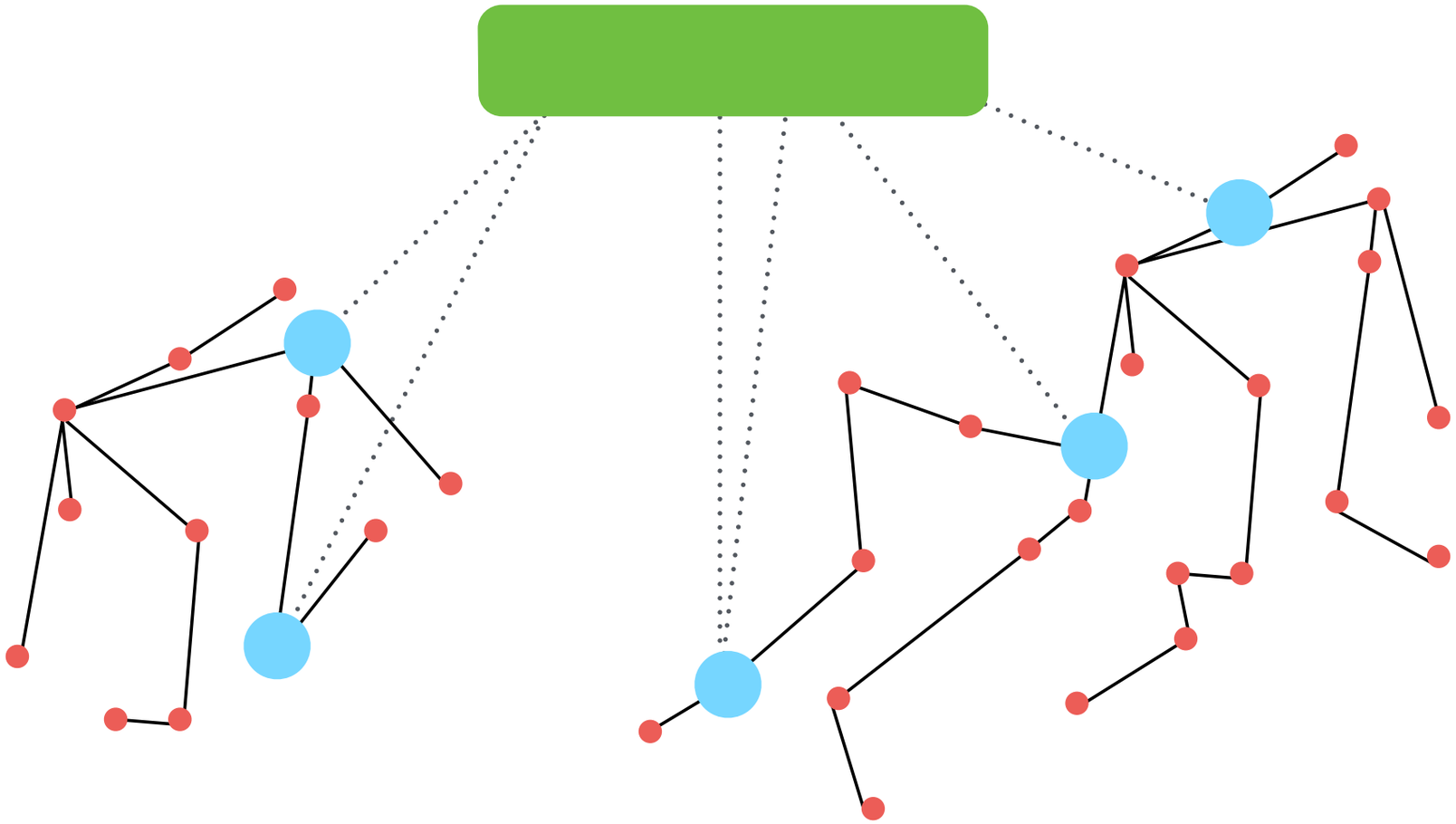
t “points of contact” between S and G\S…

at least t edges going across the two sets.



t “points of contact” between S and G\S… at least t edges going across the two sets.

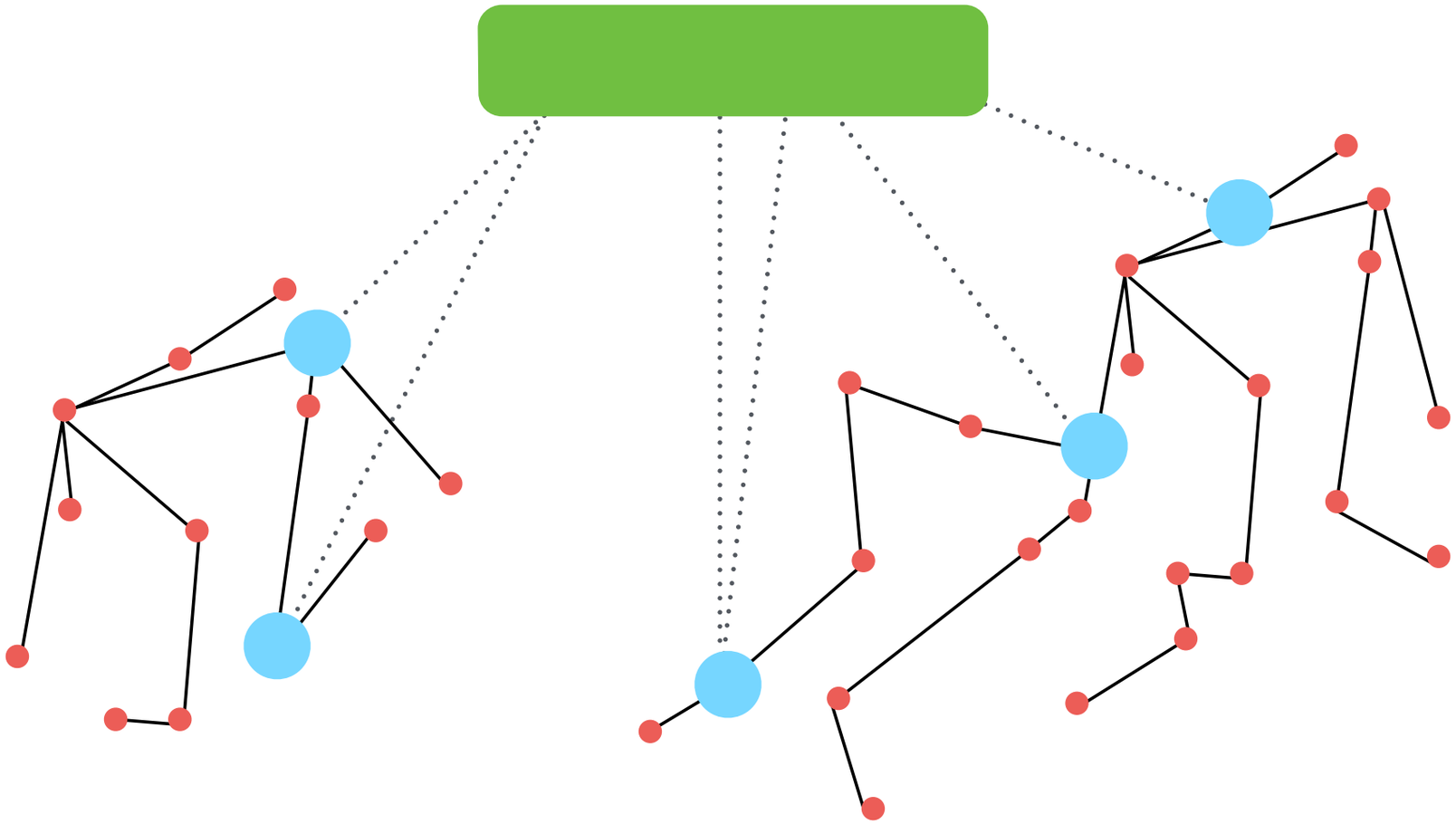
Delete the points of contact in G\S to get at most 5t “pieces”.



t “points of contact” between S and G\S… at least t edges going across the two sets.

Delete the points of contact in G\S to get at most 5t “pieces”.

Suppose each piece has constant size, say c.

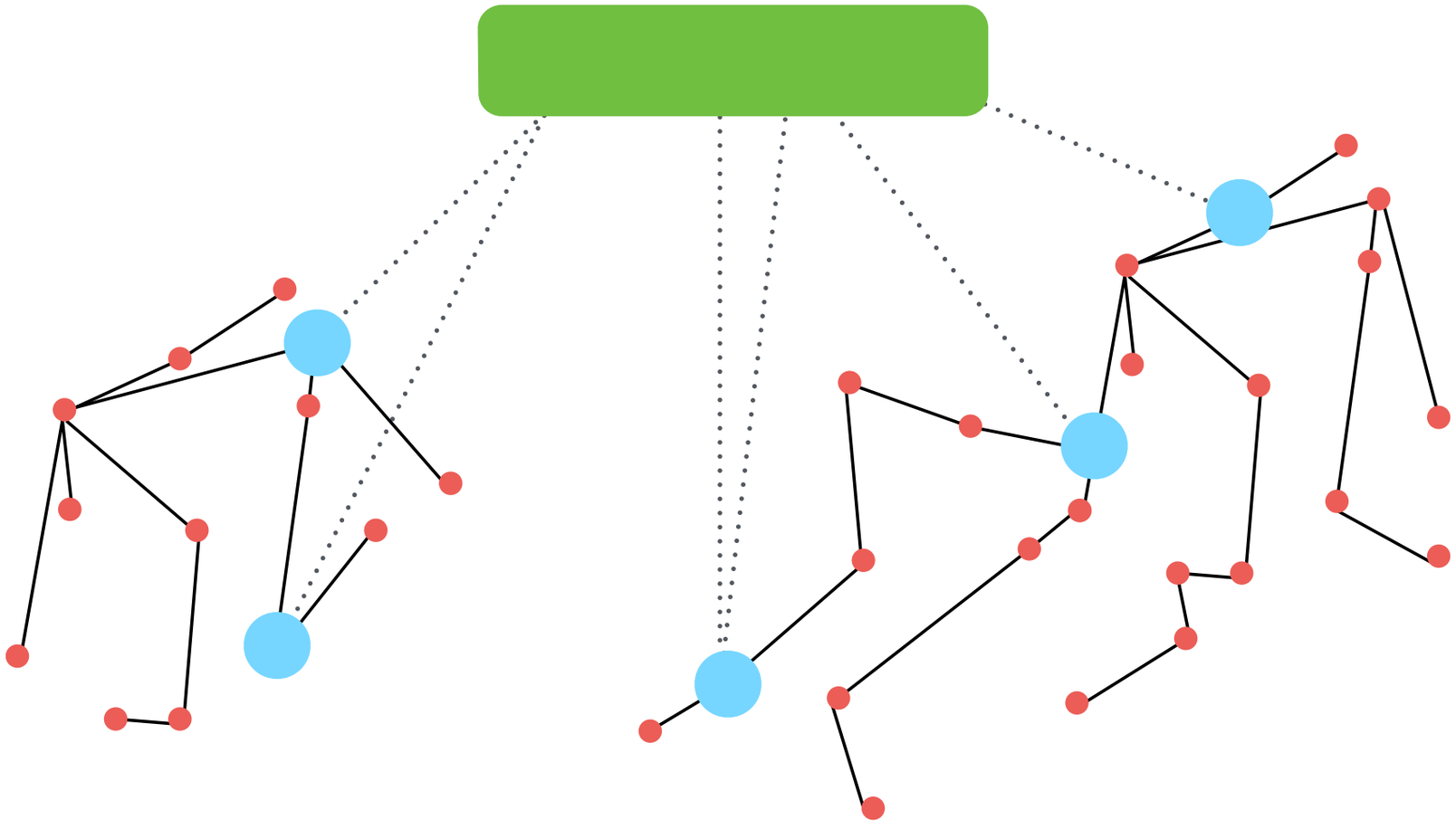


t “points of contact” between S and G\S… at least t edges going across the two sets.

Delete the points of contact in G\S to get at most 5t “pieces”.

Suppose each piece has constant size, say c.

The total number of edges in G\S is at most 5ct.



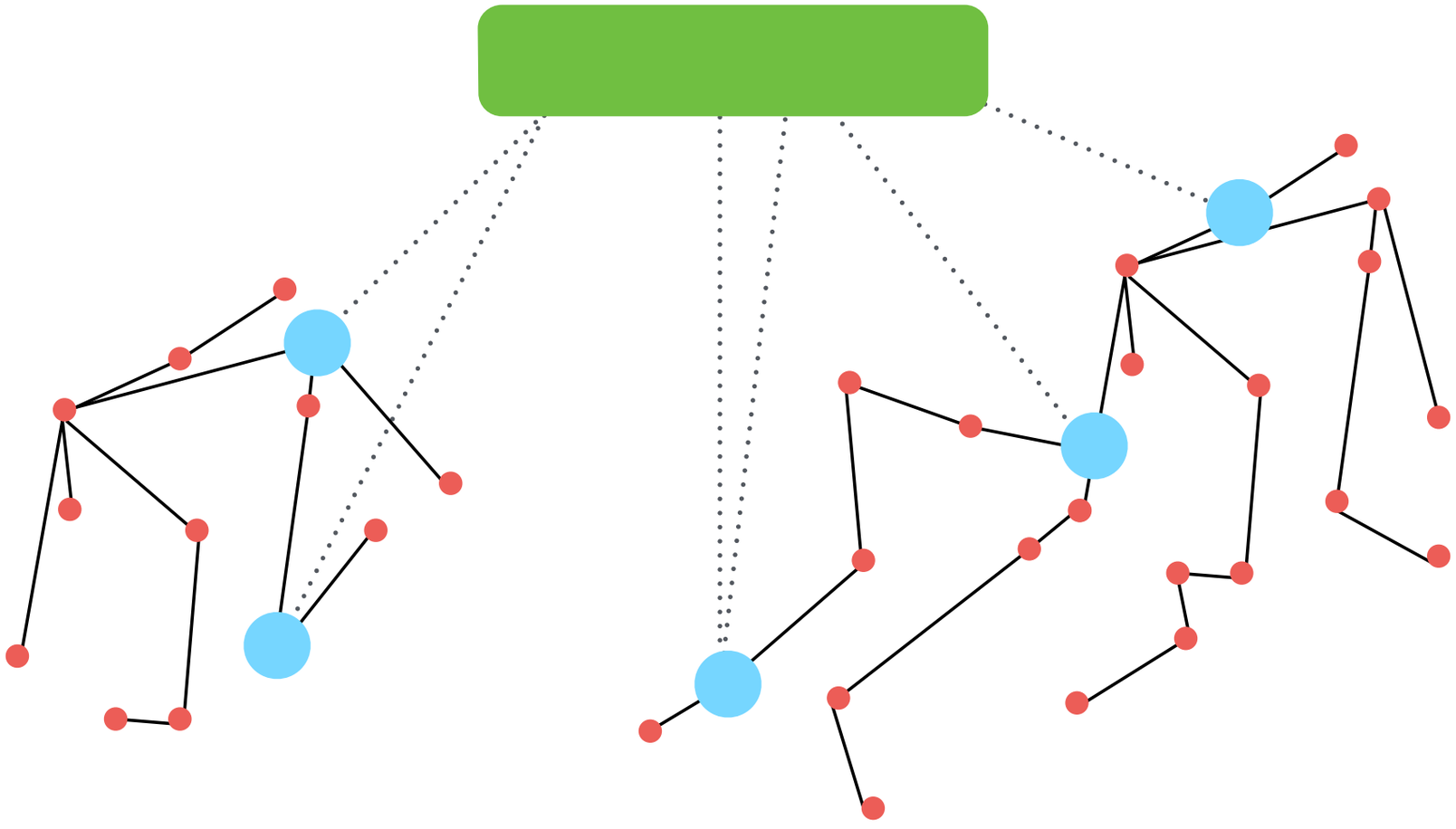
t “points of contact” between S and G\S… at least t edges going across the two sets.

Delete the points of contact in G\S to get at most 5t “pieces”.

Suppose each piece has constant size, say c.

The total number of edges in G\S is at most 5ct.

The number of **good edges is at least t,** and the number of **bad edges is at most 5ct.**



t “points of contact” between S and G\S… at least t edges going across the two sets.

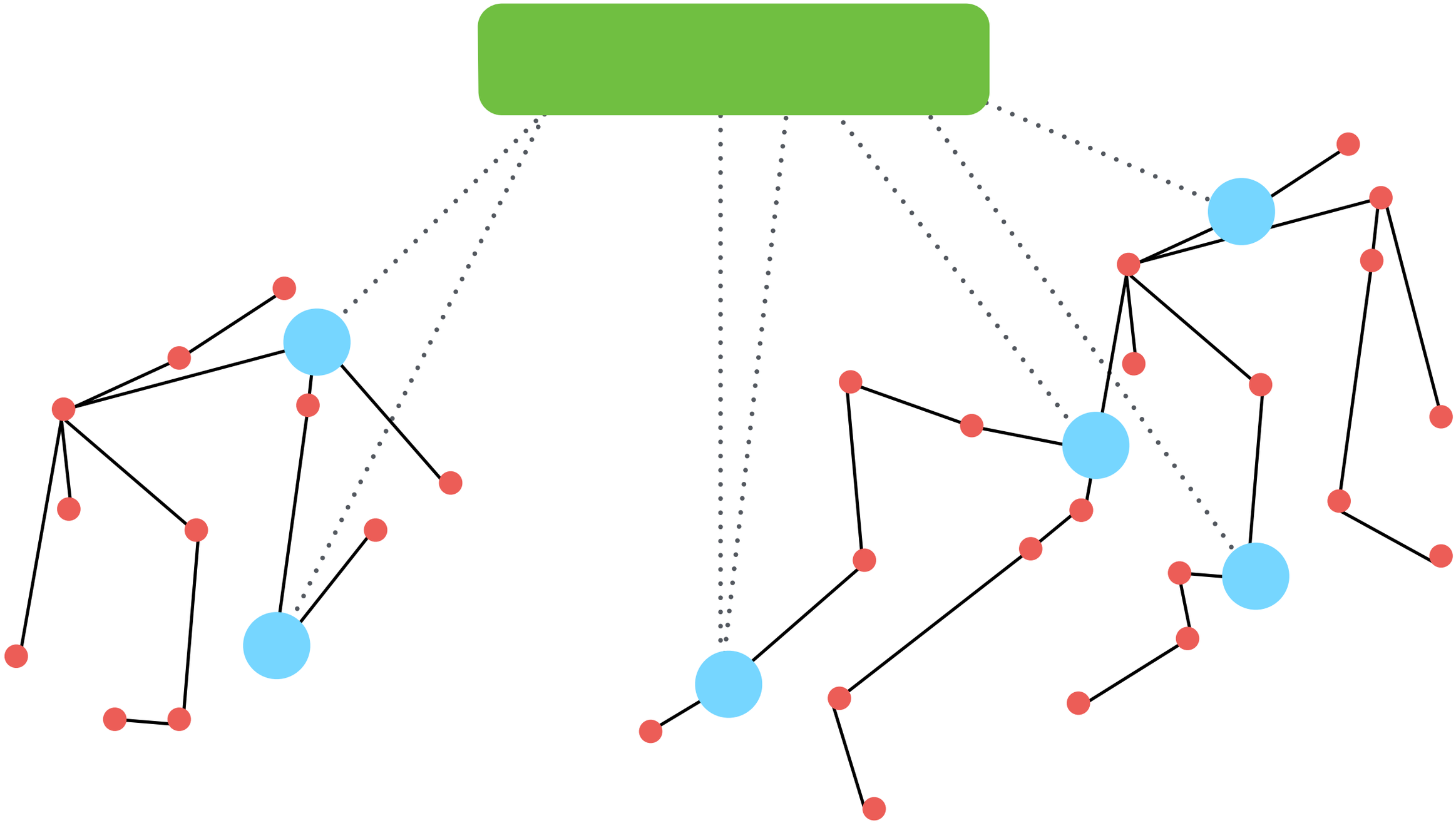
Delete the points of contact in G\S to get at most 5t “pieces”.

Suppose each piece has constant size, say c.

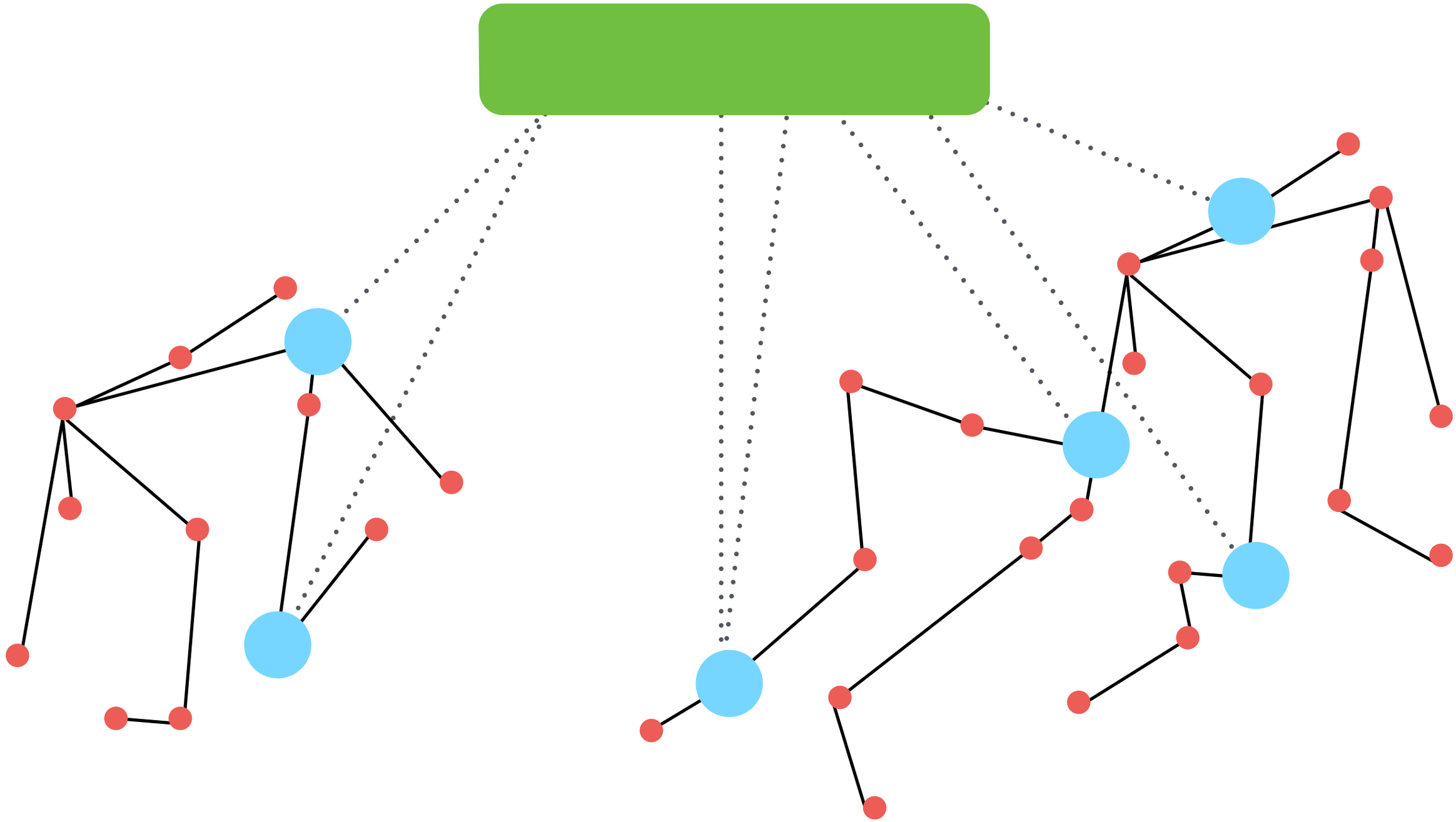
The total number of edges in G\S is at most 5ct.

The number of **good edges is at least t,** and the number of **bad edges is at most 5ct.**

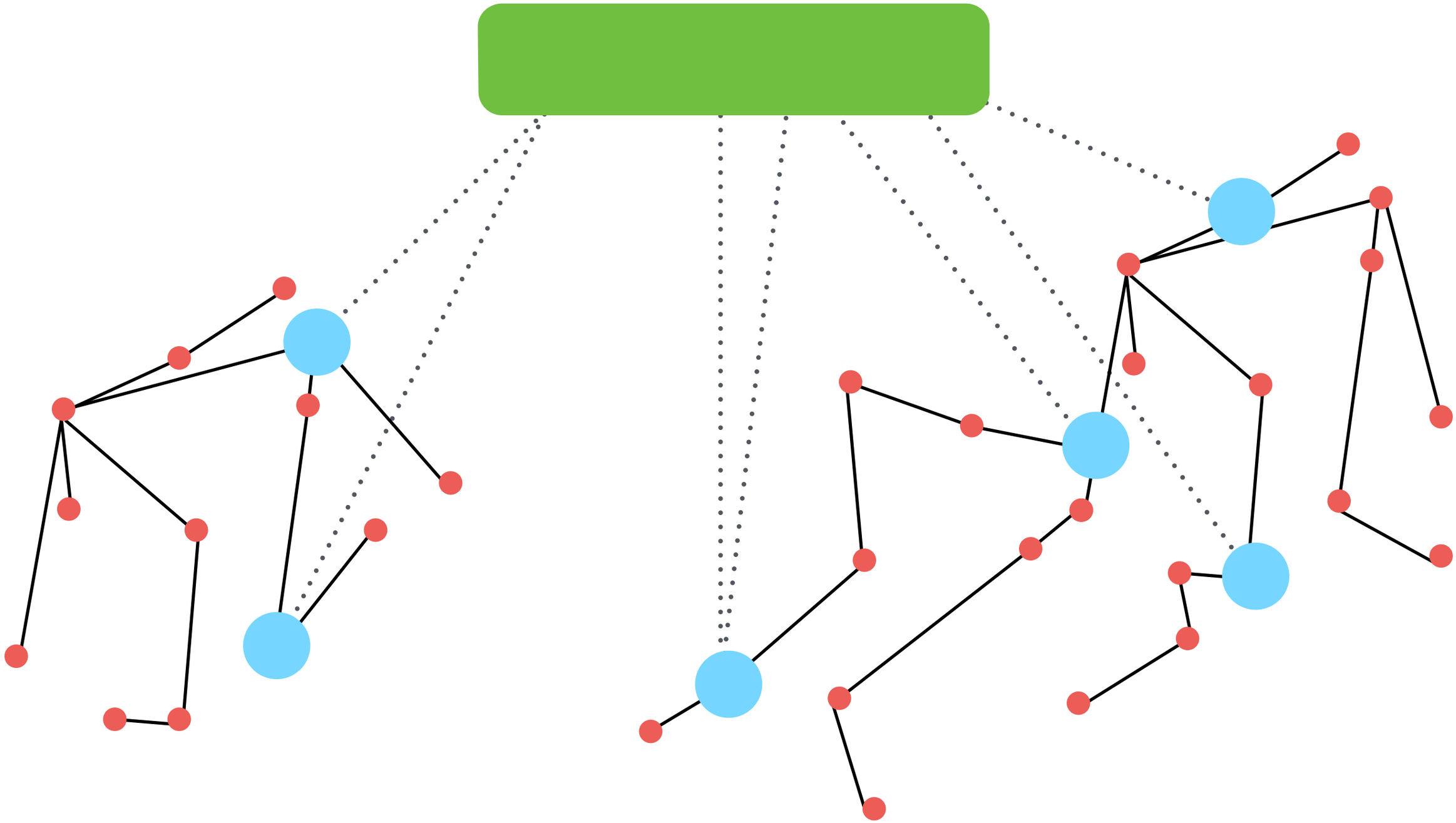
In this scenario, a randomly chosen edge will be **good with constant probability!**



Bad scenario: one of the pieces is large.



These pieces have **simple structure** and **bounded interaction** with the outer world.



These pieces have **simple structure** and **bounded interaction** with the outer world.

Such pieces are called **“protrusions”.**

A Boundary of Constant Size

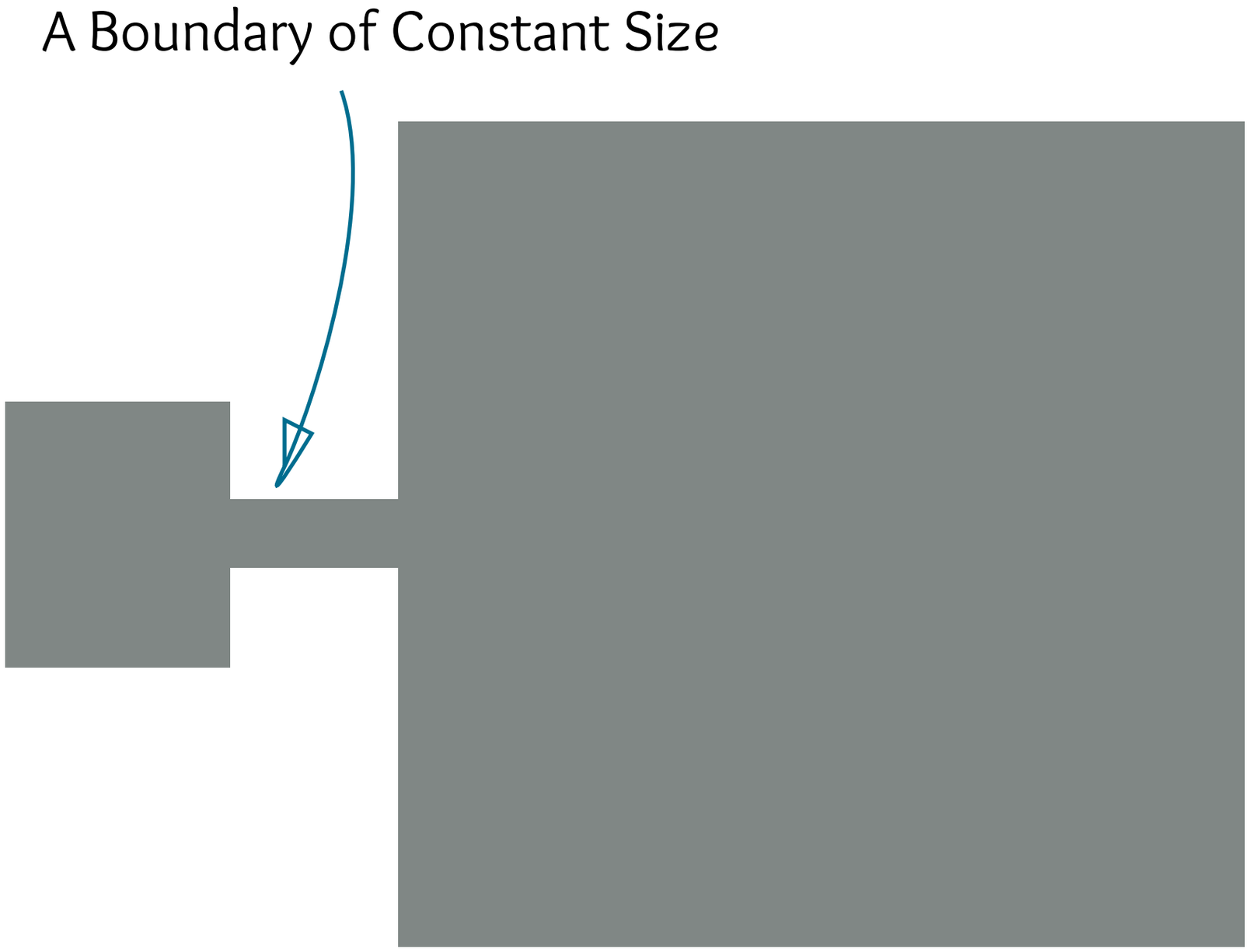


Constant Treewidth

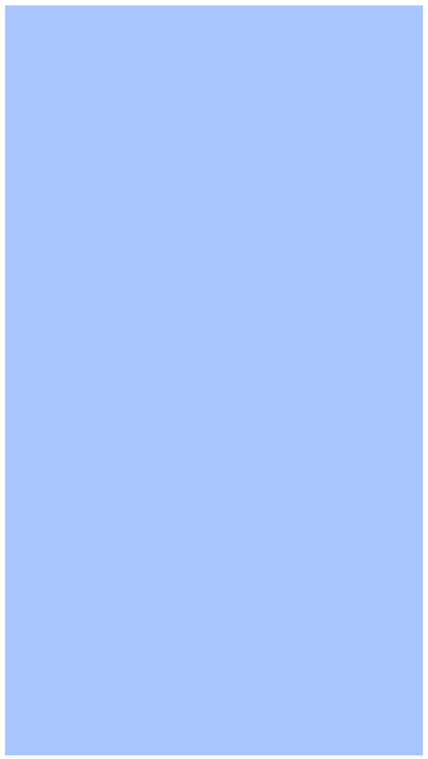
A Boundary of Constant Size

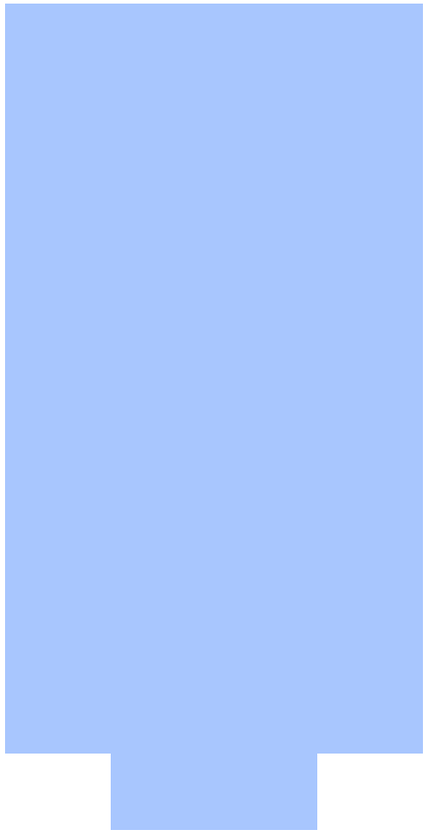


Constant Treewidth

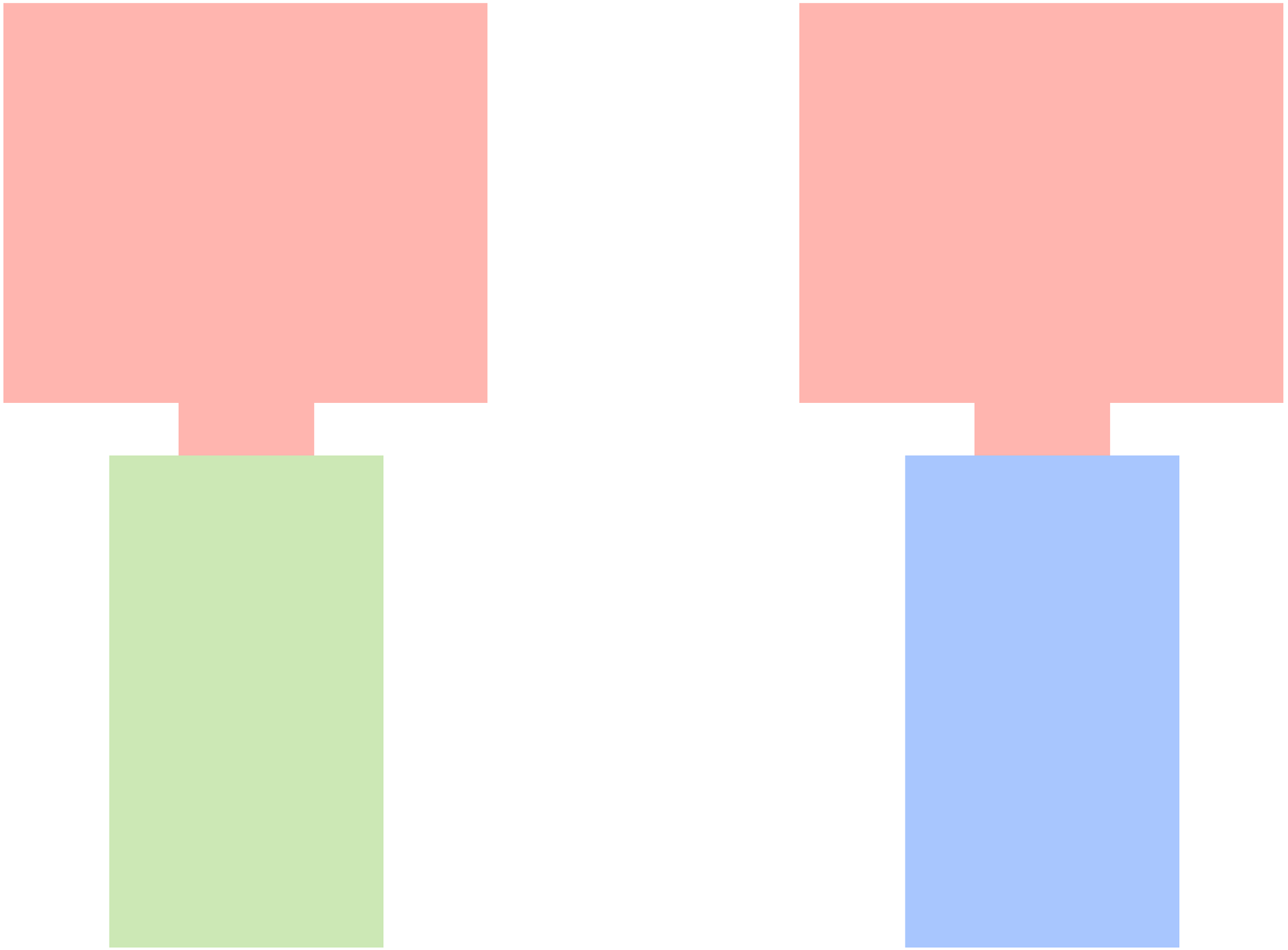


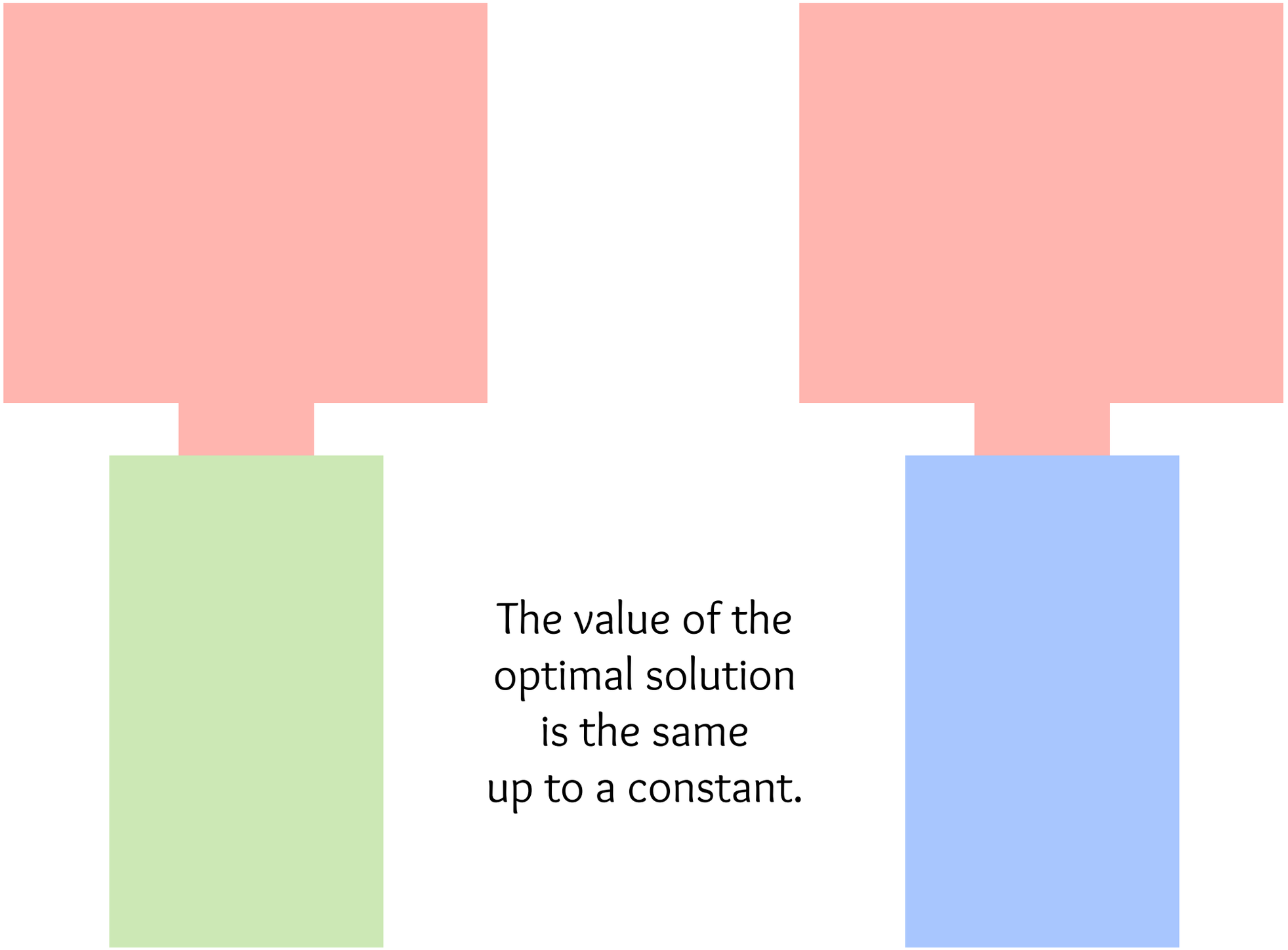
Constant Treewidth





The space of t-boundaried graphs
can be broken up into **equivalence classes**
based on how they “behave” with
the “other side” of the boundary.



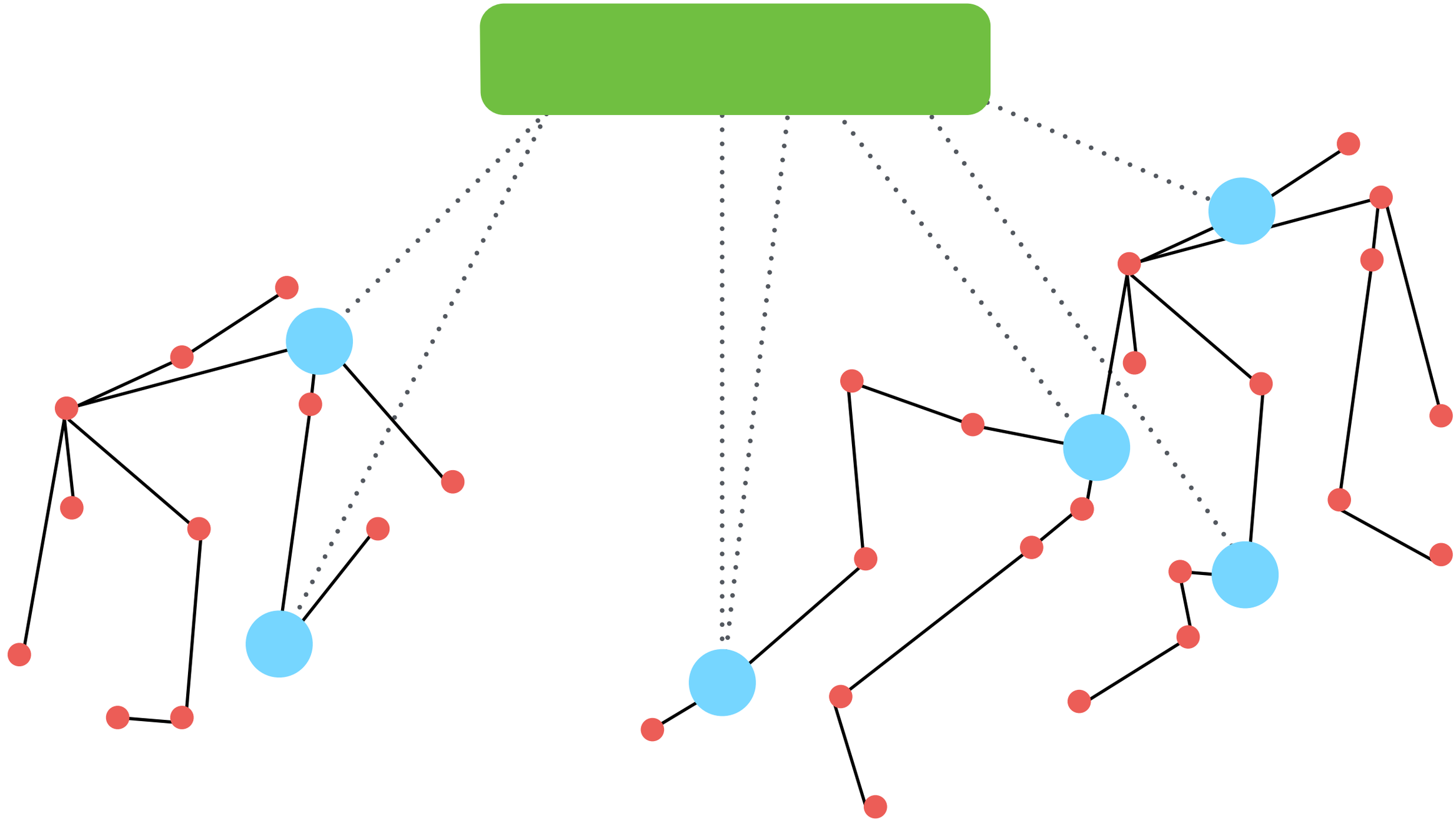


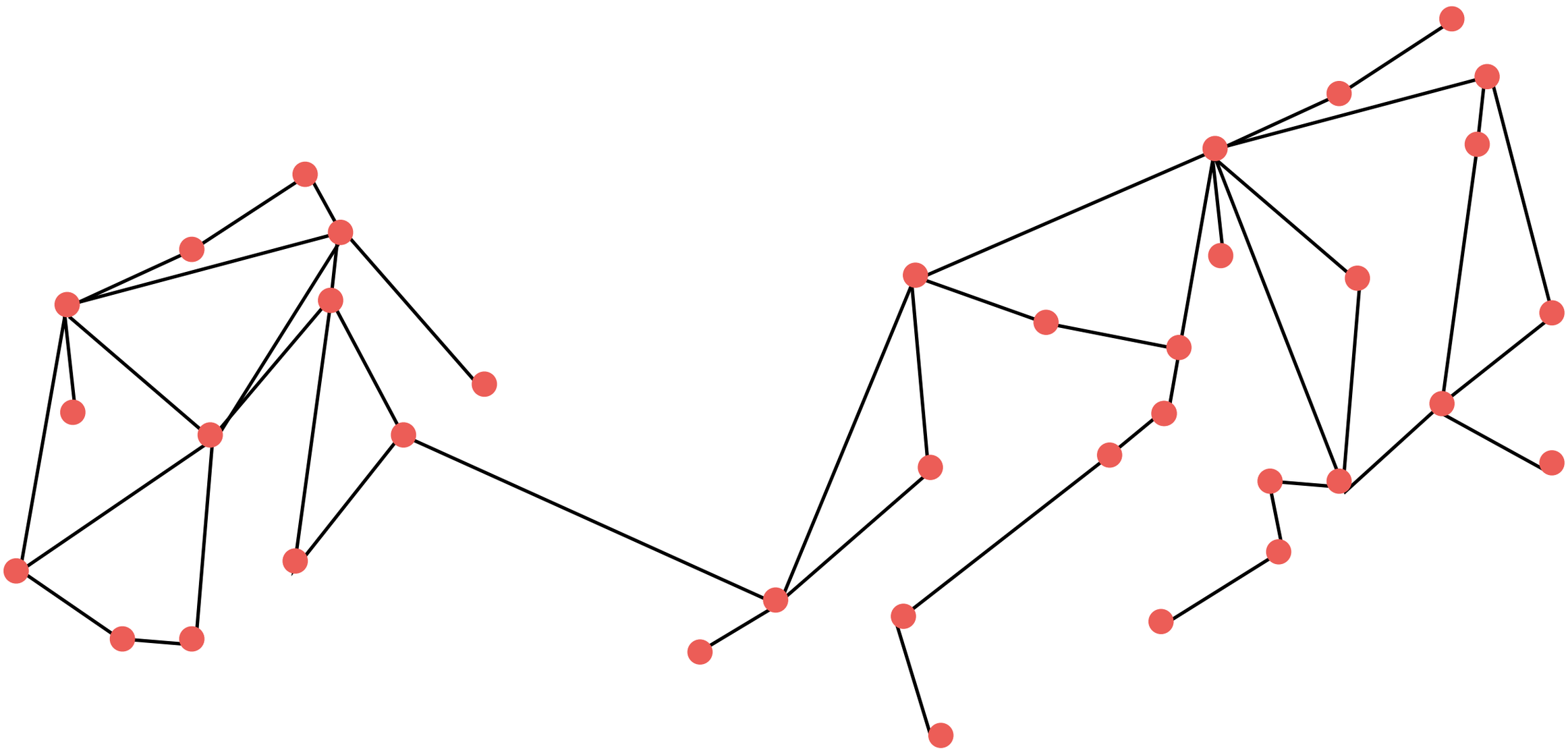
The value of the
optimal solution
is the same
up to a constant.

The space of t-boundaried graphs
can be broken up into **equivalence classes**
based on how they "behave" with
the "other side" of the boundary.

The space of t-boundaried graphs
can be broken up into **equivalence classes**
based on how they "behave" with
the "other side" of the boundary.

For some problems,
the number of equivalence classes is **finite**,
allowing us to replace protrusions in graphs.





Finding protrusions? What do we replace them with?

One Size Fits All?

A meta-algorithm for a class of graph-theoretic problems

Define a class of graph modification problems.

Implications of the Graph Minor Theorem.

Vertex Cover, Revisited.

Beyond Vertex Cover: Feedback Vertex Set
(on graphs of constant degree).

Planar F-Deletion.

Can G be made H-minor free, for all H in $\mathcal{F}$,
by the removal of at most k vertices?

Can G be made H-minor free, for all H in $\mathcal{F}$,
by the removal of at most k vertices?

Suppose G is a **YES-instance**, and further, suppose $\mathcal{F}$ contains **at least one planar graph**.

Can G be made H-minor free, for all H in $\mathcal{F}$,
by the removal of at most k vertices?

Suppose G is a **YES-instance**, and further, suppose $\mathcal{F}$ contains **at least one planar graph**.

Then G\S cannot have large grids.

Can G be made H-minor free, for all H in $\mathcal{F}$,
by the removal of at most k vertices?

Suppose G is a **YES-instance**, and further, suppose $\mathcal{F}$ contains **at least one planar graph**.

Then G\S cannot have large grids.

Thus the treewidth of G\S is **bounded by a constant** that depends only on $\mathcal{F}$.

Can G be made H-minor free, for all H in $\mathcal{F}$,
by the removal of at most k vertices?

Suppose G is a **YES-instance**, and further, suppose $\mathcal{F}$ contains **at least one planar graph**.

Then G\S cannot have large grids.

Thus the treewidth of G\S is **bounded by a constant** that depends only on $\mathcal{F}$.

The Planar $\mathcal{F}$-Deletion Problem

The Planar $\mathcal{F}$-Deletion Problem

The Planar $\mathcal{F}$-Deletion Problem

Ensure that protrusions are removed.

The Planar $\mathcal{F}$-Deletion Problem

Ensure that protrusions are removed.

Pick an edge uniformly at random.

The Planar $\mathcal{F}$-Deletion Problem

Ensure that protrusions are removed.

Pick an edge uniformly at random.

Include both endpoints.

The Planar $\mathcal{F}$-Deletion Problem

Ensure that protrusions are removed.

Pick an edge uniformly at random.

Include both endpoints.

Pick an endpoint u.a.r.

The Planar $\mathcal{F}$-Deletion Problem

Ensure that protrusions are removed.

Pick an edge uniformly at random.

Include both endpoints.

Pick an endpoint u.a.r.

Repeat till G is $\mathcal{F}$-free.

The Planar $\mathcal{F}$-Deletion Problem

Ensure that protrusions are removed.

Pick an edge uniformly at random.

Include both endpoints.

Pick an endpoint u.a.r.

(Approximation algorithm.)

Repeat till G is $\mathcal{F}$-free.

The Planar $\mathcal{F}$-Deletion Problem

Ensure that protrusions are removed.

Pick an edge uniformly at random.

Include both endpoints.

(Approximation algorithm.)

Pick an endpoint u.a.r.

(Randomized algorithm.)

Repeat till G is $\mathcal{F}$-free.

The Planar $\mathcal{F}$-Deletion Problem
accounts for several specific problems…

The Planar $\mathcal{F}$-Deletion Problem
accounts for several specific problems…

$\mathcal{F}$	Corresponding Deletion Problem
K_2	Vertex Cover
K_3	Feedback Vertex Set
((k+1)-by-(k+1)) Grids	Treewidth k
$K_{2,3}$, K_4	Outerplanar
K_4	Series-Parallel
K_3,T_2	Pathwidth-One

Thanks to **Graph Minors**, we have a $f(k)n^2$ algorithm for the $\mathcal{F}$-Deletion problem.

Thanks to **Graph Minors**, we have a $f(k)n^2$ algorithm for the $\mathcal{F}$-Deletion problem.

For **Planar** $\mathcal{F}$-Deletion, the starting point was a double-exponential algorithm.
[Bodlaender, 1997]

Thanks to **Graph Minors**, we have a $f(k)n^2$ algorithm for the $\mathcal{F}$-Deletion problem.

For **Planar** $\mathcal{F}$-Deletion, the starting point was a double-exponential algorithm.
[Bodlaender, 1997]

Many single-exponential algorithms are known for special cases of $\mathcal{F}$.

Thanks to **Graph Minors**, we have a $f(k)n^2$ algorithm for the $\mathcal{F}$-Deletion problem.

For **Planar** $\mathcal{F}$-Deletion, the starting point was a double-exponential algorithm.
[Bodlaender, 1997]

Many single-exponential algorithms are known for special cases of $\mathcal{F}$.

Cao, Chen & Liu
[2010]

Feedback Vertex Set has a $O^*(3.83^k)$ algorithm.

Thanks to **Graph Minors**, we have a $f(k)n^2$ algorithm for the $\mathcal{F}$-Deletion problem.

For **Planar** $\mathcal{F}$-Deletion, the starting point was a double-exponential algorithm.
[Bodlaender, 1997]

Many single-exponential algorithms are known for special cases of $\mathcal{F}$.

Cao, Chen & Liu [2010]

Feedback Vertex Set has a $O^*(3.83^k)$ algorithm.

Kim, Paul & Philip [2012]

$\{K_4\}$-Deletion has a $O^*(2^{O(k)})$ algorithm.

Thanks to **Graph Minors**, we have a $f(k)n^2$ algorithm for the $\mathcal{F}$-Deletion problem.

For **Planar** $\mathcal{F}$-Deletion, the starting point was a double-exponential algorithm.
[Bodlaender, 1997]

Many single-exponential algorithms are known for special cases of $\mathcal{F}$.

Cao, Chen & Liu [2010]	**Feedback Vertex Set** has a $O^*(3.83^k)$ algorithm.
Kim, Paul & Philip [2012]	**$\{K_4\}$-Deletion** has a $O^*(2^{O(k)})$ algorithm.
Cygan, Pilipczuk, Pilipczuk & Wojtaszczyk [2010]	**Pathwidth-1-Deletion** has a $O^*(4.65^k)$ algorithm.

Fomin, Lokshtanov, M. & Saurabh [FOCS; 2013]

Planar $\mathcal{F}$-Deletion admits a randomized $(2^{O(k)}n)$ algorithm when every graph in $\mathcal{F}$ is connected.

Fomin, Lokshtanov, M. & Saurabh [FOCS; 2013]

Planar $\mathcal{F}$-Deletion admits a randomized $(2^{O(k)}n)$ algorithm when every graph in $\mathcal{F}$ is connected.

Best possible under ETH.

Fomin, Lokshtanov, M. & Saurabh [FOCS; 2013]

Planar $\mathcal{F}$-Deletion admits a randomized $(2^{O(k)}n)$ algorithm when every graph in $\mathcal{F}$ is connected.

Best possible under ETH.

Planar $\mathcal{F}$-Deletion admits a deterministic $(2^{O(k)}n \log^2 n)$ algorithm when every graph in $\mathcal{F}$ is connected.

Fomin, Lokshtanov, M. & Saurabh [FOCS; 2013]

Planar $\mathcal{F}$-Deletion admits a randomized $(2^{O(k)}n)$ algorithm when every graph in $\mathcal{F}$ is connected.

Best possible under ETH.

Planar $\mathcal{F}$-Deletion admits a deterministic $(2^{O(k)}n \log^2 n)$ algorithm when every graph in $\mathcal{F}$ is connected.

Planar $\mathcal{F}$-Deletion admits an O(nm) randomized algorithm that leads us to a constant-factor approximation.

Can we get a deterministic constant-factor approximation algorithm?

Fomin, Lokshtanov, M. & Saurabh [FOCS; 2013]

Planar $\mathcal{F}$-Deletion admits a randomized $(2^{O(k)}n)$ algorithm when every graph in $\mathcal{F}$ is connected.

Best possible under ETH.

Planar $\mathcal{F}$-Deletion admits a deterministic $(2^{O(k)}n \log^2 n)$ algorithm when every graph in $\mathcal{F}$ is connected.

Planar $\mathcal{F}$-Deletion admits an O(nm) randomized algorithm that leads us to a **constant-factor approximation**.

Can we get a deterministic constant-factor approximation algorithm?

Fomin, Lokshtanov, M., Philip & Saurabh [FOCS; 2013]

Planar $\mathcal{F}$-Deletion admits an deterministic algorithm that leads us to a $O(\log^{3/2}(OPT))$ approximation.

Fomin, Lokshtanov, M. & Saurabh [FOCS; 2013]

Planar $\mathcal{F}$-Deletion admits a randomized $(2^{O(k)}n)$ algorithm when every graph in $\mathcal{F}$ is connected.

Best possible under ETH.

Planar $\mathcal{F}$-Deletion admits a deterministic $(2^{O(k)}n \log^2 n)$ algorithm when every graph in $\mathcal{F}$ is connected.

Planar $\mathcal{F}$-Deletion admits an O(nm) randomized algorithm that leads us to a constant-factor approximation.

Can we get a deterministic constant-factor approximation algorithm?

Fomin, Lokshtanov, M., Philip & Saurabh [FOCS; 2013]

Planar $\mathcal{F}$-Deletion admits an deterministic algorithm that leads us to a $O(\log^{3/2}(OPT))$ approximation.

Kim, Langer, Paul, Reidl, Rossmanith, Sau, & Sikdar [ICALP; 2013]

Planar $\mathcal{F}$-Deletion admits a deterministic $(2^{O(k)}n^2)$ algorithm.

Fomin, Lokshtanov, M. & Saurabh [FOCS; 2013]

Planar $\mathcal{F}$-Deletion admits a randomized $(2^{O(k)}n)$ algorithm when every graph in $\mathcal{F}$ is connected.

Best possible under ETH.

Fomin, Lokshtanov, M., Ramanujan & Saurabh [SODA; 2015]

Planar $\mathcal{F}$-Deletion admits a deterministic $(2^{O(k)}n)$ algorithm when every graph in $\mathcal{F}$ is connected.

Planar $\mathcal{F}$-Deletion admits an O(nm) randomized algorithm that leads us to a **constant-factor approximation**.

Can we get a deterministic constant-factor approximation algorithm?

Fomin, Lokshtanov, M., Philip & Saurabh [FOCS; 2013]

Planar $\mathcal{F}$-Deletion admits an deterministic algorithm that leads us to a $O(\log^{3/2}(OPT))$ approximation.

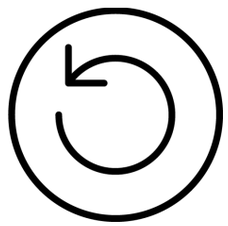
Kim, Langer, Paul, Reidl, Rossmanith, Sau, & Sikdar [ICALP; 2013]

Planar $\mathcal{F}$-Deletion admits a deterministic $(2^{O(k)}n^2)$ algorithm.

Algorithms for Planar F-Deletion and Satisfiability

Beyond Graphs

Efficient SAT algorithms using backdoors



Beyond Graphs

Efficient SAT algorithms using backdoors

Introduction: The curious success of SAT Solvers

Backdoors and Base Classes

Extending tractable classes using Backdoors

Solving SAT using backdoors
(but without finding them!)

Beyond uniform backdoors

Beyond Graphs

Efficient SAT algorithms using backdoors

Introduction: The curious success of SAT Solvers

Backdoors and Base Classes

Extending tractable classes using Backdoors

Solving SAT using backdoors
(but without finding them!)

Beyond uniform backdoors

Goal: Given a propositional formula F in CNF, over a set of variables X, determine an assignment of X that satisfies F.

Goal: Given a propositional formula F in CNF, over a set of variables X, determine an assignment of X that satisfies F.

Among the oldest problems shown to be NP-complete.

Goal: Given a propositional formula F in CNF, over a set of variables X, determine an assignment of X that satisfies F.

Among the oldest problems shown to be NP-complete.

Several applications— problems like software model checking, chip verification etc. reduced to SAT and solved using SAT solvers.

Goal: Given a propositional formula F in CNF, over a set of variables X, determine an assignment of X that satisfies F.

Among the oldest problems shown to be NP-complete.

Several applications— problems like software model checking, chip verification etc. reduced to SAT and solved using SAT solvers.

Resulting instances often have the order of a million variables and several million clauses.

Goal: Given a propositional formula F in CNF, over a set of variables X, determine an assignment of X that satisfies F.

Among the oldest problems shown to be NP-complete.

Several applications— problems like software model checking, chip verification etc. reduced to SAT and solved using SAT solvers.

Resulting instances often have the order of a million variables and several million clauses.

In stark contrast, the best-known worst-case bounds are entirely infeasible, even on instances with only hundreds of variables.

Goal: Given a propositional formula F in CNF, over a set of variables X, determine an assignment of X that satisfies F.

Is there a theoretical explanation for this disparity?

SAT Solvers

SAT Solvers

Unit Propogation. Force all inevitable assignments.

SAT Solvers

Unit Propogation. Force all inevitable assignments.

$(u \vee z)(y\ z)\ (\overline{z}\ w\ y)\ (x)\ (\overline{x}\ \overline{y})$

SAT Solvers

Unit Propogation. Force all inevitable assignments.

$$(u \vee z)(y\ z)\ (\overline{z}\ w\ y)\ (x)\ (\overline{x}\ \overline{y})$$

SAT Solvers

Unit Propogation. Force all inevitable assignments.

$$(u \vee z)(y\ z)\ (\overline{z}\ w\ y)\ (x)\ (\overline{x}\ \overline{y})$$

$$(u \vee z)\ (y\ z)\ (\overline{z}\ w\ y)\ (\overline{y})$$

SAT Solvers

Unit Propogation. Force all inevitable assignments.

$$(u \vee v \vee z)(y \vee z)\ (\overline{z} \vee w \vee y)\ (x)\ (\overline{x} \vee \overline{y})$$

$$(u \vee v \vee z)\ (y \vee z)\ (\overline{z} \vee w \vee y)\ (\overline{y})$$

SAT Solvers

Unit Propogation. Force all inevitable assignments.

$$(u \vee z)(y\ z)\ (\overline{z}\ w\ y)\ (x)\ (\overline{x}\ \overline{y})$$

$$(u \vee z)\ (y\ z)\ (\overline{z}\ w\ y)\ (\overline{y})$$

$$(u \vee z)\ (z)\ (\overline{z}\ w)$$

SAT Solvers

Unit Propogation. Force all inevitable assignments.

$$(u \vee z)(y\ z)\ (\overline{z}\ w\ y)\ (x)\ (\overline{x}\ \overline{y})$$

$$(u \vee z)\ (y\ z)\ (\overline{z}\ w\ y)\ (\overline{y})$$

$$(u \vee z)\ (z)\ (\overline{z}\ w)$$

SAT Solvers

Unit Propogation. Force all inevitable assignments.

$$(u \vee z)(y\ z)\ (\overline{z}\ w\ y)\ (x)\ (\overline{x}\ \overline{y})$$

$$(u \vee z)\ (y\ z)\ (\overline{z}\ w\ y)\ (\overline{y})$$

$$(u \vee z)\ (z)\ (\overline{z}\ w)$$

$$(w)$$

SAT Solvers

Unit Propogation. Force all inevitable assignments.

$$(u \vee z)(y\ z)\ (\overline{z}\ w\ y)\ (x)\ (\overline{x}\ \overline{y})$$

$$(u \vee z)\ (y\ z)\ (\overline{z}\ w\ y)\ (\overline{y})$$

$$(u \vee z)\ (z)\ (\overline{z}\ w)$$

$$(w)$$

Pure Literal Elimination. Assign variables that appear with uniform polarity.

SAT Solvers

Unit Propogation. Force all inevitable assignments.

$$(u \vee z)(y\ z)\ (\overline{z}\ w\ y)\ (x)\ (\overline{x}\ \overline{y})$$

$$(u \vee z)\ (y\ z)\ (\overline{z}\ w\ y)\ (\overline{y})$$

$$(u \vee z)\ (z)\ (\overline{z}\ w)$$

$$(w)$$

Pure Literal Elimination. Assign variables that appear with uniform polarity.

$$(u \vee z)(y\ z)\ (\overline{z}\ w\ y)\ (x)\ (\overline{x}\ \overline{y})\ (y\ \overline{v})\ (u\ x\ y)\ (u\ \overline{z}\ w)\ (x\ \overline{y})$$

SAT Solvers

Unit Propogation. Force all inevitable assignments.

$$(u \vee z)(y\ z)\ (\overline{z}\ w\ y)\ (x)\ (\overline{x}\ \overline{y})$$

$$(u \vee z)\ (y\ z)\ (\overline{z}\ w\ y)\ (\overline{y})$$

$$(u \vee z)\ (z)\ (\overline{z}\ w)$$

$$(w)$$

Pure Literal Elimination. Assign variables that appear with uniform polarity.

$$(u \vee z)(y\ z)\ (\overline{z}\ w\ y)\ (x)\ (\overline{x}\ \overline{y})\ (y\ \overline{v})\ (u\ x\ y)\ (u\ \overline{z}\ w)\ (x\ \overline{y})$$

SAT Solvers

Unit Propogation. Force all inevitable assignments.

Pure Literal Elimination. Assign variables that appear with uniform polarity.

SAT Solvers

Unit Propogation. Force all inevitable assignments.

Pure Literal Elimination. Assign variables that appear with uniform polarity.

SAT Solvers

Unit Propogation. Force all inevitable assignments.

Pure Literal Elimination. Assign variables that appear with uniform polarity.

Branch. Pick a variable (based on some heuristic) and try both settings.

SAT Solvers

Unit Propogation. Force all inevitable assignments.

Pure Literal Elimination. Assign variables that appear with uniform polarity.

Branch. Pick a variable (based on some heuristic) and try both settings.

F[x = 0] F[x = 1]

SAT Solvers

Unit Propogation. Force all inevitable assignments.

Pure Literal Elimination. Assign variables that appear with uniform polarity.

Branch. Pick a variable (based on some heuristic) and try both settings.

F[x = 0] F[x = 1]

SAT Solvers

SAT Solvers

Most variables seem “dependent” only on a few variables.

SAT Solvers

Most variables seem “dependent” only on a few variables.

Fixing an assignment of these variables quickly propagates to the other variables.

SAT Solvers

Most variables seem “dependent” only on a few variables.

Fixing an assignment of these variables quickly propagates to the other variables.

Lots of real world instances seem to have a small set on which the remaining variables are dependent.

SAT Solvers

Most variables seem “dependent” only on a few variables.

Fixing an assignment of these variables quickly propagates to the other variables.

Lots of real world instances seem to have a small set on which the remaining variables are dependent.

Can we capture the structure of an instance through this small set of variables ?

Beyond Graphs

Efficient SAT algorithms using backdoors

Introduction: The curious success of SAT Solvers

Backdoors and Base Classes

Extending tractable classes using Backdoors

Solving SAT using backdoors
(but without finding them!)

Beyond uniform backdoors

Base Classes

Base Classes

A **tractable base class** is a class of SAT instances
that can be resolved in polynomial time.

Base Classes

A **tractable base class** is a class of SAT instances that can be resolved in polynomial time.

EXAMPLES

Base Classes

A **tractable base class** is a class of SAT instances that can be resolved in polynomial time.

EXAMPLES

1-valid Formulas where all clauses have **at least one positive** literal.

0-valid Formulas where all clauses have **at least one negative** literal.

Base Classes

A **tractable base class** is a class of SAT instances that can be resolved in polynomial time.

EXAMPLES

1-valid Formulas where all clauses have **at least one positive** literal.

0-valid Formulas where all clauses have **at least one negative** literal.

2-CNF Formulas where all clauses have **at most two** literals.

Base Classes

A **tractable base class** is a class of SAT instances that can be resolved in polynomial time.

EXAMPLES

1-valid Formulas where all clauses have **at least one positive** literal.

0-valid Formulas where all clauses have **at least one negative** literal.

2-CNF Formulas where all clauses have **at most two** literals.

Horn Formulas where all clauses have **at most one positive** literal.

Dual Horn Formulas where all clauses have **at most one negative** literal.

Backdoors

Backdoors

$(\mathbf{u} \vee z)(y\ z)\ (\overline{z}\ w\ y)$

$(x)\ (\overline{x}\ \overline{y})\ (y\ \overline{v})\ (\mathbf{u}\ x\ y)$

$(\mathbf{u}\ \overline{z}\ w)\ (x\ \overline{y})$

$(x\ y\ \overline{u}\ v)$

$(a\ b\ u)\ (\overline{a}\ u\ x)$

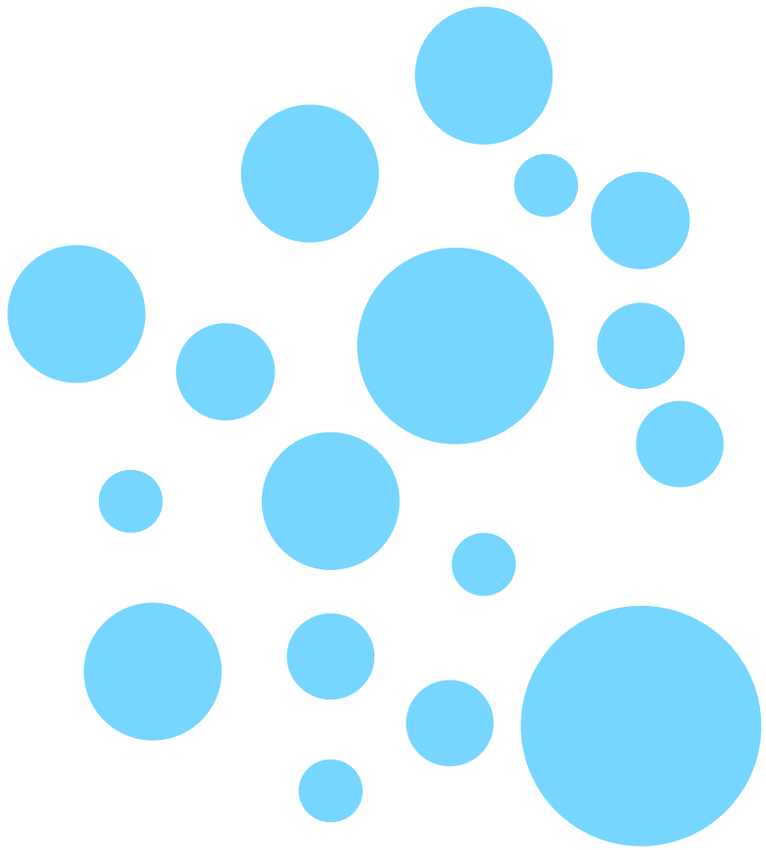
$(\overline{b}\ z\ y)\ (a\ x\ z)\ (a\ \overline{w})$

A Formula F

Backdoors

$$(\mathbf{u} \vee z)(y\ z)\ (\overline{z}\ w\ y)$$
$$(x)\ (\overline{x}\ \overline{y})\ (y\ \overline{v})\ (\mathbf{u}\ x\ y)$$
$$(\mathbf{u}\ \overline{z}\ w)\ (x\ \overline{y})$$
$$(x\ y\ \overline{u}\ v)$$
$$(a\ b\ u)\ (\overline{a}\ u\ x)$$
$$(\overline{b}\ z\ y)\ (a\ x\ z)\ (a\ \overline{w})$$

A Formula F



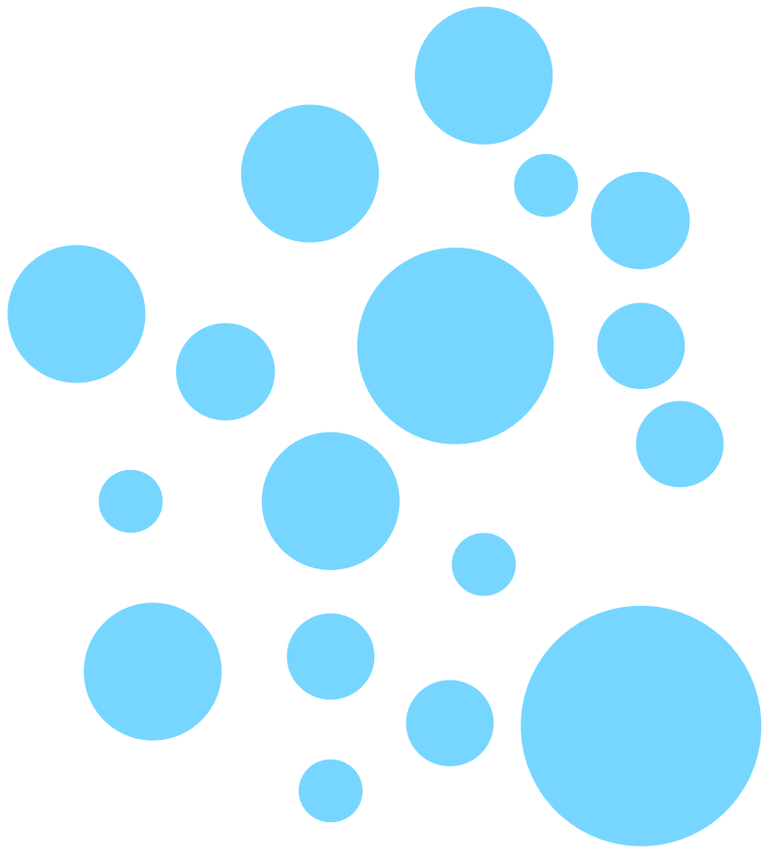
A base class **B**

Backdoors

Goal: Transform F so that it belongs to **B** with "minimum damages".

$(\mathbf{u} \vee z)(y\ z)\ (\overline{z}\ w\ y)$
$(x)\ (\overline{x}\ \overline{y})\ (y\ \overline{v})\ (\mathbf{u}\ x\ y)$
$(\mathbf{u}\ \overline{z}\ w)\ (x\ \overline{y})$
$(x\ y\ \overline{u}\ v)$
$(a\ b\ u)\ (\overline{a}\ u\ x)$
$(\overline{b}\ z\ y)\ (a\ x\ z)\ (a\ \overline{w})$

A Formula F



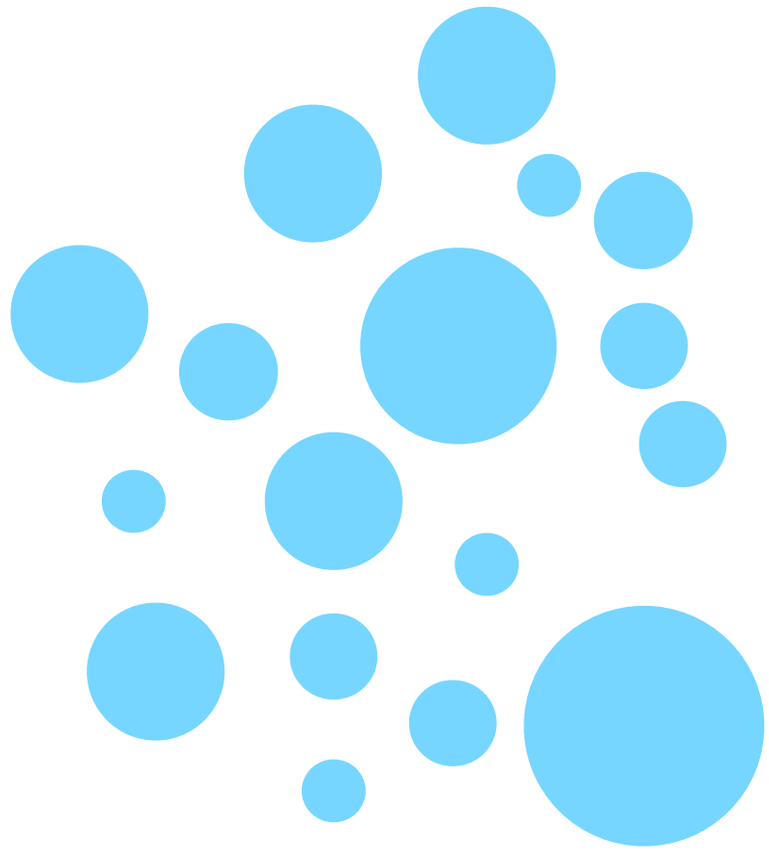
A base class **B**

Backdoors

Goal: Transform F so that it belongs to **B** with "minimum damages".

$$(\mathbf{u} \vee z)(y\ z)\ (\overline{z}\ w\ y)$$
$$(x)\ (\overline{x}\ \overline{y})\ (y\ \overline{v})\ (\mathbf{u}\ x\ y)$$
$$(\mathbf{u}\ \overline{z}\ w)\ (x\ \overline{y})$$
$$(x\ y\ \overline{u}\ v)$$
$$(a\ b\ u)\ (\overline{a}\ u\ x)$$
$$(\overline{b}\ z\ y)\ (a\ x\ z)\ (a\ \overline{w})$$

A Formula F



A base class **B**

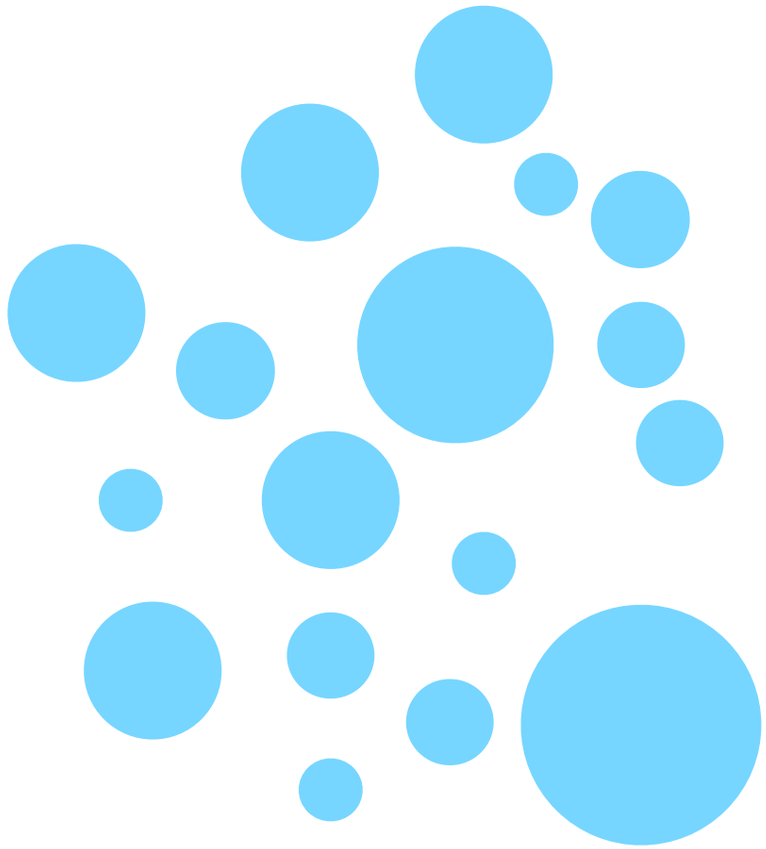
Backdoors

Goal: Transform F so that it belongs to **B** with "minimum damages".

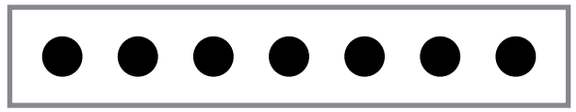
Williams, Gomes, Selman (IJCAI, 2003) and
Crama, Ekin, Hammer (Discrete Applied Math, 1997)

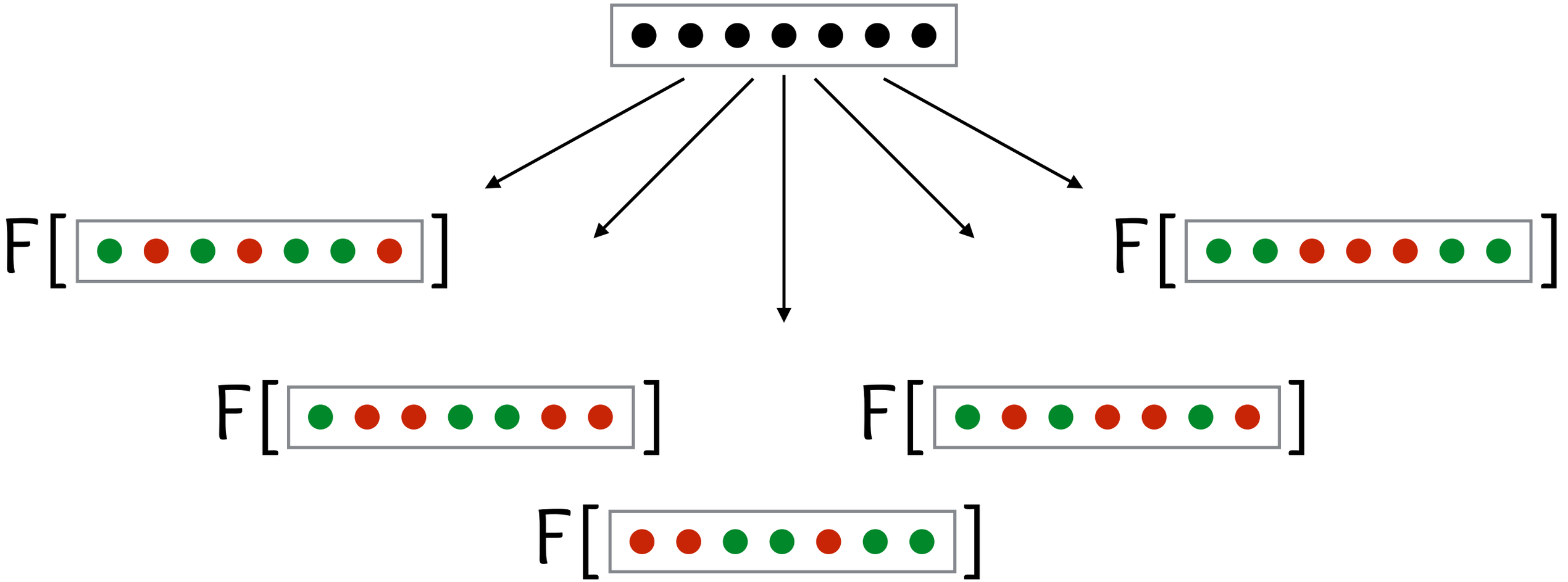
$$(\mathbf{u} \vee z)(y\ z)\ (\overline{z}\ w\ y)$$
$$(x)\ (\overline{x}\ \overline{y})\ (y\ \overline{v})\ (\mathbf{u}\ x\ y)$$
$$(\mathbf{u}\ \overline{z}\ w)\ (x\ \overline{y})$$
$$(x\ y\ \overline{u}\ v)$$
$$(a\ b\ u)\ (\overline{a}\ u\ x)$$
$$(\overline{b}\ z\ y)\ (a\ x\ z)\ (a\ \overline{w})$$

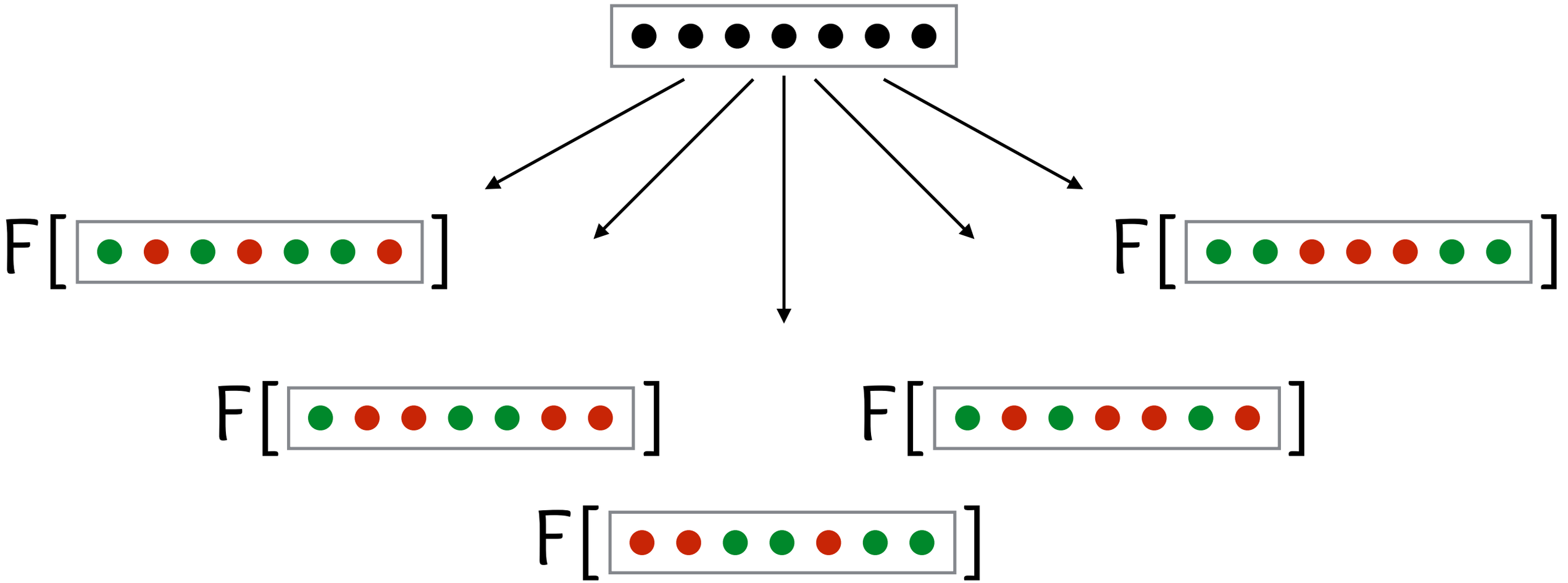
A Formula F



A base class **B**

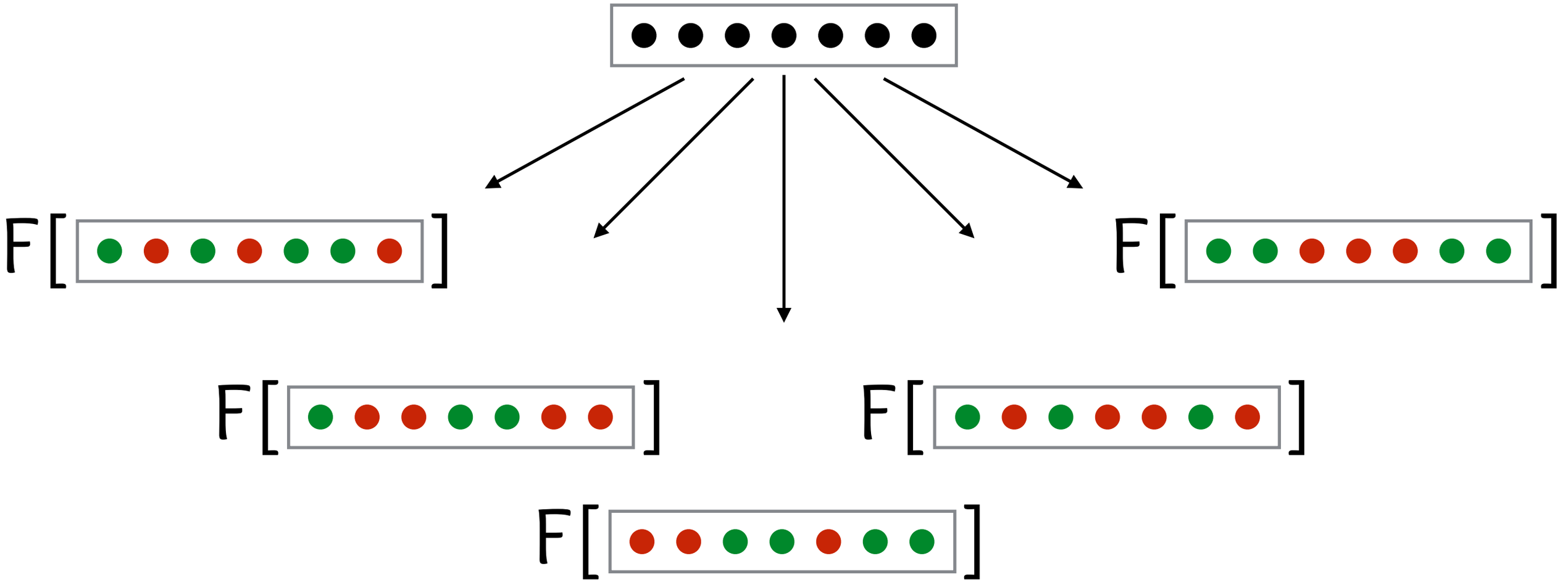



F[
F[
F[
F[
F[



All these formulas belong to **B.**

Strong Backdoor



All these formulas belong to **B.**

Strong Backdoor

At least one of them is satisfiable and in **B.**

Weak Backdoor

Beyond Graphs

Efficient SAT algorithms using backdoors

Introduction: The curious success of SAT Solvers

Backdoors and Base Classes

Extending tractable classes using Backdoors

Solving SAT using backdoors
(but without finding them!)

Beyond uniform backdoors

Backdoors

Backdoors

Let X be a subset of variables.

Backdoors

Let X be a subset of variables.

Strong Backdoor For any assignment g on X, F[g] belongs to **B.**

Backdoors

Let X be a subset of variables.

Strong Backdoor For any assignment g on X, F[g] belongs to **B.**

Weak Backdoor There is an assignment assignment g on X for which F[g] belongs to **B** *and* is satisfiable.

Backdoors

Let X be a subset of variables.

Strong Backdoor For any assignment g on X, F[g] belongs to **B.**

Weak Backdoor There is an assignment assignment g on X for which F[g] belongs to **B** *and* is satisfiable.

Try all possible assignments to the backdoor variables, and run an algorithm designed to solve SAT formulas in **B.**

Backdoors

Let X be a subset of variables.

Strong Backdoor For any assignment g on X, F[g] belongs to **B.**

Weak Backdoor There is an assignment assignment g on X for which F[g] belongs to **B** *and* is satisfiable.

Try all possible assignments to the backdoor variables, and run an algorithm designed to solve SAT formulas in **B.**

[Time to find backdoor of size k] + 2^k poly(n)

[Time to find backdoor of size k] + 2^k poly(n)

[Time to find backdoor of size k] + 2^k poly(n)

Typically, the smallest backdoors to most reasonable base classes are NP-complete to find.

[Time to find backdoor of size k] + 2^k poly(n)

Typically, the smallest backdoors to most reasonable base classes are NP-complete to find.

As might be expected, the parameterized complexity of the problem is well-explored with respect to various choices of base classes.

[Time to find backdoor of size k] + 2^k poly(n)

Typically, the smallest backdoors to most reasonable base classes are NP-complete to find.

As might be expected, the parameterized complexity of the problem is well-explored with respect to various choices of base classes.

For example, finding weak (**strong**) backdoors to any of the Schaefer Classes is W[2]-hard (**FPT**).

[Nishimura, Ragde, Szeider 2004]

[Time to find backdoor of size k] + 2^k poly(n)

Typically, the smallest backdoors to most reasonable base classes are NP-complete to find.

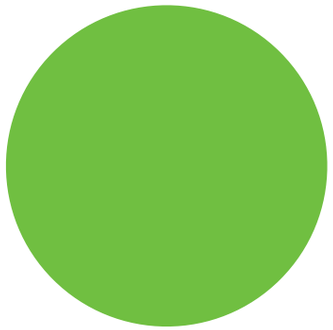
As might be expected, the parameterized complexity of the problem is well-explored with respect to various choices of base classes.

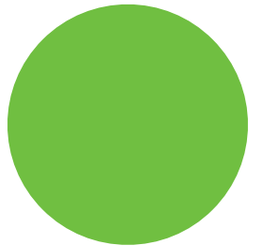
For example, finding weak (**strong**) backdoors to any of the Schaefer Classes is W[2]-hard (**FPT**).

[Nishimura, Ragde, Szeider 2004]

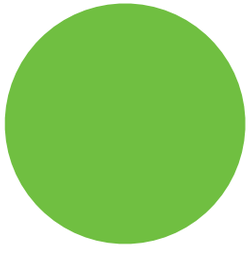
However, finding weak backdoors to any of the Schaefer Classes if FPT if the input is a 3-CNF formula.


2-SAT

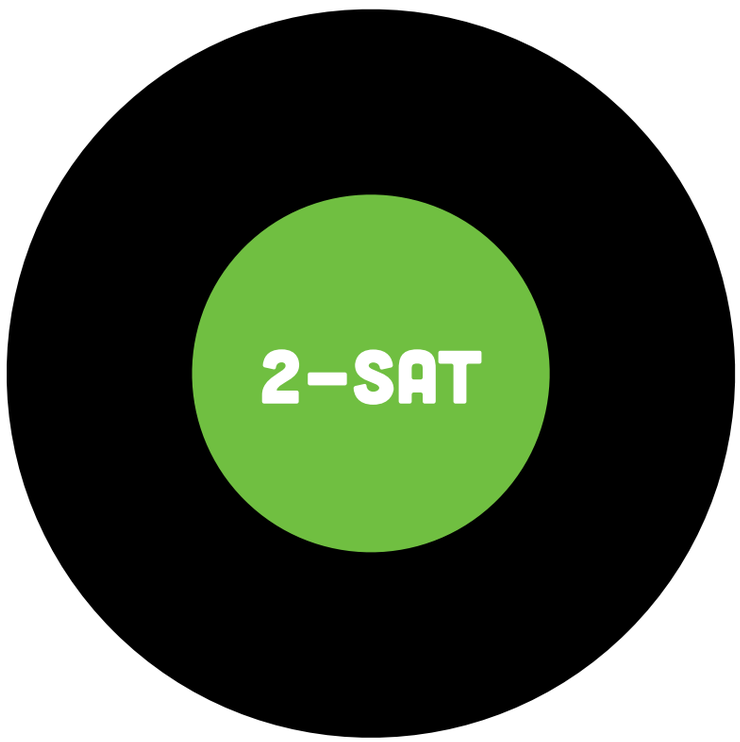


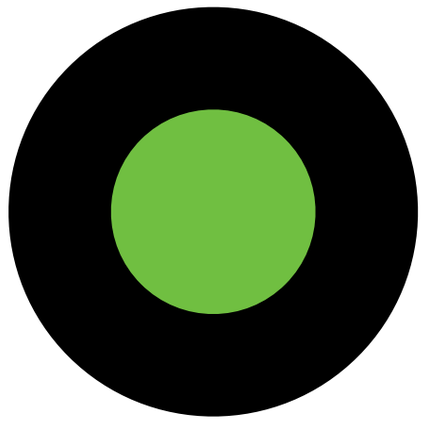


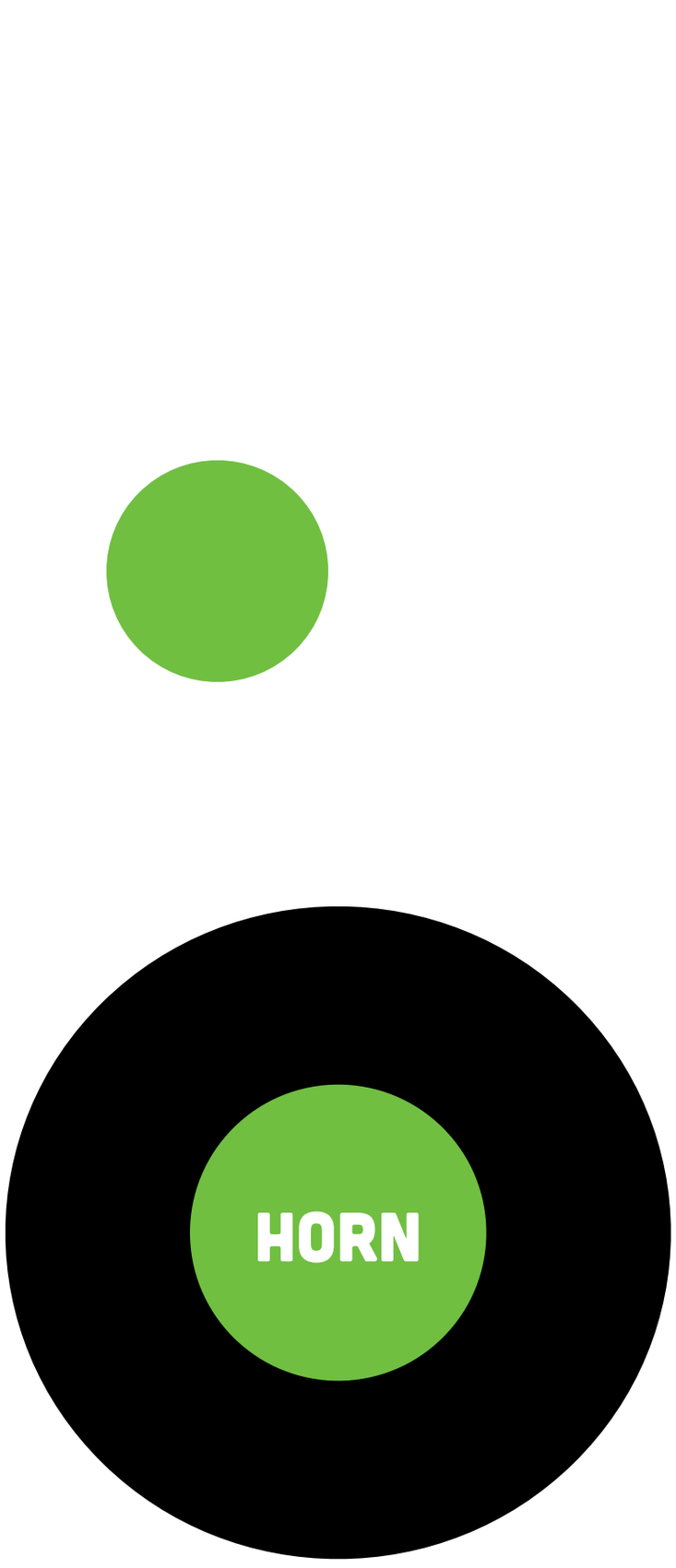





HORN


2-SAT




HORN

The base class of acyclic formulas

The base class of acyclic formulas

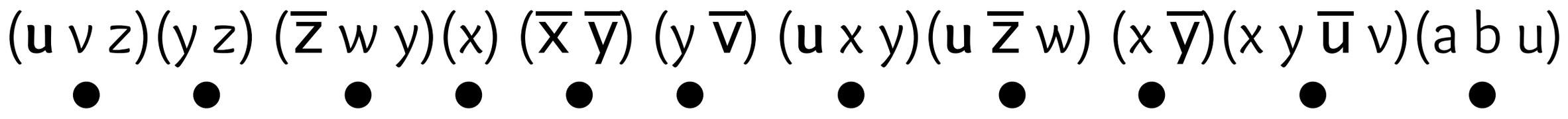
$$(\mathbf{u}\ \nu\ z)(y\ z)\ (\overline{z}\ w\ y)(x)\ (\overline{x}\ \overline{y})\ (y\ \overline{\nu})\ (\mathbf{u}\ x\ y)(\mathbf{u}\ \overline{z}\ w)\ (x\ \overline{y})(x\ y\ \overline{u}\ \nu)(a\ b\ u)$$

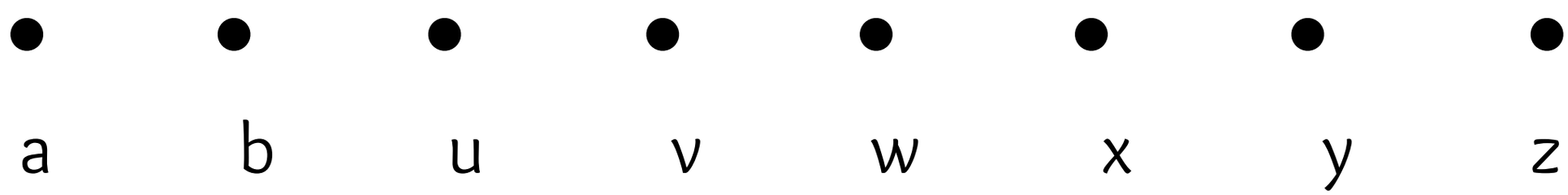
The base class of acyclic formulas

$$(\mathbf{u}\ \nu\ z)(y\ z)\ (\overline{z}\ w\ y)(x)\ (\overline{x}\ \overline{y})\ (y\ \overline{\nu})\ (\mathbf{u}\ x\ y)(\mathbf{u}\ \overline{z}\ w)\ (x\ \overline{y})(x\ y\ \overline{u}\ \nu)(a\ b\ u)$$

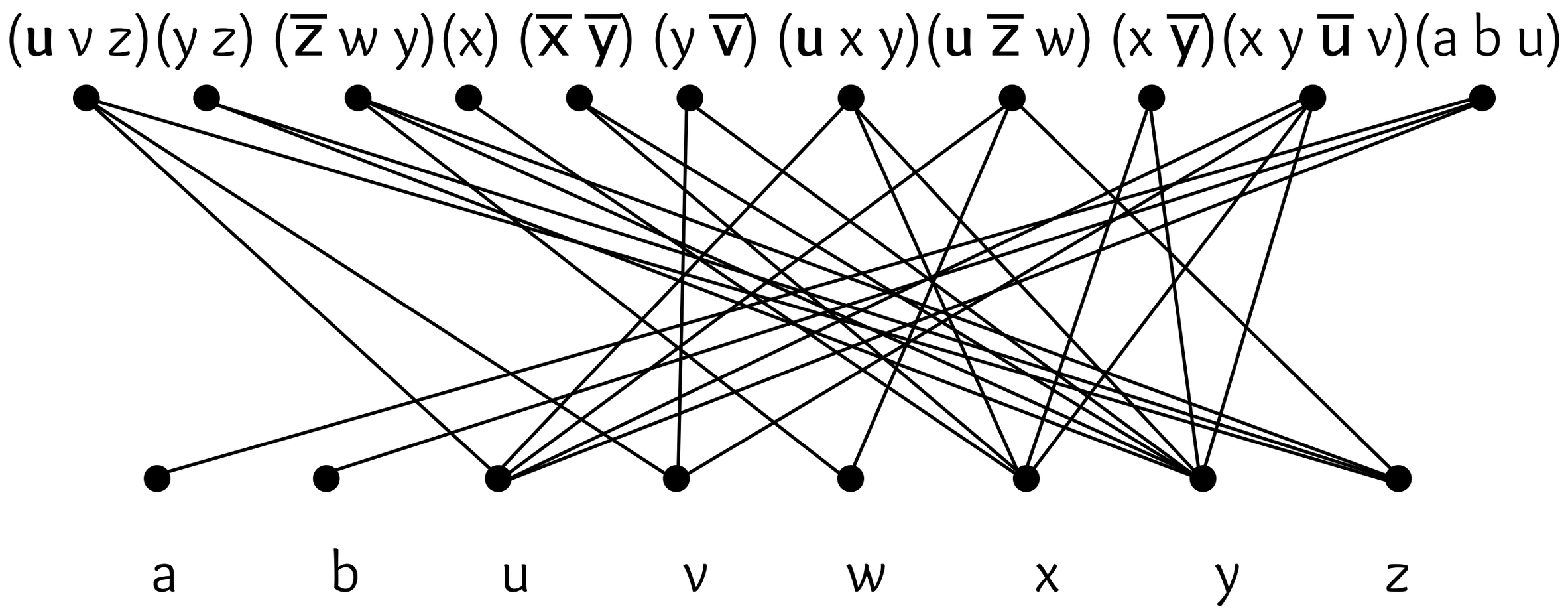
a b u ν w x y z

The base class of acyclic formulas

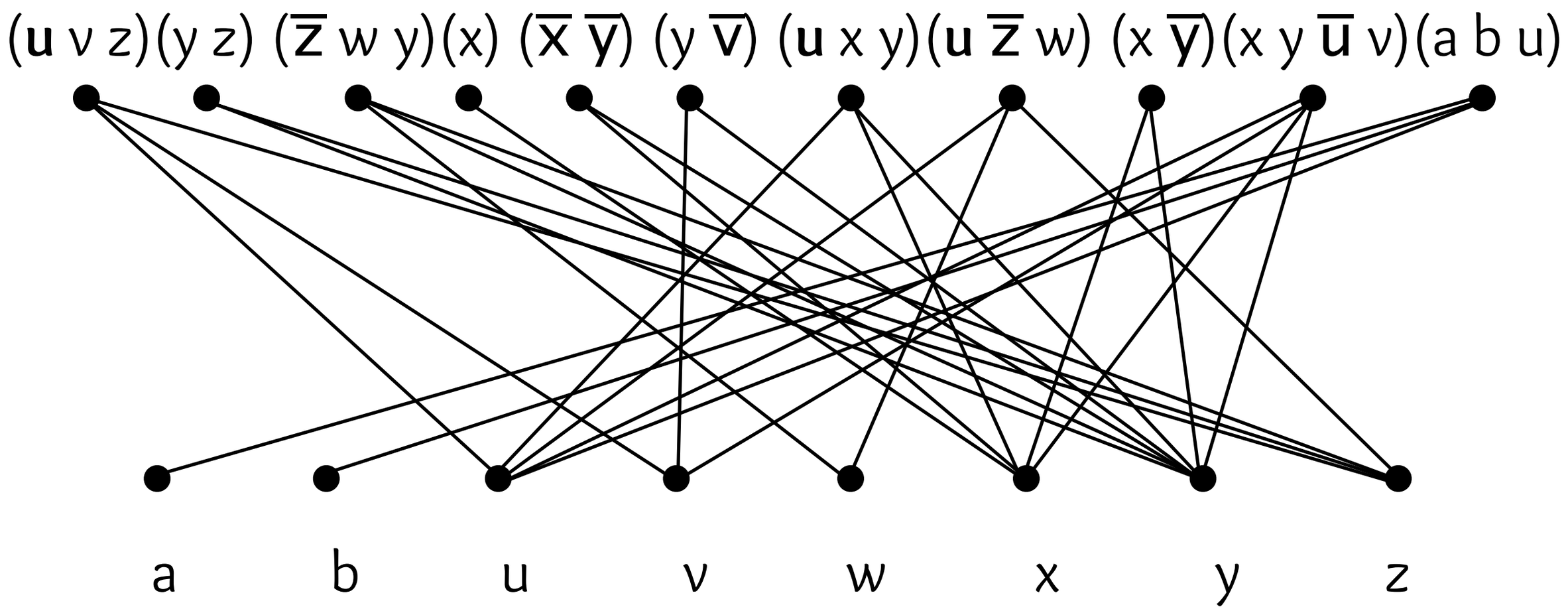




The base class of acyclic formulas



The base class of acyclic formulas



If the incidence graph is **acycilc**,
then the formula is solvable in polynomial time.

(Fischer, Makowsky, Ravve 2008)

Backdoors to acyclic formulas

Backdoors to acyclic formulas

If the incidence graph is **acycilc**,
then the formula is solvable in polynomial time.
(Fischer, Makowsky, Ravve; J. of Discrete Applied Math, 2008)

Backdoors to acyclic formulas

If the incidence graph is **acycilc**,
then the formula is solvable in polynomial time.
(Fischer, Makowsky, Ravve; J. of Discrete Applied Math, 2008)

Weak backdoor detection to acyclic SAT is W[2]-hard.
Weak backdoor detection to acyclic 3-SAT is FPT.
Strong backdoor detection to acyclic SAT is **FPT-approximable.**
Gaspers and Szeider (ICALP 2012)

Backdoors to acyclic formulas

If the incidence graph is **acycilc**,
then the formula is solvable in polynomial time.
(Fischer, Makowsky, Ravve; J. of Discrete Applied Math, 2008)

Weak backdoor detection to acyclic SAT is W[2]-hard.
Weak backdoor detection to acyclic 3-SAT is FPT.
Strong backdoor detection to acyclic SAT is **FPT-approximable.**
Gaspers and Szeider (ICALP 2012)

In FPT time, either conclude there is no strong backdoor of size k or compute a strong backdoor of size 2^k.

Backdoors to bounded treewidth formulas

Backdoors to bounded treewidth formulas

If the incidence graph is **tree-like**,
then the formula is solvable in polynomial time.
(Fischer, Makowsky, Ravve; J. of Discrete Applied Math, 2008)

Backdoors to bounded treewidth formulas

If the incidence graph is **tree-like**,
then the formula is solvable in polynomial time.
(Fischer, Makowsky, Ravve; J. of Discrete Applied Math, 2008)

Strong backdoor detection to tw-SAT is
FPT-approximable.
Gaspers and Szeider (ICALP 2012)

Backdoors to bounded treewidth formulas

If the incidence graph is **tree-like**,
then the formula is solvable in polynomial time.
(Fischer, Makowsky, Ravve; J. of Discrete Applied Math, 2008)

Strong backdoor detection to tw-SAT is
FPT-approximable.
Gaspers and Szeider (ICALP 2012)

SAT is FPT parameterized by the size of a SBD to tw-SAT.

Backdoors to bounded treewidth formulas

If the incidence graph is **tree-like**,
then the formula is solvable in polynomial time.
(Fischer, Makowsky, Ravve; J. of Discrete Applied Math, 2008)

Strong backdoor detection to tw-SAT is
FPT-approximable.
Gaspers and Szeider (ICALP 2012)

SAT is FPT parameterized by the size of a SBD to tw-SAT.

$$O(2^{2^k} n^3)$$

Beyond Graphs

Efficient SAT algorithms using backdoors

Introduction: The curious success of SAT Solvers

Backdoors and Base Classes

Extending tractable classes using Backdoors

Solving SAT using backdoors
(but without finding them!)

Beyond uniform backdoors

3-SAT parameterized by k=min{sbd,wbd} to tw 3-SAT can be solved in time $2^{O(k)}m$.

(Fomin, Lokshtanov, M, Ramanujan, Saurabh; SODA 2015)

3-SAT parameterized by k=min{sbd,wbd} to tw 3-SAT can be solved in time $2^{O(k)}m$.

(Fomin, Lokshtanov, M, Ramanujan, Saurabh; SODA 2015)

Williams et al. proposed that SAT solvers encounter backdoor sets without actually searching for them.

3-SAT parameterized by k=min{sbd,wbd} to tw 3-SAT can be solved in time $2^{O(k)}m$.

(Fomin, Lokshtanov, M, Ramanujan, Saurabh; SODA 2015)

Williams et al. proposed that SAT solvers encounter backdoor sets without actually searching for them.

Our algorithm revisits this perspective.

In the spirit of SAT solvers

In the spirit of SAT solvers

In the spirit of SAT solvers

If tw is bounded by a constant. Resolve the formula in polynomial time.

In the spirit of SAT solvers

If tw is bounded by a constant. Resolve the formula in polynomial time.

Protrusion Elimination. Remove protrusions from the incidence graph.

In the spirit of SAT solvers

If tw is bounded by a constant. Resolve the formula in polynomial time.

Protrusion Elimination. Remove protrusions from the incidence graph.

Branch. Pick a variable (uniformly at random) and try both settings.

In the spirit of SAT solvers

If tw is bounded by a constant. Resolve the formula in polynomial time.

Protrusion Elimination. Remove protrusions from the incidence graph.

Branch. Pick a variable (uniformly at random) and try both settings.

F[x = 0] F[x = 1]

In the spirit of SAT solvers

If tw is bounded by a constant. Resolve the formula in polynomial time.

Protrusion Elimination. Remove protrusions from the incidence graph.

Branch. Pick a variable (uniformly at random) and try both settings.

F[x = 0] F[x = 1]

In the spirit of SAT solvers

In the spirit of SAT solvers

Protrusion replacement takes the place of
Unit Propogation and Pure Literal Elimination.

In the spirit of SAT solvers

Protrusion replacement takes the place of
Unit Propogation and Pure Literal Elimination.

Since the base class is more complex, the
preprocessing is also involved.

In the spirit of SAT solvers

Protrusion replacement takes the place of
Unit Propogation and Pure Literal Elimination.

Since the base class is more complex, the
preprocessing is also involved.

But intuition remains the same: Remove ‘irrelevant’ parts
of the formula or at the very least replace them with a
‘small’ equivalent formula.

In the spirit of SAT solvers

In the spirit of SAT solvers

First FPT algorithm for SAT which does not depend on computing a backdoor set first.

In the spirit of SAT solvers

First FPT algorithm for SAT which does not depend on computing a backdoor set first.

The algorithm achieves optimal running time both with respect to the parameter and input size.

In the spirit of SAT solvers

First FPT algorithm for SAT which does not depend on computing a backdoor set first.

The algorithm achieves optimal running time both with respect to the parameter and input size.

Again, techniques developed here have other applications: improving several kernelization and FPT algorithms to linear time.

In the spirit of SAT solvers

First FPT algorithm for SAT which does not depend on computing a backdoor set first.

The algorithm achieves optimal running time both with respect to the parameter and input size.

Again, techniques developed here have other applications: improving several kernelization and FPT algorithms to linear time.

Expanding the scope of the algorithm to general SAT formulas is a pertinent open problem.

Beyond Graphs

Efficient SAT algorithms using backdoors

Introduction: The curious success of SAT Solvers

Backdoors and Base Classes

Extending tractable classes using Backdoors

Solving SAT using backdoors
(but without finding them!)

Beyond uniform backdoors

$$(x\ \overline{a}_1\ \overline{a}_2\ \overline{a}_3\ \ldots\ \overline{a}_n)$$
$$(\overline{x}\ b_1\ c_1)\ (\overline{x}\ b_2\ c_2)\ (\overline{x}\ b_3\ c_3)\ \ldots\ (\overline{x}\ b_n\ c_n)$$

$$(x\ \overline{a}_1\ \overline{a}_2\ \overline{a}_3\ \ldots\ \overline{a}_n)$$
$$(\overline{x}\ b_1\ c_1)\ (\overline{x}\ b_2\ c_2)\ (\overline{x}\ b_3\ c_3)\ \ldots\ (\overline{x}\ b_n\ c_n)$$

$(\overline{a}_1\ \overline{a}_2\ \overline{a}_3\ \ldots\ \overline{a}_n)$

$F[x = 0]$

$$(x\ \overline{a}_1\ \overline{a}_2\ \overline{a}_3\ \ldots\ \overline{a}_n)$$
$$(\overline{x}\ b_1\ c_1)\ (\overline{x}\ b_2\ c_2)\ (\overline{x}\ b_3\ c_3)\ \ldots\ (\overline{x}\ b_n\ c_n)$$

$(\overline{a}_1\ \overline{a}_2\ \overline{a}_3\ \ldots\ \overline{a}_n)$

$F[x = 0]$

$(b_1\ c_1)\ (b_2\ c_2)\ (b_3\ c_3)\ \ldots\ (b_n\ c_n)$

$F[x = 1]$

$$(x\ \overline{a}_1\ \overline{a}_2\ \overline{a}_3\ \ldots\ \overline{a}_n)$$
$$(\overline{x}\ b_1\ c_1)\ (\overline{x}\ b_2\ c_2)\ (\overline{x}\ b_3\ c_3)\ \ldots\ (\overline{x}\ b_n\ c_n)$$

$(\overline{a}_1\ \overline{a}_2\ \overline{a}_3\ \ldots\ \overline{a}_n)$

$F[x = 0]$

Horn

$(b_1\ c_1)\ (b_2\ c_2)\ (b_3\ c_3)\ \ldots\ (b_n\ c_n)$

$F[x = 1]$

2CNF

$$(x\ \overline{a}_1\ \overline{a}_2\ \overline{a}_3\ \ldots\ \overline{a}_n)$$
$$(\overline{x}\ b_1\ c_1)\ (\overline{x}\ b_2\ c_2)\ (\overline{x}\ b_3\ c_3)\ \ldots\ (\overline{x}\ b_n\ c_n)$$

$(\overline{a}_1\ \overline{a}_2\ \overline{a}_3\ \ldots\ \overline{a}_n)$

$F[x = 0]$

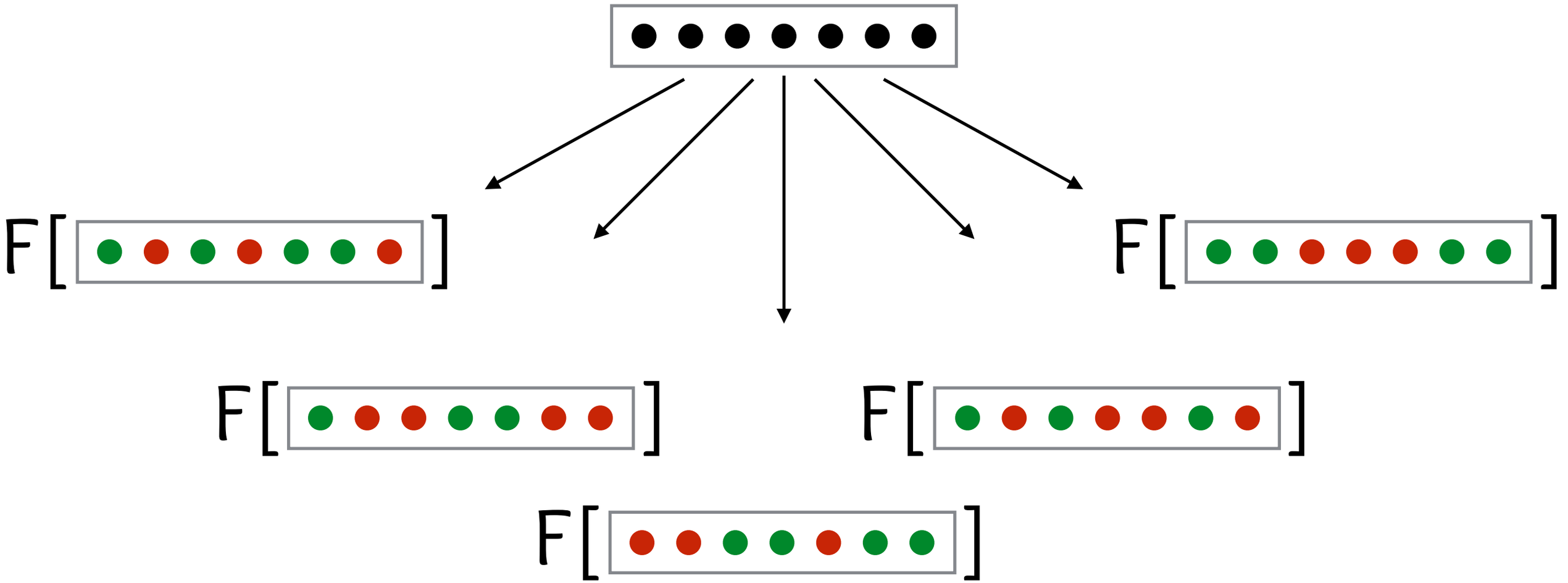
Horn

$(b_1\ c_1)\ (b_2\ c_2)\ (b_3\ c_3)\ \ldots\ (b_n\ c_n)$

$F[x = 1]$

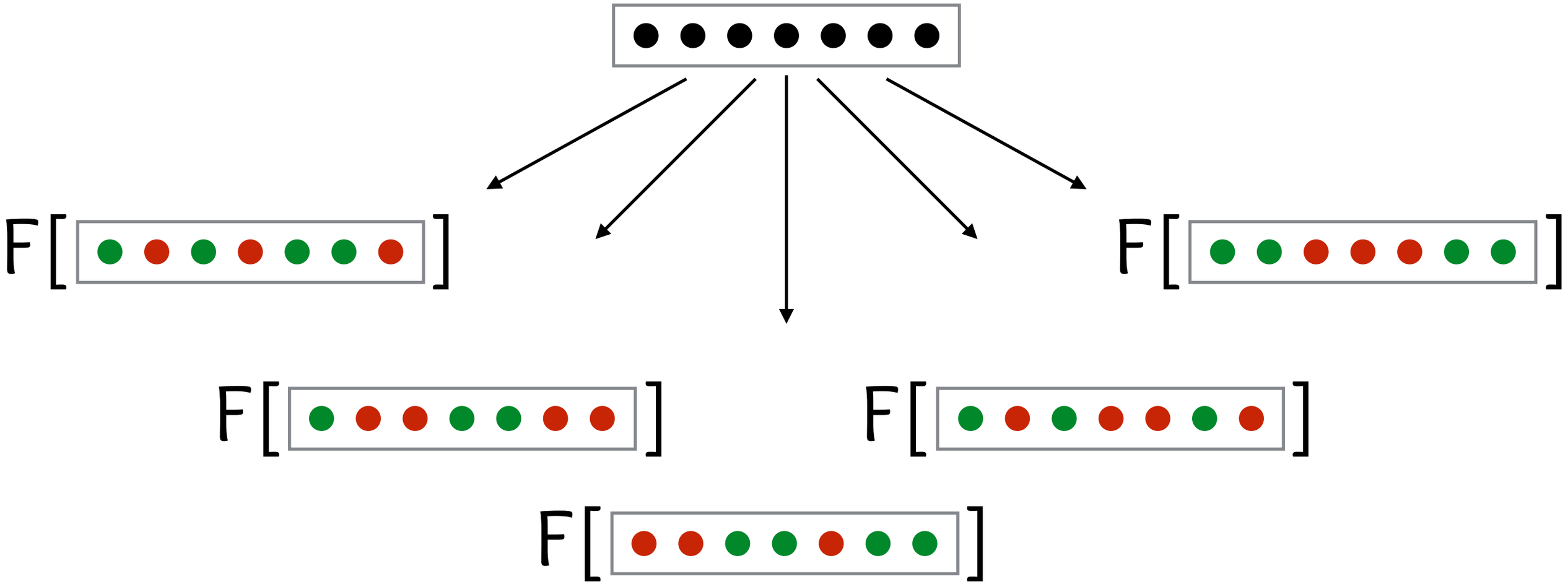
2CNF

Can be checked that the size of the smallest strong backdoor to either Horn or 2CNF is O(n).



All these formulas belong to **B.**

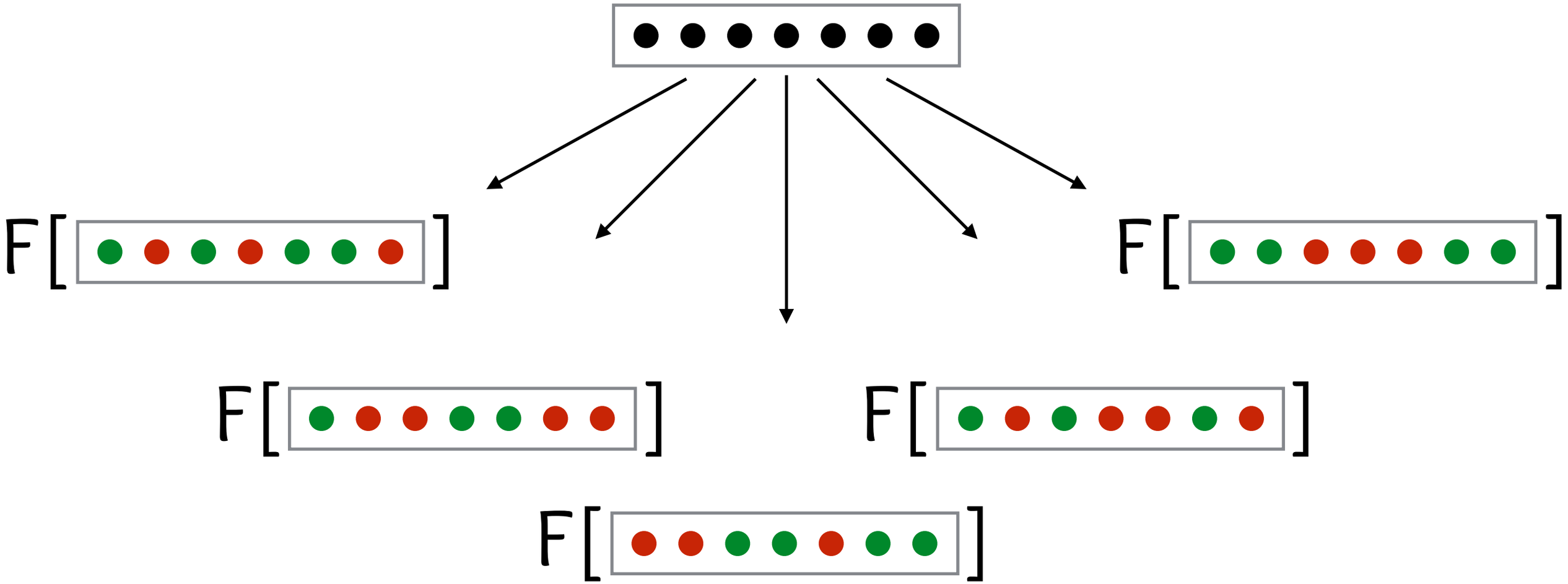
Strong Backdoor



All these formulas belong to some $\mathbf{B_i}$ **in** $\overline{\mathbf{B}}$**.**

Strong Backdoor

(Here $\overline{\mathbf{B}}$ is some collection of tractable base classes.)



All these formulas belong to some $\mathbf{B_i}$ **in** $\overline{\mathbf{B}}$**.**

Heterogeneous Backdoor

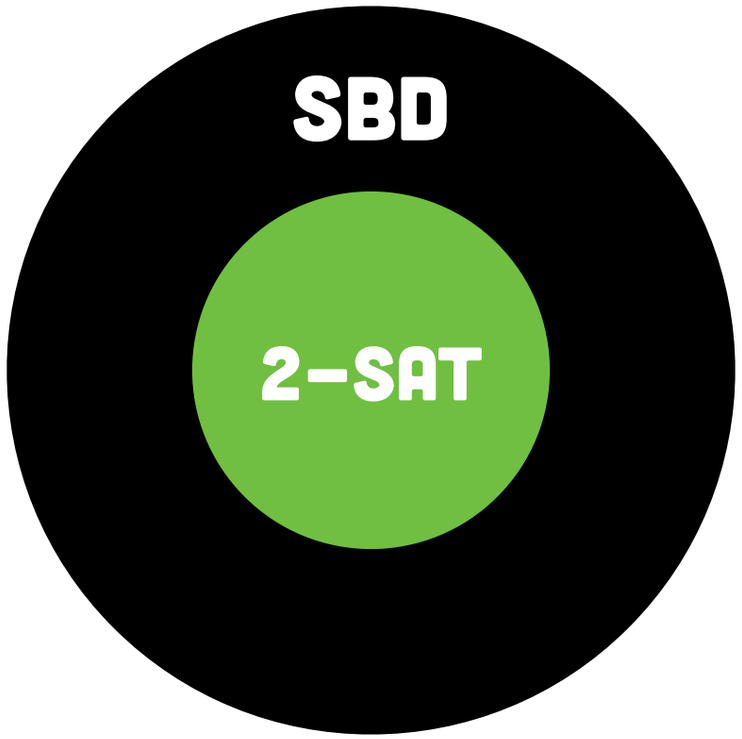
(Here $\overline{\mathbf{B}}$ is some collection of tractable base classes.)

Heterogeneous backdoors can be arbitrarily smaller than strong backdoors.

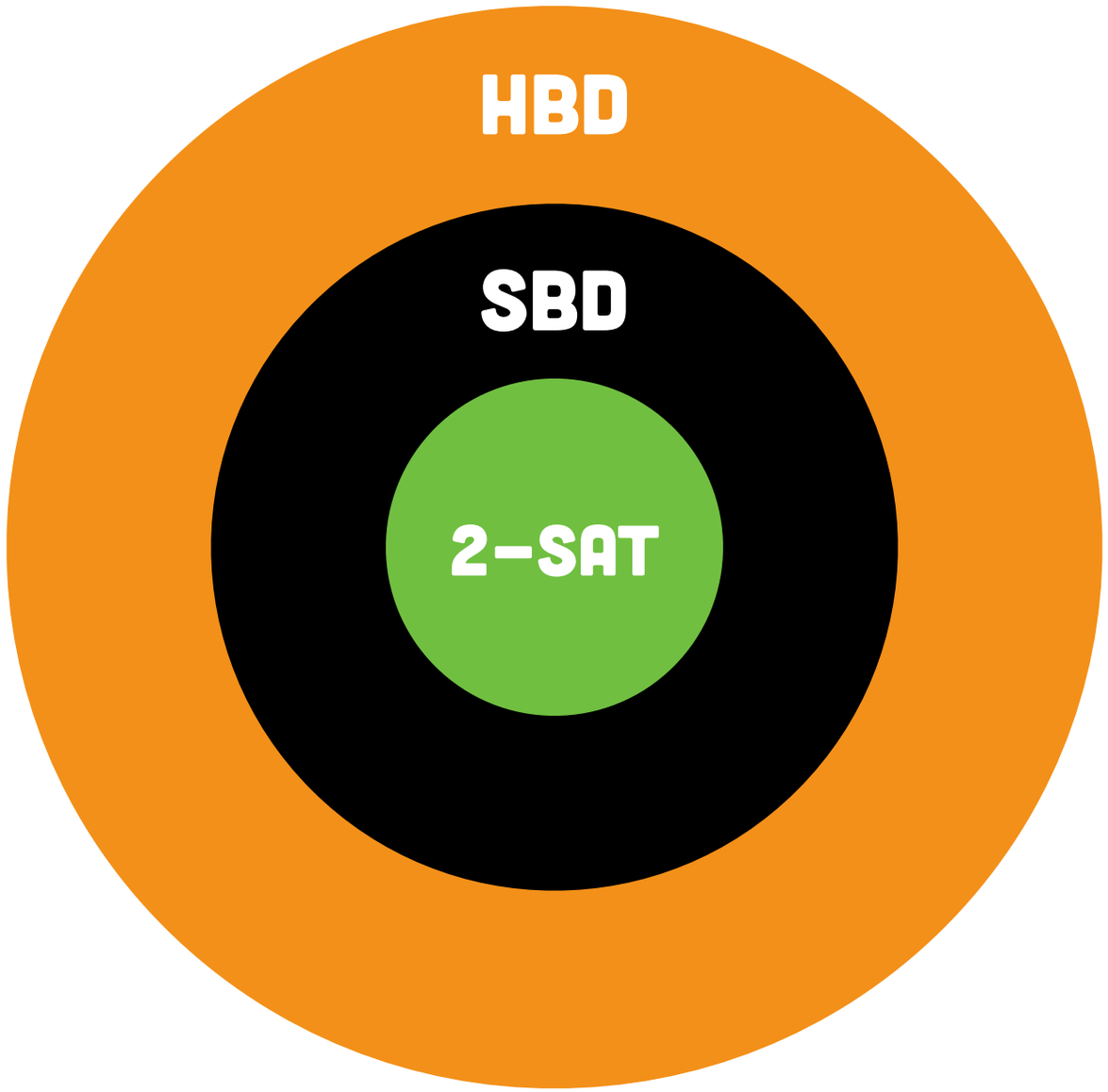
Heterogeneous backdoors can be arbitrarily smaller than strong backdoors.



Heterogeneous backdoors can be arbitrarily smaller than strong backdoors.



Heterogeneous backdoors can be arbitrarily smaller than strong backdoors.



If $\overline{\mathbf{B}}$ does not contain a “bad pair” of Schaefer classes, then detecting heterogenous backdoors to H is FPT .

For every other combination of Schaefer classes, detecting heterogenous backdoors to H is W[2]-hard.

(The problem is also FPT for 3-CNF formulas.)

Gaspers, Ordyniak, M., Szeider, Zivny (AAAI 2014)

Algorithms for Planar F-Deletion and Satisfiability

Preference Aggregation

Strategic behavior in the context of voting



Preference Aggregation

Strategic behavior in the context of voting

Finding a consensus from a crowd

Examples of strategic behavior

Tractable subclasses

Preference Aggregation

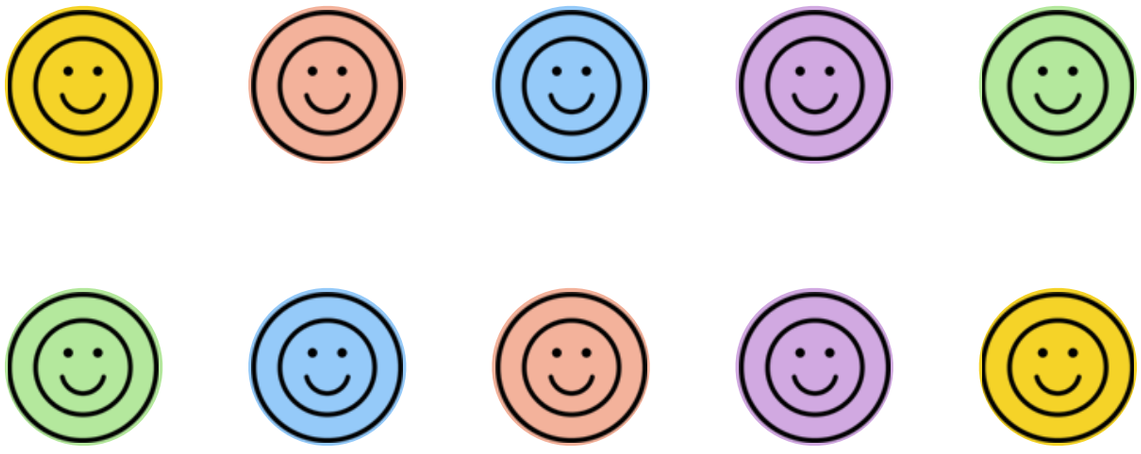
Strategic behavior in the context of voting

Finding a consensus from a crowd

Examples of strategic behavior

Tractable subclasses



















Possible Goals



Possible Goals

Determine a
favorite candidate.





Possible Goals

Determine a
favorite candidate.



Determine a
winning committee.



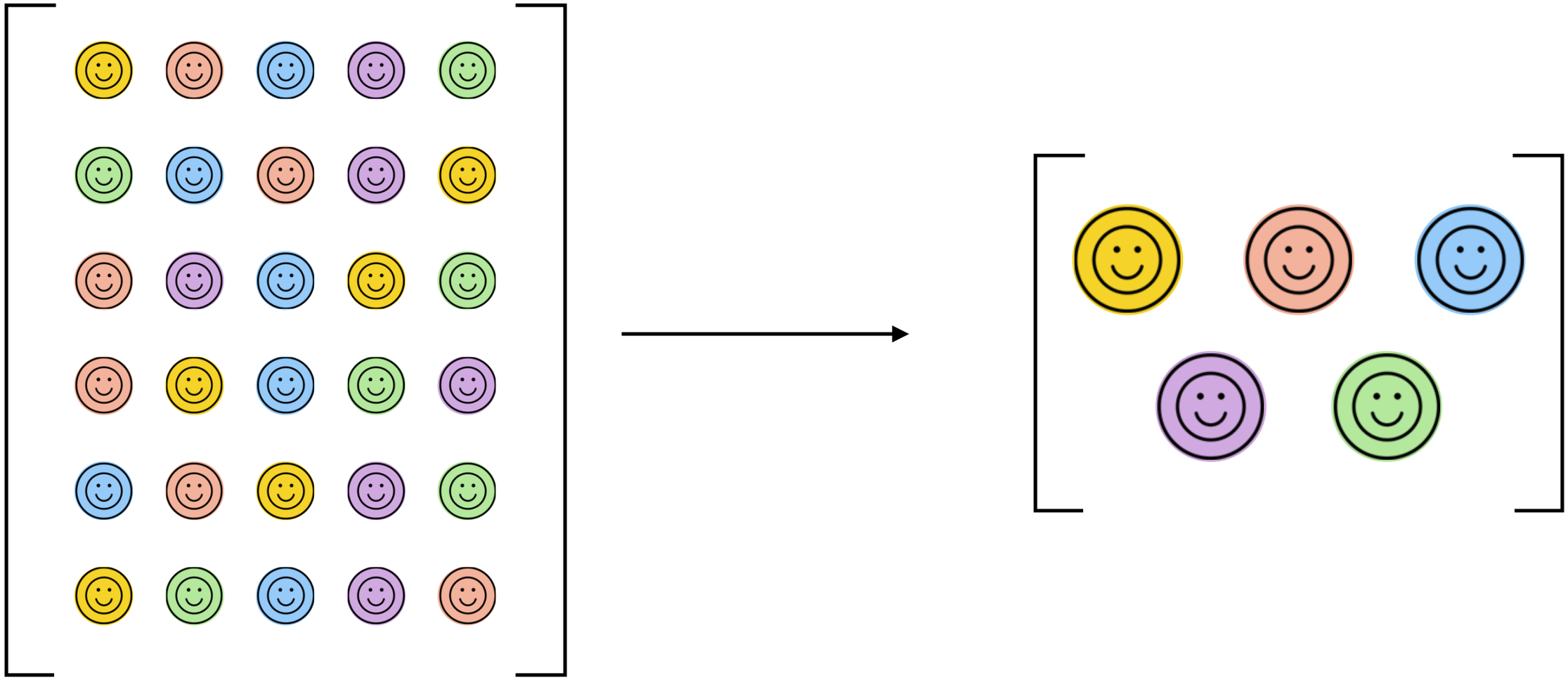
Possible Goals

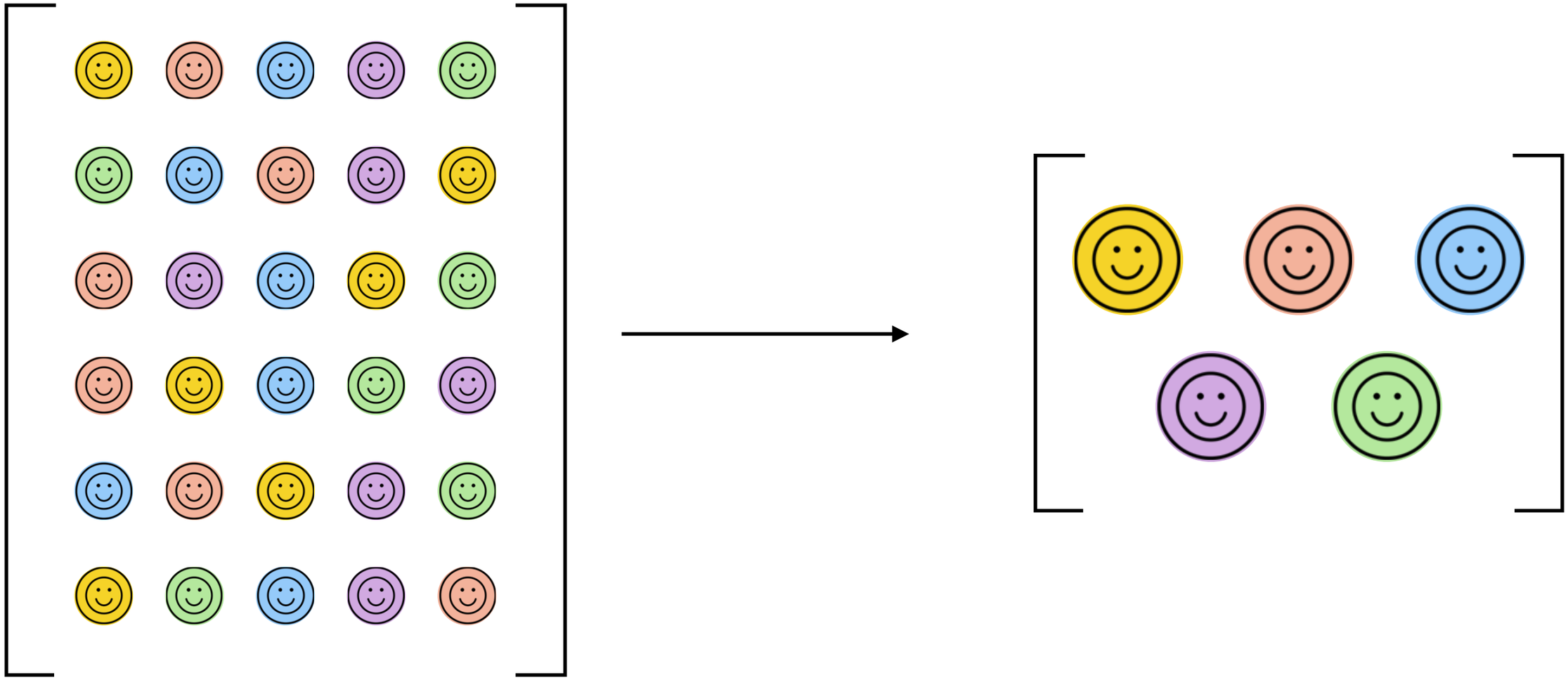
Determine a
favorite candidate.



Determine a
winning committee.

Determine a
consensus ranking.



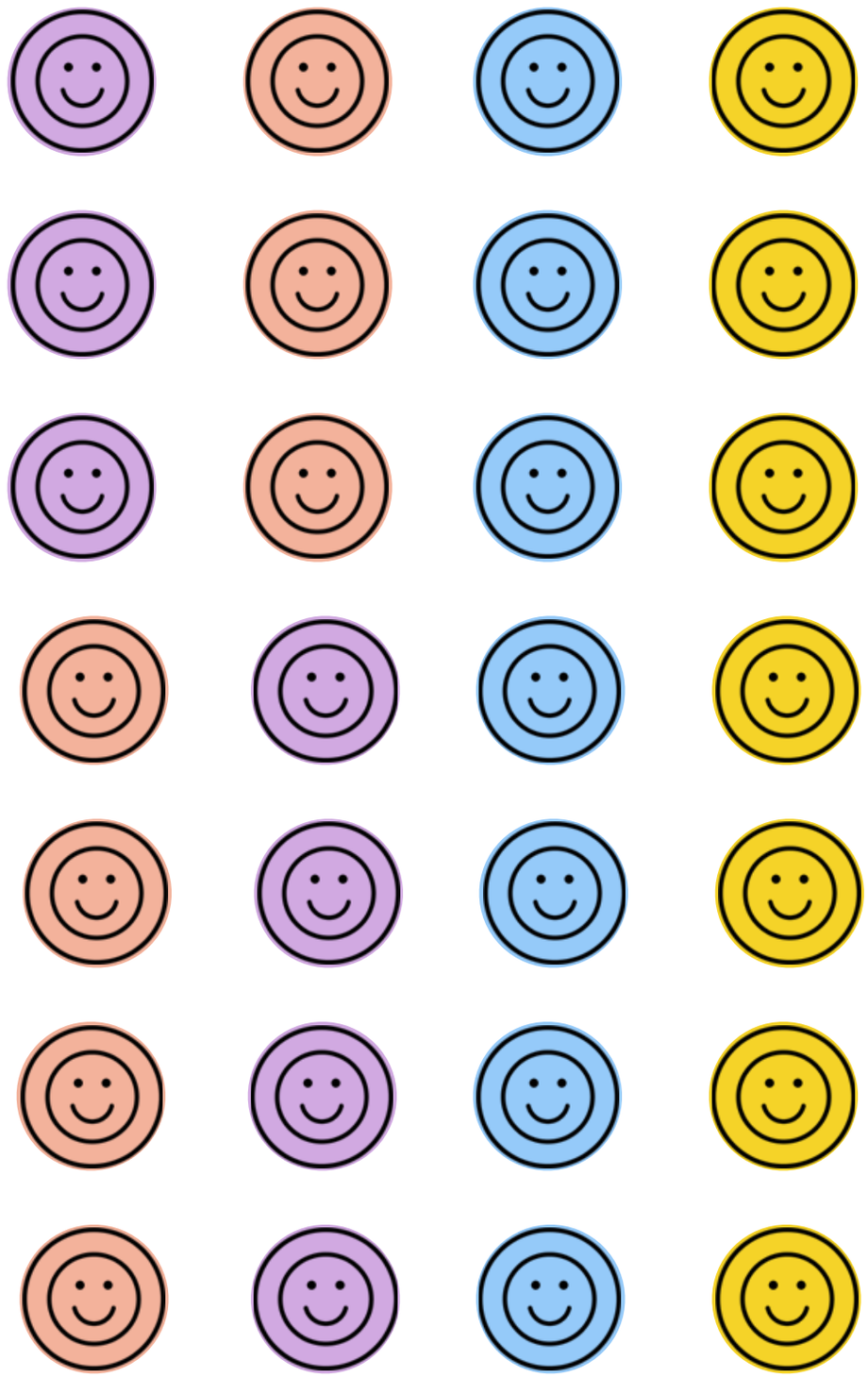


A **social choice function** maps a preference profile to a candidate.

Why not just pick the majority winner?

Why not just pick the majority winner?

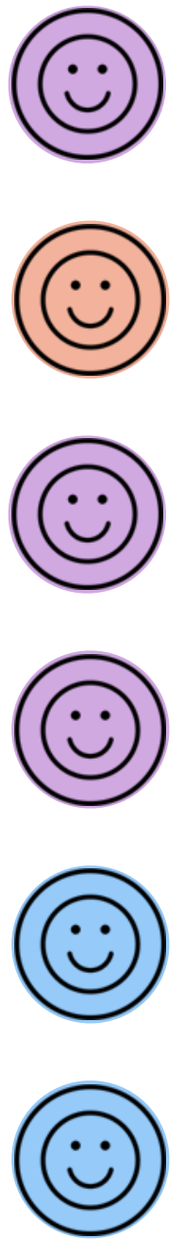




Why not just pick the majority winner?



Why not just pick the majority winner?





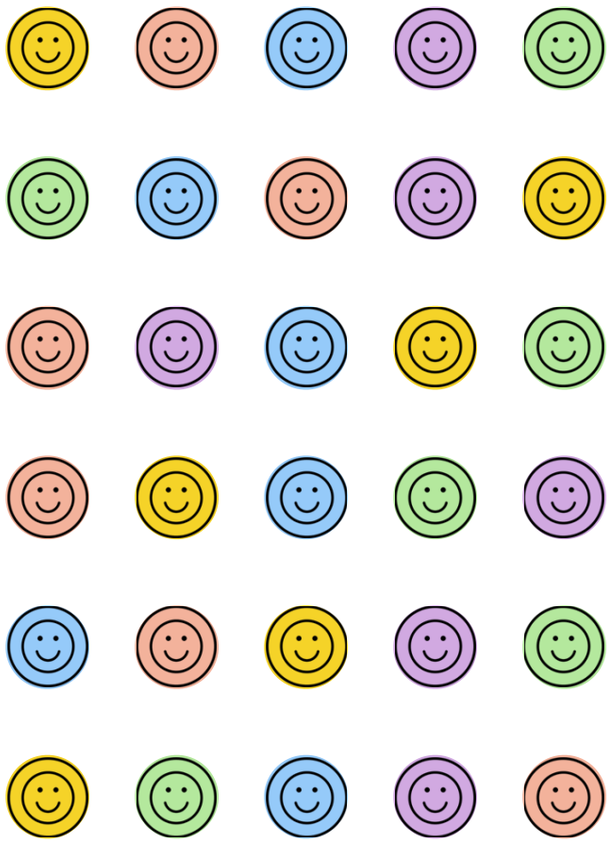
Examples of social choice functions

Examples of social choice functions

Scoring rules: Specified by a vector $(s_1, s_2, \ldots, s_m)$, this rule assigns s_j points to candidate i if i is placed at the jth position of voter j.

Examples of social choice functions

Scoring rules: Specified by a vector $(s_1, s_2, \ldots, s_m)$, this rule assigns s_j points to candidate i if i is placed at the jth position of voter j.





$s_3 + s_2 + s_3 + s_3 + s_1 + s_3$

Examples of social choice functions

Examples of social choice functions

Maximin: A candidate's maximin score is the number of candidates it beats in its **worst pairwise election**.

Examples of social choice functions

Maximin: A candidate's maximin score is the number of candidates it beats in its **worst pairwise election**.

Copeland: A candidate's Copeland score is the number of candidates it beats in pairwise elections.

The axiomatic perspective

The axiomatic perspective

Find social choice functions that satisfy desirable properties.

(Eg, unanimity; non-dictatorial;
condorcet consistency; strategyproof; etc.)

The axiomatic perspective

Find social choice functions that satisfy desirable properties.

(Eg, unanimity; non-dictatorial;
condorcet consistency; strategyproof; etc.)

Unfortunately, there are several results showing that certain combinations of desirable properties are **impossible to achieve simultaneously.**

The axiomatic perspective

Find social choice functions that satisfy desirable properties.

(Eg, unanimity; non-dictatorial;
condorcet consistency; strategyproof; etc.)

Unfortunately, there are several results showing that certain combinations of desirable properties are **impossible to achieve simultaneously.**

For example, there is no social choice function that is both non-dictatorial and strategyproof.*

The computational perspective

The computational perspective

Find social choice functions that are
easy to compute, and **hard to control or manipulate.**

The computational perspective

Find social choice functions that are
easy to compute, and **hard to control or manipulate.**

Also, rules that have many desirable properties but are hard to compute are studied often from the parameterized perspective.

The computational perspective

Find social choice functions that are
easy to compute, and **hard to control or manipulate.**

Also, rules that have many desirable properties but are hard to compute are studied often from the parameterized perspective.

On the flip side, some rules that are hard to manipulate or control in the traditional setting are tractable in the parameterized setting.

Preference Aggregation

Strategic behavior in the context of voting

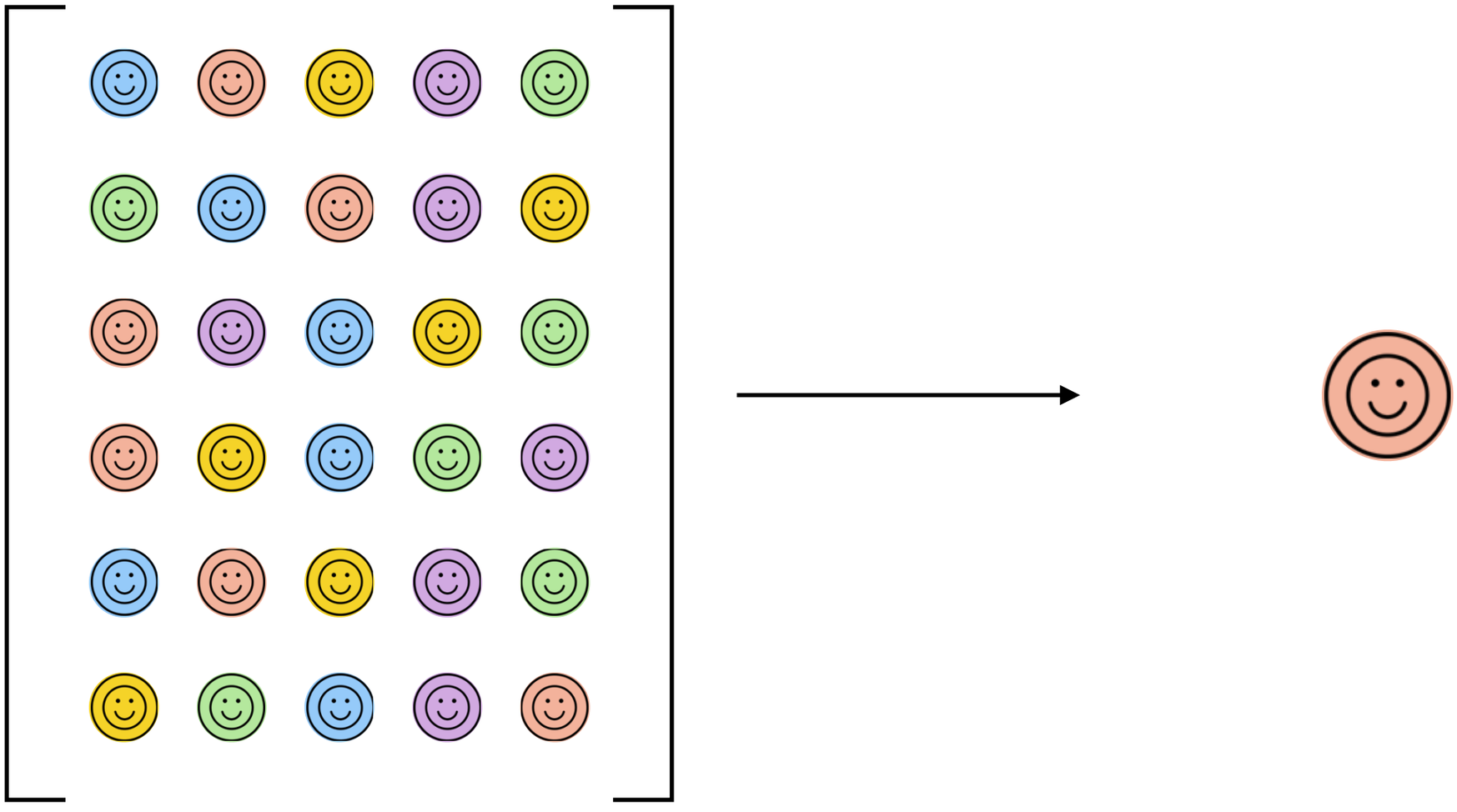
Finding a consensus from a crowd

Examples of strategic behavior

Tractable subclasses

Can a voter benefit by misrepresenting his vote?

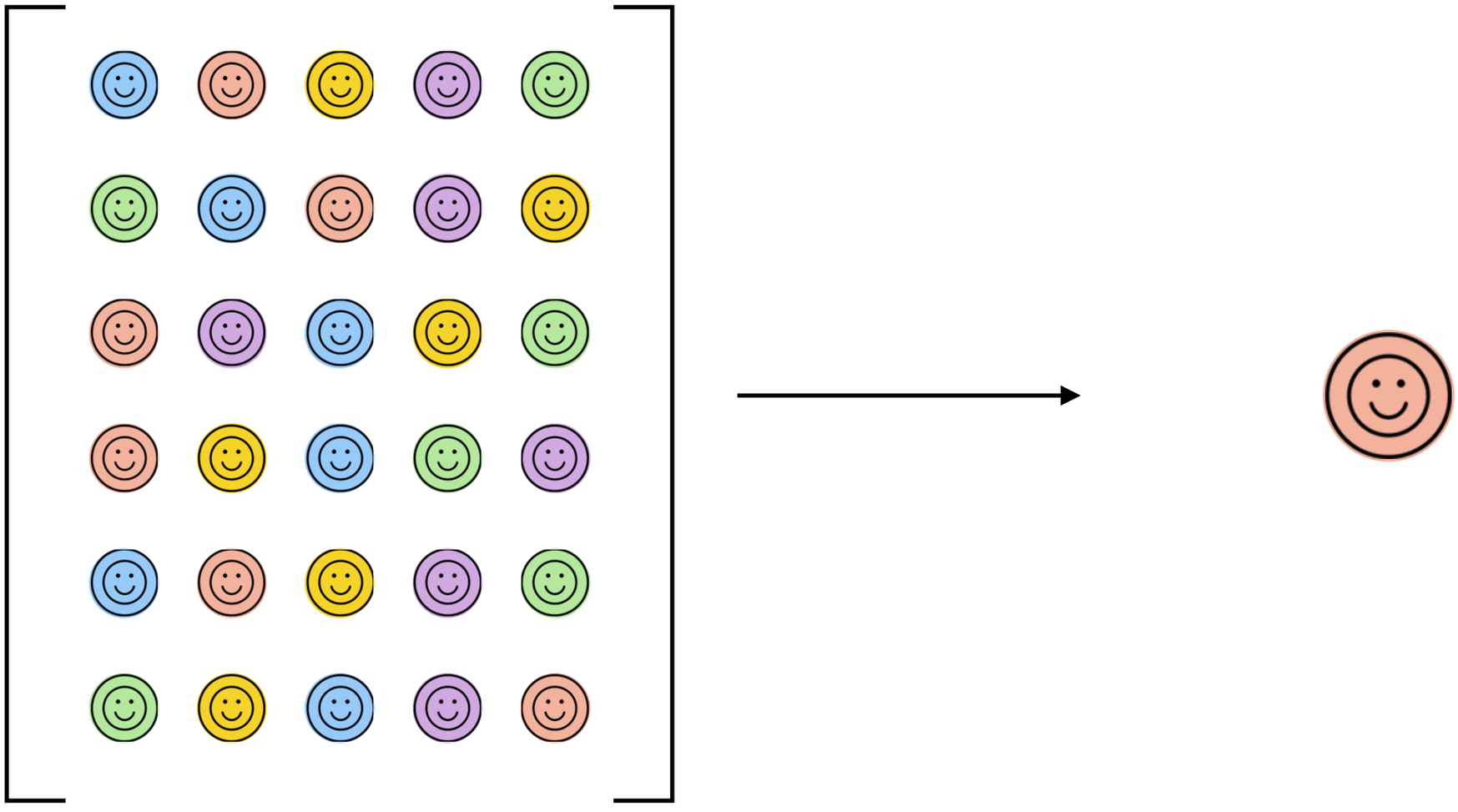
Can a voter benefit by misrepresenting his vote?



Plurality with tie-breaking order:



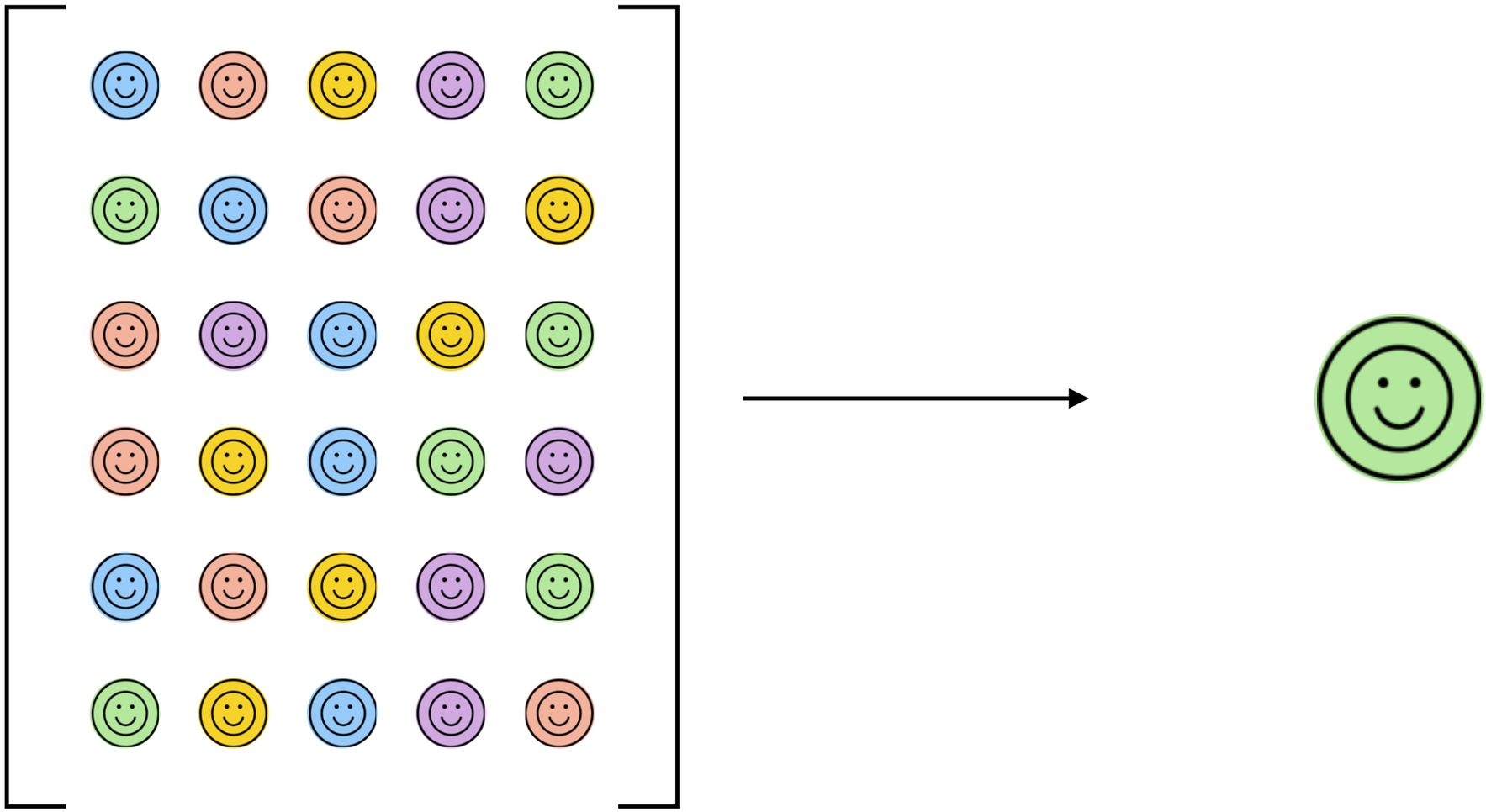
Can a voter benefit by misrepresenting his vote?



Plurality with tie-breaking order:



Can a voter benefit by misrepresenting his vote?



Plurality with tie-breaking order:



Manipulation

Let E be an election with outcome c.

A voter has a manipulation strategy if there exists a preference v*, different from his truthful preference v, for which the outcome of the election is a candidate that was **preferred over c in v.**

Manipulation: The computational problem

Let E be an election with outcome c,
and let p be a designated “favorite”.

Is there a collection of t preferences, say X, such that p wins the election E*, which consists of the original votes and the preferences in X?

Manipulation: The computational problem

Let E be an election with outcome c,
and let p be a designated "favorite".

Is there a collection of t preferences, say X, such that p wins the election E*, which consists of the original votes and the preferences in X?

A greedy manipulation strategy works for rules like Plurality.

On the other hand, Maximin is NP-hard to manipulate with two or more manipulators.

Manipulation: Parameterized by the number of candidates

For a bounded number of manipulators, we show that maximin, Copeland, Bucklin, ranked pairs, and a class of scoring rules that include the Borda voting rule admit polynomial kernels with the number of candidates as the parameter.

Dey, M., Narahari (AAMAS 2015)

Preference Aggregation

Strategic behavior in the context of voting

Finding a consensus from a crowd

Examples of strategic behavior

Tractable subclasses

Structure in the Electorate

When the preference profile exhibits special properties
(eg, single-peakedness, single-crossing, etc),
many known NP-hard problems become polynomially solvable.

Brandt, Brill, Hemaspaandra and Hemaspaandra; AAAI 2010

Recently, FPT algorithms have been established when the structure is generalized (eg, **bounded single-peaked width**).

Yang; AAMAS 2015

An interesting direction at this point is “deletion into good structure”, in the spirit of Graph Modification and Backdoors - that is, are there a small number of voters whose removal makes the profile single peaked?

Algorithms for Planar F-Deletion and Satisfiability

Algorithms for Planar F-Deletion and Satisfiability

