Fixed Parameter Algorithms

Dániel Marx

Tel Aviv University, Israel

International Workshop on Tractability

July 5, 2010, Microsoft Research, Cambridge, UK

Parameterized complexity

Main idea: Instead of expressing the running time as a function $T(n)$ of n, we express it as a function $T(n, k)$ of the input size n and some parameter k of the input.

In other words: we do not want to be efficient on all inputs of size n, only for those where k is small.

Parameterized complexity

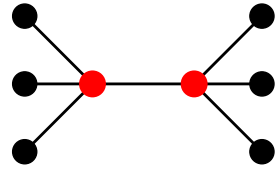
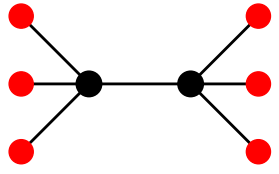
Main idea: Instead of expressing the running time as a function $T(n)$ of n, we express it as a function $T(n, k)$ of the input size n and some parameter k of the input.

In other words: we do not want to be efficient on all inputs of size n, only for those where k is small.

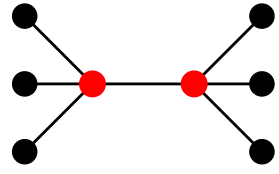
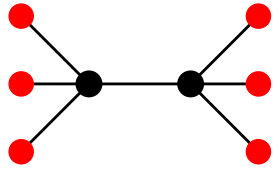
What can be the parameter k?

- The size k of the solution we are looking for.
- The maximum degree of the input graph.
- The diameter of the input graph.
- The length of clauses in the input Boolean formula.
- …

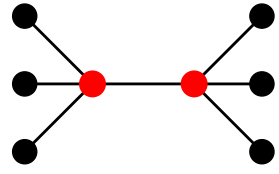
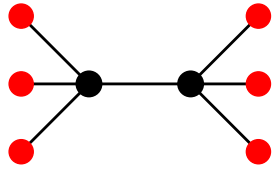
Parameterized complexity

Problem:	VERTEX COVER	INDEPENDENT SET
Input:	Graph G, integer k	Graph G, integer k
Question:	Is it possible to cover the edges with k vertices? 	Is it possible to find k independent vertices? 
Complexity:	NP-complete	NP-complete

Parameterized complexity

Problem:	VERTEX COVER	INDEPENDENT SET
Input:	Graph G, integer k	Graph G, integer k
Question:	Is it possible to cover the edges with k vertices? 	Is it possible to find k independent vertices? 
Complexity:	NP-complete	NP-complete
Complete enumeration:	$O(n^k)$ possibilities	$O(n^k)$ possibilities

Parameterized complexity

	VERTEX COVER	INDEPENDENT SET
Problem:	VERTEX COVER	INDEPENDENT SET
Input:	Graph G, integer k	Graph G, integer k
Question:	Is it possible to cover the edges with k vertices?	Is it possible to find k independent vertices?
		
Complexity:	NP-complete	NP-complete
Complete enumeration:	$O(n^k)$ possibilities	$O(n^k)$ possibilities
	$O(2^k n^2)$ algorithm exists	No $n^{o(k)}$ algorithm known
		

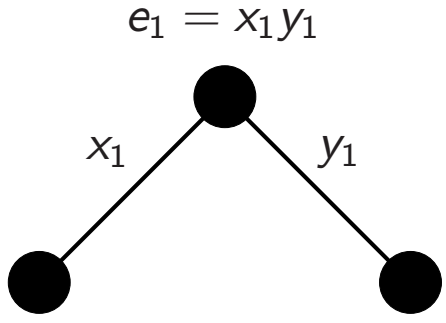
Bounded search tree method

Algorithm for VERTEX COVER:

$$e_1 = x_1 y_1$$

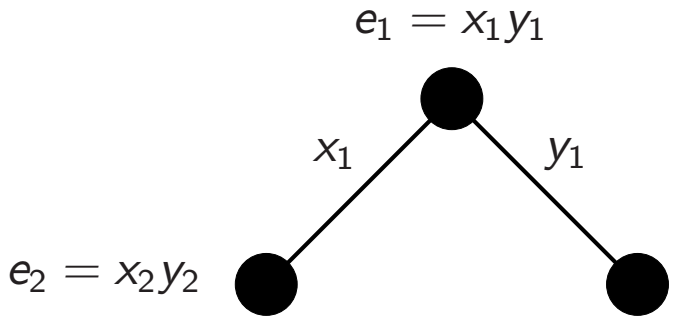
Bounded search tree method

Algorithm for VERTEX COVER:



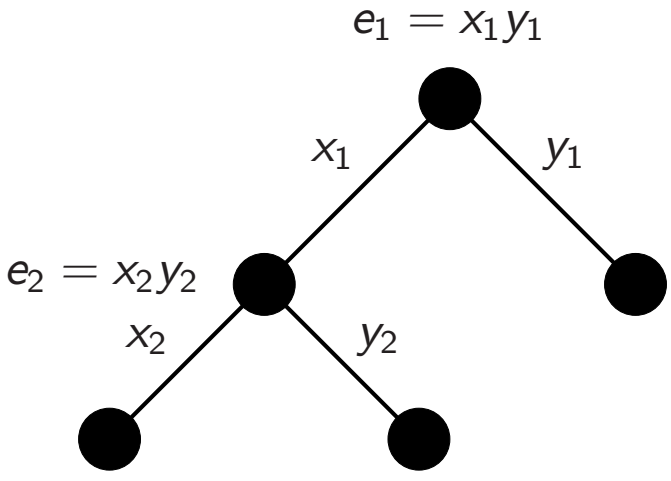
Bounded search tree method

Algorithm for VERTEX COVER:



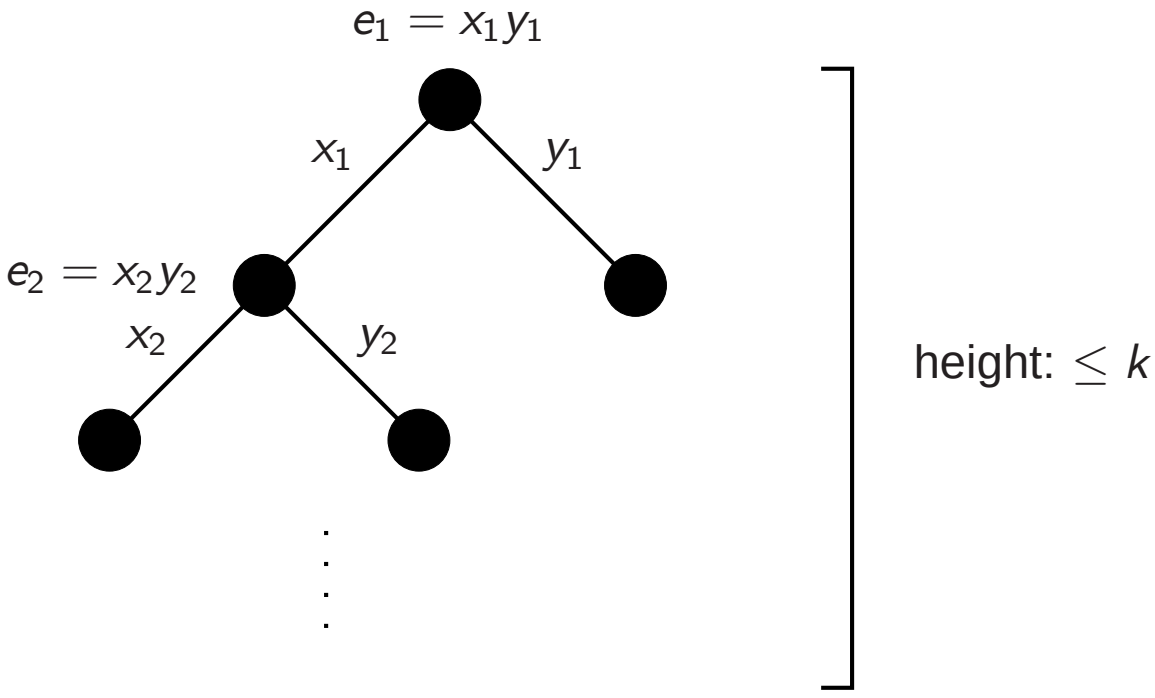
Bounded search tree method

Algorithm for VERTEX COVER:



Bounded search tree method

Algorithm for VERTEX COVER:



Height of the search tree is $\leq k \Rightarrow$ number of leaves is $\leq 2^k \Rightarrow$ complete search requires $2^k \cdot$ poly steps.

Fixed-parameter tractability

Definition: A **parameterization** of a decision problem is a function that assigns an integer parameter k to each input instance x.

The parameter can be

- explicit in the input (for example, if the parameter is the integer k appearing in the input (G, k) of VERTEX COVER), or
- implicit in the input (for example, if the parameter is the diameter d of the input graph G).

Main definition:

> A parameterized problem is **fixed-parameter tractable (FPT)** if there is an $f(k)n^c$ time algorithm for some constant c.

Fixed-parameter tractability

Definition: A **parameterization** of a decision problem is a function that assigns an integer parameter k to each input instance x.

Main definition:

> A parameterized problem is **fixed-parameter tractable (FPT)** if there is an $f(k)n^c$ time algorithm for some constant c.

Example: VERTEX COVER parameterized by the required size k is FPT:
we have seen that it can be solved in time $O(2^k \cdot n^2)$.

Better algorithms are known: e.g, $O(1.2832^k k + k|V|)$.

Main goal of parameterized complexity: to find FPT problems.

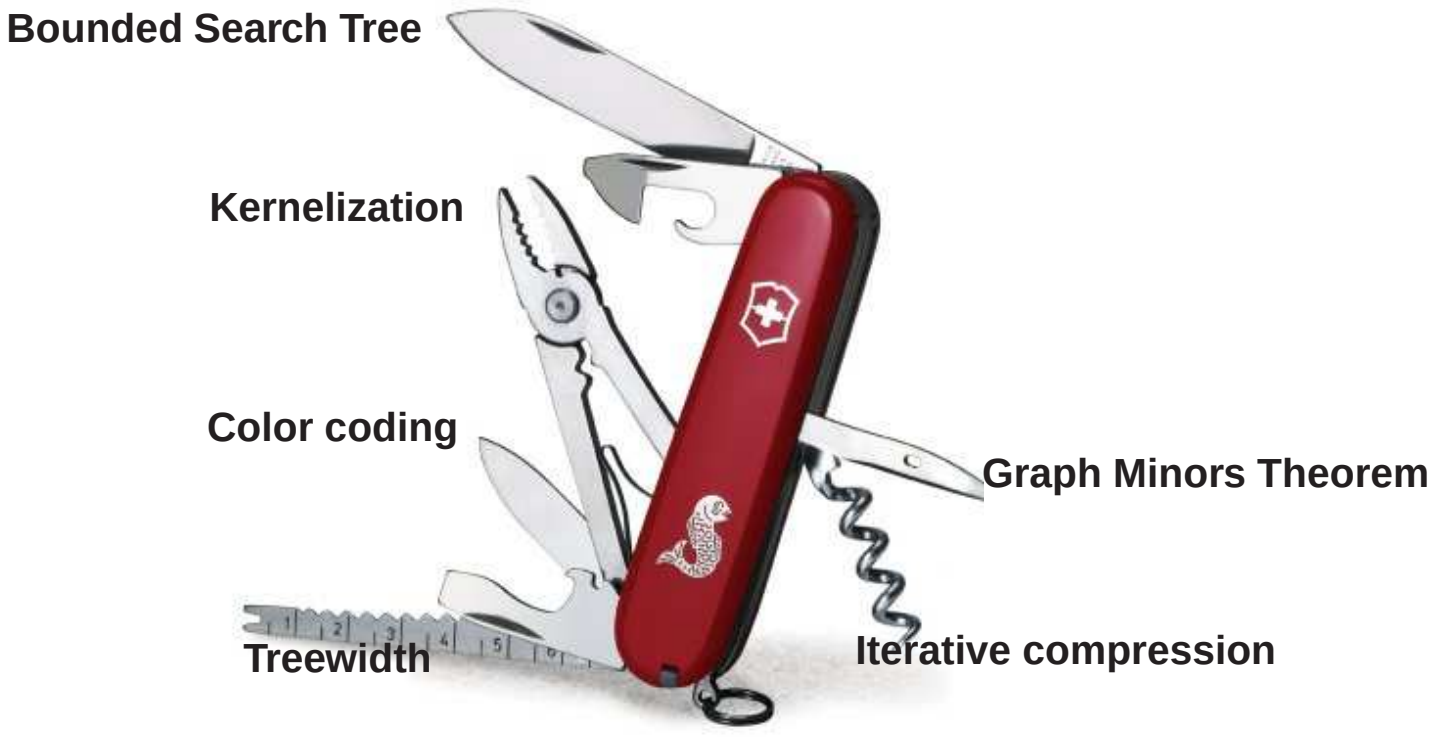
FPT problems

Examples of NP-hard problems that are FPT:

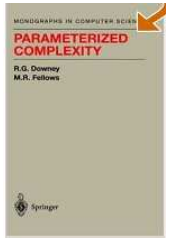
- Finding a vertex cover of size k.
- Finding a path of length k.
- Finding k disjoint triangles.
- Drawing the graph in the plane with k edge crossings.
- Finding disjoint paths that connect k pairs of points.
- …

FPT algorithmic techniques

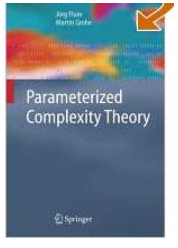
- Significant advances in the past 20 years or so (especially in recent years).
- Powerful toolbox for designing FPT algorithms:



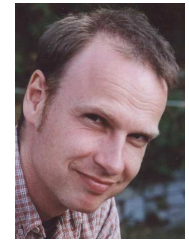
Books



Downey-Fellows: Parameterized Complexity, Springer, 1999



Flum-Grohe: Parameterized Complexity Theory, Springer, 2006

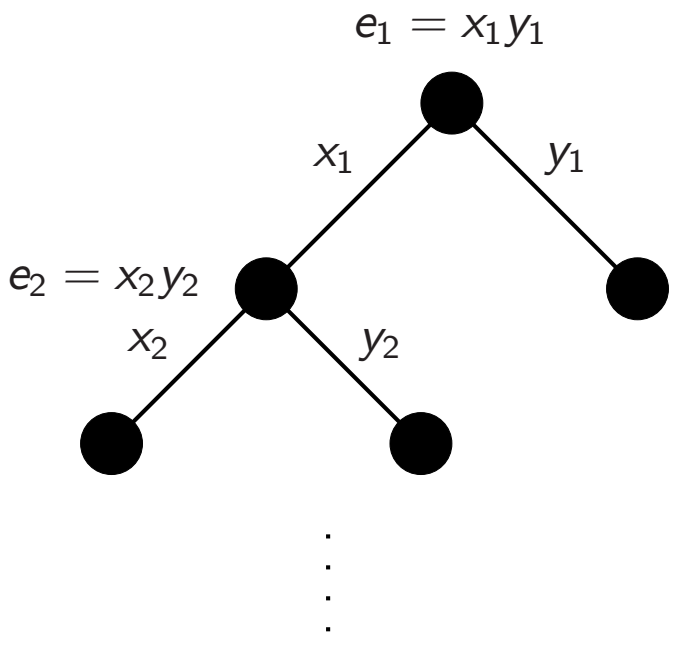


Niedermeier: Invitation to Fixed-Parameter Algorithms, Oxford University Press, 2006.

Bounded search trees

Bounded search tree method

Recall how we solved VERTEX COVER:



height: $\leq k$

If we branch into a bounded number of directions a bounded number of times, then we get an FPT algorithm!

MIN ONES SAT

VERTEX COVER can be interpreted as a CSP problem:

> "Given constraints $x \vee y$ on Boolean variables, find a satisfying assignment of weight at most k."

More generally:

MIN ONES SAT: Given an r-SAT formula and an integer k, find a satisfying assignment of weight at most k.

MIN ONES SAT

VERTEX COVER can be interpreted as a CSP problem:

> "Given constraints $x \vee y$ on Boolean variables, find a satisfying assignment of weight at most k."

More generally:

MIN ONES SAT: Given an r-SAT formula and an integer k, find a satisfying assignment of weight at most k.

The bounded search tree approach gives:

Fact: MIN ONES SAT on r-SAT formulas can be solved in time $O(r^k \cdot n)$, i.e., problem is FPT parameterized jointly by r and k.

Fact: MIN ONES SAT(Γ) is FPT parameterized by k for every Γ.

Color coding

k-PATH

Task: Given a graph G, an integer k, two vertices s, t, find a **simple** s-t path with exactly k internal vertices.

Note: Finding such a **walk** can be done easily in polynomial time.

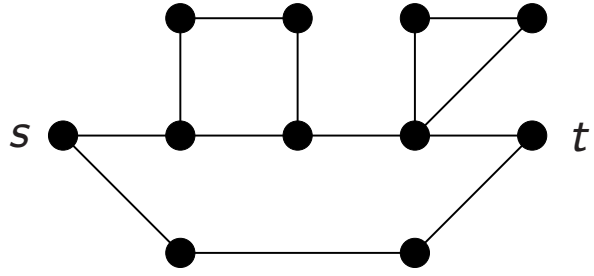
Note: The problem is clearly NP-hard, as it contains the s-t HAMILTONIAN PATH problem.

The k-PATH algorithm can be used to check if there is a cycle of length exactly k in the graph.

Fact: [Alon-Yuster-Zwick 1995] k-PATH can be solved in time $c^k \cdot n^{O(1)}$ for some c.

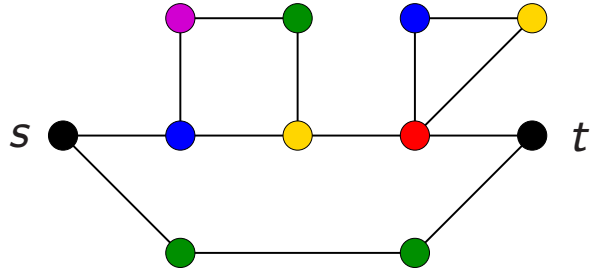
k-PATH

- Assign colors from $\{1, \ldots, k\}$ to vertices $V(G) \setminus \{s, t\}$ uniformly and independently at random.



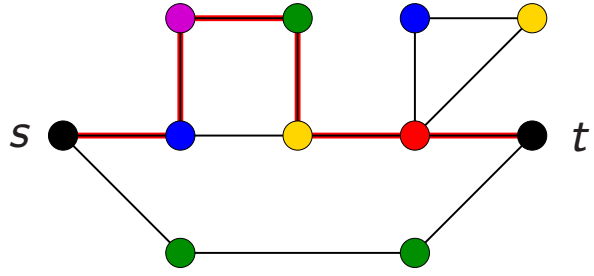
k-PATH

- Assign colors from $\{1, \ldots, k\}$ to vertices $V(G) \setminus \{s, t\}$ uniformly and independently at random.



k-PATH

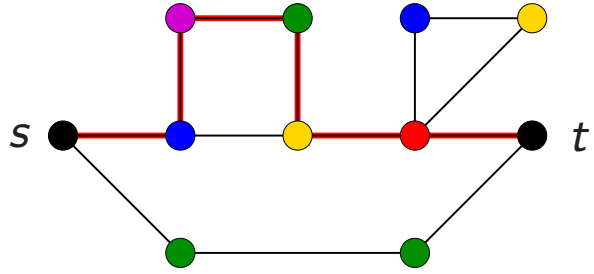
- Assign colors from $\{1, \ldots, k\}$ to vertices $V(G) \setminus \{s, t\}$ uniformly and independently at random.



- Check if there is a **colorful** s-t path: a path where each color appears exactly once on the internal vertices; output “YES” or “NO”.

k-PATH

- Assign colors from $\{1, \ldots, k\}$ to vertices $V(G) \setminus \{s, t\}$ uniformly and independently at random.



- Check if there is a **colorful** s-t path: a path where each color appears exactly once on the internal vertices; output "YES" or "NO".
  - If there is no s-t k-path: no such colorful path exists $\Rightarrow$ "NO".
  - If there is an s-t k-path: the probability that such a path is colorful is

$$\frac{k!}{k^k} > \frac{\left(\frac{k}{e}\right)^k}{k^k} = e^{-k},$$

  thus the algorithm outputs "YES" with at least that probability.

Error probability

- **Useful fact:** If the probability of success is at least p, then the probability that the algorithm **does not** say "YES" after $1/p$ repetitions is at most

$$(1-p)^{1/p} < (e^{-p})^{1/p} = 1/e \approx 0.38$$

- Thus if $p > e^{-k}$, then error probability is at most $1/e$ after e^k repetitions.
- Repeating the whole algorithm a constant number of times can make the error probability an arbitrary small constant.
- For example, by trying $100 \cdot e^k$ random colorings, the probability of a wrong answer is at most $1/e^{100}$.

Error probability

- **Useful fact:** If the probability of success is at least p, then the probability that the algorithm **does not** say "YES" after $1/p$ repetitions is at most

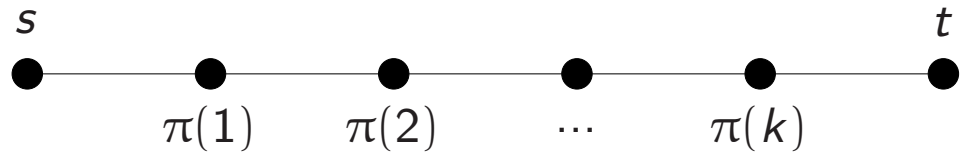
$$(1-p)^{1/p} < (e^{-p})^{1/p} = 1/e \approx 0.38$$

- Thus if $p > e^{-k}$, then error probability is at most $1/e$ after e^k repetitions.
- Repeating the whole algorithm a constant number of times can make the error probability an arbitrary small constant.
- For example, by trying $100 \cdot e^k$ random colorings, the probability of a wrong answer is at most $1/e^{100}$.

It remains to see how a colorful s-t path can be found.

Finding a colorful path

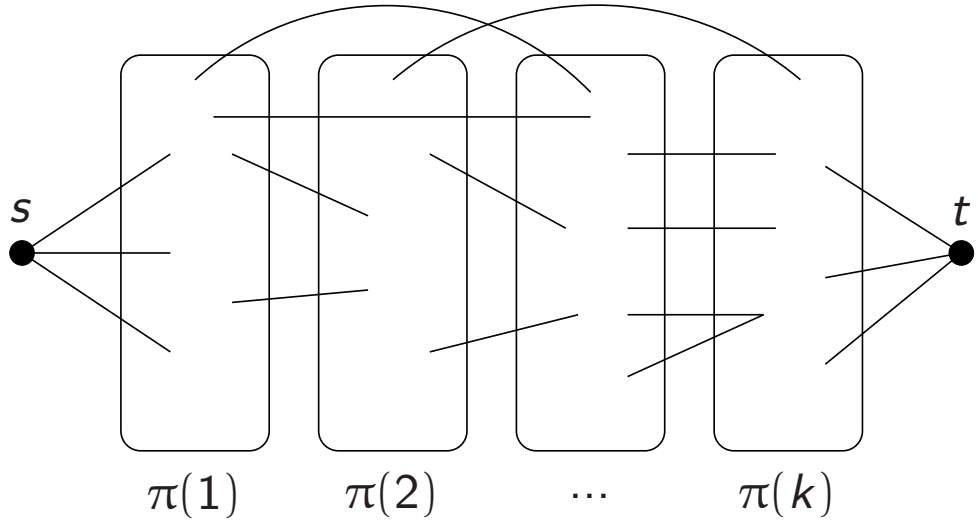
The colors encountered on a colorful s-t path form a permutation π of $\{1, 2, \ldots, k\}$:



We try all possible $k!$ permutations. For a fixed π, it is easy to check if there is a path with this order of colors.

Finding a colorful path

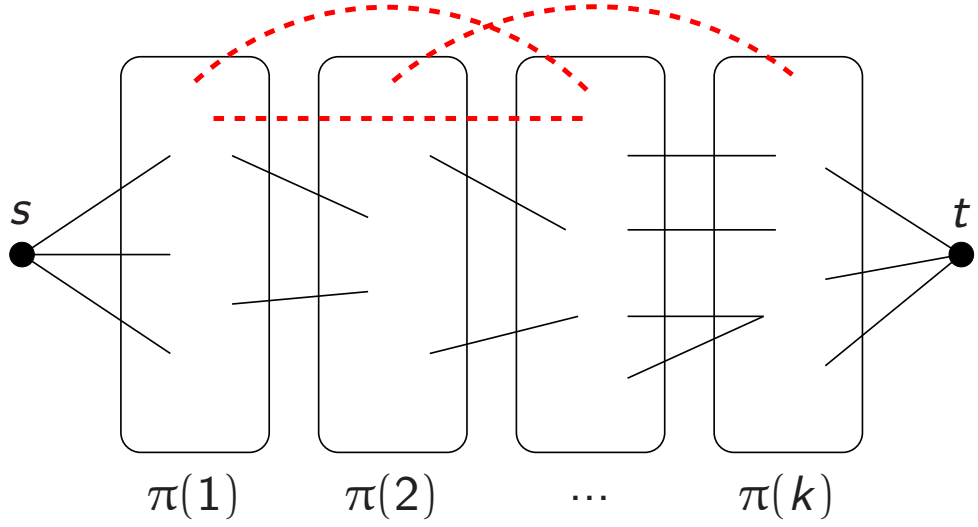
We try all possible $k!$ permutations. For a fixed π, it is easy to check if there is a path with this order of colors.



- Edges connecting nonadjacent color classes are removed.
- The remaining edges are directed.
- All we need to check if there is a directed s-t path.
- Running time is $O(k! \cdot |E(G)|)$ (can be improved to $O(2^k \cdot |E(G)|)$).

Finding a colorful path

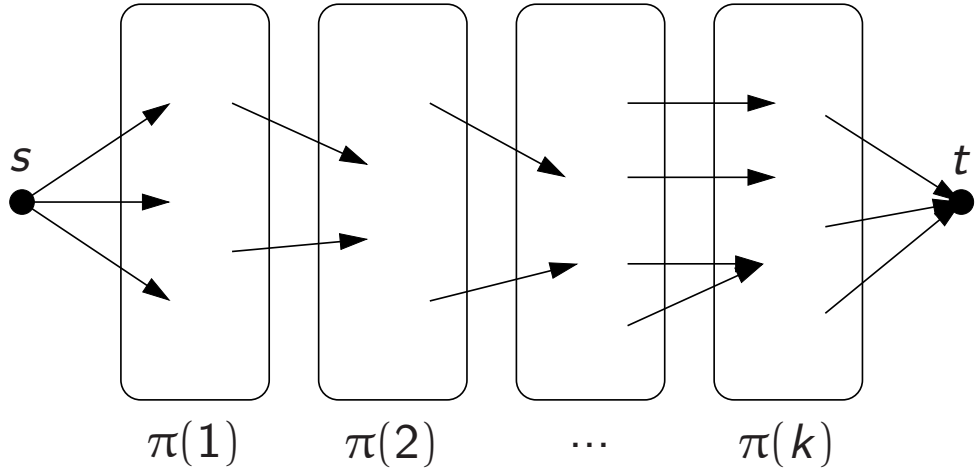
We try all possible $k!$ permutations. For a fixed π, it is easy to check if there is a path with this order of colors.



- Edges connecting nonadjacent color classes are removed.
- The remaining edges are directed.
- All we need to check if there is a directed s-t path.
- Running time is $O(k! \cdot |E(G)|)$ (can be improved to $O(2^k \cdot |E(G)|)$).

Finding a colorful path

We try all possible $k!$ permutations. For a fixed π, it is easy to check if there is a path with this order of colors.



- Edges connecting nonadjacent color classes are removed.
- The remaining edges are directed.
- All we need to check if there is a directed s-t path.
- Running time is $O(k! \cdot |E(G)|)$ (can be improved to $O(2^k \cdot |E(G)|)$).

Color coding

- Algorithm has small probability of error, but can be derandomized using k-perfect hash functions.
- More generally: finding graphs of low treewidth.
- Color coding is useful to ensure that certain objects are disjoint.

Iterative compression

BIPARTITE DELETION

BIPARTITE DELETION: Given a graph G and an integer k, delete k vertices to make G bipartite (2-colorable).

Fixed-parameteter tractability was open until Reed, Smith, and Vetta showed it in 2004 using the technique of **iterative compression.**

BIPARTITE DELETION

Solution based on iterative compression:

- **Step 1:**

  Solve the **annotated problem** for bipartite graphs:

  Given a bipartite graph G, two sets $B, W \subseteq V(G)$, and an integer k, find a set S of at most k vertices such that $G \setminus S$ has a 2-coloring where $B \setminus S$ is black and $W \setminus S$ is white.

BIPARTITE DELETION

Solution based on iterative compression:

- **Step 1:**

  Solve the **annotated problem** for bipartite graphs:

  Given a bipartite graph G, two sets $B, W \subseteq V(G)$, and an integer k, find a set S of at most k vertices such that $G \setminus S$ has a 2-coloring where $B \setminus S$ is black and $W \setminus S$ is white.

- **Step 2:**

  Solve the **compression problem** for general graphs:

  Given a graph G, an integer k, and a set S' of $k+1$ vertices such that $G \setminus S'$ is bipartite, find a set S of k vertices such that $G \setminus S$ is bipartite.

BIPARTITE DELETION

Solution based on iterative compression:

- **Step 1:**

  Solve the **annotated problem** for bipartite graphs:

  Given a bipartite graph G, two sets $B, W \subseteq V(G)$, and an integer k, find a set S of at most k vertices such that $G \setminus S$ has a 2-coloring where $B \setminus S$ is black and $W \setminus S$ is white.

- **Step 2:**

  Solve the **compression problem** for general graphs:

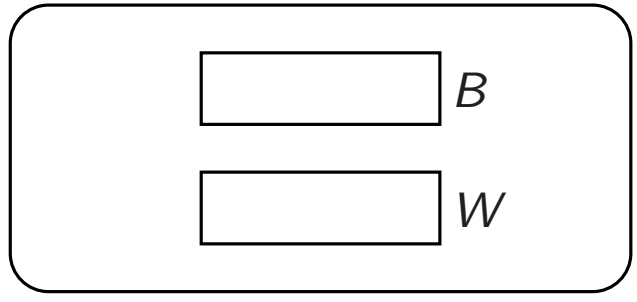
  Given a graph G, an integer k, and a set S' of $k+1$ vertices such that $G \setminus S'$ is bipartite, find a set S of k vertices such that $G \setminus S$ is bipartite.

- **Step 3:**

  Apply the magic of iterative compression...

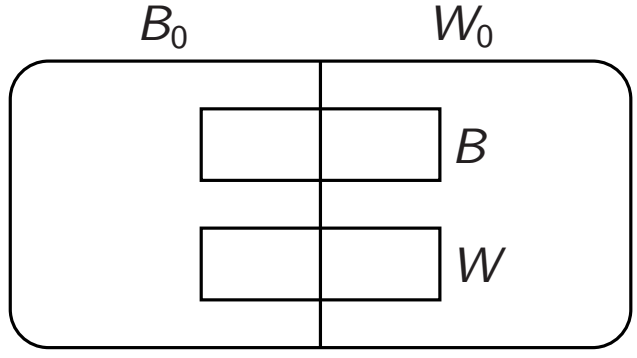
Step 1: The annotated problem

Given a bipartite graph G, two sets $B, W \subseteq V(G)$, and an integer k, find a set S of at most k vertices such that $G \setminus S$ has a 2-coloring where $B \setminus S$ is black and $W \setminus S$ is white.



Step 1: The annotated problem

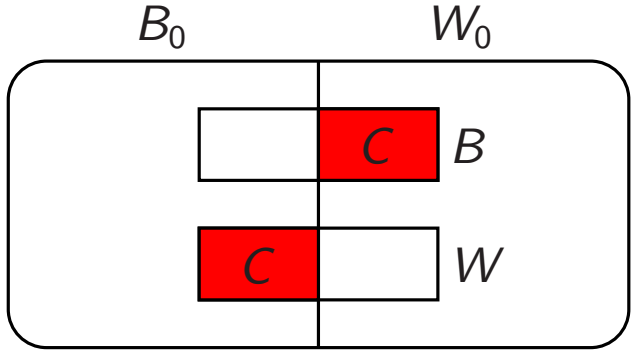
Given a bipartite graph G, two sets $B, W \subseteq V(G)$, and an integer k, find a set S of at most k vertices such that $G \setminus S$ has a 2-coloring where $B \setminus S$ is black and $W \setminus S$ is white.



Find an arbitrary 2-coloring (B_0, W_0) of G.

Step 1: The annotated problem

Given a bipartite graph G, two sets $B, W \subseteq V(G)$, and an integer k, find a set S of at most k vertices such that $G \setminus S$ has a 2-coloring where $B \setminus S$ is black and $W \setminus S$ is white.

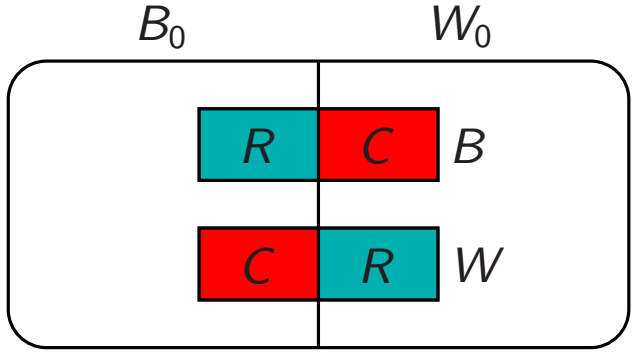


Find an arbitrary 2-coloring (B_0, W_0) of G.

$C := (B_0 \cap W) \cup (W_0 \cap B)$ should change color, while $R := (B_0 \cap B) \cup (W_0 \cap W)$ should remain the same color.

Step 1: The annotated problem

Given a bipartite graph G, two sets $B, W \subseteq V(G)$, and an integer k, find a set S of at most k vertices such that $G \setminus S$ has a 2-coloring where $B \setminus S$ is black and $W \setminus S$ is white.



Find an arbitrary 2-coloring (B_0, W_0) of G.
$C := (B_0 \cap W) \cup (W_0 \cap B)$ should change color, while $R := (B_0 \cap B) \cup (W_0 \cap W)$ should remain the same color.

Lemma: $G \setminus S$ has the required 2-coloring if and only if S separates C and R, i.e., no component of $G \setminus S$ contains vertices from both $C \setminus S$ and $R \setminus S$.

Step 1: The annotated problem

Lemma: $G \setminus S$ has the required 2-coloring if and only if S separates C and R, i.e., no component of $G \setminus S$ contains vertices from both $C \setminus S$ and $R \setminus S$.

Proof:

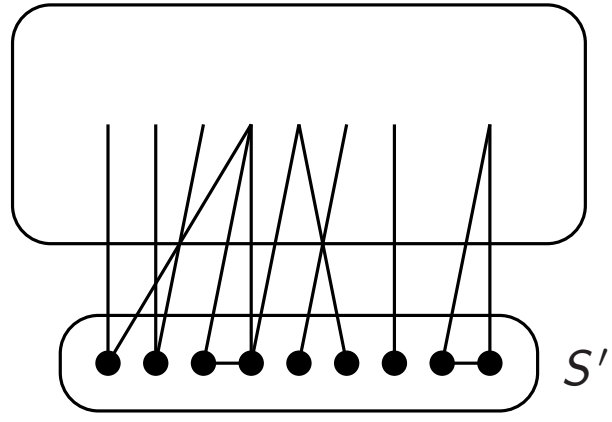
$\Rightarrow$ In a 2-coloring of $G \setminus S$, each vertex either remained the same color or changed color. Adjacent vertices do the same, thus every component either changed or remained.

$\Leftarrow$ Flip the coloring of those components of $G \setminus S$ that contain vertices from $C \setminus S$. No vertex of R is flipped.

Algorithm: Using max-flow min-cut techniques, we can check if there is a set S that separates C and R. It can be done in time $O(k|E(G)|)$ using k iterations of the Ford-Fulkerson algorithm.

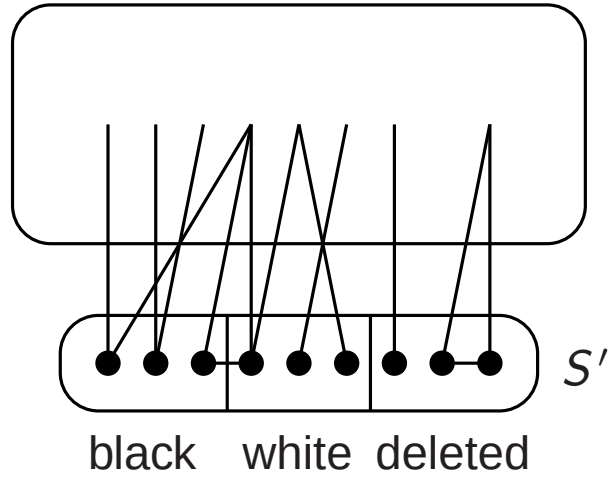
Step 2: The compression problem

Given a graph G, an integer k, and a set S' of $k+1$ vertices such that $G \setminus S'$ is bipartite, find a set S of k vertices such that $G \setminus S$ is bipartite.



Step 2: The compression problem

Given a graph G, an integer k, and a set S' of $k+1$ vertices such that $G \setminus S'$ is bipartite, find a set S of k vertices such that $G \setminus S$ is bipartite.

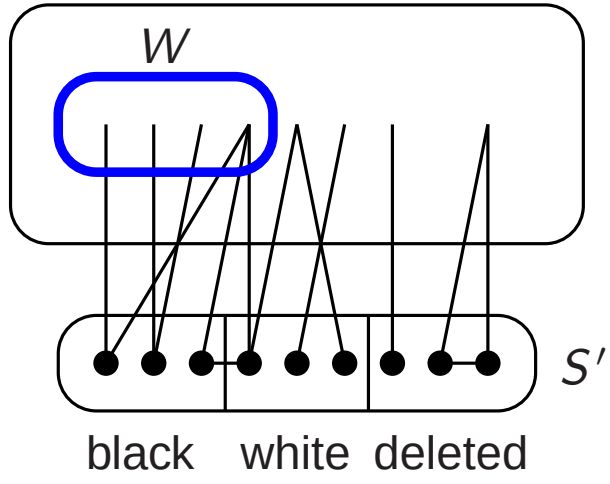


Branch into 3^{k+1} cases: each vertex of S' is either black, white, or deleted.

Trivial check: no edge between two black or two white vertices.

Step 2: The compression problem

Given a graph G, an integer k, and a set S' of $k+1$ vertices such that $G \setminus S'$ is bipartite, find a set S of k vertices such that $G \setminus S$ is bipartite.



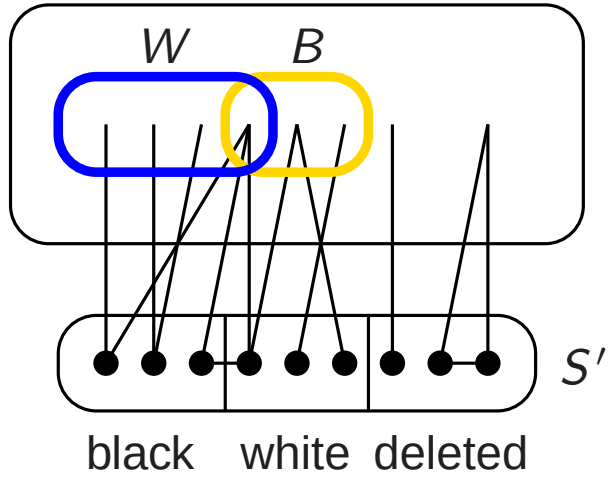
Branch into 3^{k+1} cases: each vertex of S' is either black, white, or deleted.

Trivial check: no edge between two black or two white vertices.

Neighbors of the black vertices in S' should be white and the neighbors of the white vertices in S' should be black.

Step 2: The compression problem

Given a graph G, an integer k, and a set S' of $k+1$ vertices such that $G \setminus S'$ is bipartite, find a set S of k vertices such that $G \setminus S$ is bipartite.



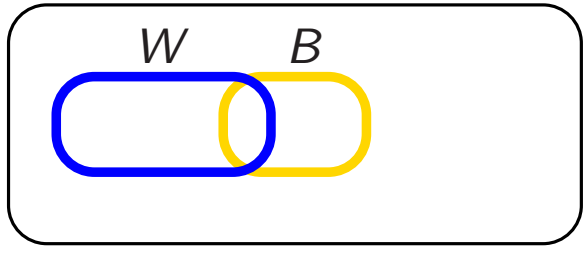
Branch into 3^{k+1} cases: each vertex of S' is either black, white, or deleted.

Trivial check: no edge between two black or two white vertices.

Neighbors of the black vertices in S' should be white and the neighbors of the white vertices in S' should be black.

Step 2: The compression problem

Given a graph G, an integer k, and a set S' of $k+1$ vertices such that $G \setminus S'$ is bipartite, find a set S of k vertices such that $G \setminus S$ is bipartite.



The vertices of S' can be disregarded. Thus we need to solve the annotated problem on the bipartite graph $G \setminus S'$.

Running time: $O(3^k \cdot k|E(G)|)$ time.

Step 3: Iterative compression

How do we get a solution of size $k+1$?

Step 3: Iterative compression

How do we get a solution of size $k+1$?

Dirty trick : we get it for free!

Step 3: Iterative compression

How do we get a solution of size $k+1$?

Dirty trick : we get it for free!

Let $V(G) = \{v_1, \ldots, v_n\}$ and let G_i be the graph induced by $\{v_1, \ldots, v_i\}$.

For every i, we find a set S_i of size k such that $G_i \setminus S_i$ is bipartite.

- For G_k, the set $S_k = \{v_1, \ldots, v_k\}$ is a trivial solution.
- If S_{i-1} is known, then $S_{i-1} \cup \{v_i\}$ is a set of size $k+1$ whose deletion makes G_i bipartite $\Rightarrow$ We can use the compression algorithm to find a suitable S_i in time $O(3^k \cdot k|E(G_i)|)$.

Step 3: Iterative Compression

Bipartite-Deletion(G, k)

1. $S_k = \{v_1, \ldots, v_k\}$
2. for $i := k + 1$ to n
3. Invariant: $G_{i-1} \setminus S_{i-1}$ is bipartite.
4. Call Compression$(G_i, S_{i-1} \cup \{v_i\})$
5. If the answer is “NO” $\Rightarrow$ return “NO”
6. If the answer is a set $X \Rightarrow S_i := X$
7. Return the set S_n

Running time: the compression algorithm is called n times and everything else can be done in linear time

$\Rightarrow O(3^k \cdot k|V(G)| \cdot |E(G)|)$ time algorithm.

Almost 2-SAT

BIPARTITE EDGE DELETION: Delete k edges to make the graph bipartite.

Fact: BIPARTITE EDGE DELETION can be solved in time $O(2^k \cdot k|V(G)| \cdot |E(G)|)$.

BIPARTITE EDGE DELETION can be seen as a CSP problem:

> "Given Boolean variables and constraints of the form $x \neq y$, find an assignment that satisfies all but at most k constraints."

Almost 2-SAT

BIPARTITE EDGE DELETION: Delete k edges to make the graph bipartite.

Fact: BIPARTITE EDGE DELETION can be solved in time $O(2^k \cdot k|V(G)| \cdot |E(G)|)$.

BIPARTITE EDGE DELETION can be seen as a CSP problem:

> "Given Boolean variables and constraints of the form $x \neq y$, find an assignment that satisfies all but at most k constraints."

More generally:

ALMOST 2-SAT: Given a 2SAT formula and an integer k, find an assignment that satisfies all but at most k clauses.

Fact: [O'Sullivan and Razgon 2008] ALMOST 2-SAT is FPT.

Next: A simplified $7^k \cdot n^{O(1)}$ time algorithm.

Almost 2-Sat

Solution based on iterative compression:

- **Step 1:**

  Solve the **annotated problem** for 0-valid formulas

  Given a 0-valid formula φ, two sets V_0, V_1 of variables, and an integer k, find a set S of at most k clauses such that $\varphi \setminus S$ has a satisfying assignment where V_0 is 0 and V_1 is 1.

- **Step 2:**

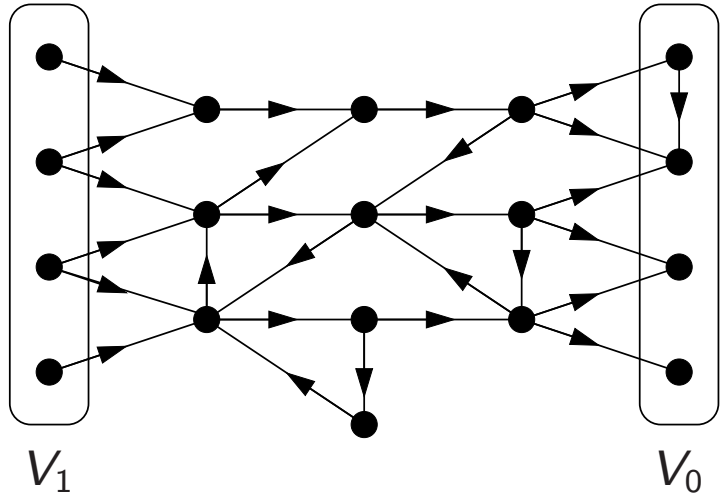
  Solve the **compression problem** for general formulas:

  Given a formula φ, an integer k, and a set S' of $k+1$ clauses such that $G \setminus S$ is satisfiable, find a set S of k clauses such that $G \setminus S$ is satisfiable.

Step 1: The annotated problem

Given a 0-valid formula φ, two sets V_0, V_1 of variables, and an integer k, find a set S of at most k clauses such that $\varphi \setminus S$ has a satisfying assignment where V_0 is 0 and V_1 is 1.

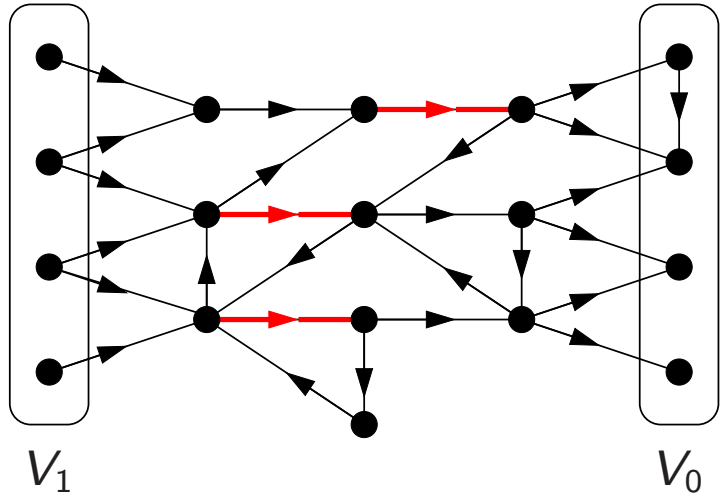
If φ is 0-valid, then clauses are of form $x \rightarrow y$ and $\overline{x} \vee \overline{y}$. Graph of implications:



Step 1: The annotated problem

Given a 0-valid formula φ, two sets V_0, V_1 of variables, and an integer k, find a set S of at most k clauses such that $\varphi \setminus S$ has a satisfying assignment where V_0 is 0 and V_1 is 1.

If φ is 0-valid, then clauses are of form $x \rightarrow y$ and $\overline{x} \vee \overline{y}$. Graph of implications:

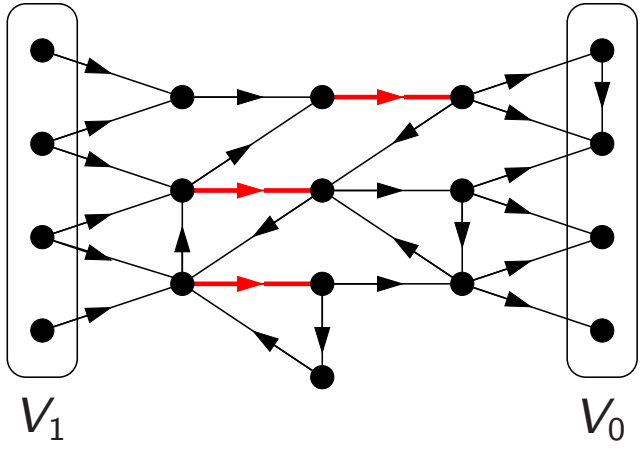


Observation: Any solution S is a (V_1, V_0)-cut (necessary, but not sufficient).

Step 1: The annotated problem

Let λ be the size of the minimum (V_1, V_0)-cut.

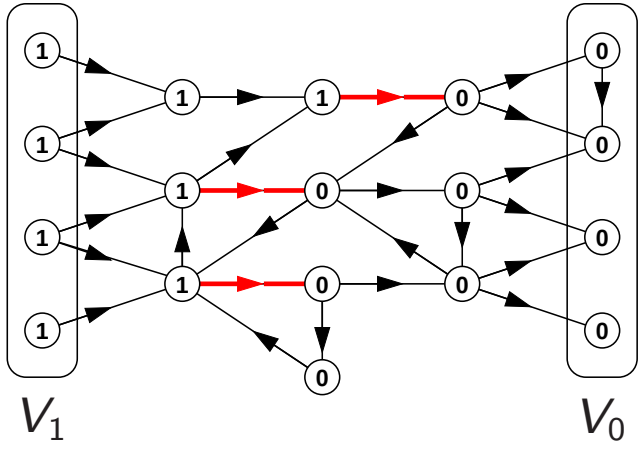
Fact: There is a unique cut of size λ such that the set of vertices reachable from V_1 has to be reachable in every cut of size λ.



Step 1: The annotated problem

Let λ be the size of the minimum (V_1, V_0)-cut.

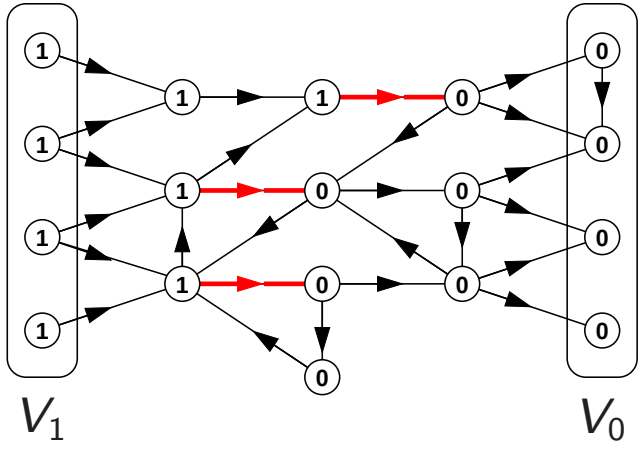
Fact: There is a unique cut of size λ such that the set of vertices reachable from V_1 has to be reachable in every cut of size λ.



Step 1: The annotated problem

Let λ be the size of the minimum (V_1, V_0)-cut.

Fact: There is a unique cut of size λ such that the set of vertices reachable from V_1 has to be reachable in every cut of size λ.

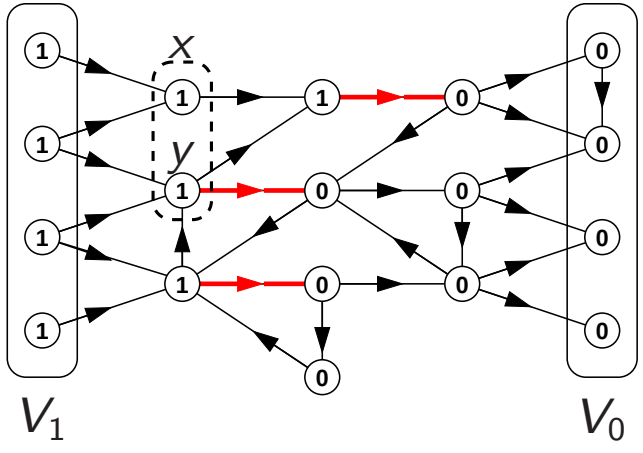


If this assignment satisfies every $\overline{x} \vee \overline{y}$ clause, then we are done. Otherwise we branch into 3 directions.

Step 1: The annotated problem

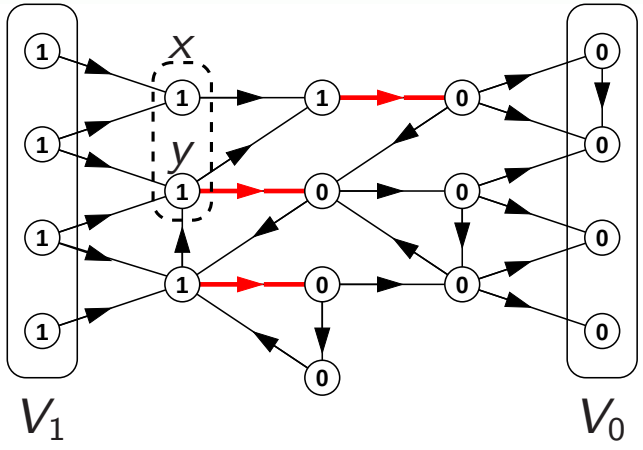
Let λ be the size of the minimum (V_1, V_0)-cut.

Fact: There is a unique cut of size λ such that the set of vertices reachable from V_1 has to be reachable in every cut of size λ.



If this assignment satisfies every $\overline{x} \vee \overline{y}$ clause, then we are done. Otherwise we branch into 3 directions.

Step 1: The annotated problem



If this assignment satisfies every $\overline{x} \vee \overline{y}$ clause, then we are done. Otherwise we branch into 3 directions.

1. Remove the clause $\overline{x} \vee \overline{y}$ and decrease k.
2. Put x into V_0 (increases λ).
3. Put y into V_0 (increases λ).

We can branch at most k times: λ cannot be larger than k

$\Rightarrow O(3^k \cdot m)$ time algorithm for the annotated problem.

Step 2: The compression problem

Given a formula φ, an integer k, and a set S' of $k+1$ clauses such that $\varphi \setminus S'$ is satisfiable, find a set S of k clauses such that $\varphi \setminus S'$ is satisfiable.

Small trick: by negating variables, we can assume that $\varphi \setminus S'$ is 0-valid.

Step 2: The compression problem

Given a formula φ, an integer k, and a set S' of $k+1$ clauses such that $\varphi \setminus S'$ is satisfiable, find a set S of k clauses such that $\varphi \setminus S'$ is satisfiable.

Small trick: by negating variables, we can assume that $\varphi \setminus S'$ is 0-valid.

- Let V be the variables involved in S'. Let us guess the values of these variables in the solution ($\leq 2^{2(k+1)}$ possibilities).
- Suppose that $V = V_0 \cup V_1$. Remove S' and decrease k by the number of clauses in S' that are unsatisfied by (V_0, V_1).
- Task: find a solution of $\varphi \setminus S'$ that is compatible with V_0 and V_1
  $\Rightarrow$ can be solved in time $O(3^k \cdot k \cdot m)$ by the annotated problem.

Running time: $O(2^{2(k+1)} \cdot 3^k \cdot k \cdot m) = O(12^k \cdot k \cdot m)$.

Slightly better analysis gives a $O(7^k \cdot k \cdot m)$ bound.

Step 3: Iterative compression

We get for free the solution of size $k+1$.

Let $C_1, \ldots, C_m$ be the clauses and let φ_i be the formula containing only the first i clauses.

For every i, we find a set S_i of size k such that $\varphi_i \setminus S_i$ is satisfiable.

- For φ_k, the set $S_k = \{C_1, \ldots, C_k\}$ is a trivial solution.
- If S_{i-1} is known, then $S_{i-1} \cup \{C_i\}$ is a set of size $k+1$ whose deletion makes φ_i satisfiable $\Rightarrow$ We can use the compression algorithm to find a suitable S_i in time $O(7^k \cdot km)$.

Running time: $O(7^k \cdot k \cdot m^2)$ for a formula with m clauses.

Kernelization

Kernelization for VERTEX COVER

General strategy: We devise a list of reduction rules, and show that if none of the rules can be applied and the size of the instance is still larger than $s(k)$, then the answer is trivial.

$\Rightarrow$ We can assume that the instance is of size at most $s(k)$ and can solved by brute force $\Rightarrow$ FPT parameterized by k.

Kernelization for VERTEX COVER

General strategy: We devise a list of reduction rules, and show that if none of the rules can be applied and the size of the instance is still larger than $s(k)$, then the answer is trivial.

$\Rightarrow$ We can assume that the instance is of size at most $s(k)$ and can solved by brute force $\Rightarrow$ FPT parameterized by k.

Reduction rules for VERTEX COVER instance (G, k):

Rule 1: If v is an isolated vertex $\Rightarrow (G \setminus v, k)$

Rule 2: If $d(v) > k \Rightarrow (G \setminus v, k-1)$

Kernelization for VERTEX COVER

General strategy: We devise a list of reduction rules, and show that if none of the rules can be applied and the size of the instance is still larger than $s(k)$, then the answer is trivial.

$\Rightarrow$ We can assume that the instance is of size at most $s(k)$ and can solved by brute force $\Rightarrow$ FPT parameterized by k.

Reduction rules for VERTEX COVER instance (G, k):

Rule 1: If v is an isolated vertex $\Rightarrow (G \setminus v, k)$
Rule 2: If $d(v) > k \Rightarrow (G \setminus v, k-1)$

If neither Rule 1 nor Rule 2 can be applied:

- If $|V(G)| > k(k+1) \Rightarrow$ There is no solution (every vertex should be the neighbor of at least one vertex of the cover).
- Otherwise, $|V(G)| \leq k(k+1)$ and we have a $k(k+1)$ vertex kernel.

Kernelization

Definition: **Kernelization** is a polynomial-time transformation that maps an instance (I, k) to an instance (I', k') such that

- (I, k) is a yes-instance if and only if (I', k') is a yes-instance,
- $k' \leq k$, and
- $|I'| \leq f(k)$ for some function $f(k)$.

In other words: a polynomial-time preprocessing technique with some provable guarantee.

We are especially interested in kernelizations where $f(k)$ is polynomial in k: which problems have **polynomial kernels?**

Kernelization for MIN ONES SAT

We have seen that VERTEX COVER is a special case of MIN ONES SAT and MIN ONES SAT(Γ) is FPT for every Γ.

Do we have a polynomial kernelization for MIN ONES SAT(Γ) for every Γ?

Kernelization for MIN ONES SAT

We have seen that VERTEX COVER is a special case of MIN ONES SAT and MIN ONES SAT(Γ) is FPT for every Γ.

Do we have a polynomial kernelization for MIN ONES SAT(Γ) for every Γ?

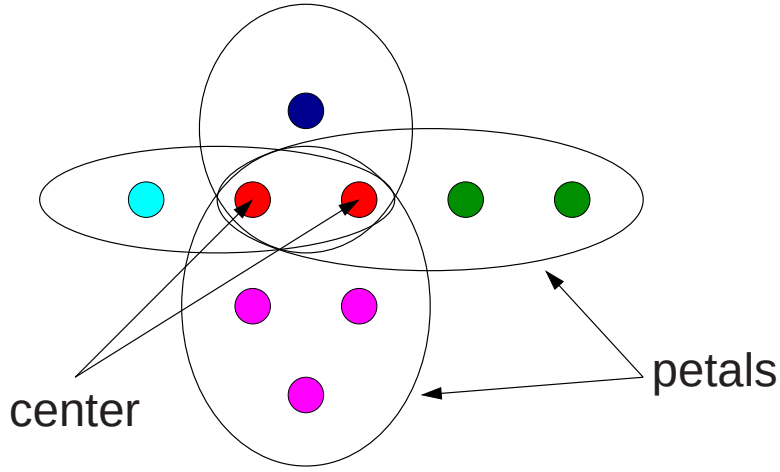
First another special case:

d-HITTING SET: Given a collection $\mathcal{S}$ of sets of size at most d and an integer k, find a set S of k elements that intersects every set of $\mathcal{S}$.

Fact: d-HITTING SET has a polynomial kernel for every fixed d.

Sunflower lemma

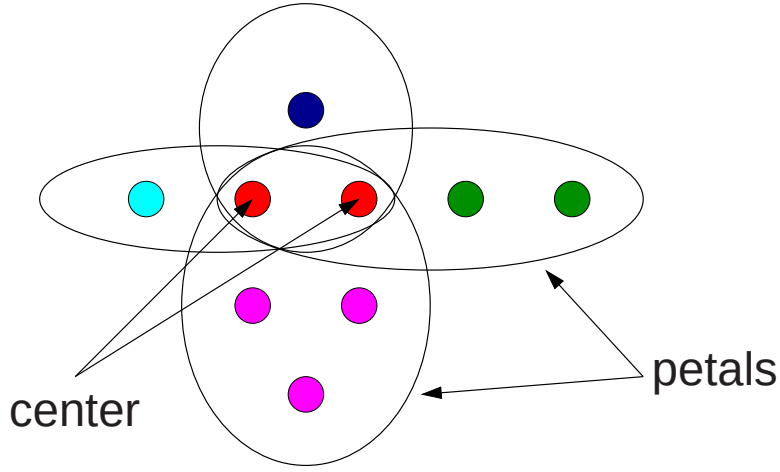
Definition: Sets $S_1, S_2, \ldots, S_k$ form a **sunflower** if the sets $S_i \setminus (S_1 \cap S_2 \cap \cdots \cap S_k)$ are disjoint.



Lemma: [Erdős and Rado, 1960] If the size of a set system is greater than $(p-1)^d \cdot d!$ and it contains only sets of size at most d, then the system contains a sunflower with p petals. Furthermore, in this case such a sunflower can be found in polynomial time.

Sunflowers and d-HITTING SET

d-HITTING SET: Given a collection $\mathcal{S}$ of sets of size at most d and an integer k, find a set S of k elements that intersects every set of $\mathcal{S}$.



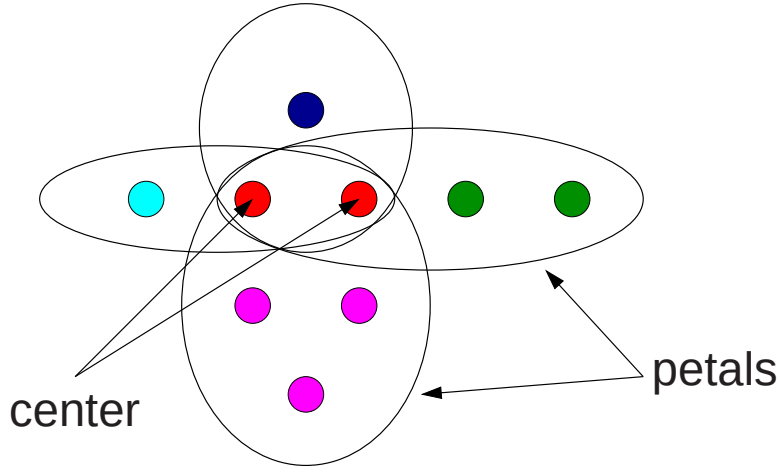
Reduction Rule: If $k+1$ sets form a sunflower, then remove these sets from $\mathcal{S}$ and add the center C to $\mathcal{S}$ (S does not hit one of the petals, thus it has to hit the center).

Note: if the center is empty (the sets are disjoint), then there is no solution.

If the rule cannot be applied, then there are at most $O(k^d)$ sets.

Sunflowers and d-HITTING SET

d-HITTING SET: Given a collection $\mathcal{S}$ of sets of size at most d and an integer k, find a set S of k elements that intersects every set of $\mathcal{S}$.



Reduction Rule (variant): Suppose more than $k+1$ sets form a sunflower.

- If the sets are disjoint $\Rightarrow$ No solution.
- Otherwise, keep only $k+1$ of the sets.

If the rule cannot be applied, then there are at most $O(k^d)$ sets.

Kernelization for MIN ONES SAT

The story very briefly:

- We have seen that MIN ONES SAT(Γ) is FPT for every Γ.
- Not every FPT problem admits a polynomial kernel and there is a technology for proving that (modulo some complexity assumption).
- Some cases of MIN ONES SAT(Γ), including d-HITTING SET, have polynomial kernels.
- MIN ONES SAT(Γ) has a polynomial kernel if and only if Γ is mergable [Kratsch and Wahlström 2010].

Lots of possibilities for Schaefer-style dichotomy theorems in parameterized complexity!

Conclusions

- Nice techniques for fixed parameter algorithms.
- Lots of interesting NP-hard problems are FPT.
- Bad news: W[1]-hardness theory.
- New kind of questions.