Sorting With Forbidden Intermediates

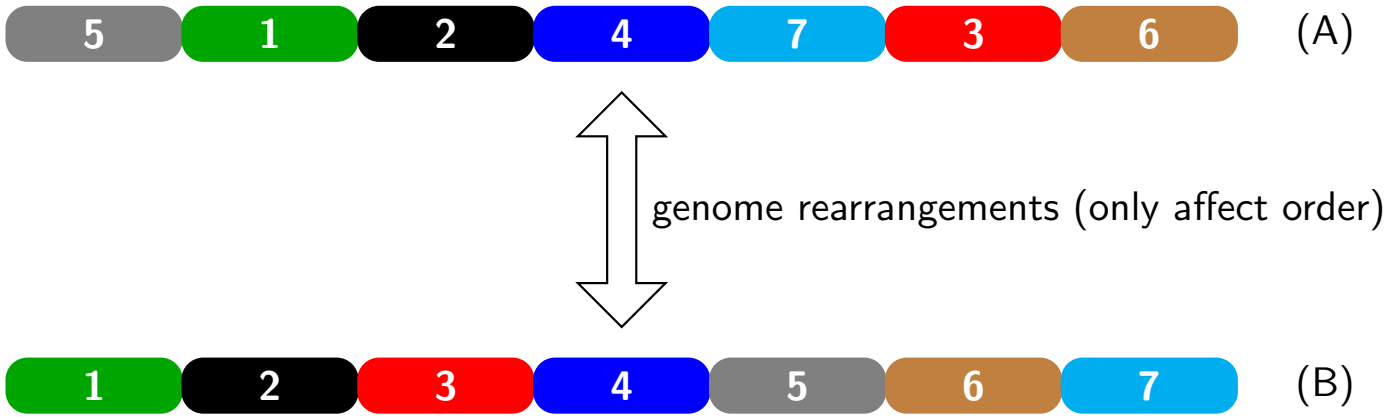
Carlo Comin Anthony Labarre Romeo Rizzi Stéphane Vialette

SCHLOSS DAGSTUHL
Leibniz-Zentrum für Informatik

February 15th, 2016

Genome rearrangements for permutations

- Permutations model genomes with the same "contents" without duplication:



- The actual numbering is irrelevant, so we assume either genome is the **identity permutation** $\iota = \langle 1\ 2\ \cdots\ n \rangle$;
- The classical (family of) problem(s):

GENOME REARRANGEMENT (PERMUTATIONS)

Input: a permutation π in S_n, a set S of (per)mutations;
Goal: find a shortest sorting sequence of elements of S for π.

Three examples

- Let us sort $\pi = \langle 3\ 2\ 1\ 9\ 8\ 7\ 6\ 5\ 4 \rangle$ using three different sets of operations:

Reversals

$\langle \underline{3\ 2\ 1}\ 9\ 8\ 7\ 6\ 5\ 4 \rangle$
$\langle 1\ 2\ 3\ \underline{9\ 8\ 7\ 6\ 5\ 4} \rangle$
$\langle 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9 \rangle$

Block-interchanges

$\langle 3\ 2\ 1\ 9\ 8\ 7\ 6\ 5\ 4 \rangle$
$\langle 1\ 9\ 8\ 7\ 6\ 5\ 2\ 3\ 4 \rangle$
$\langle 1\ 2\ 3\ 4\ 8\ 7\ 6\ 5\ 9 \rangle$
$\langle 1\ 2\ 3\ 4\ 5\ 7\ 6\ 8\ 9 \rangle$
$\langle 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9 \rangle$

Block-transpositions

$\langle 3\ 2\ 1\ 9\ 8\ 7\ 6\ 5\ 4 \rangle$
$\langle 3\ 4\ 2\ 1\ 9\ 8\ 7\ 6\ 5 \rangle$
$\langle 1\ 9\ 8\ 3\ 4\ 2\ 7\ 6\ 5 \rangle$
$\langle 1\ 2\ 7\ 6\ 9\ 8\ 3\ 4\ 5 \rangle$
$\langle 1\ 2\ 7\ 8\ 3\ 4\ 5\ 6\ 9 \rangle$
$\langle 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9 \rangle$

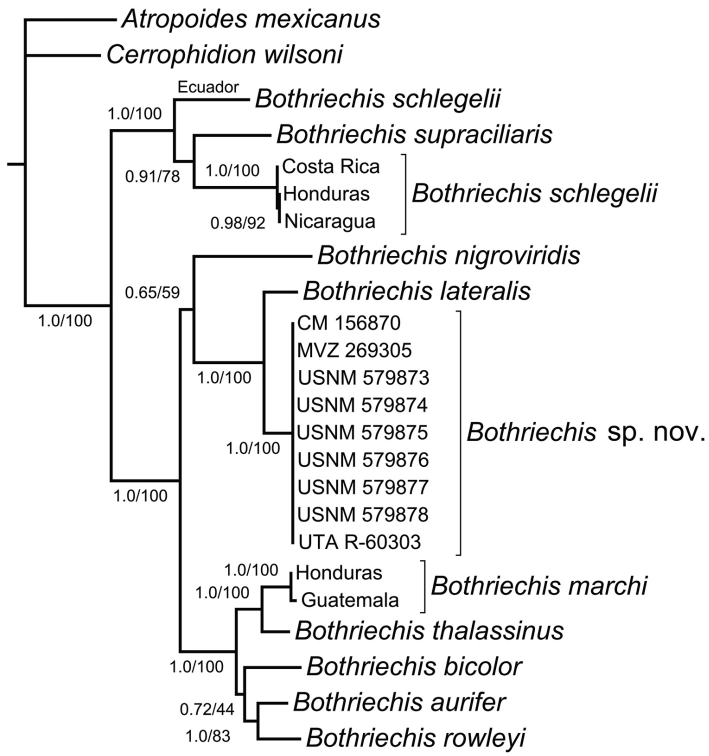
- All these sequences are optimal (proofs omitted);
- The **distance** of π is the length of an optimal sequence;

Issues with the model

- The overall approach is criticised for various reasons:
  - permutations are too restricted ;
    ... but many other models exist
  - operations are too restricted;
    ... but we can consider several of them at once
  - complexity issues;
    ... but we have SAT and LP solvers if need be
  - ...
- Another critical issue needs addressing (next slide);

Phylogenies

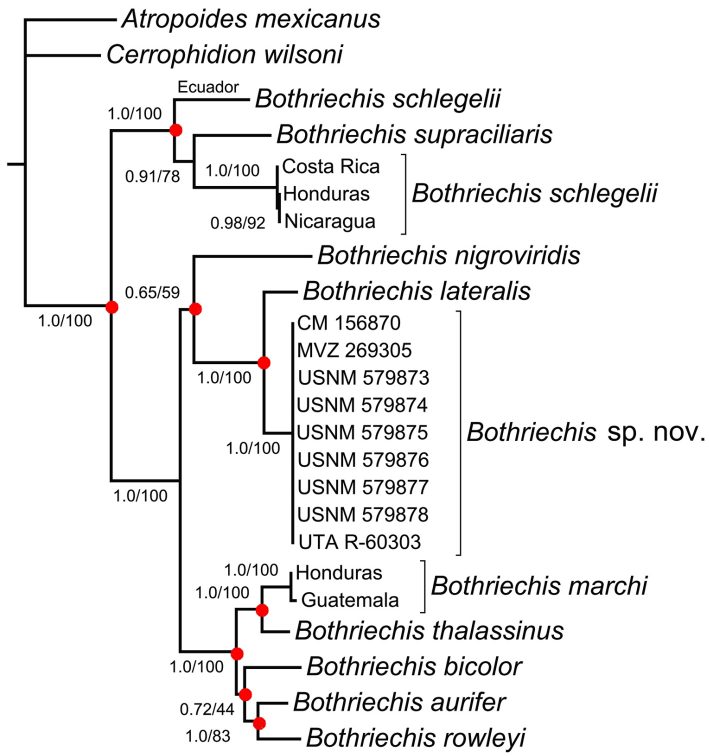
- One motivation for measuring similarities between genomes is to reconstruct ancestral genomes and phylogenies;



source: `https://commons.wikimedia.org/wiki/File:Drynarioid_phylogeny.png`

Phylogenies

- One motivation for measuring similarities between genomes is to reconstruct ancestral genomes and phylogenies;



source: `https://commons.wikimedia.org/wiki/File:Drynarioid_phylogeny.png`

(except the ●'s)

- But some mutations are lethal;
- Which means some ancestors cannot exist and therefore cannot have led to present-day species;

A more realistic model

- We must therefore forbid some intermediate configurations in our search for a sorting sequence;
- Our problem becomes:

GUIDED SORTING (PERMUTATIONS)

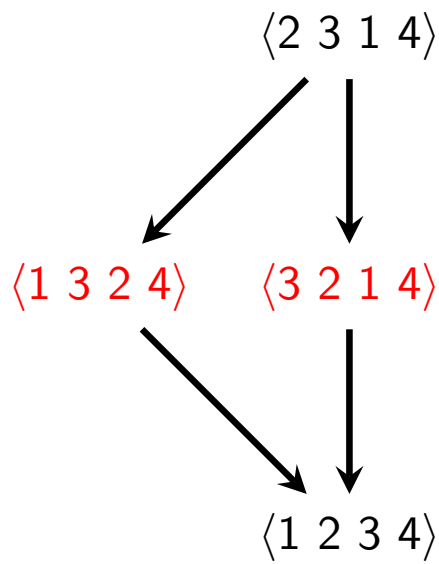
Input: a permutation π in S_n, a set S of (per)mutations, *a set F of forbidden permutations*;

Goal: find a shortest sorting sequence of elements of S for π *that avoids all elements of F*;

- Here "shortest" means "as if F were empty";
- Note: we do not try to restrict operations themselves or the structure of genomes;

Example

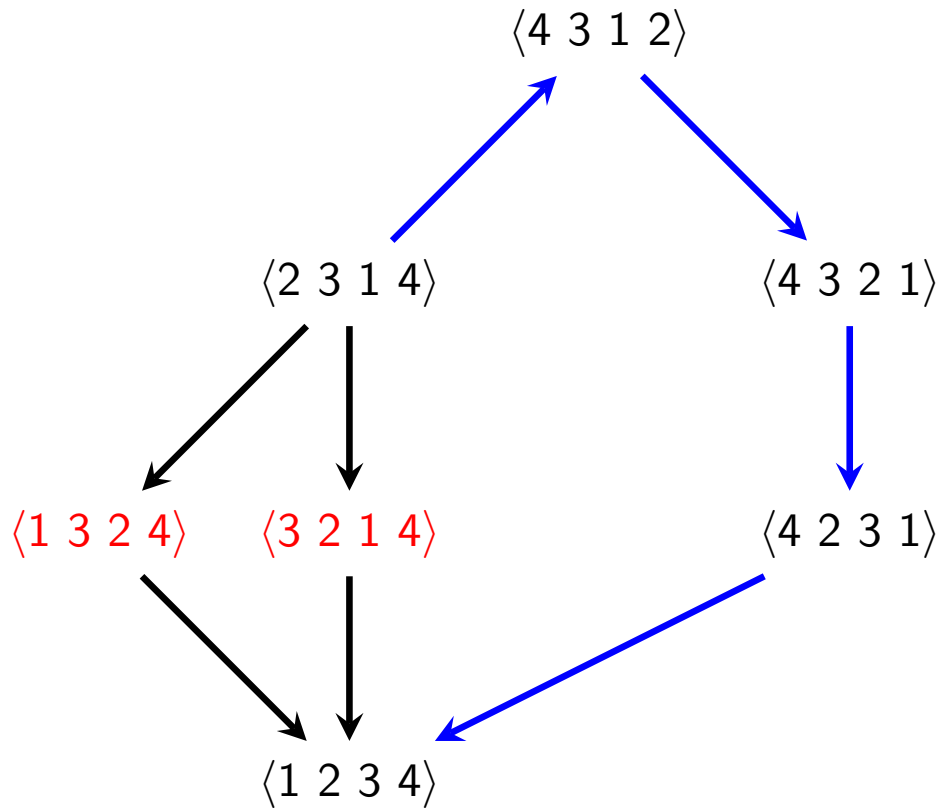
- If $\pi = \langle 2\ 3\ 1\ 4 \rangle$, $S = \{\text{exchanges}\}$ and $F = \{\langle 1\ 3\ 2\ 4 \rangle, \langle 3\ 2\ 1\ 4 \rangle\}$:



the black paths are optimal but do not avoid F

Example

- If $\pi = \langle 2\ 3\ 1\ 4 \rangle$, $S = \{\text{exchanges}\}$ and $F = \{\langle 1\ 3\ 2\ 4 \rangle, \langle 3\ 2\ 1\ 4 \rangle\}$:

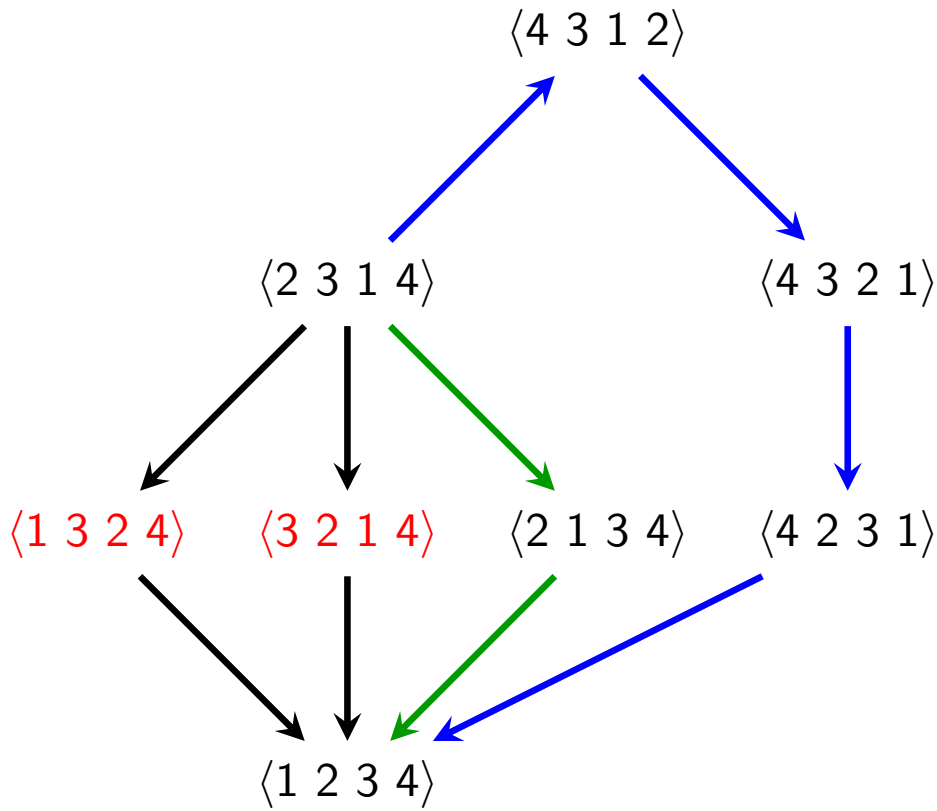


the black paths are optimal but do not avoid F

the blue path avoids F but is not optimal

Example

- If $\pi = \langle 2\ 3\ 1\ 4 \rangle$, $S = \{\text{exchanges}\}$ and $F = \{\langle 1\ 3\ 2\ 4 \rangle, \langle 3\ 2\ 1\ 4 \rangle\}$:



the black paths are optimal but do not avoid F

the blue path avoids F but is not optimal

the green path avoids F and is optimal

In this talk

- We focus on "exchanges" (i.e. algebraic transpositions);
  - strongly connected to cycles of permutations;
  - hopefully some connections carry on to cycles in *breakpoint graphs*;
- We give a polynomial-time algorithm for solving the problem on *involutions*;

Obvious and generic solution: Cayley graph

Definition

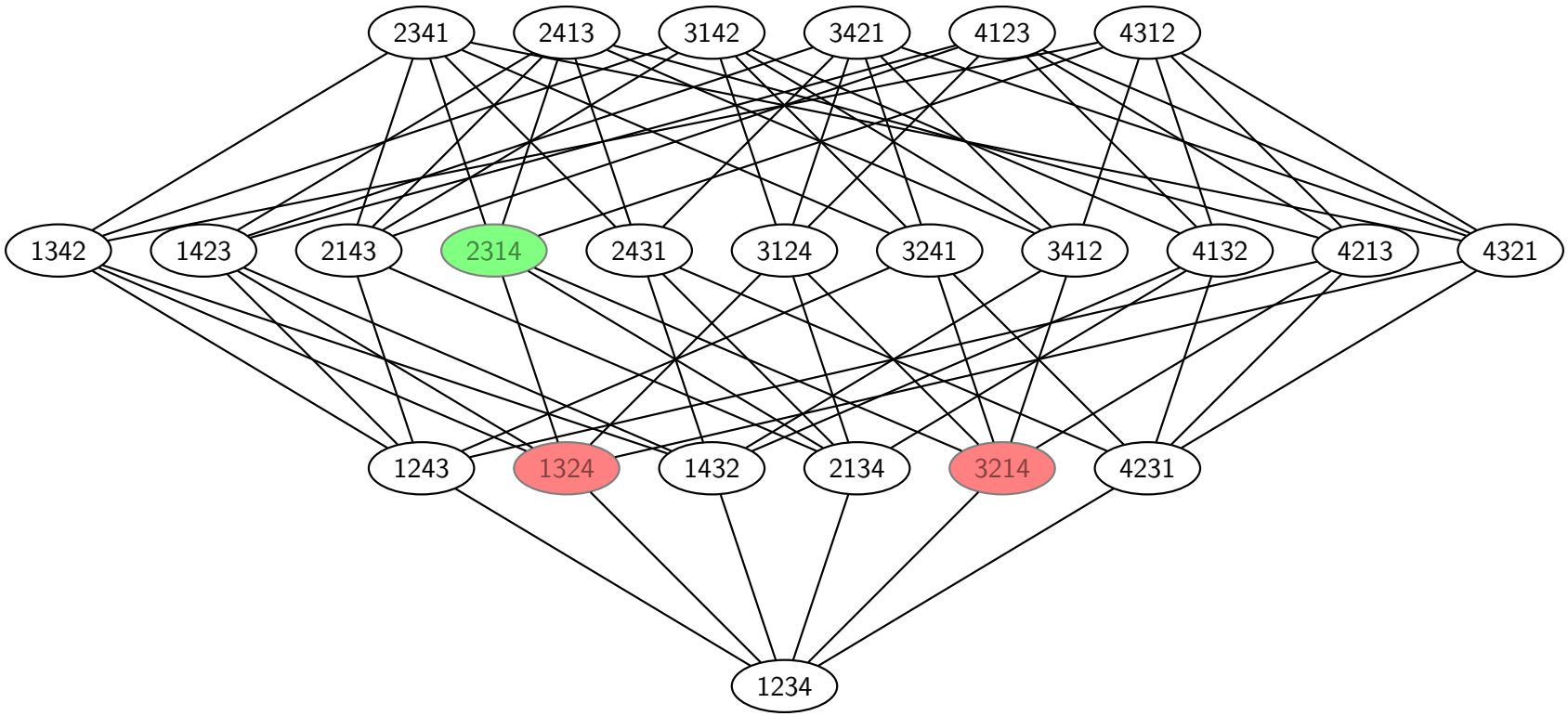
The **Cayley graph** G of S_n with generating set S is defined by:

1. $V(G) = \{\pi \mid \pi \in S_n\}$;
2. $E(G) = \{\{\pi, \sigma\} \mid d_S(\pi, \sigma) = 1\}$.

- Here's a straightforward solution to all variants of GUIDED SORTING:
  1. build the part of the Cayley graph we are interested in;
  2. find a shortest path between π and ι (e.g. Dijkstra);

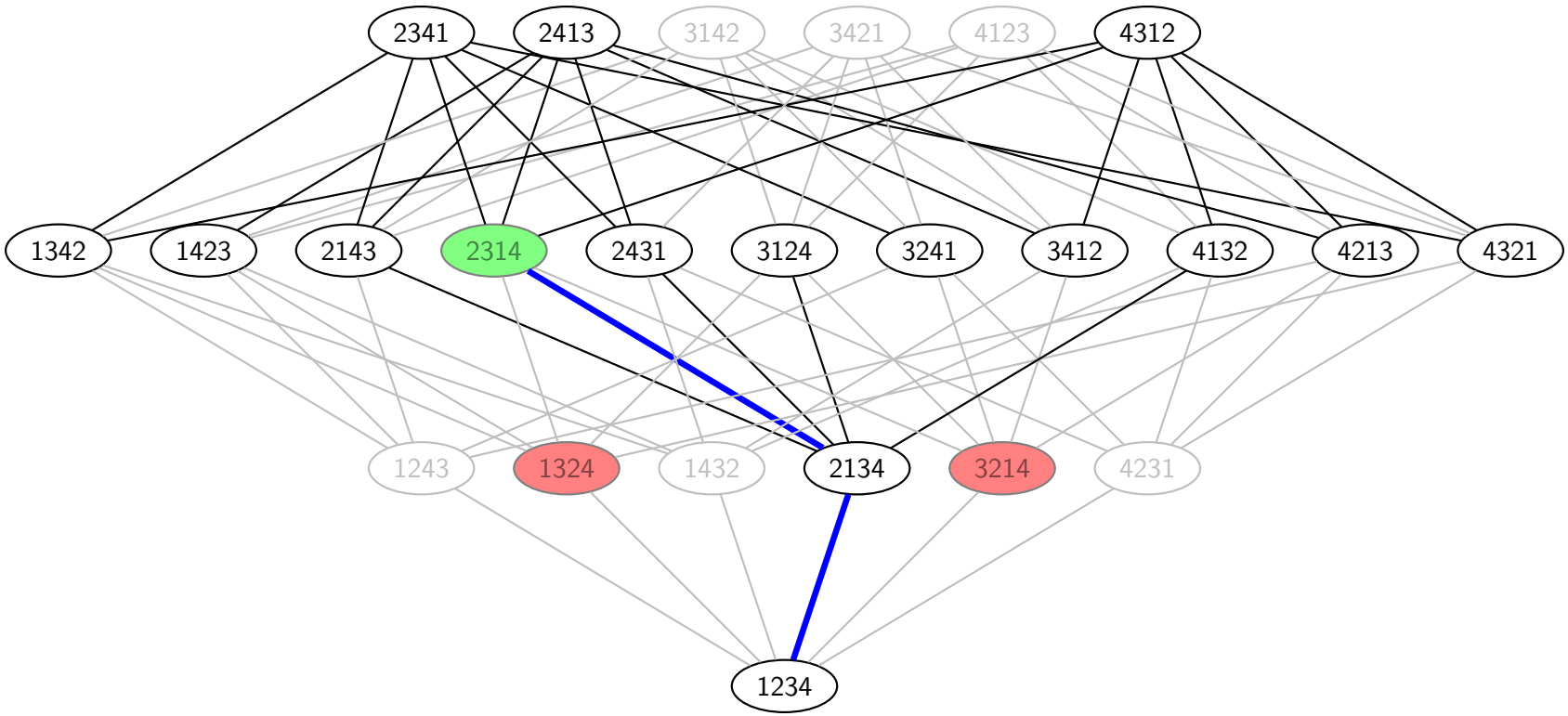
The Cayley graph approach in action

- Here's what would happen using our previous example:



The Cayley graph approach in action

- Here's what would happen using our previous example:

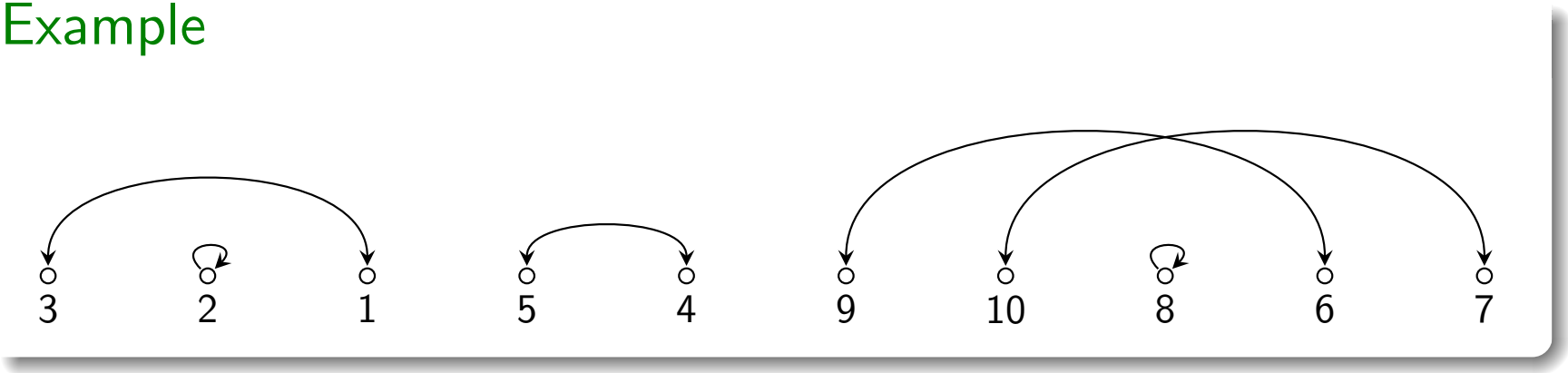


- Obviously, the approach does not scale ($O(n!)$ vertices, $O(n!|S|)$ edges);

Involutions

- An **involution** is a permutation π such that $\pi = \pi^{-1}$;
- Equivalently: all its **cycles** have length ≤ 2;

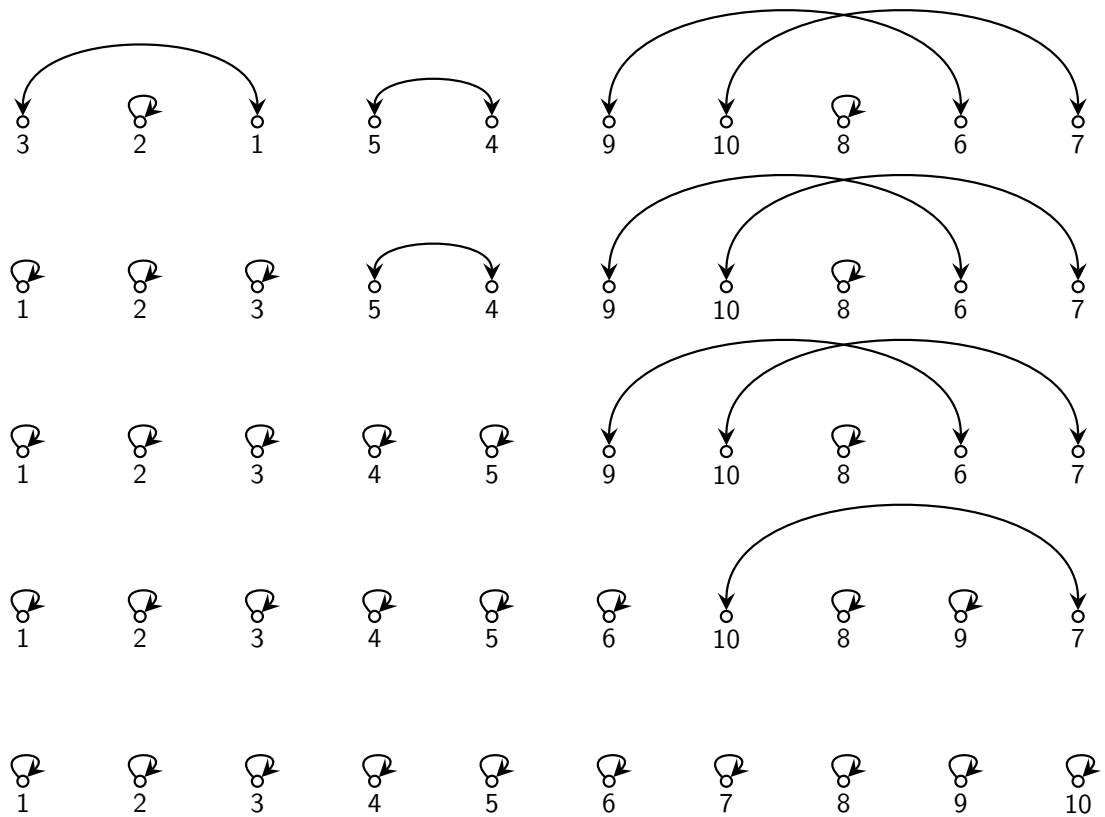
Example



A simpler view of sorting by exchanges

- Involutions are "conceptually simpler" to sort:
  - 1-cycles are left alone;
  - a single exchange splits a 2-cycle, and we can always find one;

Example (split 2-cycles from left to right)



From involutions to the hypercube graph

- π is an involution $\Rightarrow$ we only need to worry about forbidden involutions whose 2-cycles appear in π;

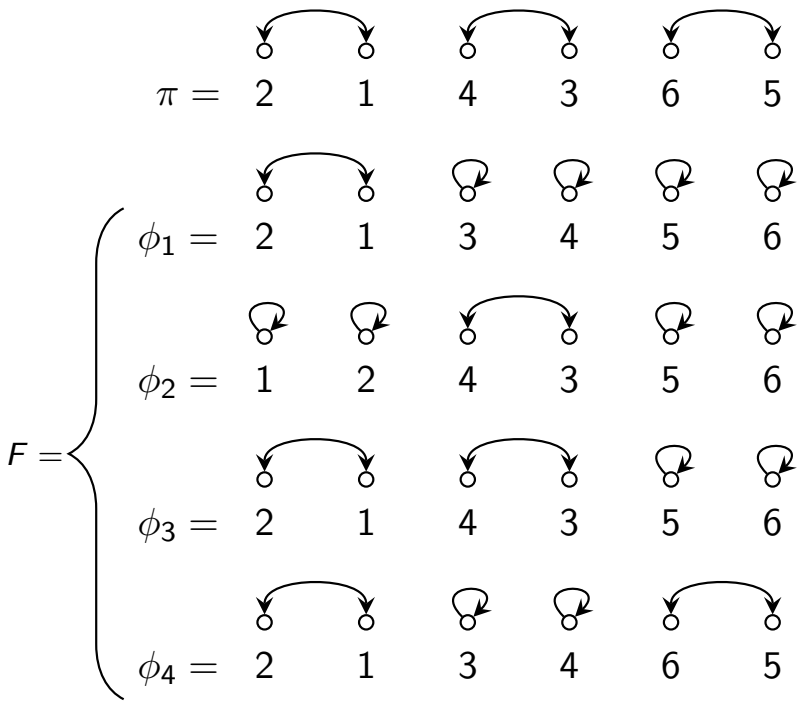
From involutions to the hypercube graph

- π is an involution $\Rightarrow$ we only need to worry about forbidden involutions whose 2-cycles appear in π;
- We map (π, F) onto $([k], F')$, where:
  - k is the number of 2-cycles of π;
  - F' is a collection of forbidden subsets of $[k]$;

From involutions to the hypercube graph

- π is an involution $\Rightarrow$ we only need to worry about forbidden involutions whose 2-cycles appear in π;
- We map (π, F) onto $([k], F')$, where:
  - k is the number of 2-cycles of π;
  - F' is a collection of forbidden subsets of $[k]$;

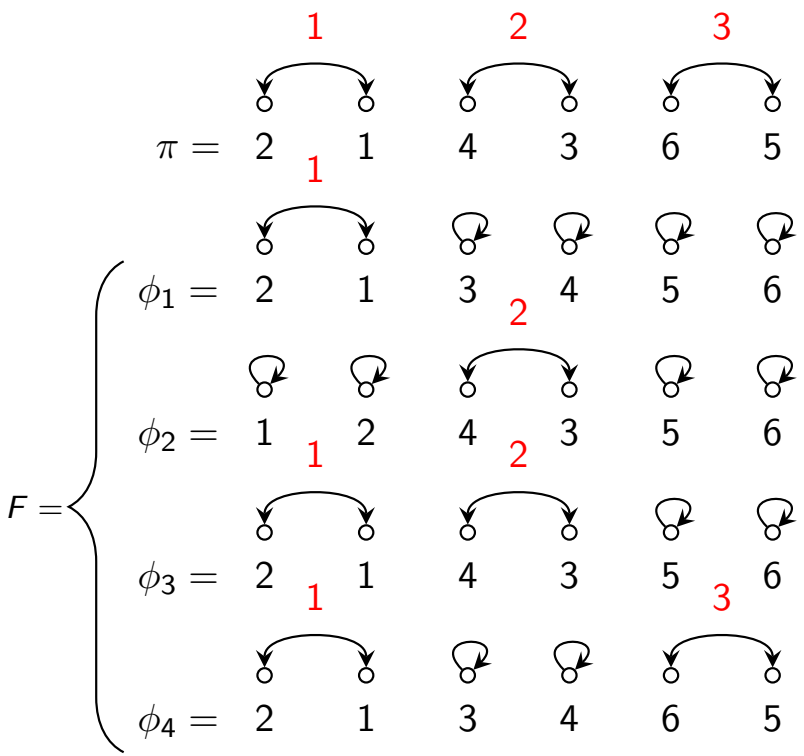
Example



From involutions to the hypercube graph

- π is an involution $\Rightarrow$ we only need to worry about forbidden involutions whose 2-cycles appear in π;
- We map (π, F) onto $([k], F')$, where:
  - k is the number of 2-cycles of π;
  - F' is a collection of forbidden subsets of $[k]$;

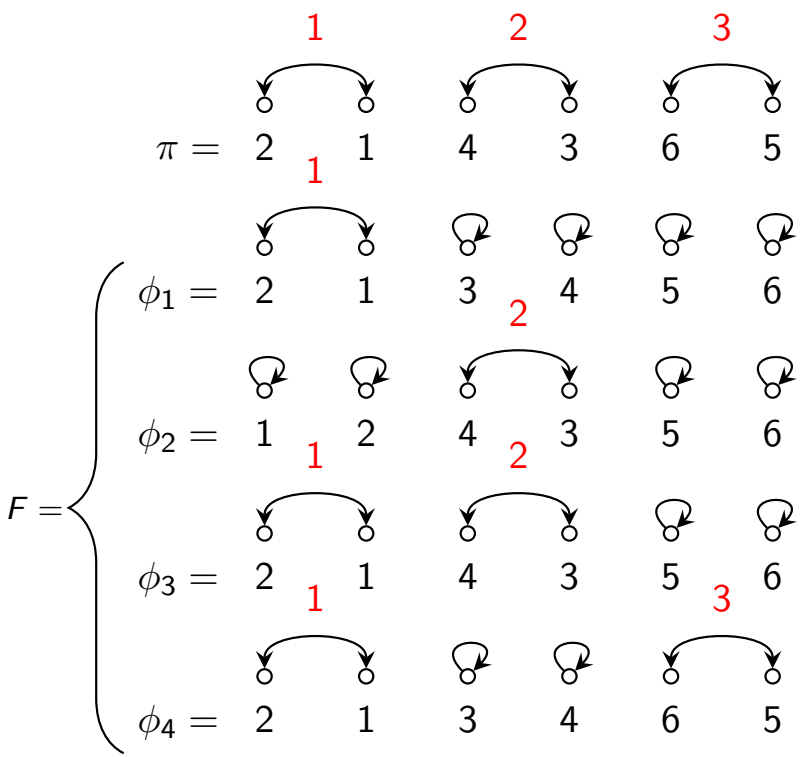
Example

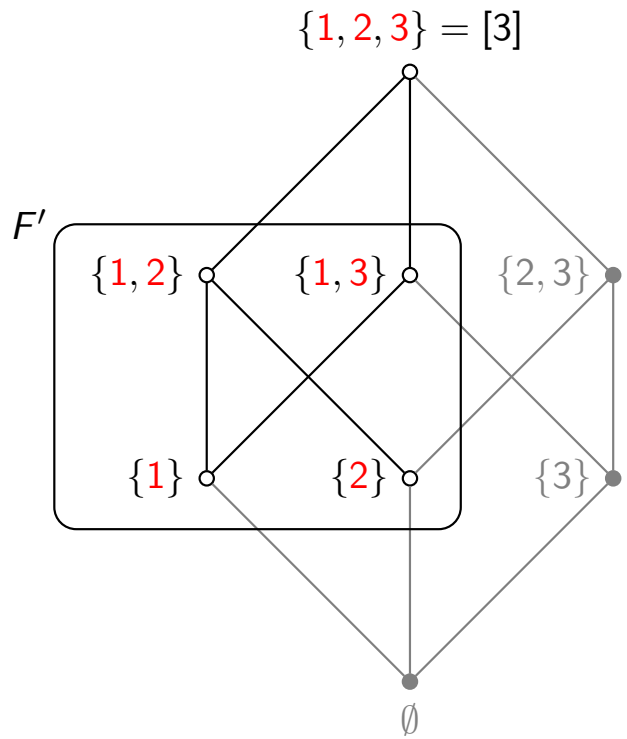


From involutions to the hypercube graph

- π is an involution $\Rightarrow$ we only need to worry about forbidden involutions whose 2-cycles appear in π;
- We map (π, F) onto $([k], F')$, where:
  - k is the number of 2-cycles of π;
  - F' is a collection of forbidden subsets of $[k]$;

Example





Properties of involutions and exchanges

- Involutions behave nicely with respect to exchanges;
- Only elements of F whose 2-cycles all appear in π need to be considered;
- GUIDED SORTING under these hypotheses reduces to the following problem:

(s, t)-PATHS IN HYPERCUBE NETWORK

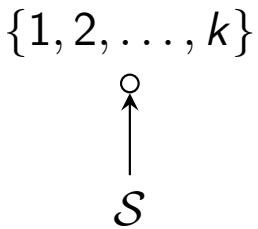
Input: the set $[k] = \{1, 2, \ldots, k\}$, a collection $\mathcal{F}$ of subsets of $[k]$;
Goal: find a sequence of element deletions for $[k]$ that empties it while avoiding $\mathcal{F}$.

Overview of the main algorithm

- We follow the Cayley graph approach but avoid its explicit construction;
  - otherwise: $O(2^k)$ vertices and $O(2^{k-1}k)$ edges;
- The main algorithm goes as follows:

Overview of the main algorithm

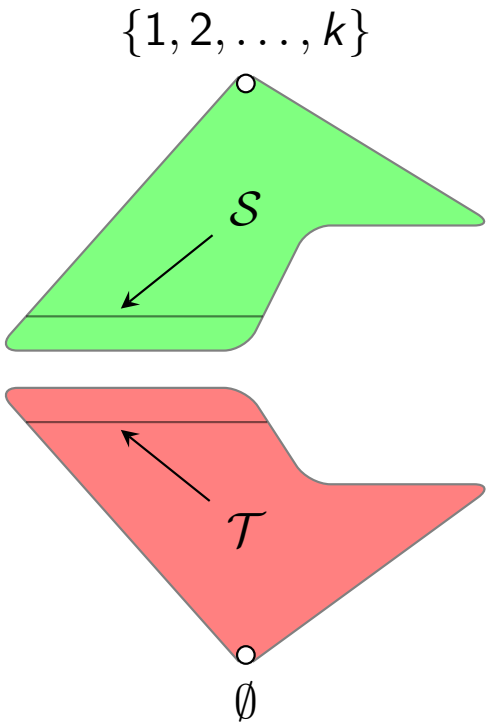
- We follow the Cayley graph approach but avoid its explicit construction;
  - otherwise: $O(2^k)$ vertices and $O(2^{k-1}k)$ edges;
- The main algorithm goes as follows:



1. $\mathcal{S} \leftarrow \{[k]\}$, $\mathcal{T} \leftarrow \{\emptyset\}$;

Overview of the main algorithm

- We follow the Cayley graph approach but avoid its explicit construction;
  - otherwise: $O(2^k)$ vertices and $O(2^{k-1}k)$ edges;
- The main algorithm goes as follows:



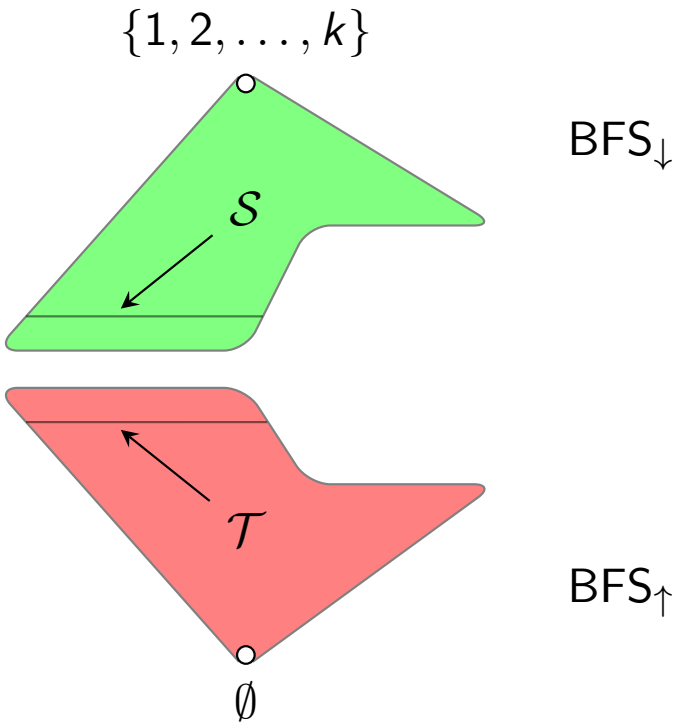
$\mathrm{BFS}_{\downarrow}$

$\mathrm{BFS}_{\uparrow}$

1. $\mathcal{S} \leftarrow \{[k]\}$, $\mathcal{T} \leftarrow \{\emptyset\}$;
2. launch a double BFS on $\mathcal{S}$ and $\mathcal{T}$;

Overview of the main algorithm

- We follow the Cayley graph approach but avoid its explicit construction;
  - otherwise: $O(2^k)$ vertices and $O(2^{k-1}k)$ edges;
- The main algorithm goes as follows:



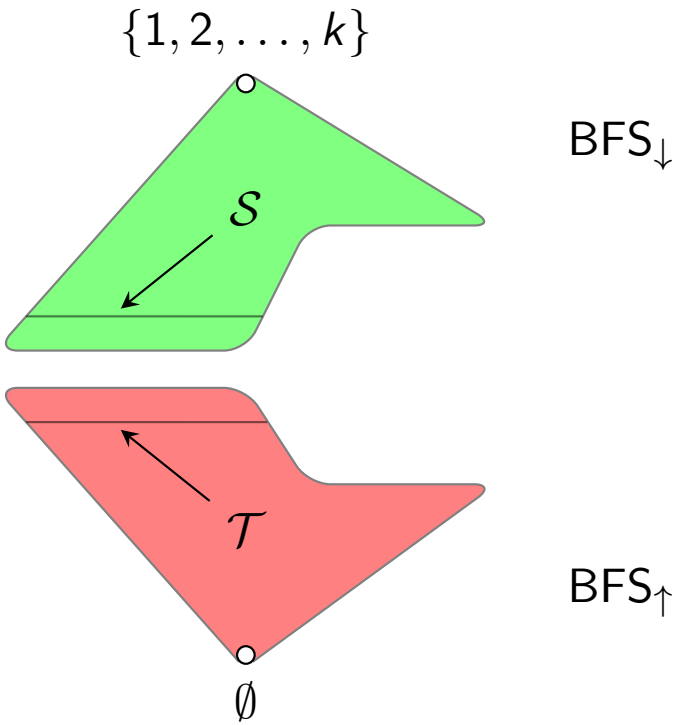
$\mathrm{BFS}_{\downarrow}$

$\mathrm{BFS}_{\uparrow}$

1. $\mathcal{S} \leftarrow \{[k]\}$, $\mathcal{T} \leftarrow \{\emptyset\}$;
2. launch a double BFS on $\mathcal{S}$ and $\mathcal{T}$;
3. if a solution exists: return it;

Overview of the main algorithm

- We follow the Cayley graph approach but avoid its explicit construction;
  - otherwise: $O(2^k)$ vertices and $O(2^{k-1}k)$ edges;
- The main algorithm goes as follows:



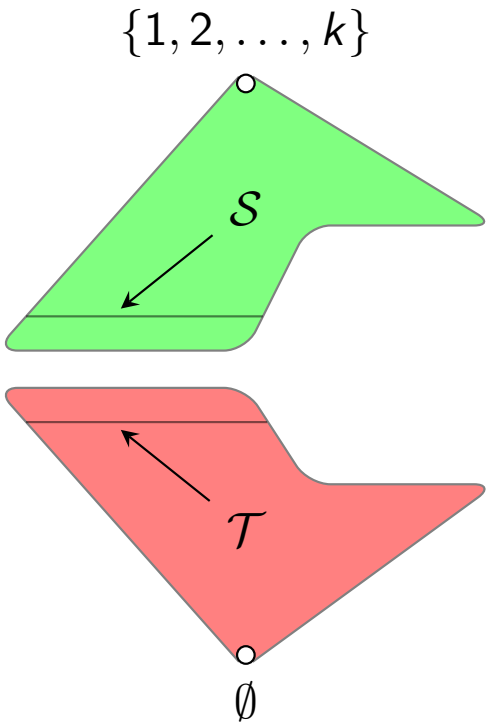
$\text{BFS}_{\downarrow}$

$\text{BFS}_{\uparrow}$

1. $\mathcal{S} \leftarrow \{[k]\}$, $\mathcal{T} \leftarrow \{\emptyset\}$;
2. launch a double BFS on $\mathcal{S}$ and $\mathcal{T}$;
3. if a solution exists: return it;
4. if no solution exists: return **NO**;

Overview of the main algorithm

- We follow the Cayley graph approach but avoid its explicit construction;
  - otherwise: $O(2^k)$ vertices and $O(2^{k-1}k)$ edges;
- The main algorithm goes as follows:



BFS$_\downarrow$

BFS$_\uparrow$

1. $\mathcal{S} \leftarrow \{[k]\}$, $\mathcal{T} \leftarrow \{\emptyset\}$;
2. launch a double BFS on $\mathcal{S}$ and $\mathcal{T}$;
3. if a solution exists: return it;
4. if no solution exists: return **NO**;
5. otherwise: compress $\mathcal{S}$ and $\mathcal{T}$ and go back to 2;

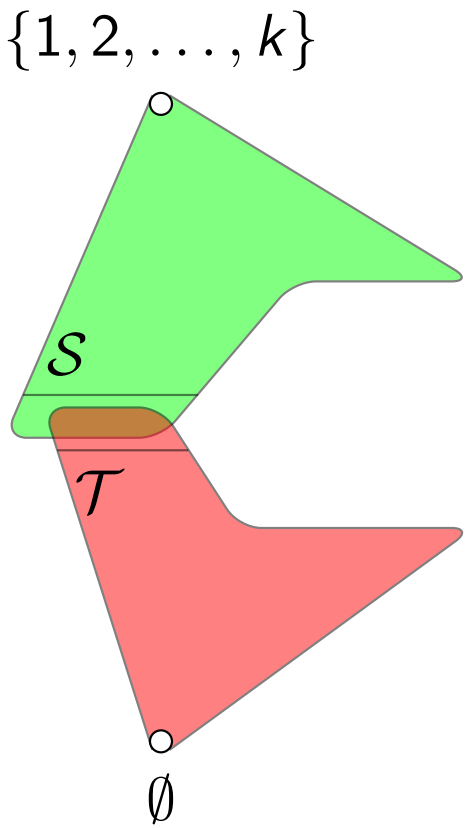
- The compression phase ensures the running time remains polynomial;

The double BFS phase

- Classical breadth-first searches, skipping elements from $\mathcal{F}$:
  1. one upwards from the current bottom;
  2. one downwards from the current top;
- To keep the running time polynomial, searches stop when we have $O(|\mathcal{F}|dn)$ vertices (d is the difference in cardinality between $\mathcal{S}$ and $\mathcal{T}$);

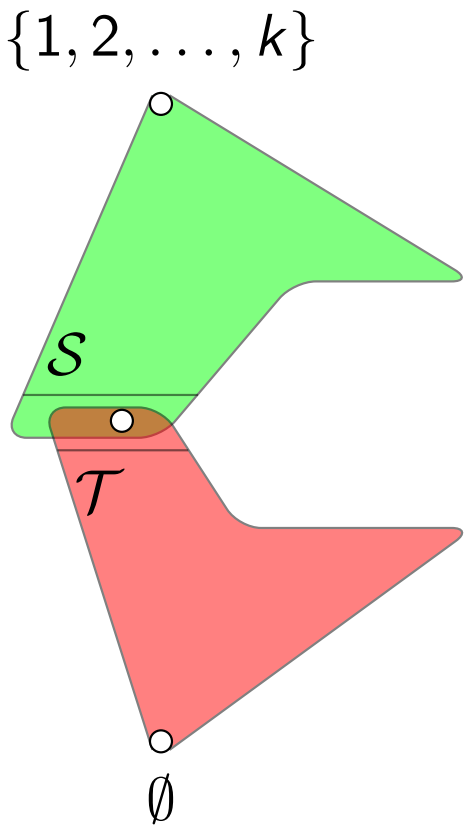
Obvious case where a solution exists

- If $\mathcal{S} \cap \mathcal{T} \neq \emptyset$, then a solution exists;



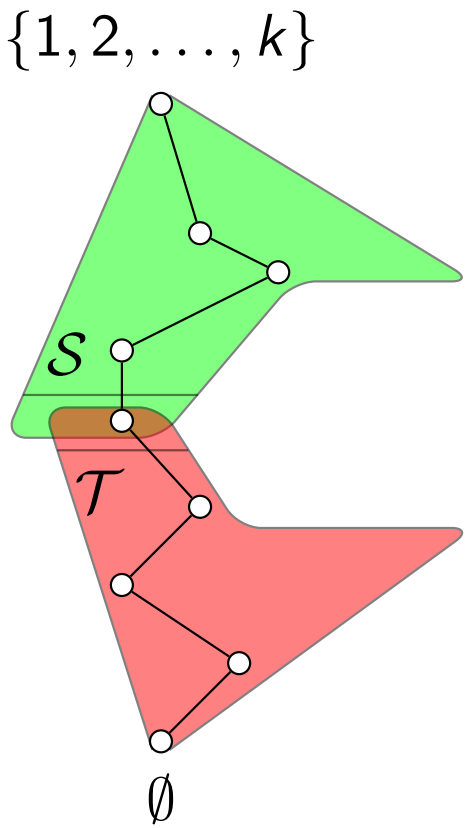
Obvious case where a solution exists

- If $\mathcal{S} \cap \mathcal{T} \neq \emptyset$, then a solution exists;



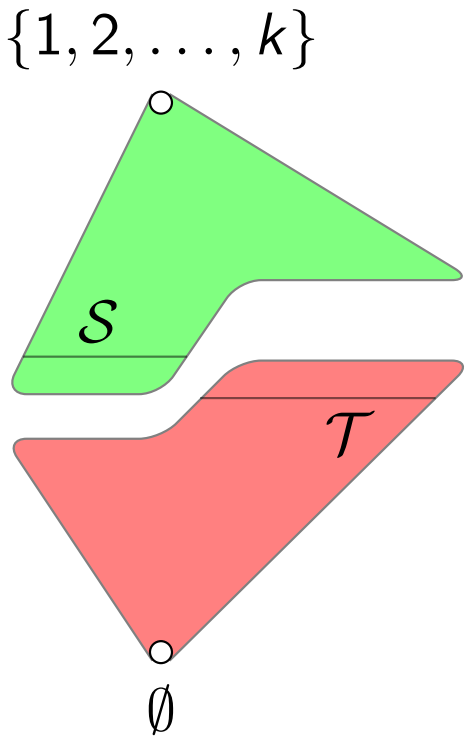
Obvious case where a solution exists

- If $\mathcal{S} \cap \mathcal{T} \neq \emptyset$, then a solution exists;



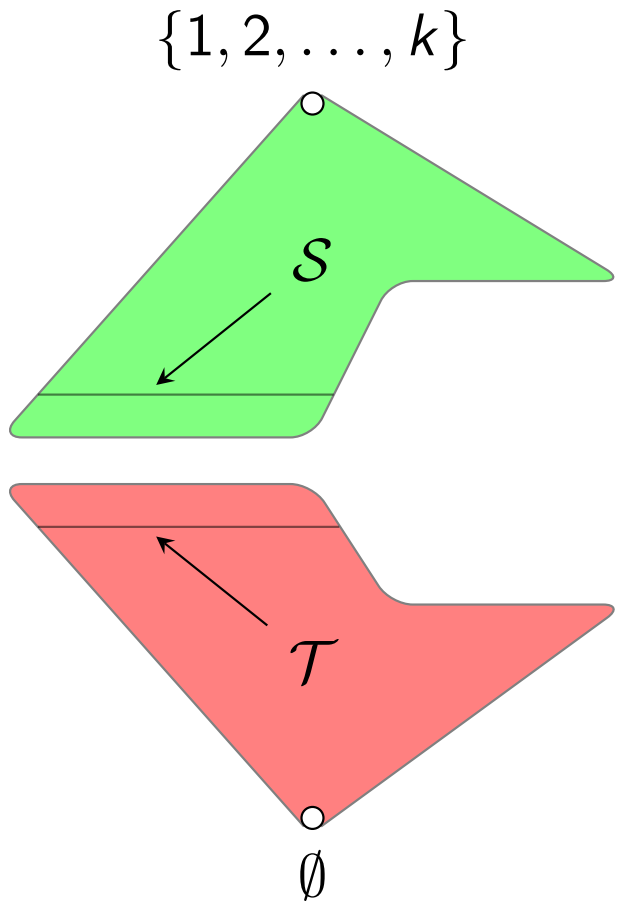
Obvious cases where no solution exists

- If $\mathcal{S}$ or $\mathcal{T}$ is empty, then no solution exists;
- If we've gone "deep (resp. high) enough" and $\mathcal{S} \cap \mathcal{T}$ is empty, then no solution exists:



The other cases

- We may have collected enough vertices to stop the BFS's, but $\mathcal{S}$ and $\mathcal{T}$ don't intersect yet:



- In this case, we may either compute a solution, or launch the compression and keep going;

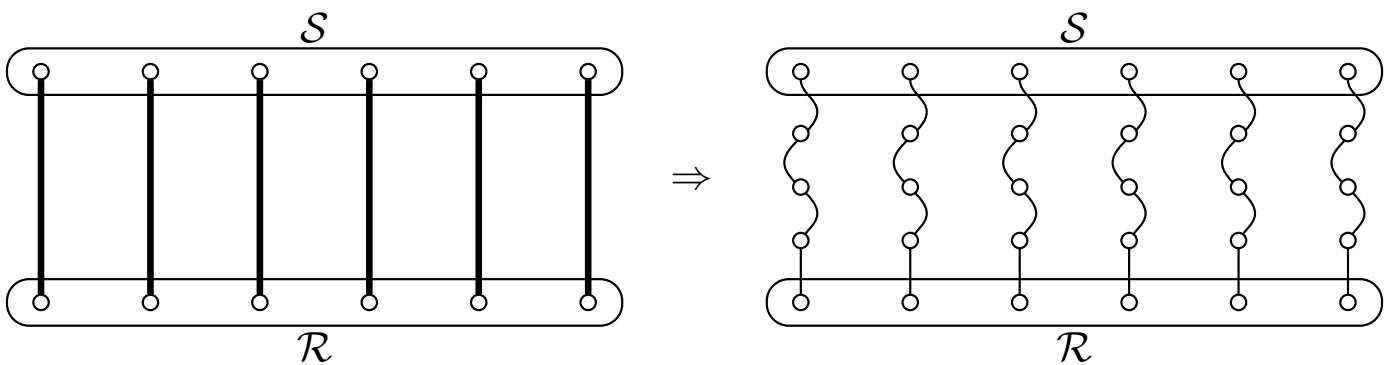
Interlude: Lehman and Ron's theorem

We need the following result[1].

Theorem

Given $n, m \in \mathbb{N}$, consider two families of sets $\mathcal{R} \subseteq \mathcal{H}_n^{(r)}$ and $\mathcal{S} \subseteq \mathcal{H}_n^{(s)}$ where $|\mathcal{R}| = |\mathcal{S}| = m$ and $0 \leq r < s \leq n$. Assume there exists a bijection $\varphi : \mathcal{S} \to \mathcal{R}$ such that $\varphi(S) \subset S$ for every $S \in \mathcal{S}$. Then there exist m vertex-disjoint directed paths in $\mathcal{H}_n$ whose union contains all the subsets in $\mathcal{S}$ and $\mathcal{R}$.

In other words:



[1] E. Lehman and D. Ron, "On Disjoint Chains of Subsets", *Journal of Combinatorial Theory, Series A*, 94(2):399–404, 2001.

Finding a solution with Theorem 1

1. Build a bipartite graph B with:
   - vertex set $\mathcal{S} \cup \mathcal{T}$;
   - edges connecting each element s of $\mathcal{S}$ with an element t of $\mathcal{T}$ if $t \subset s$;
2. Compute a maximum matching $\mathcal{M}$ of B;
3. If $|\mathcal{M}| > |\mathcal{F}|$, there is at least one $(\mathcal{S}, \mathcal{T})$-path that avoids $\mathcal{F}$ (thanks to Lehman and Ron's theorem);
4. Otherwise, we keep going but reduce the size of $\mathcal{T}$ by removing "non essential" vertices;

The compression phase

- Compute a minimum vertex cover $\mathcal{X} = \mathcal{X}_\mathcal{S} \cup \mathcal{X}_\mathcal{T}$ of B;
  - $(\mathcal{X}_\mathcal{S} = \mathcal{X} \cap \mathcal{S}$, $\mathcal{X}_\mathcal{T} = \mathcal{X} \cap \mathcal{T})$
- Since $\mathcal{X}$ is a vertex cover, no relevant path from $S \setminus \mathcal{X}_\mathcal{S}$ to $T \setminus \mathcal{X}_\mathcal{T}$ exists;
- We then search for a solution using $\mathcal{X}_\mathcal{S}$ and $\mathcal{T}$, and repeat the process until we find one or reach the threshold of $|\mathcal{F}|dn$ vertices;
- If no solution has been found, we return to the main algorithm with $\mathcal{T}' = \bigcup_i \mathcal{X}_\mathcal{T}^{(i)}$;
  - $(\mathcal{X}_\mathcal{T}^{(i)}$ is the $\mathcal{X}_\mathcal{T}$ computed at the i^{th} iteration)

Summary of results

- We can solve GUIDED SORTING by exchanges on involutions in time:
  - $O(\min(\sqrt{|\mathcal{F}|\, d\, k}, |\mathcal{F}|)\, |\mathcal{F}|^2\, d^4\, k^2)$ ("decision version");
  - $O(\min(\sqrt{|\mathcal{F}|\, d\, k}, |\mathcal{F}|)\, |\mathcal{F}|^2\, d^4\, k^2 + |\mathcal{F}|^{5/2} k^{3/2} d)$ ("search version");
- (k is the number of 2-cycles in π);

Future work

- Complexity of (variants of) GUIDED SORTING?
- Other tractable cases?
- What if we relax "optimal" to "minimal"?
- Do the algorithms generalise?
- Can we compute or benefit from an "implicit" encoding of F?
  - $F = Av_n$(some patterns);
  - $F =<$ some generators $> \setminus$ some small set;
  - ...