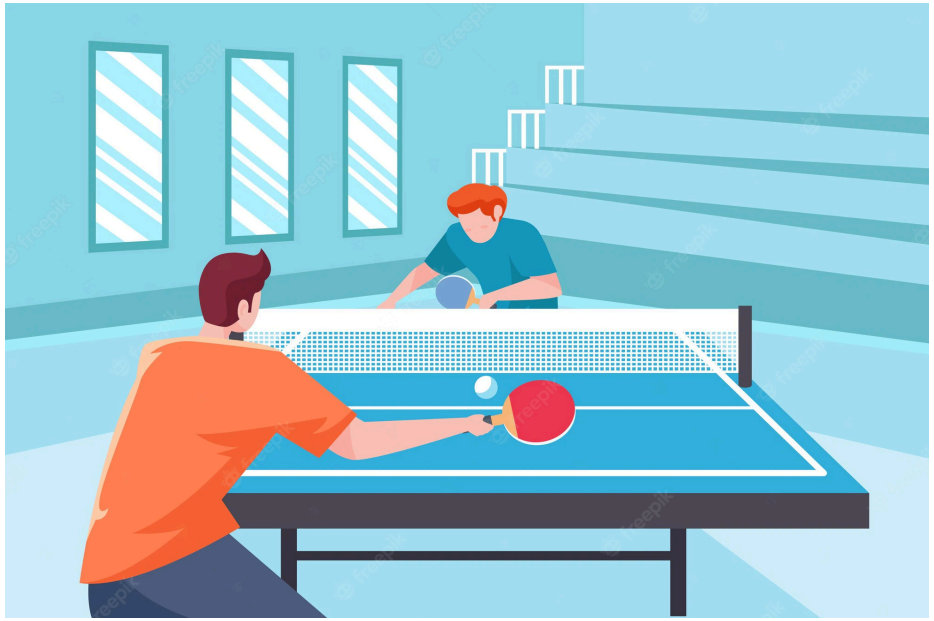
PCCR 2022

The Tournament Fixing Problem



Meirav Zehavi

Outline

Background

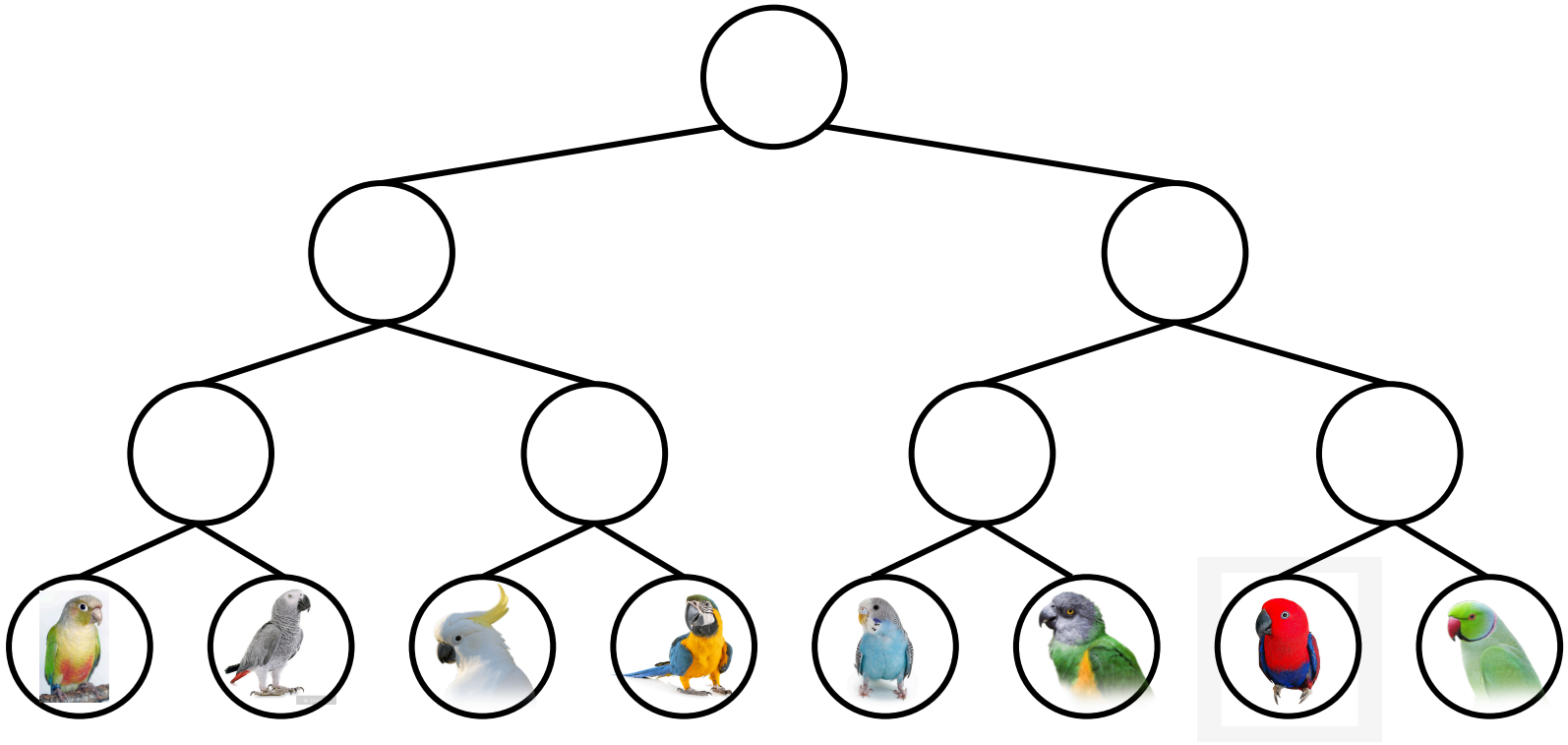
FPT algorithm for the TF problem

Kernel for the TF problem

Background

Single-Elimination (or Knockout) Tournament

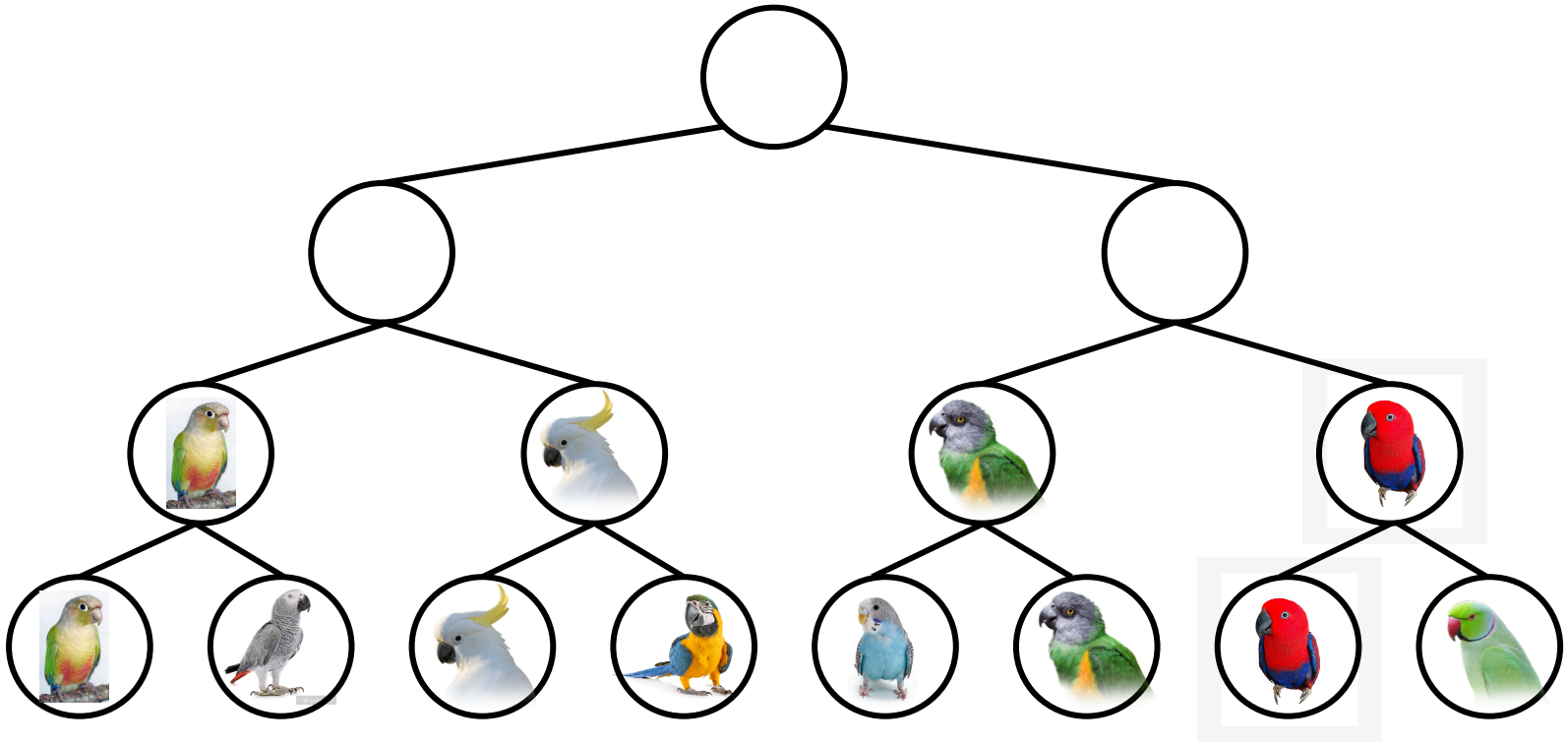
Competition involving *n* (power of 2) players, conducted as follows:



Background

Single-Elimination (or Knockout) Tournament

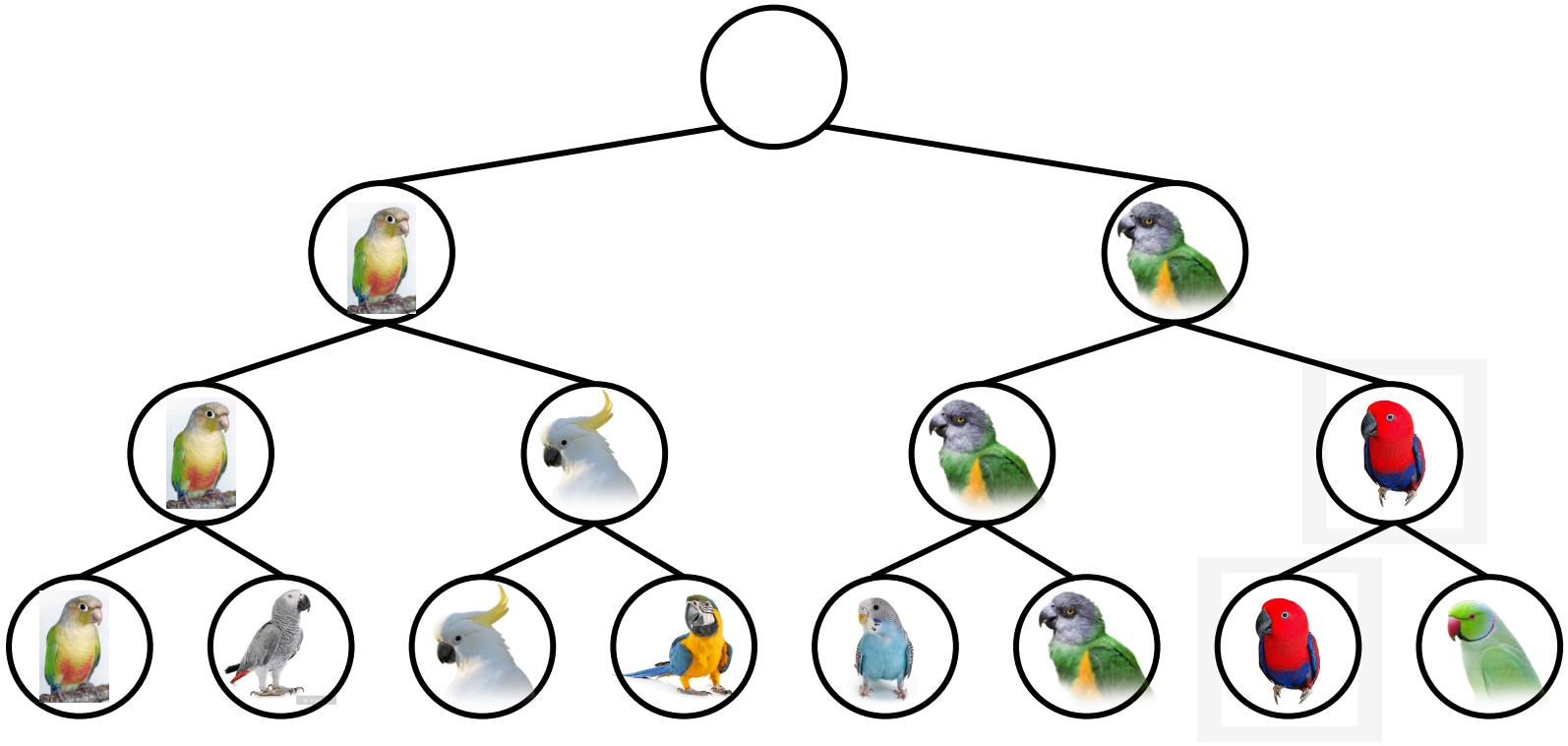
Competition involving *n* (power of 2) players, conducted as follows:



Background

Single-Elimination (or Knockout) Tournament

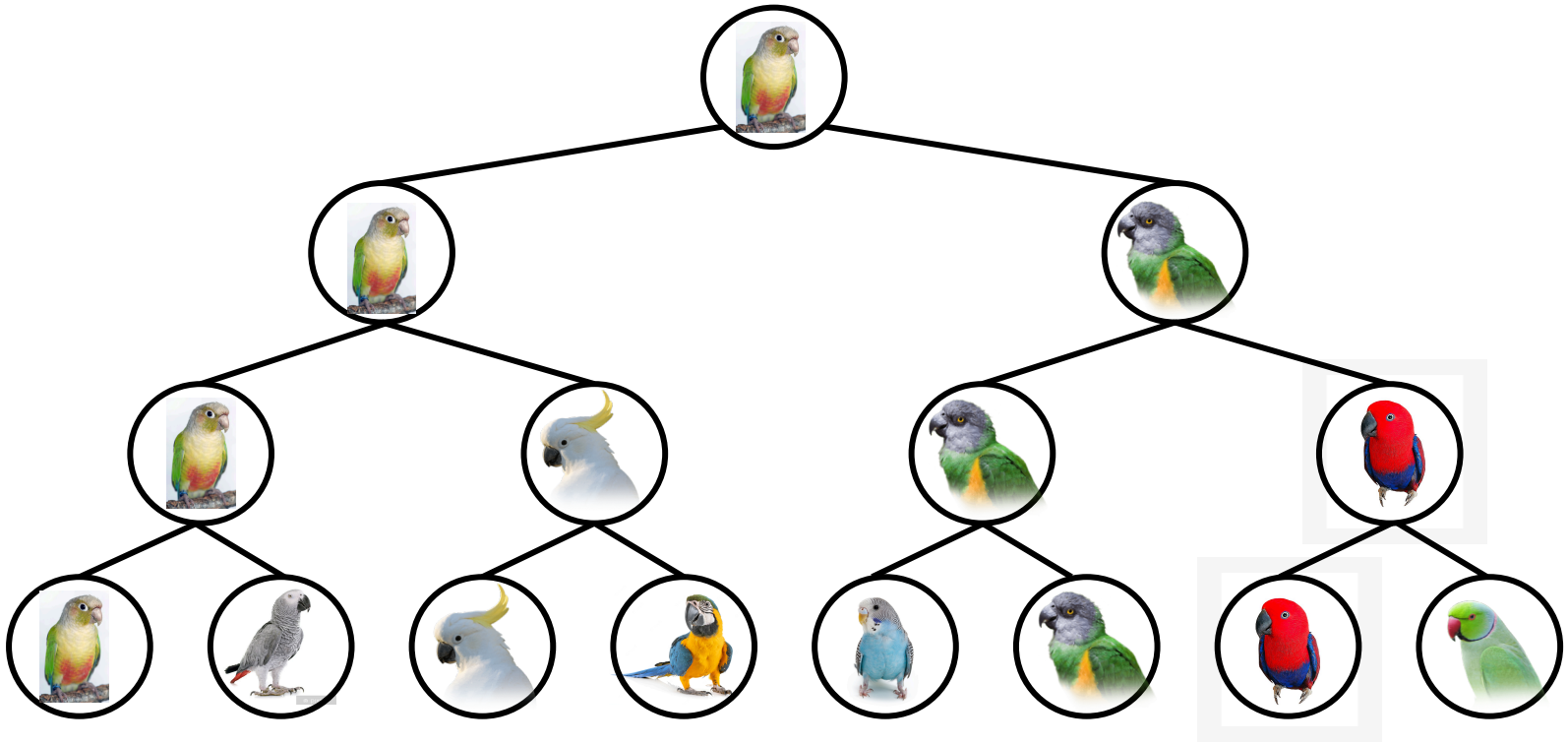
Competition involving *n* (power of 2) players, conducted as follows:



Background

Single-Elimination (or Knockout) Tournament

Competition involving *n* (power of 2) players, conducted as follows:

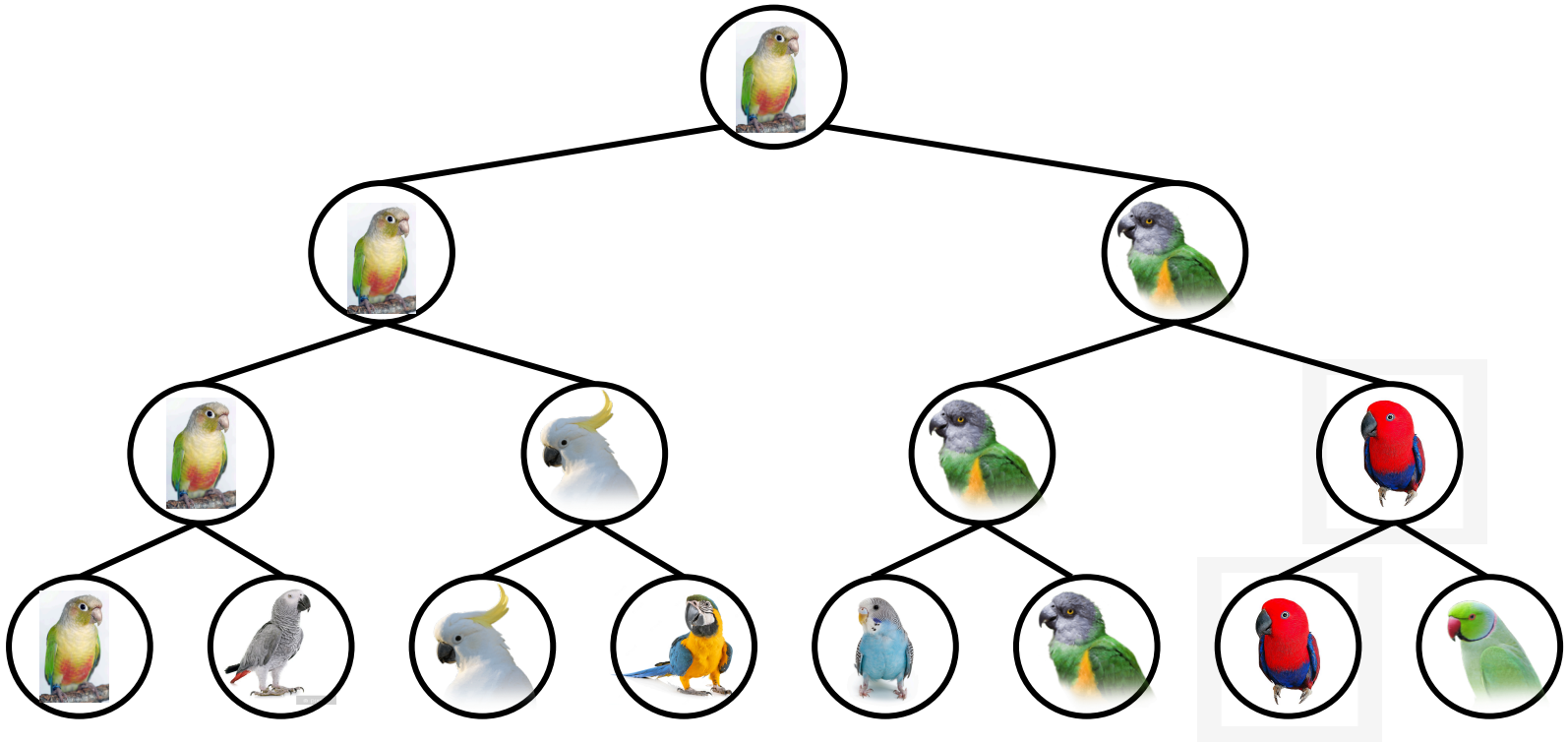


Background

Single-Elimination (or Knockout) Tournament

Competition involving *n* (power of 2) **players**, conducted as follows:

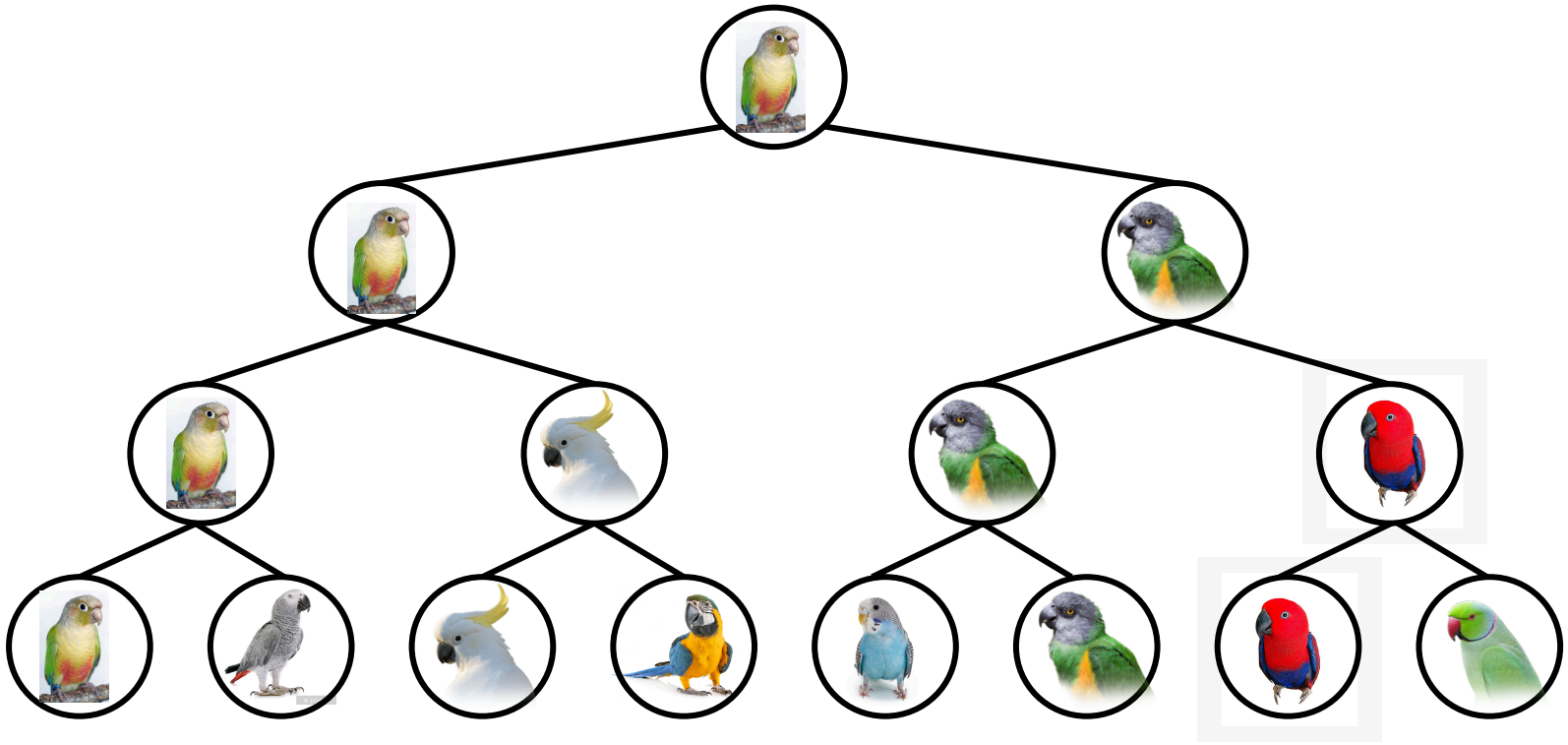
People, companies, ideas, … (Essentially, anything.)



Background

Single-Elimination (or Knockout) Tournament

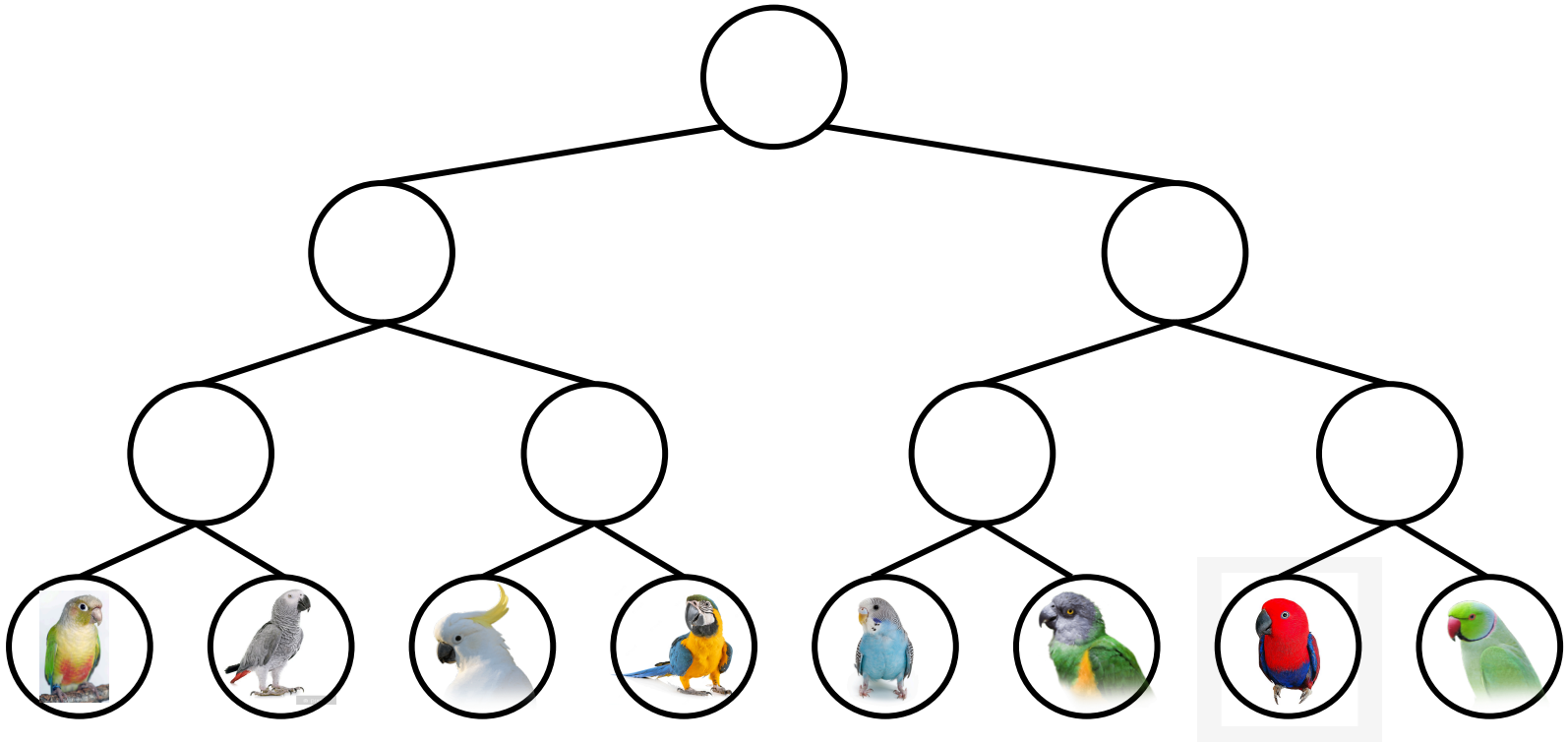
Motivation: Competitions (sports, music, …), voting, webpage ranking, biological interactions, …



Background

Single-Elimination (or Knockout) Tournament

Seeding (or bracket): Assignment of players to the leaves.

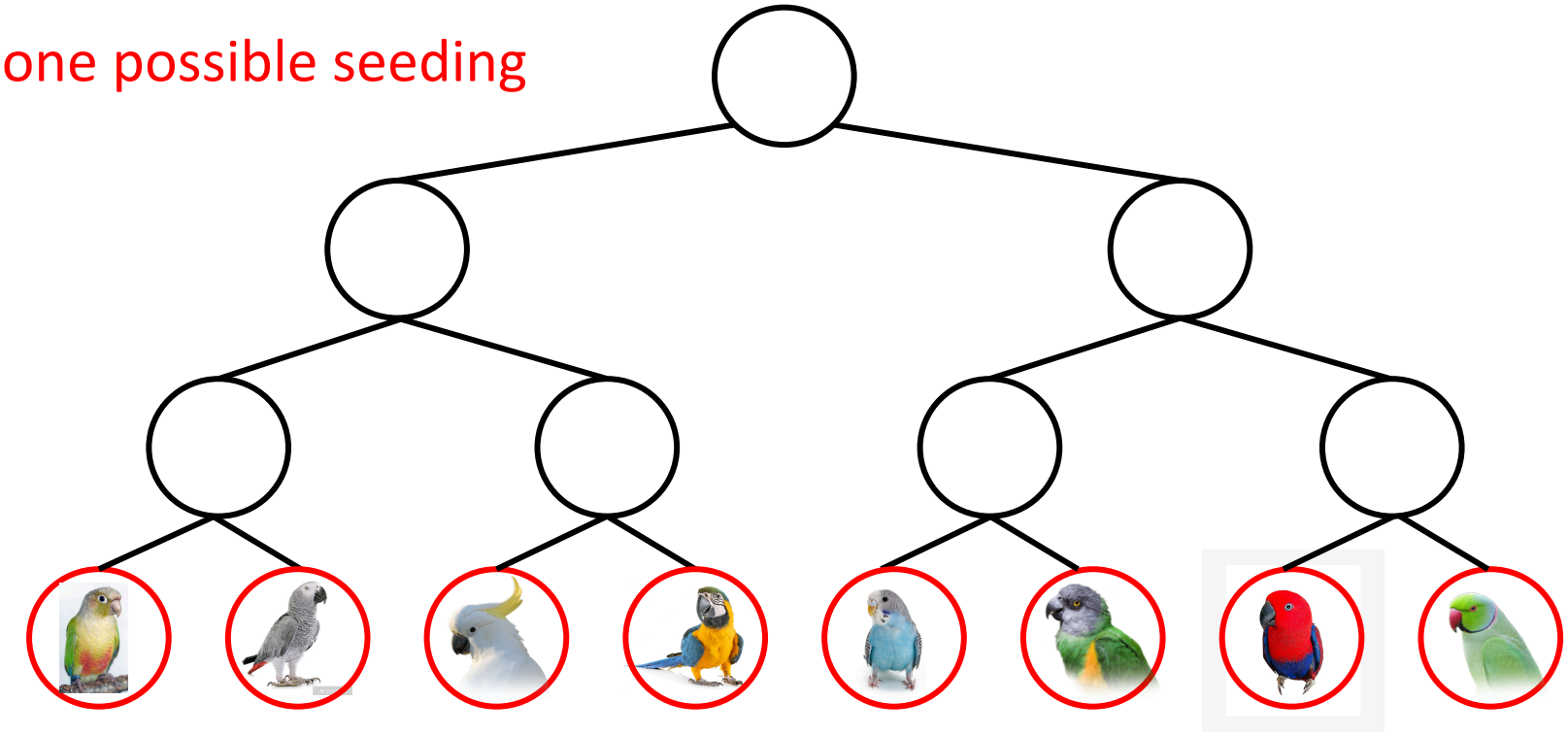


Background

Single-Elimination (or Knockout) Tournament

Seeding (or bracket): Assignment of players to the leaves.

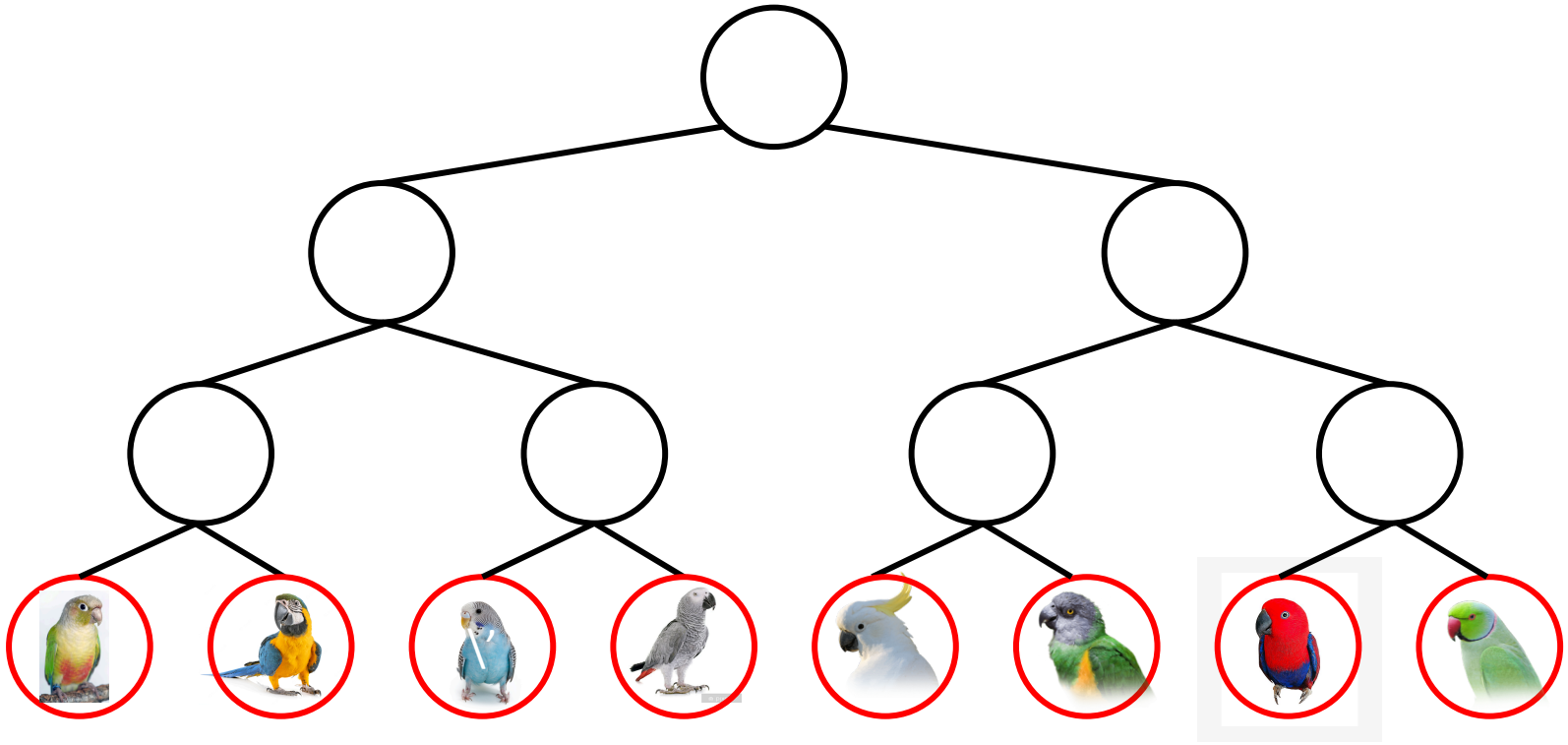
one possible seeding



Background

Single-Elimination (or Knockout) Tournament

Seeding (or bracket): Assignment of players to the leaves.

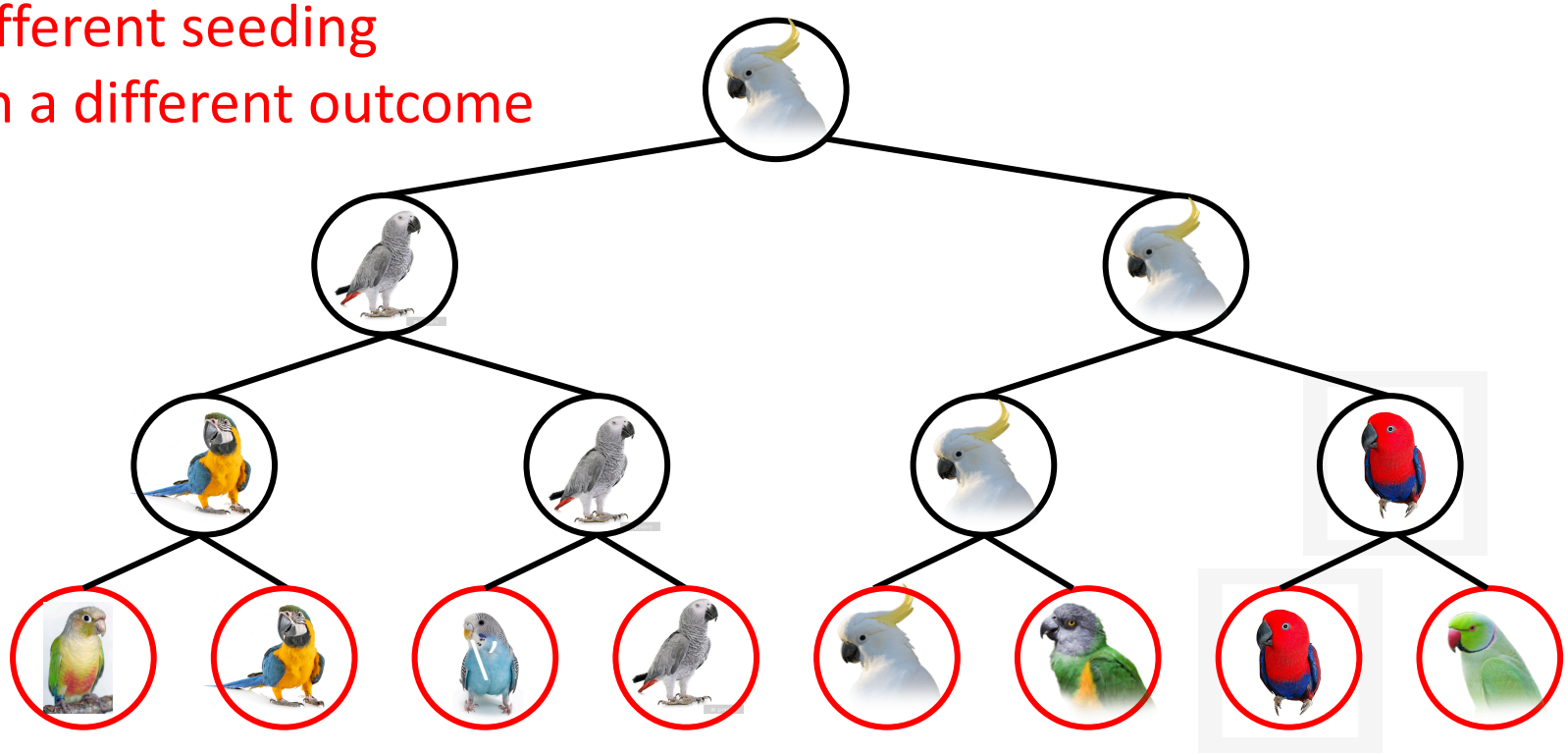


Background

Single-Elimination (or Knockout) Tournament

Seeding (or bracket): Assignment of players to the leaves.

a different seeding
with a different outcome

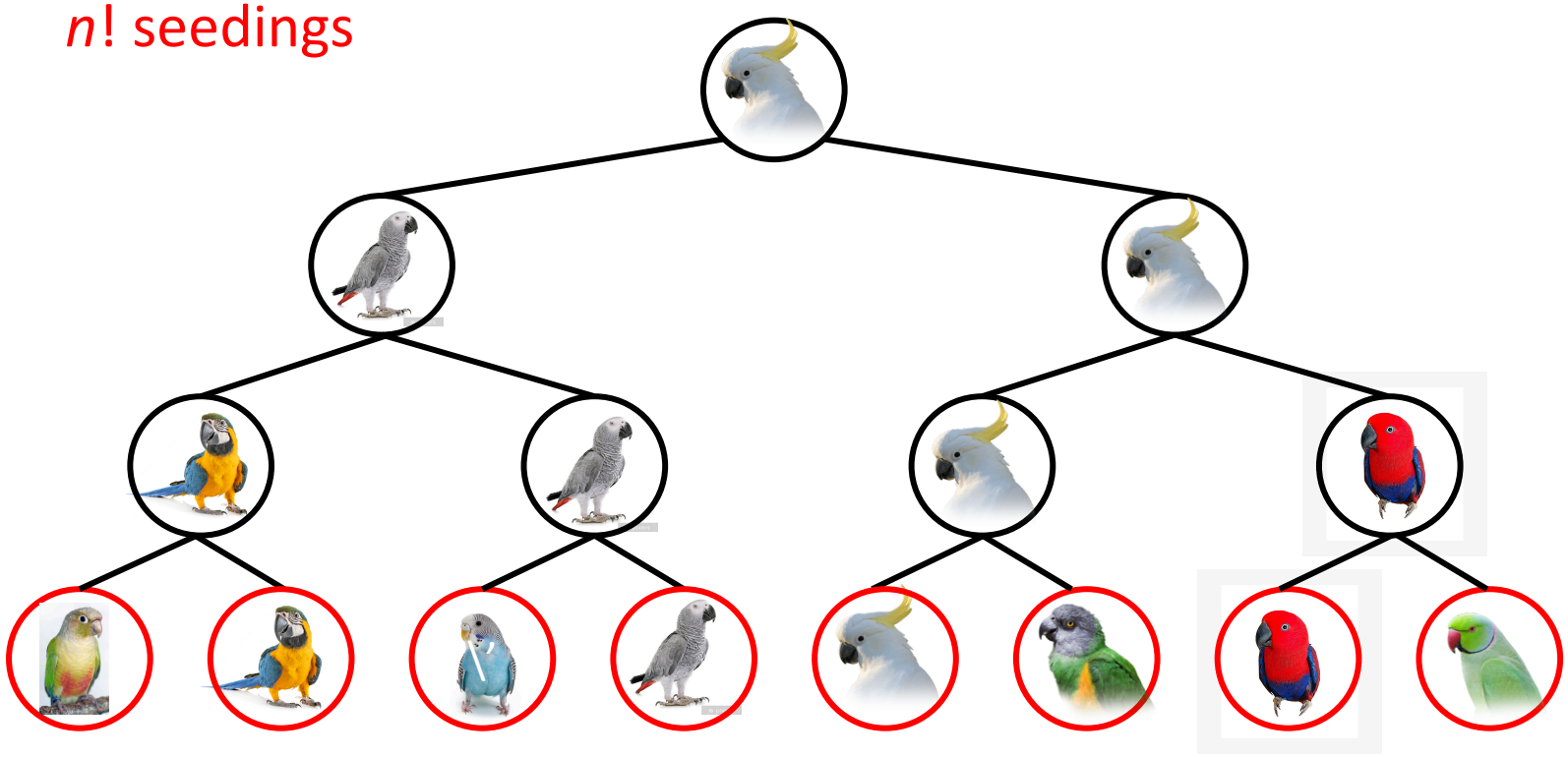


Background

Single-Elimination (or Knockout) Tournament

Seeding (or bracket): Assignment of players to the leaves.

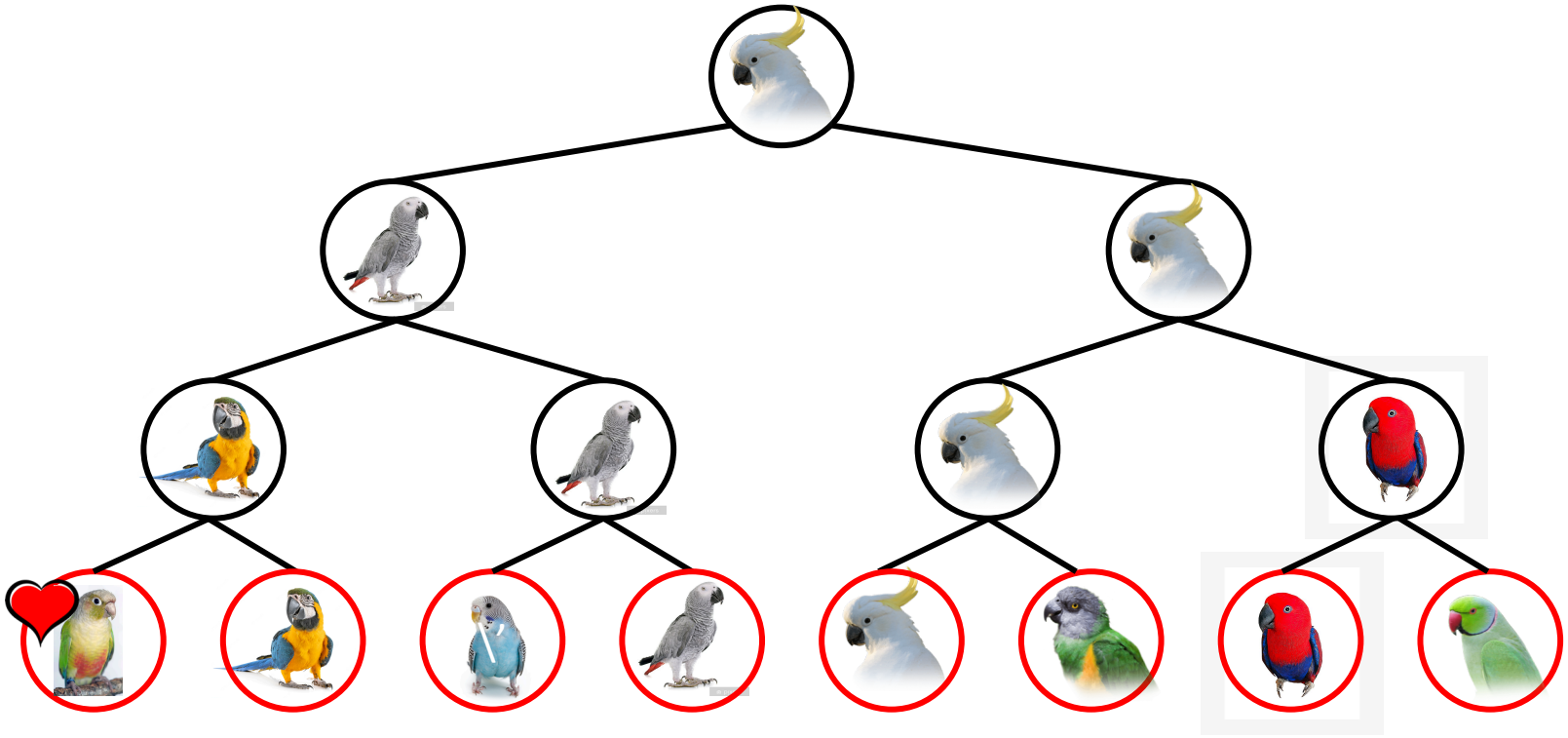
n! seedings



Background

Single-Elimination (or Knockout) Tournament

Objective: Make our favorite player win!



Background

Single-Elimination (or Knockout) Tournament

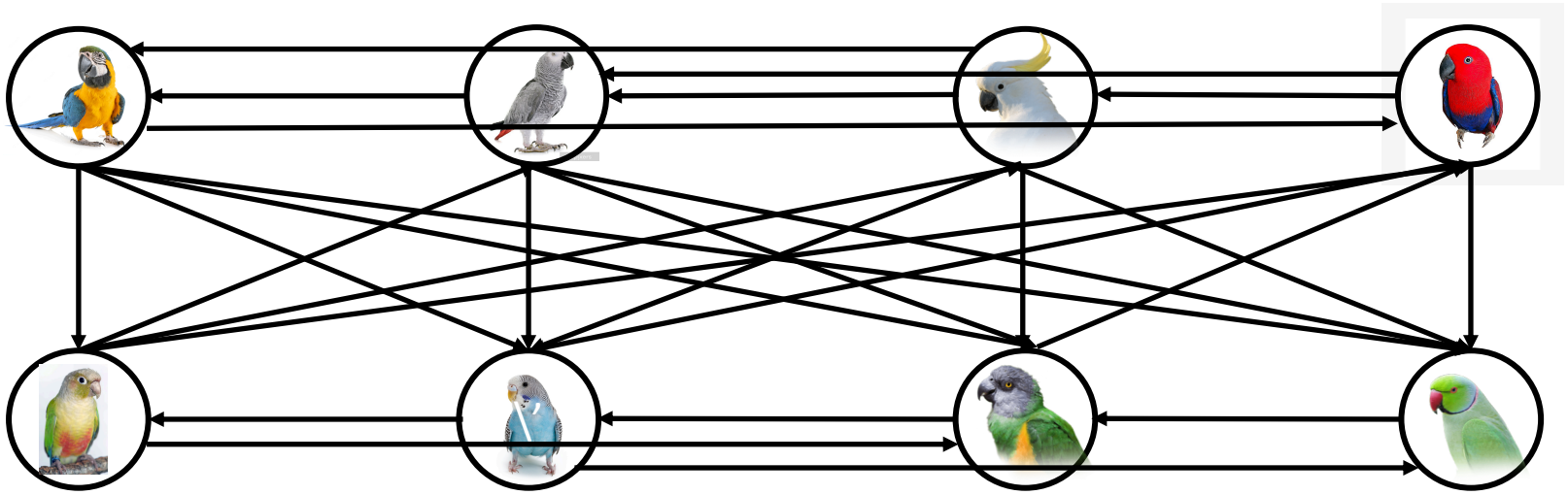
Prediction of outcomes of matches: Tournament graph on the set of players.

Background

Single-Elimination (or Knockout) Tournament

Prediction of outcomes of matches: Tournament graph on the set of players.

Digraph containing, for every pair of vertices *u,v*, either (*u,v*) or (*v,u*).

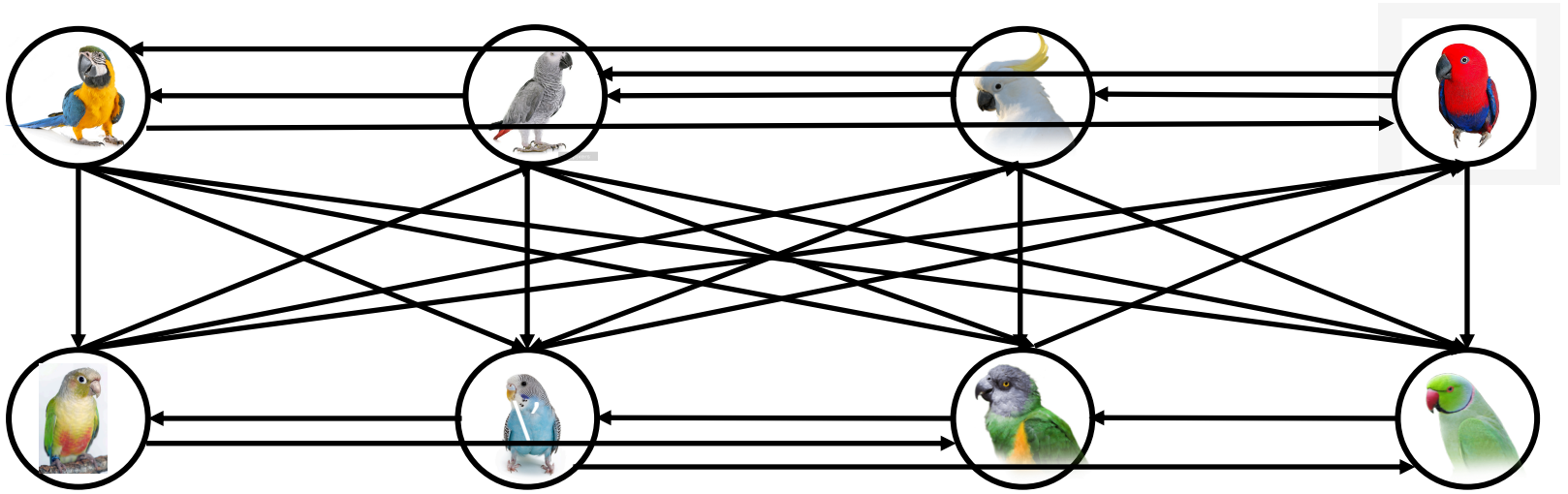


Background

Single-Elimination (or Knockout) Tournament

Prediction of outcomes of matches: Tournament graph on the set of players.

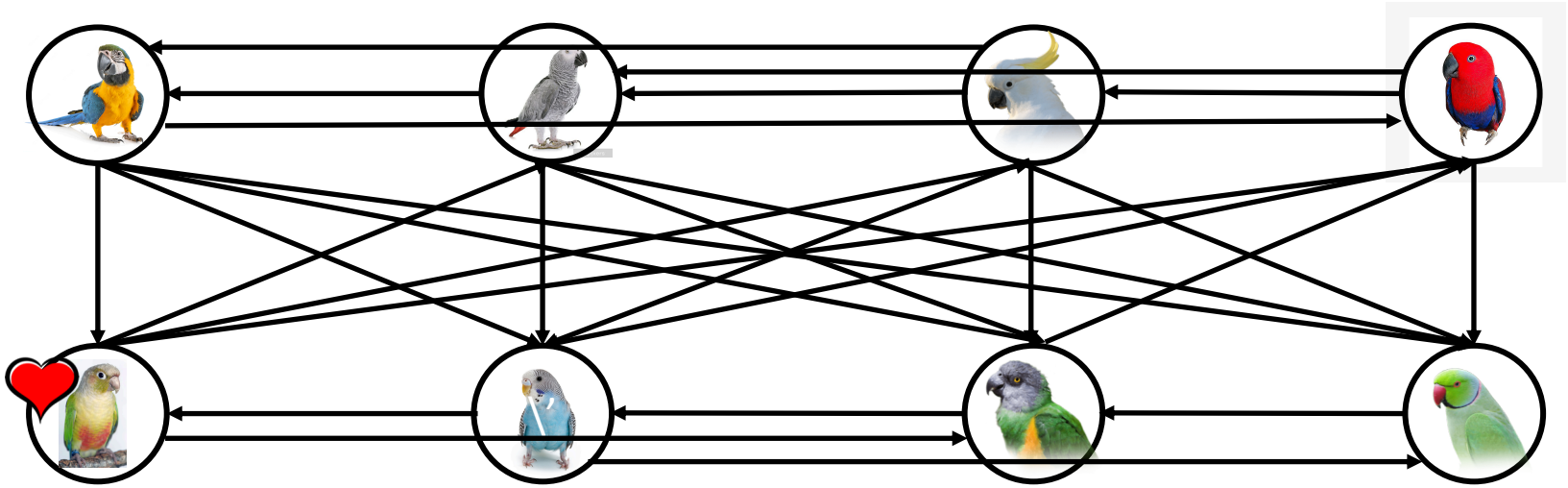
Certainty? Probabilistic models (not discussed here).



Background

Tournament Fixing Problem

Input: Tournament graph on the set of players; favorite player *w*.

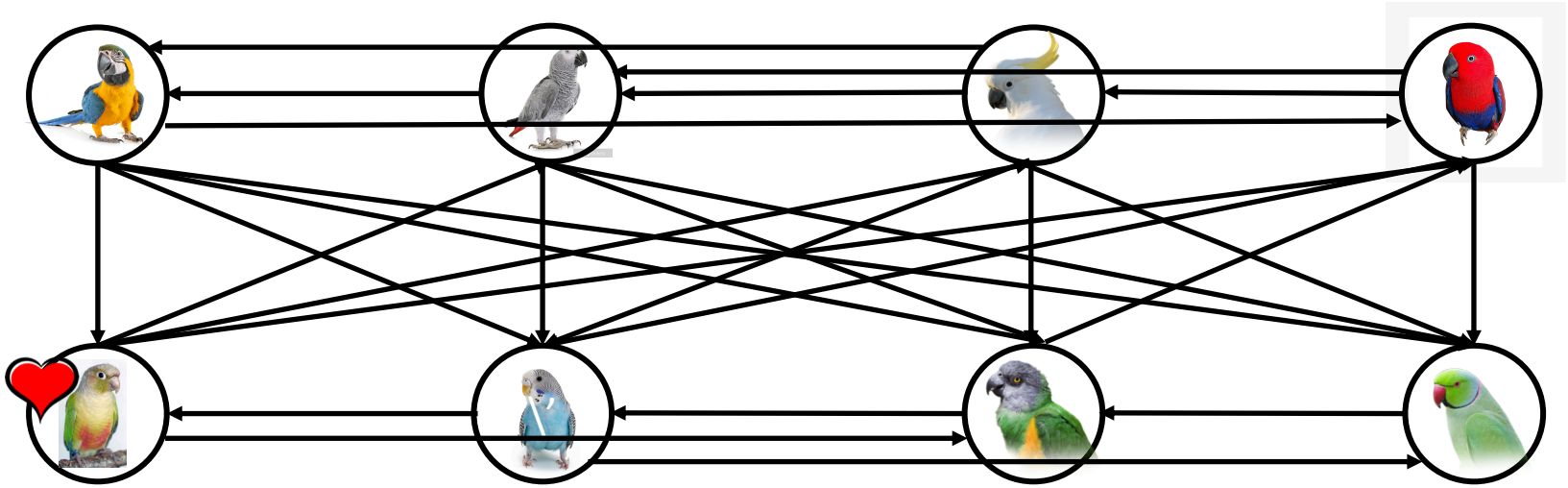


Question: Does there exist a seeding that makes *w* win?

Background

Tournament Fixing Problem

Input: Tournament graph on the set of players; favorite player *w*.



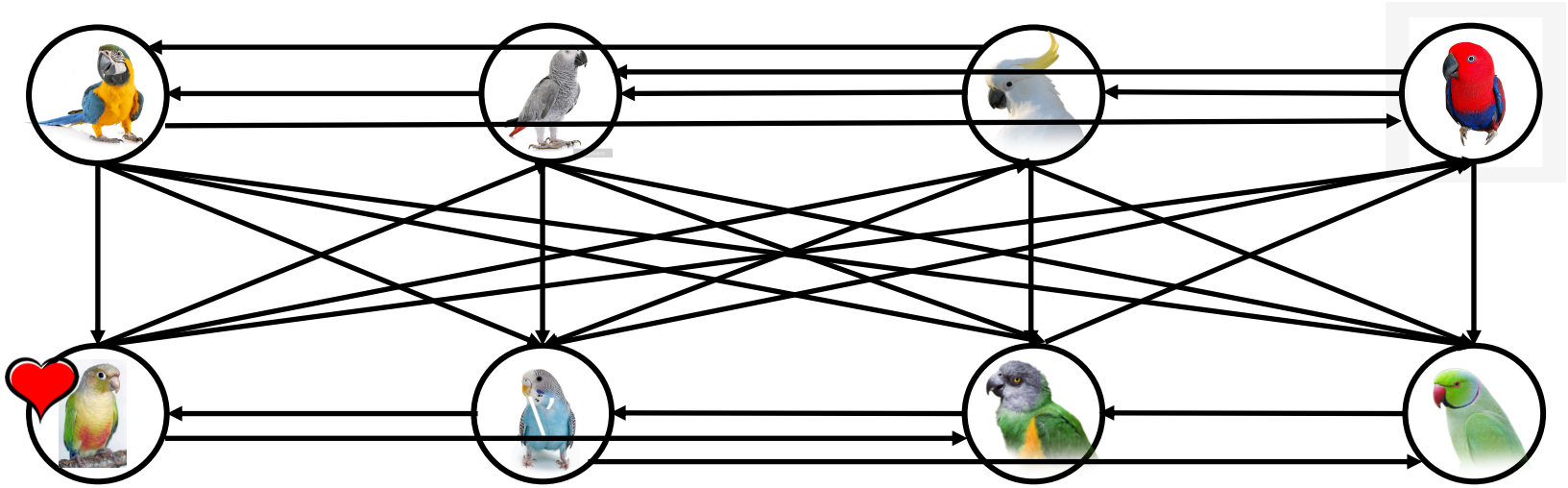
Question: Does there exist a seeding that makes *w* win?

Counting Version: How many seedings make *w* win?

Background

Tournament Fixing Problem

Input: Tournament graph on the set of players; favorite player *w*.



Question: Does there exist a seeding that makes *w* win?

Counting Version: How many seedings make *w* win?

Surveys: [Suksompong, IJCAI'21], [Williams, Handbook of Computational Social Choice, 2016]

Background

Tournament Fixing Problem: Complexity

Theorem. TFP is NP-hard. [AGMMSW, AAAI'14]

- TFP is NP-hard even if *w* is a "king" that beats at least *n*/4 other players. [KW, IJCAI'15]
- #TFP does not admit a fully polynomial-time randomized approximation scheme (FPRAS) unless RP=NP. [AGMMSW, AAAI'14]

For every player *p*, *w* beats *p* or *w* beats a player that beats *p*.

Background

Tournament Fixing Problem: Exact Exponential-Time Algorithms

Theorem. #TFP is solvable in $2^n n^{O(1)}$ time and space. [KW, IJCAI'15]

- Earlier time-space tradeoff by [AGMMSW, AAAI'14]: $O(2.83^n)$ time and $O(1.75^n)$ space, $4^{n+o(n)}$ time and polynomial-space.

Theorem. (Bribery) TFP is solvable in $2^n n^{O(1)}$ time and polynomial-space. [GRSZ, IJCAI'18a]

Background

Tournament Fixing Problem: Exact Exponential-Time Algorithms

Theorem. #TFP is solvable in $2^n n^{O(1)}$ time and space. [KW, IJCAI'15]

- Earlier time-space tradeoff by [AGMMSW, AAAI'14]: $O(2.83^n)$ time and $O(1.75^n)$ space, $4^{n+o(n)}$ time and polynomial-space.

Theorem. (Bribery) TFP is solvable in $2^n n^{O(1)}$ time and polynomial-space. [GRSZ, IJCAI'18a]

Question: Can TFP be solved in time $1.999^n n^{O(1)}$?

Background

Tournament Fixing Problem: Parameterized Complexity

Parameter *k*: Feedback arc set number (fas).

(The minimum number of arcs to remove from the tournament graph to obtain a DAG.)

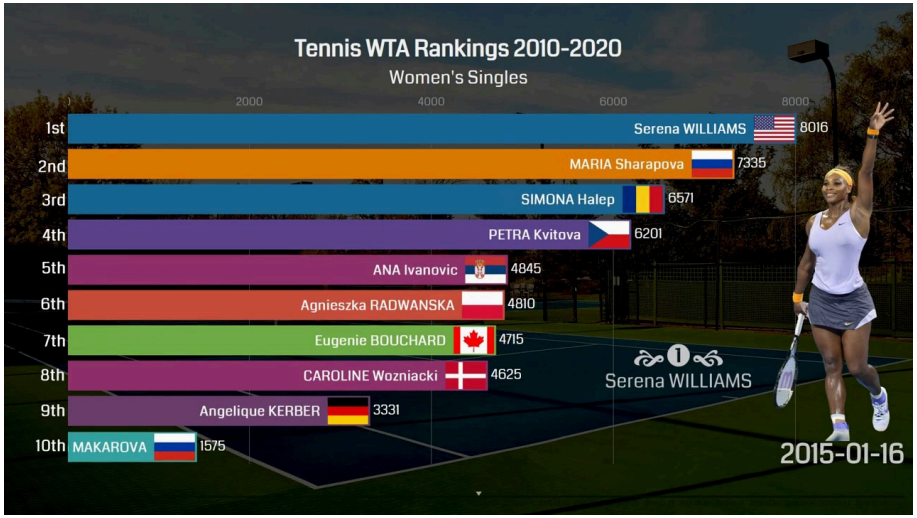
Background

Tournament Fixing Problem: Parameterized Complexity

Parameter *k*: Feedback arc set number (fas).
(The minimum number of arcs to remove from the tournament graph to obtain a DAG.)

Motivation: We know of a (rough) ranking of the players.

- ``Expect'' only few occurrences of a player that beats a player ranked higher than it.



Background

Tournament Fixing Problem: Parameterized Complexity

Parameter *k*: Feedback arc set number (fas).
(The minimum number of arcs to remove from the tournament graph to obtain a DAG.)

Theorem. TFP is solvable in $2^{O(k\log k)}n^{O(1)}$ time. [GRSZ, IJCAI'18b]

- Previously: $2^{O(k^2\log k)}n^{O(1)}$ time. [RS, AAAI'17]

Theorem. TFP admits a polynomial kernel. [GRSZ, IJCAI'19]

Background

Tournament Fixing Problem: Parameterized Complexity

Parameter *k*: Feedback arc set number (fas).
(The minimum number of arcs to remove from the tournament graph to obtain a DAG.)

Theorem. TFP is solvable in $2^{O(k\log k)}n^{O(1)}$ time. [GRSZ, IJCAI'18b]

- Previously: $2^{O(k^2\log k)}n^{O(1)}$ time. [RS, AAAI'17]

Theorem. TFP admits a polynomial kernel. [GRSZ, IJCAI'19]

Question. Better (or best) running time? (**Subexponential?**)

Question. Is TFP in FPT parameterized by the feedback **vertex** set number?

Background

Tournament Fixing Problem: Some Structural Results

- ***w* is a ``superking'':** For every player *p*, *w* beats *p* or *w* beats at least $\log_2 n$ players that beat *p*. [W, AAAI'10]
- ***w* is a king of ``high-degree'':** A king with outdegree *d* that loses to fewer than *d* players with outdegree >*d*. [SW, IJCAI'11]

➢ Generalization by [KSW, AAAI'16]

- ***w* is a 3-king + other restrictions:** [KW, IJCAI'15; KSW, AAAI'16].

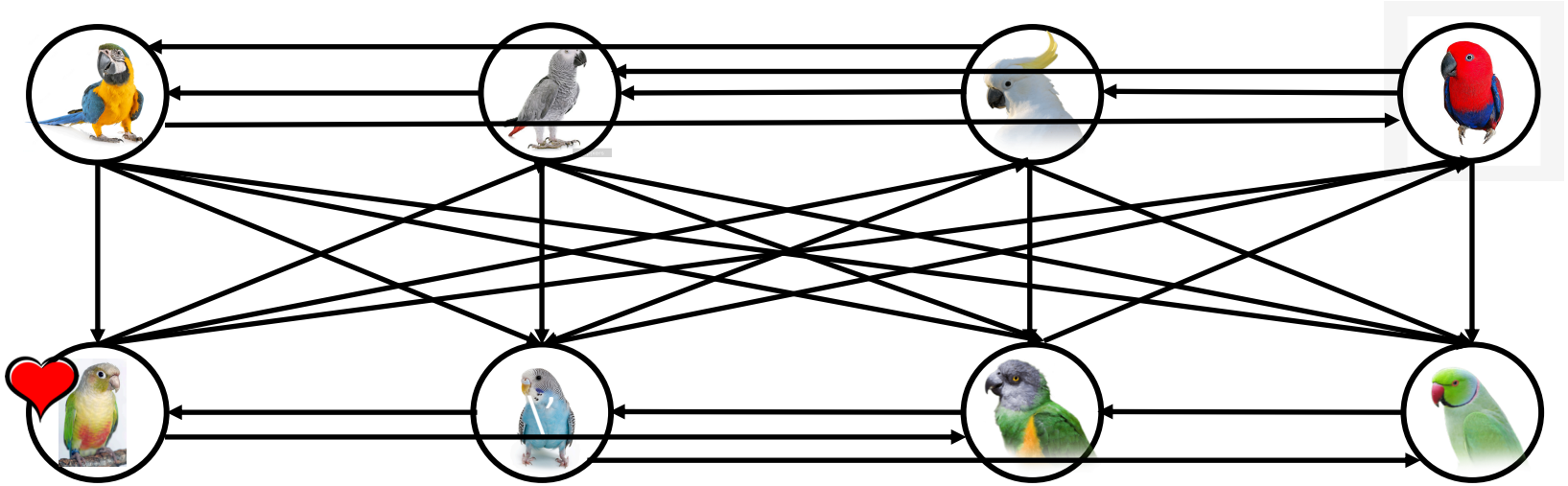
(A 3-king is a player that can reach any other player via a path of length 3 in the tournament graph.)

- **Note:** A 3-king may not be able to win even if it beats *n*-3 other players.

Background

Bribery Tournament Fixing Problem

Input: Tournament graph on the set of players; favorite player *w;* budget *b*.

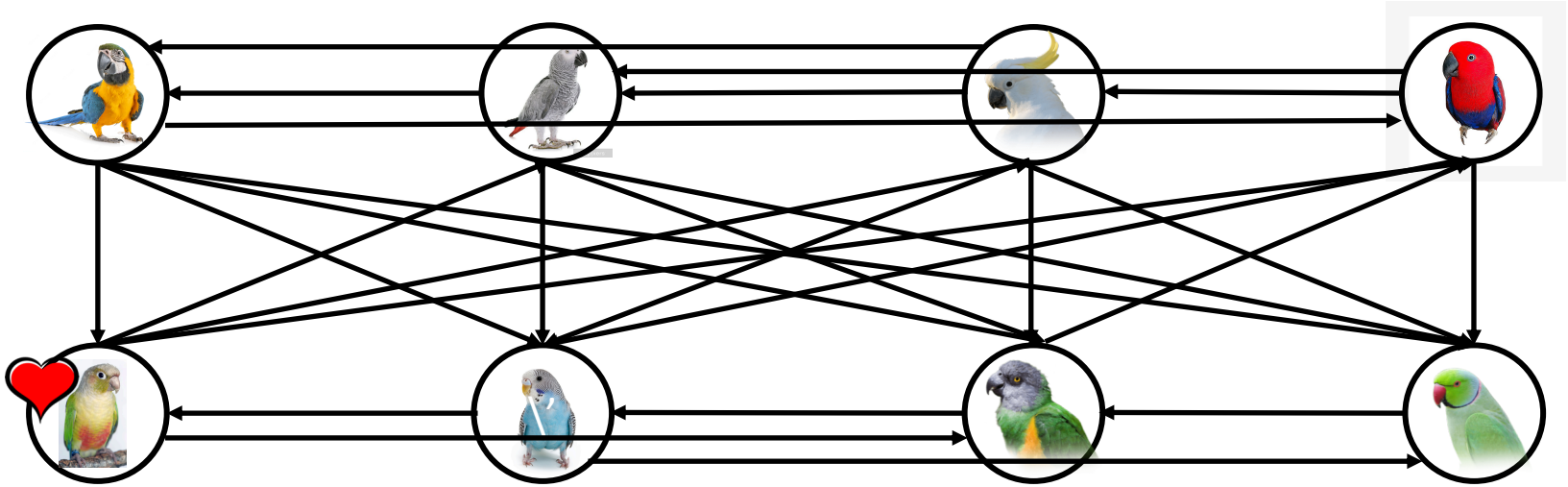


Question: Does there exist a seeding such that, after changing the outcomes of at most *b* matches, *w* wins?

Background

Bribery Tournament Fixing Problem

Input: Tournament graph on the set of players; favorite player *w;* budget *b*.



Question: Does there exist a seeding such that, after changing the outcomes of at most *b* matches, *w* wins?

Note: TFP is the special case of BTFP where *b*=0.

Background

Bribery Tournament Fixing Problem: Complexity

Corollary. BTFP is NP-hard when b=0.

Observation. BTFP is polynomial-time solvable when $b \geq \log_2 n$. (Output Yes.)

Theorem. For any fixed ε>0, BTFP is NP-hard when $b \leq (1- \varepsilon)\log_2 n$. [KW, IJCAI'15]

Background

Bribery Tournament Fixing Problem: Complexity

Corollary. BTFP is NP-hard when b=0.

Observation. BTFP is polynomial-time solvable when $b \geq \log_2 n$. (Output Yes.)

Theorem. For any fixed ε>0, BTFP is NP-hard when $b \leq (1- \varepsilon)\log_2 n$. [KW, IJCAI'15]

Question(s). Non-trivial results for the parameter $\log_2 n$-b?

Background

Bribery Tournament Fixing Problem: Exponential-Time Algorithms and Parameterized Complexity

Parameter *k*: Feedback arc set number (fas).

Not harder than TFP:

Theorem. I. BTFP is solvable in $2^n n^{O(1)}$ time and polynomial-space.
II. BTFP is solvable in $2^{O(k\log k)} n^{O(1)}$ time. [GRSZ, IJCAI'18b]

In the manuscript: $2^{O(k^2\log k)} n^{O(1)}$.

Background

Bribery Tournament Fixing Problem: Structural Results

Theorem (informal). If there is a solution, then there is one manipulating only edges $\{w,v\}$ where v belongs to an "elite club". [GRSZ, IJCAI'18b]

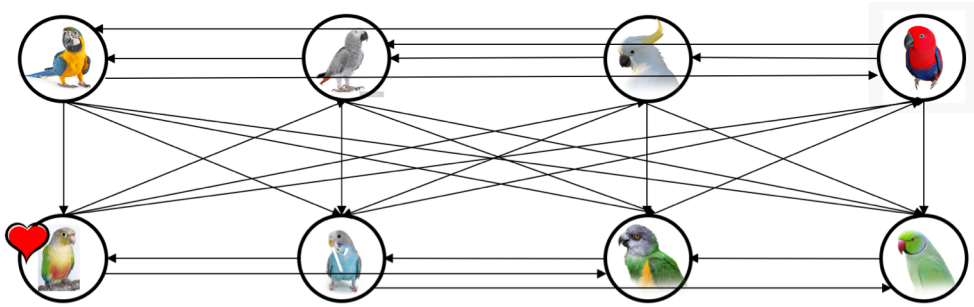
Also in [GRSZ, IJCAI'18b]:

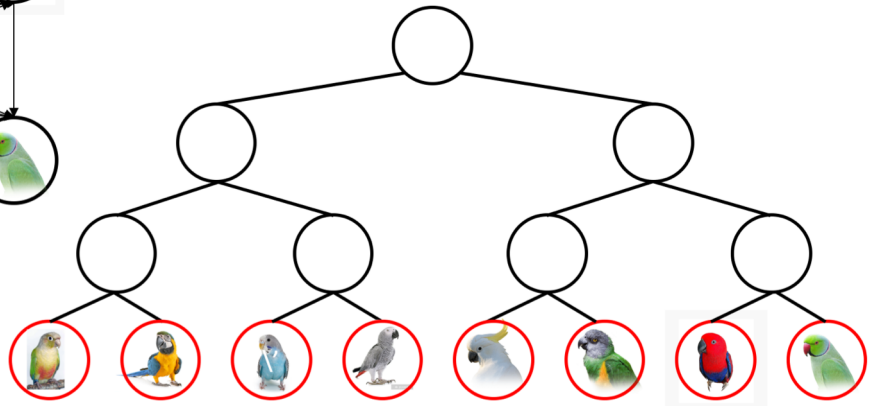
- Characterization of Yes-instances. (Yields a simple formula for DAGs.)
- ``Obfuscation'' operations.

Background

Bribery Tournament ~~Fixing~~ Problem

Input: Tournament graph on the set of players; favorite player *w;* budget *b*; seeding.





Question: Is it possible to change the outcomes of at most *b* matches so that *w* wins?

Background

Bribery Tournament ~~Fixing~~ Problem: Complexity

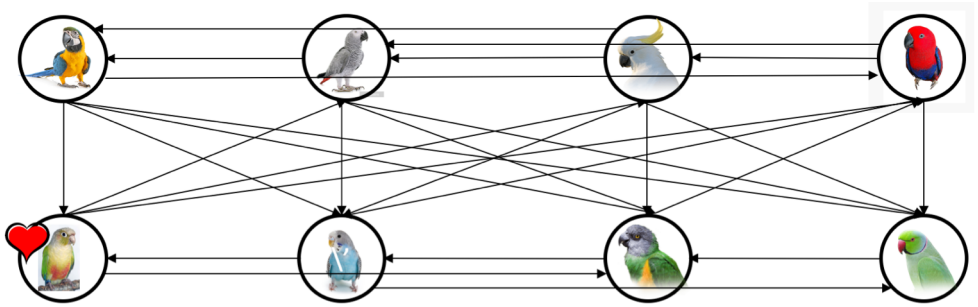
Theorem. BTP is solvable in polynomial time.[RW, ADT'09]

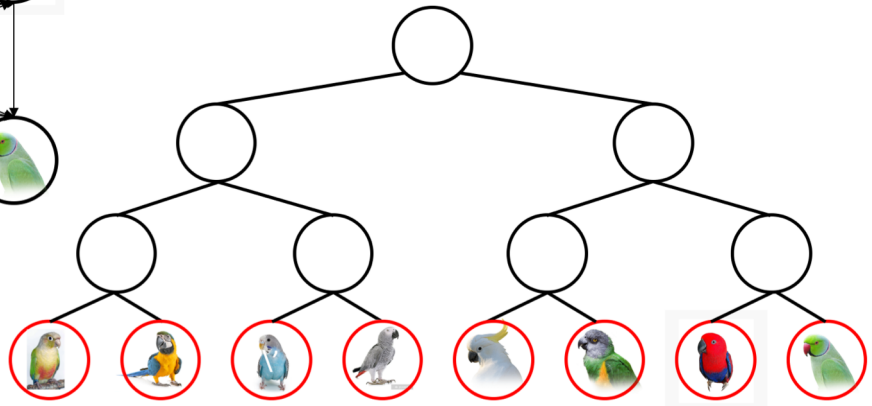
- **Also:** The ``destructive'' version (prevent *w* from winning) is also solvable in polynomial time.

Background

``Closest'' Tournament Fixing Problem

Input: Tournament graph on the set of players; favorite player *w;* seeding; distance threshold *d*.



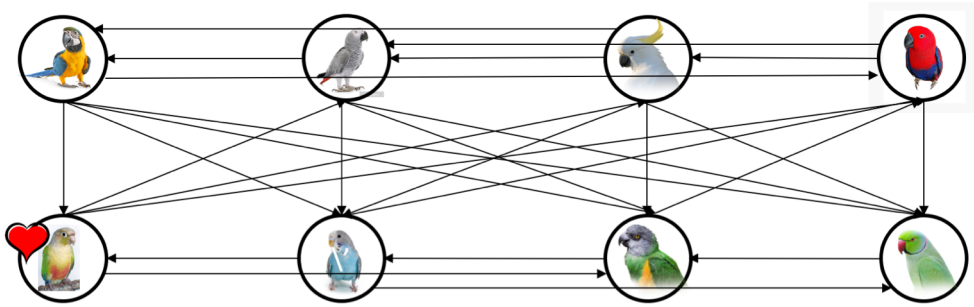


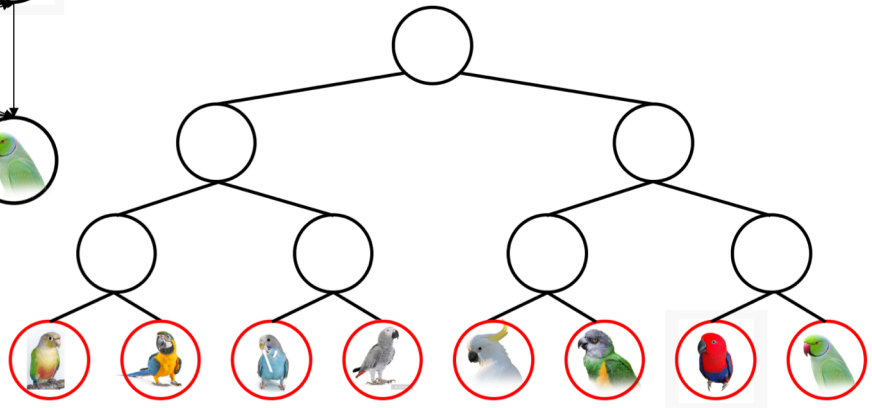
Question: Does there exist a seeding of Hamming distance at most *d* from the given seeding that makes *w* win?

Background

``Closest'' Tournament Fixing Problem

Input: Tournament graph on the set of players; favorite player *w;* seeding; distance threshold *d*.





Question: Does there exist a seeding of Hamming distance at most *d* from the given seeding that makes *w* win?

Note: TFP is the special case of CTFP where *d*=*n*.

Background

``Closest'' Tournament Fixing Problem: Motivation

Why is the input seeding attractive? Might have been already agreed upon, might be based on customs or traditions, …

Remark 1: Cyclic shifts. ABCDEFGH is similar to HABCDEFG, but all starting matches are different.

Remark 2: Extreme case. All starting matches are the same in ABCDEFGH and BADCFEHG, but the Hamming distance is *n*.

- Still, the order of each pair might be important. (Who plays first, who plays at what court, …).

Background

Bribery Tournament Fixing Problem: Complexity & Algorithms

Corollary. CTFP is NP-hard when $d=n$.

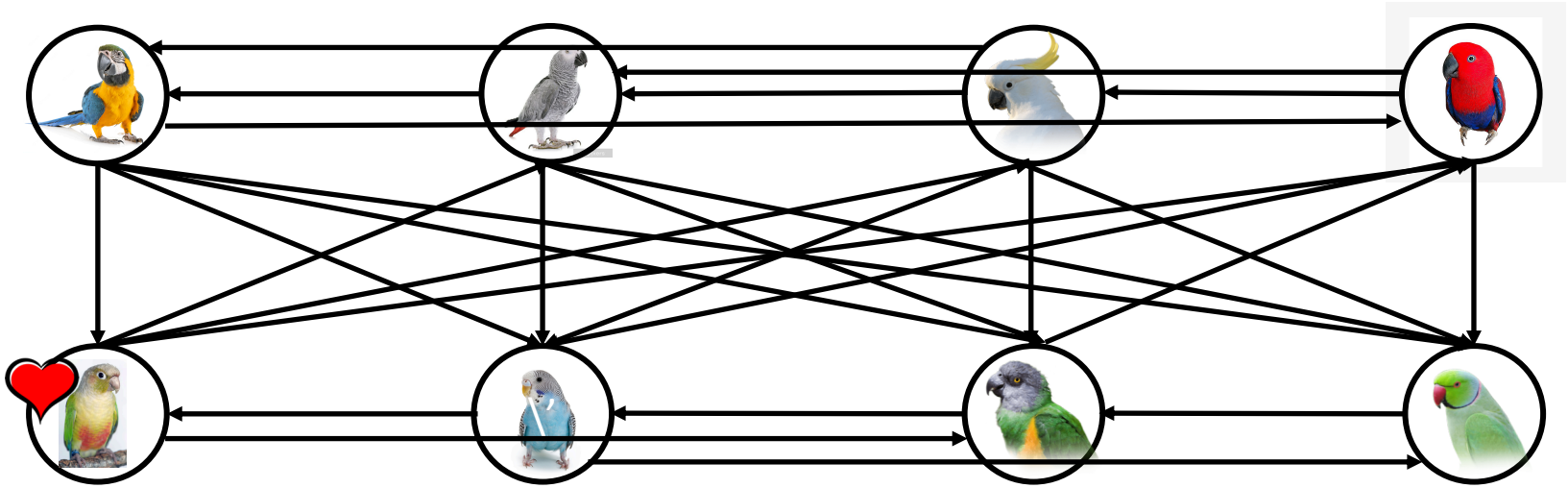
Theorem. CTFP is W[1]-hard when parameterized by d.
[GSZ, Manuscript]

Theorem. CTFP is solvable in $2^{O(n)}$ time. [GSZ, Manuscript]

Background

k-Wins Tournament Fixing Problem

Input: Tournament graph on the set of players; favorite player *w*; *k*.



Question: Does there exist a seeding that makes *w* win at least *k* matches?

Note: TFP is the special case of WTFP where $k=\log_2 n$.

Background

k-Wins Tournament Fixing Problem: Complexity & Algorithms

Corollary. WTFP is NP-hard when $k=\log_2 n$.

Theorem. WTFP is FPT when parameterized by t, the maximum number of players that a player beats.
[ACS, Annals of Mathematics and Artificial Intelligence, 2017]

- Also in [ACS, …'17]: Generalizations of WTFP. One of them is proved to be W[1]-hard wrt k.

Background

k-Wins Tournament Fixing Problem: Complexity & Algorithms

Corollary. WTFP is NP-hard when $k=\log_2 n$.

Theorem. WTFP is FPT when parameterized by t, the maximum number of players that a player beats.
[ACS, Annals of Mathematics and Artificial Intelligence, 2017]

- Also in [ACS, …'17]: Generalizations of WTFP. One of them is proved to be W[1]-hard wrt k.

Question. Is WTFP W[1]-hard or in FPT wrt k?

Background

Remark. Knockout tournament is only one out of several formats of competition that were studied in the literature.

Round-Robin Tournament. Every player plays against every other player. Usually, the winner(s) is the player that has won the largest number of matches.

The Fair Non-Eliminating Tournament Problem. Given the infrastructure of the tournament, and the rankings of the players (multiset of integers), determine if there exists an assignment of the players to the vertices so that the sum of the rankings of the neighbors of every vertex is the same.

- Analyzed the parameterized complexity of this problem wrt several parameters (*tw, fvs, vc,* Δ*,* α) and combinations thereof. [GZ, AAMAS'21]

Outline

Background

FPT algorithm for the TF problem

Kernel for the TF problem

FPT Algorithm for TFP

Theorem. TFP is solvable in $2^{O(k \log k)} n^{O(1)}$ time. [GRSZ, IJCAI'18b]

FPT Algorithm for TFP

Theorem. TFP is solvable in $2^{O(k\log k)}n^{O(1)}$ time. [GRSZ, IJCAI'18b]

Phase I: Guessing

First, compute an fas F of D:

Option 1: Branch in $3^k n^{O(1)}$ time, or use a known $2^{O(\sqrt{k})}n^{O(1)}$-time algorithm.

Option 2: Use a polynomial-time 3-approximation algorithm, or use a known PTAS.

FPT Algorithm for TFP

Theorem. TFP is solvable in $2^{O(k\log k)}n^{O(1)}$ time. [GRSZ, IJCAI'18b]

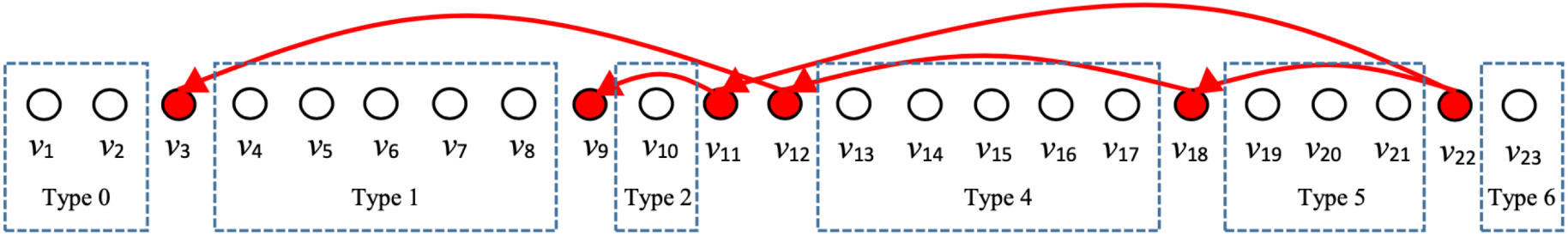
Phase I: Guessing

First, compute an fas F of D:

Option 1: Branch in $3^k n^{O(1)}$ time, or use a known $2^{O(\sqrt{k})}n^{O(1)}$-time algorithm.

Option 2: Use a polynomial-time 3-approximation algorithm, or use a known PTAS.

Why is an fas useful?



FPT Algorithm for TFP

Theorem. TFP is solvable in $2^{O(k\log k)}n^{O(1)}$ time. [GRSZ, IJCAI'18b]

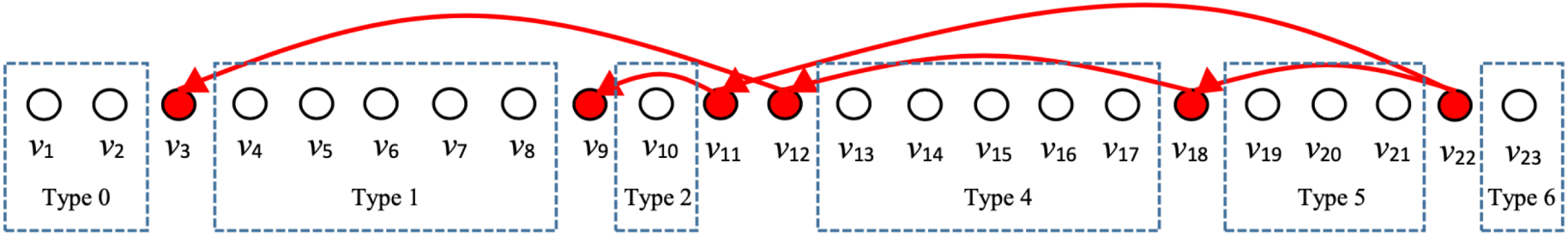
Phase I: Guessing

First, compute an fas F of D:

Option 1: Branch in $3^k n^{O(1)}$ time, or use a known $2^{O(\sqrt{k})}n^{O(1)}$-time algorithm.

Option 2: Use a polynomial-time 3-approximation algorithm, or use a known PTAS.

Why is an fas useful?



For simplicity, suppose that w belongs to V_F.

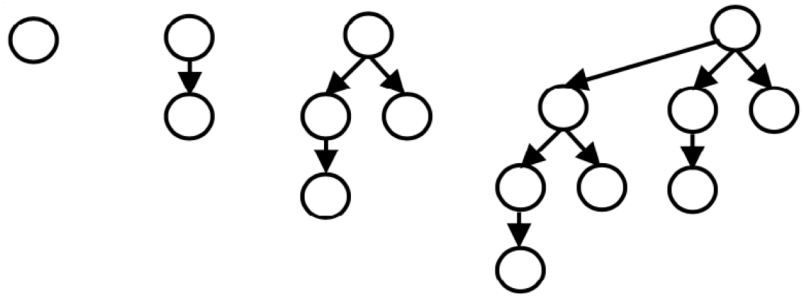
FPT Algorithm for TFP

Phase I: Guessing

Unlabeled Binomial Arborescence (UBA). Rooted tree B, defined recursively:

- A single vertex.
- Given two disjoint UBAs of equals size, B_v and B_u, the addition of an arc from v to u results in a BA rooted at v.

Given a digraph D such that $V(B)=V(D)$, we say that B is a (labeled) spanning binomial arborescence (SBA) of D.



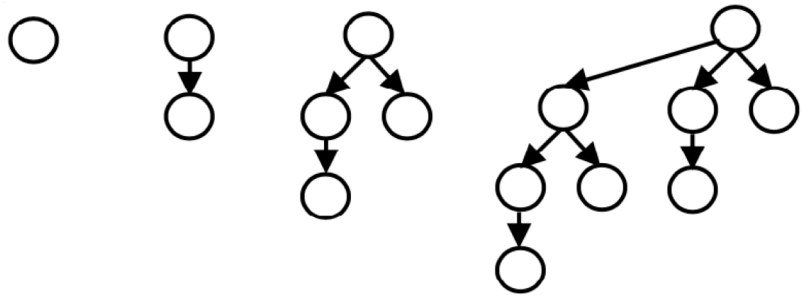
FPT Algorithm for TFP

Phase I: Guessing

Unlabeled Binomial Arborescence (UBA). Rooted tree B, defined recursively:

- A single vertex.
- Given two disjoint UBAs of equals size, B_v and B_u, the addition of an arc from v to u results in a BA rooted at v.

Given a digraph D such that $V(B)=V(D)$, we say that B is a (labeled) spanning binomial arborescence (SBA) of D.



Theorem. (D,w) is a Yes-instance iff D has an SBA rooted at w. [W, AAAI'10]

FPT Algorithm for TFP

Phase I: Guessing

Objective. Guess the relative position of the vertex set of F, V_F, in the (unknown) SBA B rooted at w.

FPT Algorithm for TFP

Phase I: Guessing

Objective. Guess the relative position of the vertex set of F, V_F, in the (unknown) SBA B rooted at w.

LCA-closure. Given a rooted tree T and a subset M of $V(T)$, LCA(M) is defined as follows. Initially, $M'=M$. As long as there exist two vertices in M whose lca is not in M', add it to M'.

FPT Algorithm for TFP

Phase I: Guessing

Objective. Guess the relative position of the vertex set of F, V_F, in the (unknown) SBA B rooted at w.

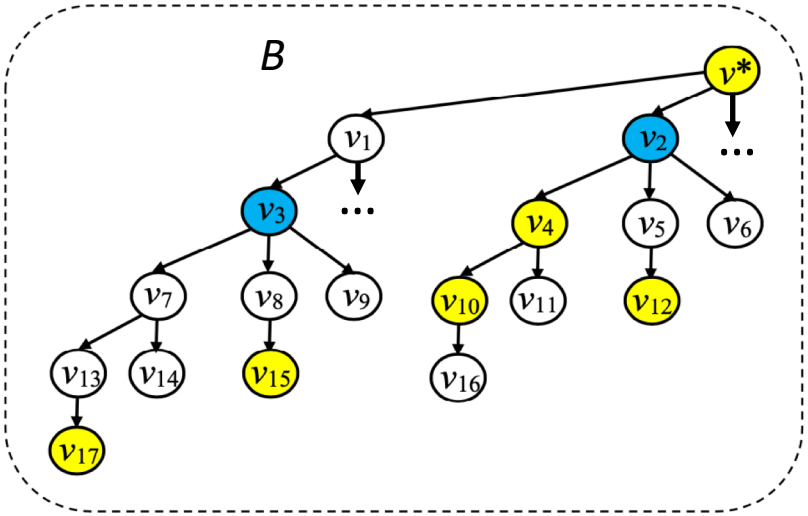
LCA-closure. Given a rooted tree T and a subset M of $V(T)$, LCA(M) is defined as follows. Initially, $M'=M$. As long as there exist two vertices in M whose lca is not in M', add it to M'.

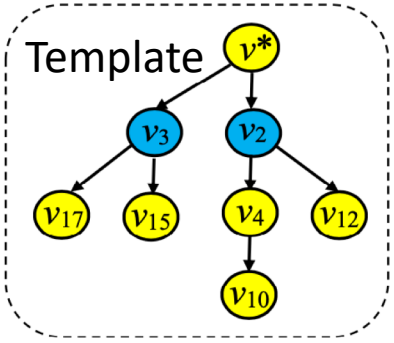
Folklore. $|\text{LCA}(M)| \leq 2|M|$, and every connected component of T-LCA(M) has at most two neighbors in LCA(M).

FPT Algorithm for TFP

Phase I: Guessing

Template of V_F in B. Remove the vertices outside V_F without descendants from V_F; ``dissolve'' the remaining (degree-2) vertices outside LCA(V_F).



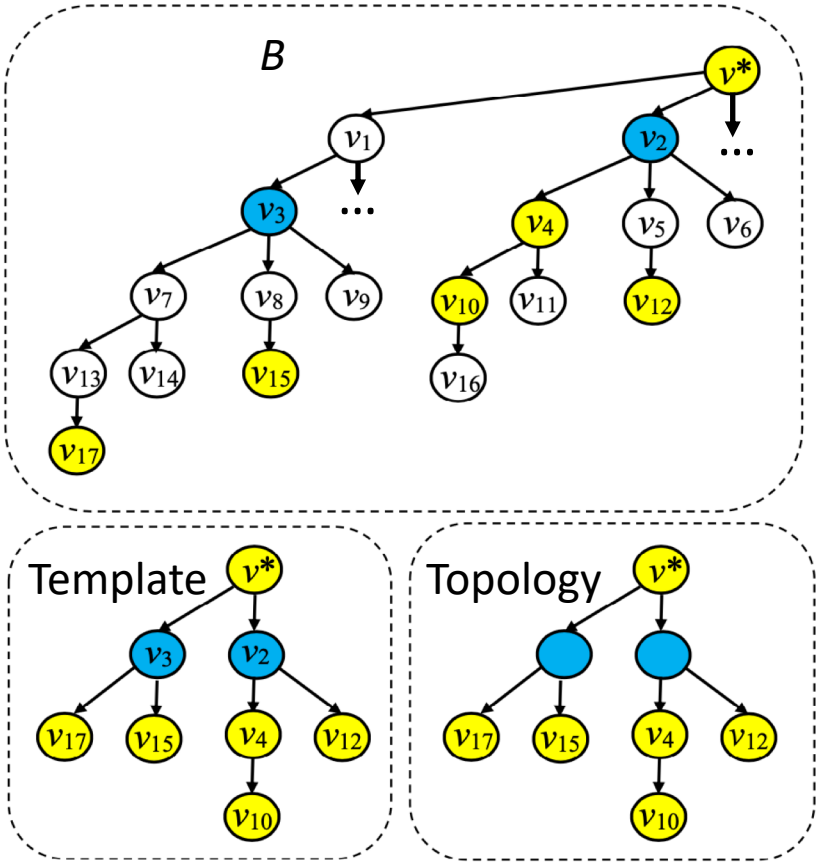


FPT Algorithm for TFP

Phase I: Guessing

Template of V_F in B. Remove the vertices outside V_F without descendants from V_F; ``dissolve'' the remaining (degree-2) vertices outside LCA(V_F).

Topology of V_F in B. Forget the identity of the vertices in LCA(V_F)\V_F.



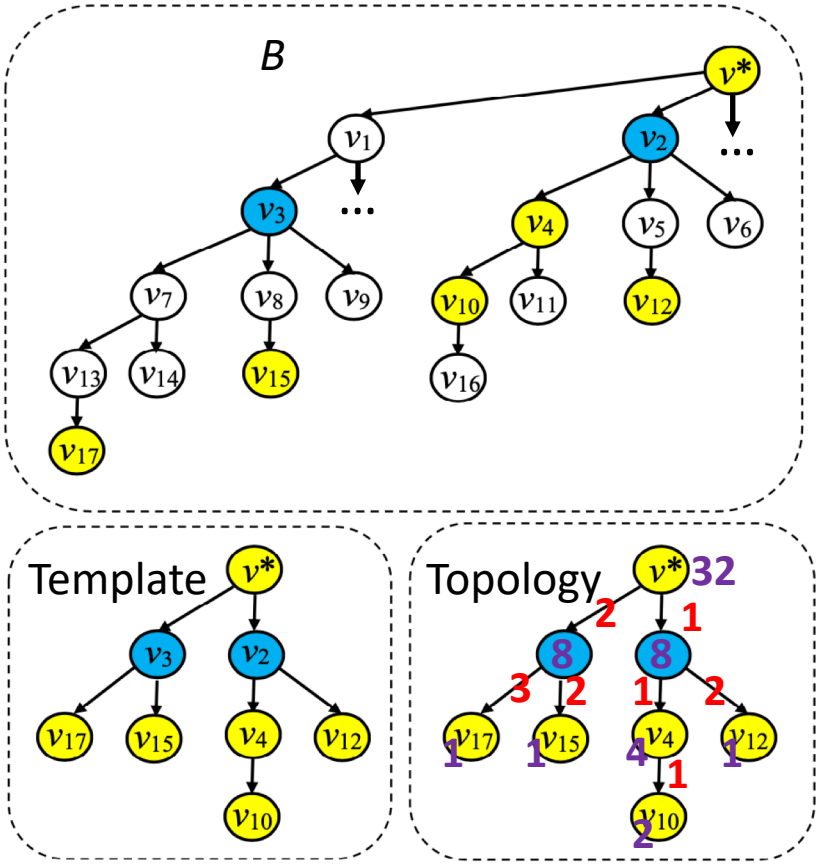
FPT Algorithm for TFP

Phase I: Guessing

Template of V_F in B. Remove the vertices outside V_F without descendants from V_F; ``dissolve'' the remaining (degree-2) vertices outside LCA(V_F).

Topology of V_F in B. Forget the identity of the vertices in LCA(V_F)\V_F.

Lengths+Sizes. For every edge: assign the **length** of its path in B. For every vertex: assign the **size** of its subtree in B.



FPT Algorithm for TFP

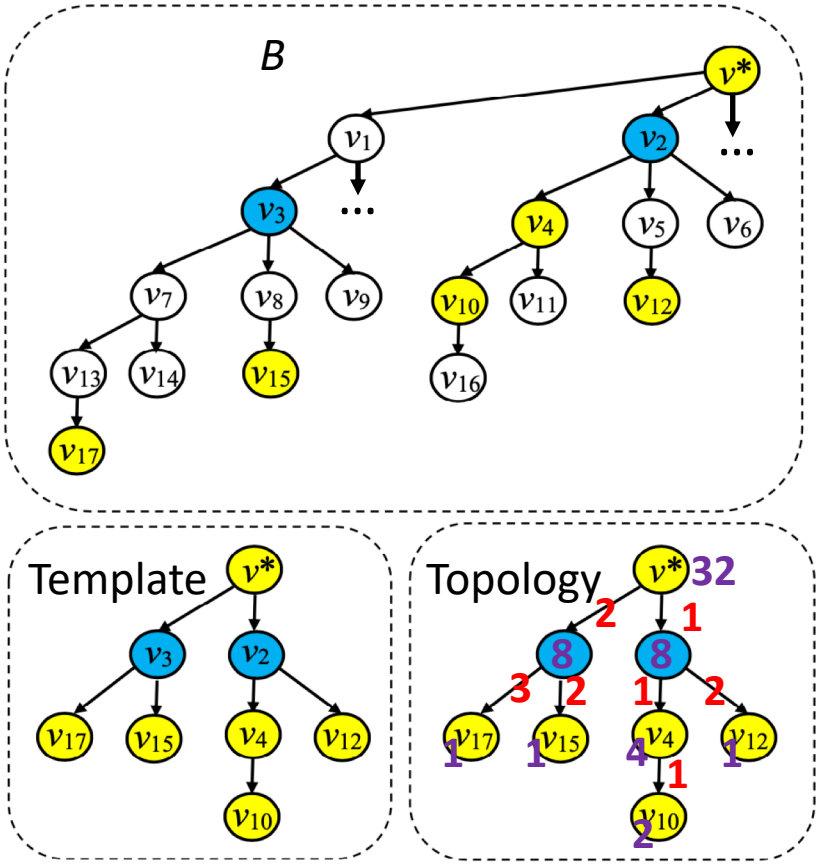
Phase I: Guessing

Template of V_F in B. Remove the vertices outside V_F without descendants from V_F; ``dissolve'' the remaining (degree-2) vertices outside LCA(V_F).

Topology of V_F in B. Forget the identity of the vertices in LCA(V_F)\V_F.

Lengths+Sizes. For every edge: assign the **length** of its path in B. For every vertex: assign the **size** of its subtree in B.

Note. All lengths belong to {1,2,…,log$n$}, and all sizes belong to $\{2^0,2^1,\ldots,2^{\log n}\}$.



FPT Algorithm for TFP

Phase I: Guessing

Template of V_F in B. Remove the vertices outside V_F without descendants from V_F; ``dissolve'' the remaining (degree-2) vertices outside LCA(V_F).

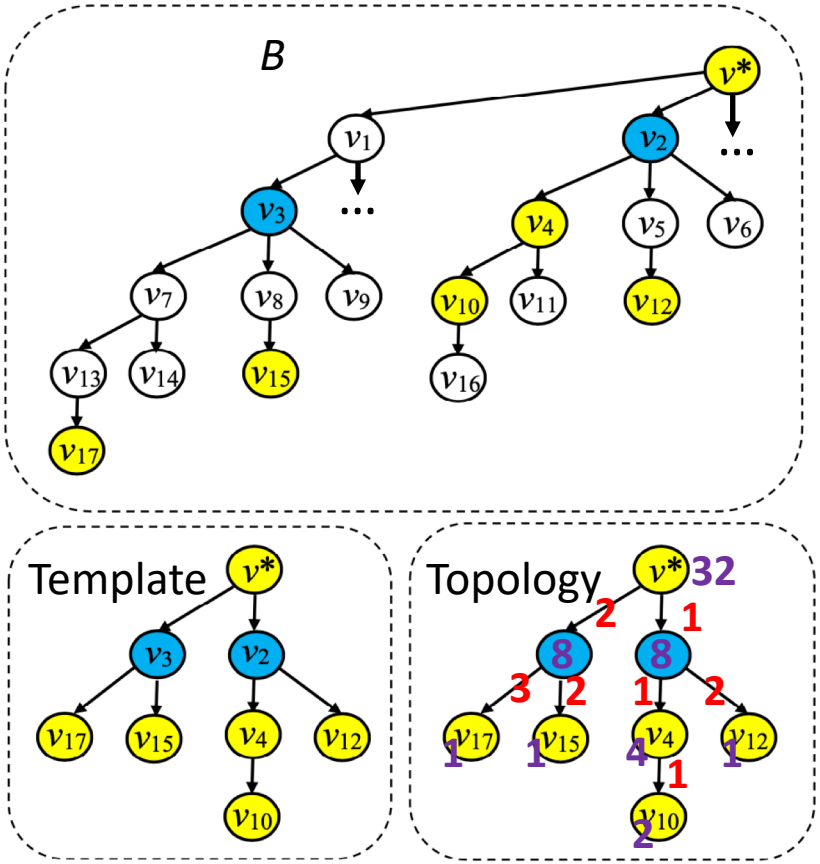
Topology of V_F in B. Forget the identity of the vertices in LCA(V_F)\V_F.

Lengths+Sizes. For every edge: assign the **length** of its path in B. For every vertex: assign the **size** of its subtree in B.

Note. All lengths belong to {1,2,…,log$n$}, and all sizes belong to $\{2^0,2^1,\ldots,2^{\log n}\}$.

Algorithm. Guess the labeled topology.

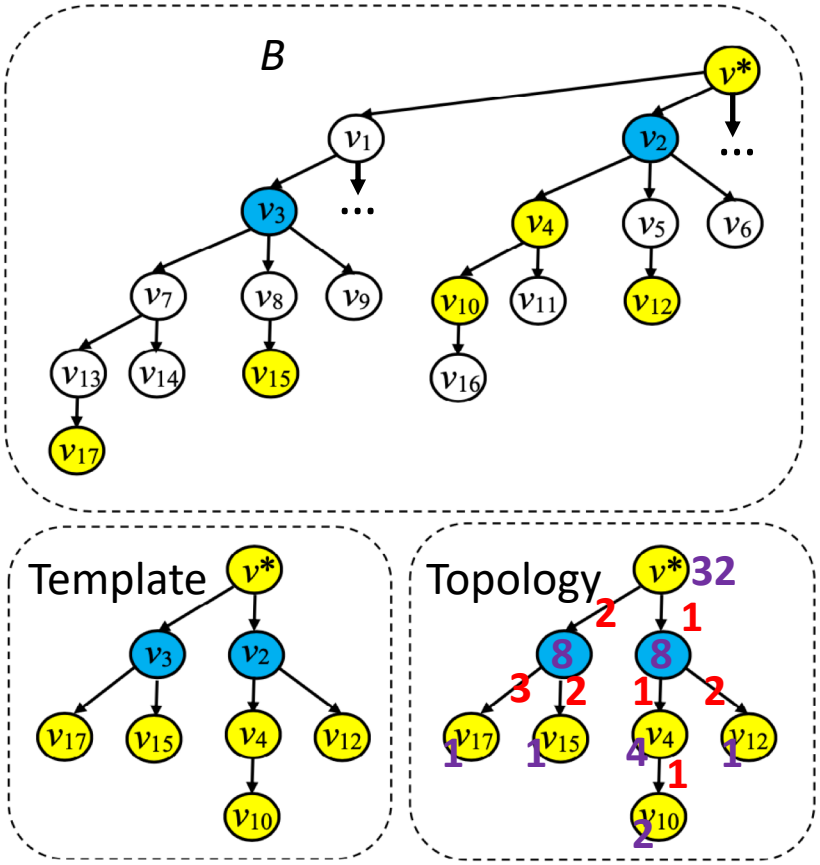
Only $k^{O(k)}$ options! (Suppose log$n$<k.)



FPT Algorithm for TFP

Phase II: Verify Realizability

When is a guess realizable? There exists an unlabeled BA to which we can map the vertices of the topology in compliance with the lengths + sizes.

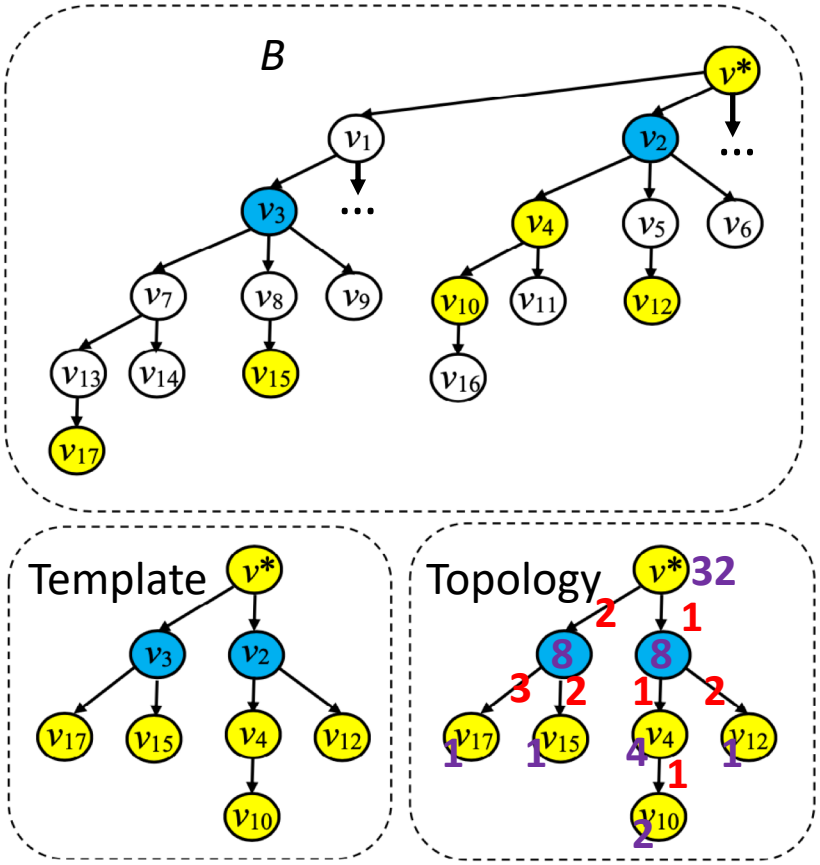


FPT Algorithm for TFP

Phase II: Verify Realizability

When is a guess realizable? There exists an unlabeled BA to which we can map the vertices of the topology in compliance with the lengths + sizes.

Lemma. Realizability can be verified in polynomial time. (Dynamic prog.).



FPT Algorithm for TFP

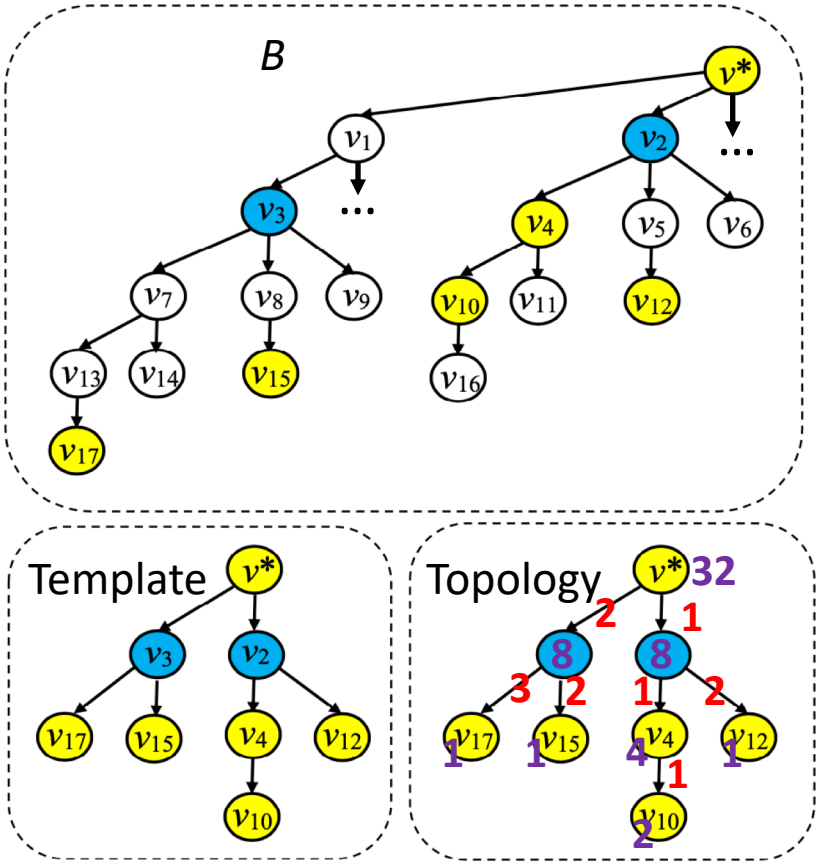
Phase II: Verify Realizability

When is a guess realizable? There exists an unlabeled BA to which we can map the vertices of the topology in compliance with the lengths + sizes.

Lemma. Realizability can be verified in polynomial time. (Dynamic prog.).

Note. A guess can be realized by many unlabeled BAs.

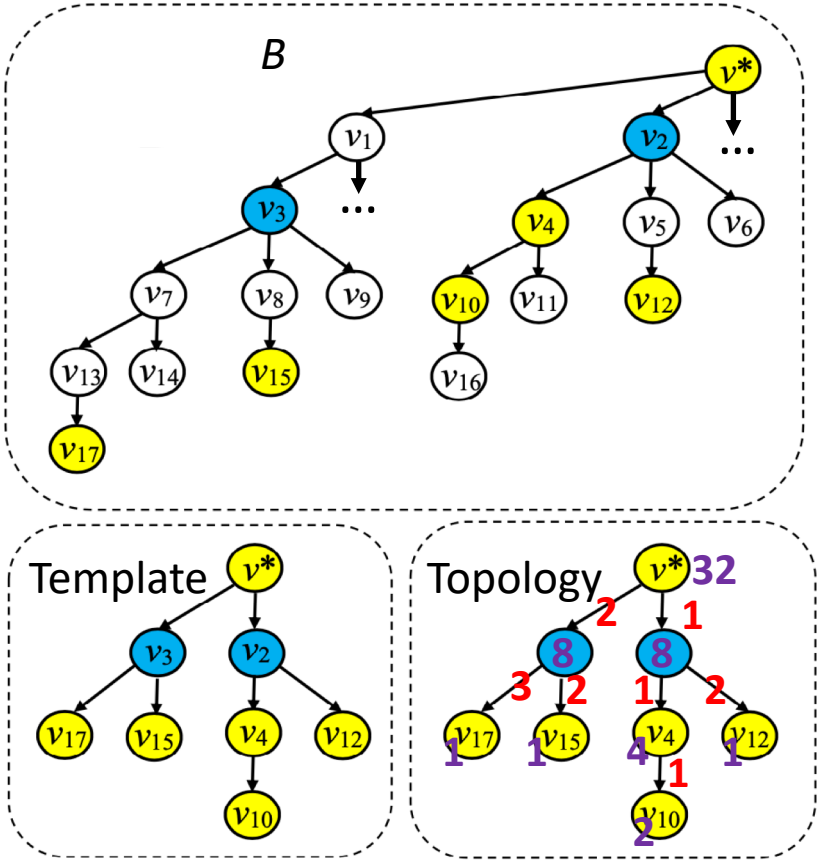
Surprisingly, we can ``work'' with any one that we want! (Later.)



FPT Algorithm for TFP

Phase III: Greedy Resolution of Paths

- An edge is **resolved** if it was replaced by a degree-2 path of its length.
- A V_F-vertex is **resolved** if all edges between it and its V_F-parent are resolved.

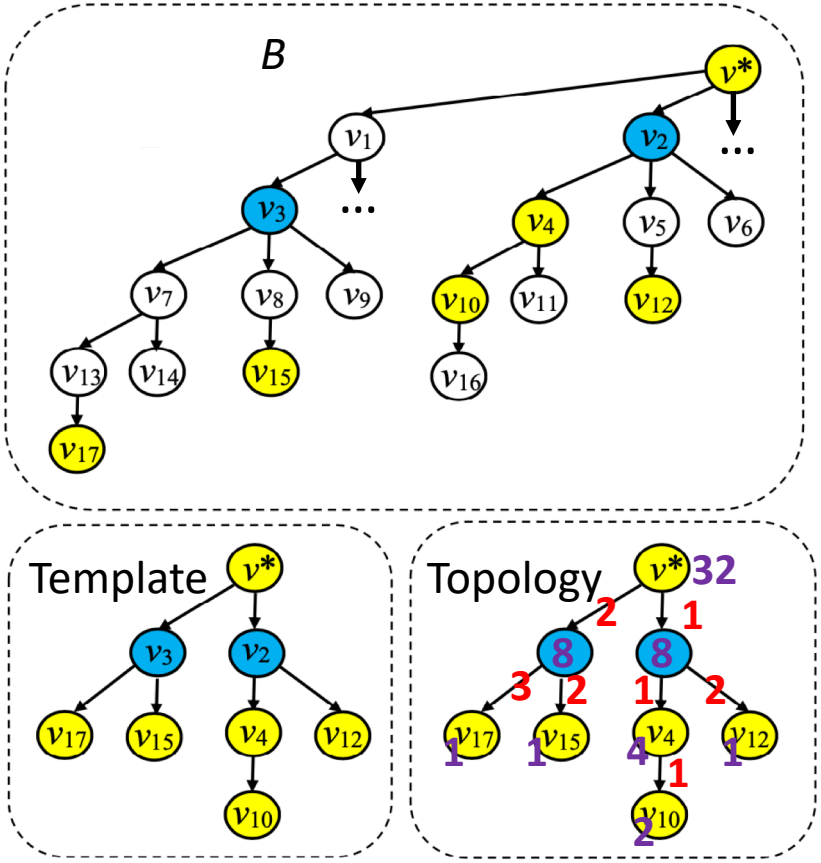


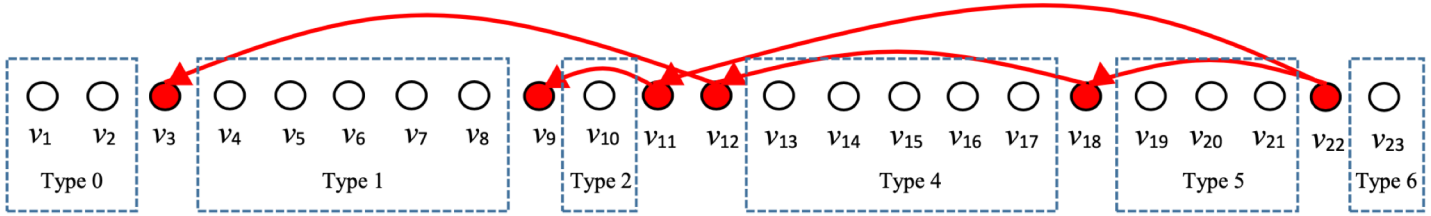
FPT Algorithm for TFP

Phase III: Greedy Resolution of Paths

- An edge is **resolved** if it was replaced by a degree-2 path of its length.
- A V_F-vertex is **resolved** if all edges between it and its V_F-parent are resolved.

Alg. While there is an unresolved V_F-vertex:



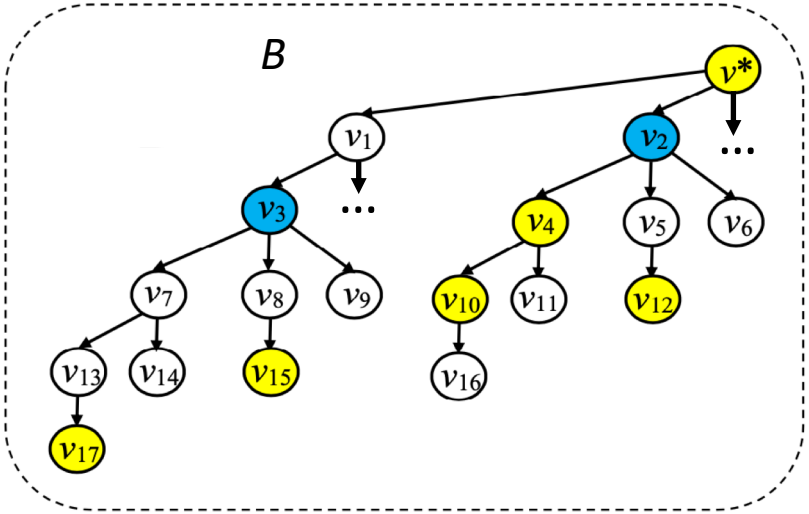


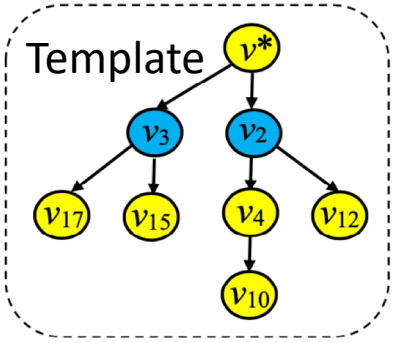
Phase III: Greedy Resolution of Paths

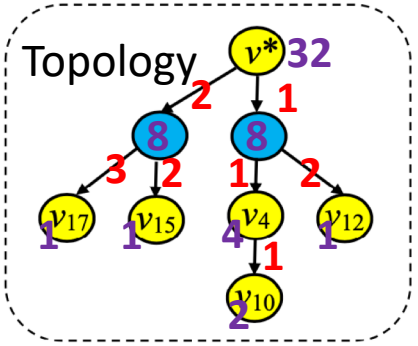
- An edge is **resolved** if it was replaced by a degree-2 path of its length.
- A V_F-vertex is **resolved** if all edges between it and its V_F-parent are resolved.

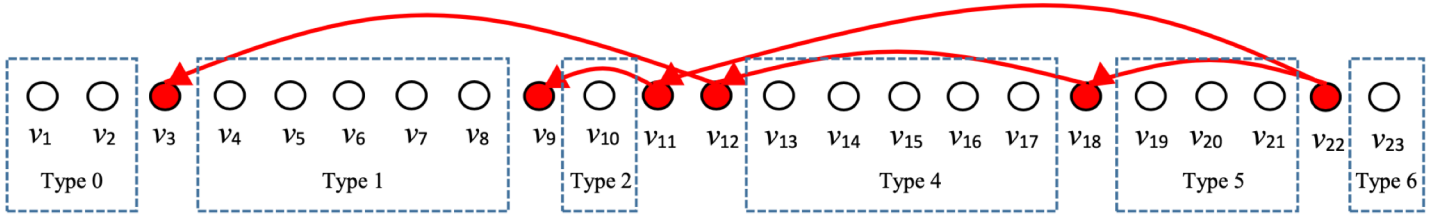
Alg. While there is an unresolved V_F-vertex:

- ***v*:** Such a vertex highest in the topo. order of D with F reversed.







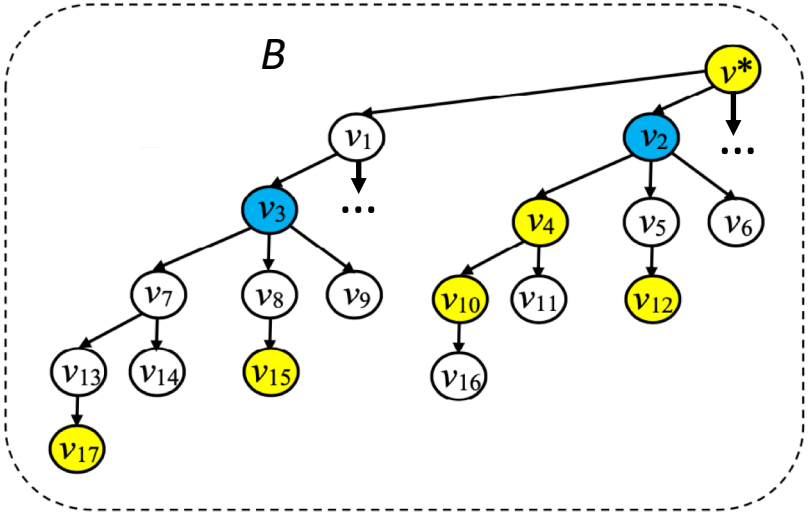


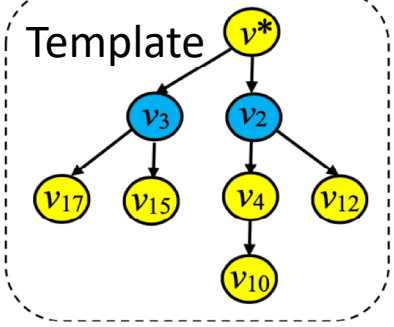
Phase III: Greedy Resolution of Paths

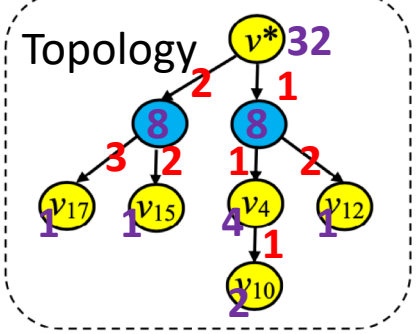
- An edge is **resolved** if it was replaced by a degree-2 path of its length.
- A V_F-vertex is **resolved** if all edges between it and its V_F-parent are resolved.

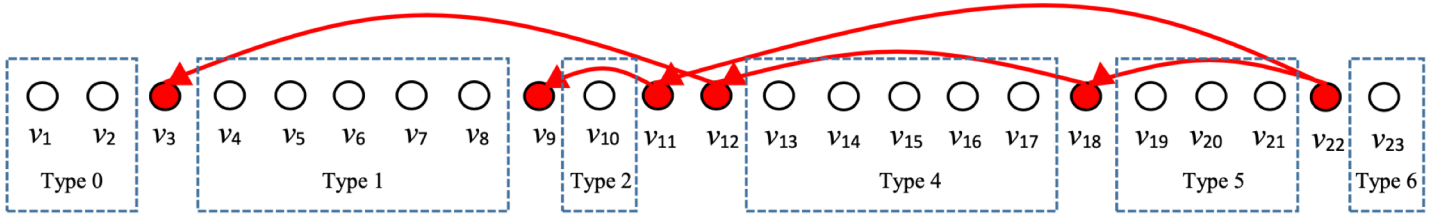
Alg. While there is an unresolved V_F-vertex:

- ***v***: Such a vertex highest in the topo. order of *D* with *F* reversed.
- ***u***: The labeled ancestor of *v* that is closest to *v*.







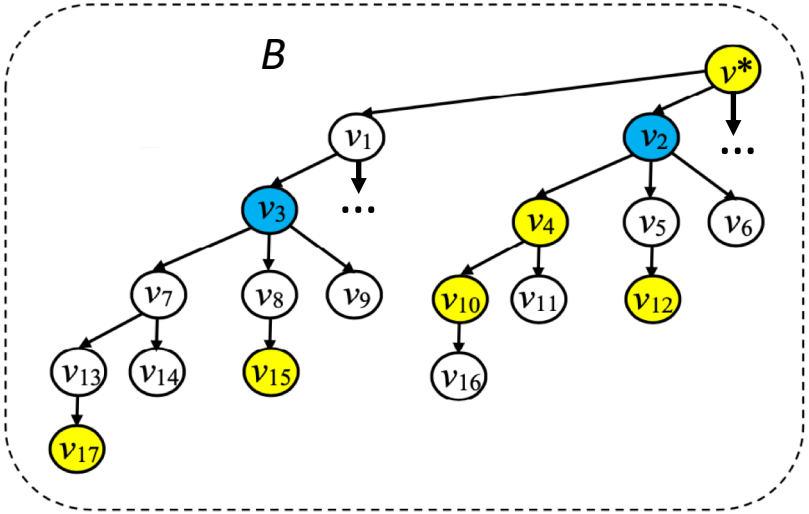


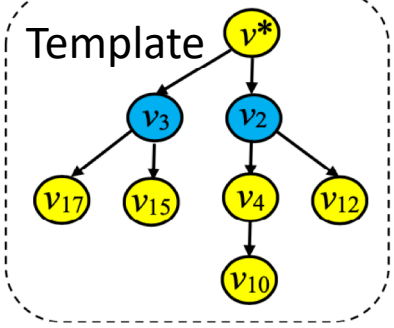
Phase III: Greedy Resolution of Paths

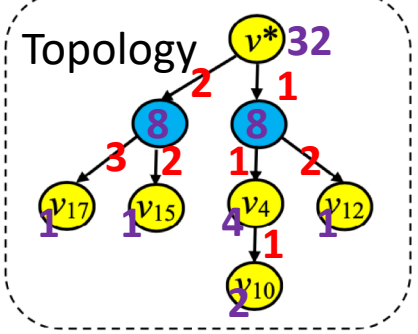
- An edge is **resolved** if it was replaced by a degree-2 path of its length.
- A V_F-vertex is **resolved** if all edges between it and its V_F-parent are resolved.

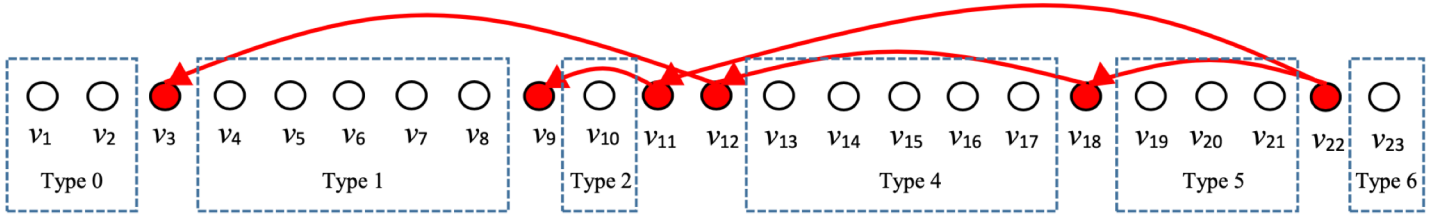
Alg. While there is an unresolved V_F-vertex:

- ***v***: Such a vertex highest in the topo. order of *D* with *F* reversed.
- ***u***: The labeled ancestor of *v* that is closest to *v*.
- ***x***: The sum of the lengths-1 of the edges between *u* and *v*.









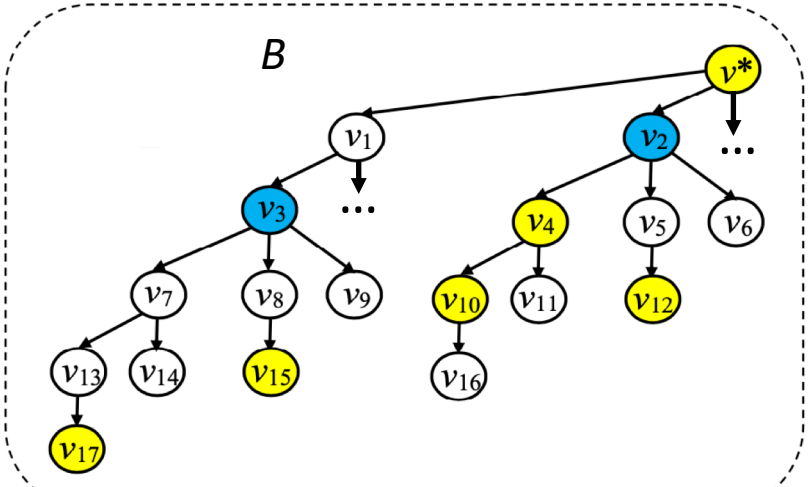
Phase III: Greedy Resolution of Paths

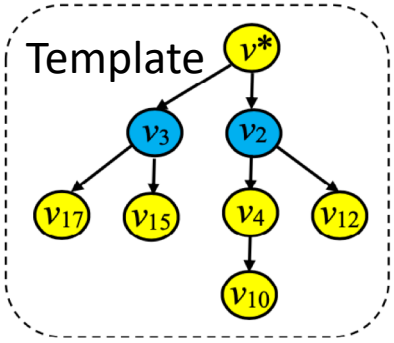
- An edge is **resolved** if it was replaced by a degree-2 path of its length.
- A V_F-vertex is **resolved** if all edges between it and its V_F-parent are resolved.

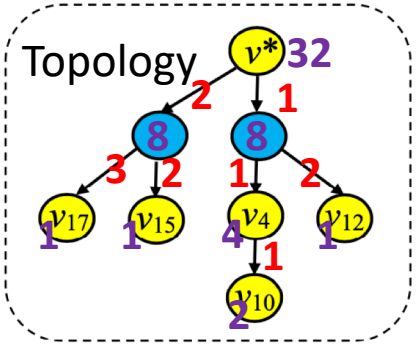
Alg. While there is an unresolved V_F-vertex:

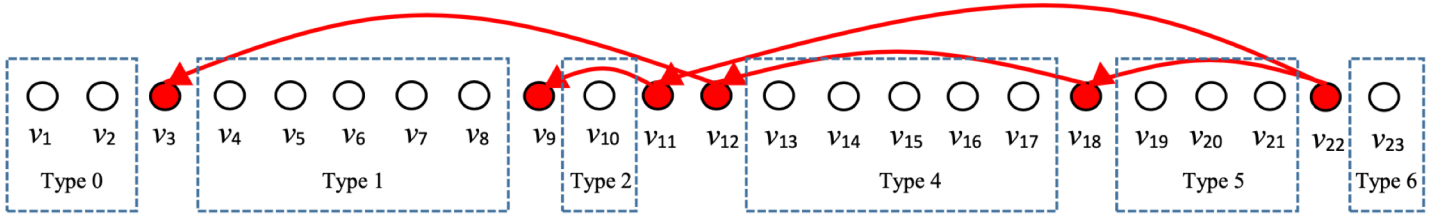
- ***v*:** Such a vertex highest in the topo. order of *D* with *F* reversed.
- ***u*:** The labeled ancestor of *v* that is closest to *v*.
- ***x*:** The sum of the lengths-1 of the edges between *u* and *v*.

➢ **Resolution:** Pick the *x* ``free'' vertices between *u* and *v* in closest to *u* in the topo. order of *D* with *F* reversed.
Replace the edges between *u* and *v* by the unique path using these *x* vertices.









Phase III: Greedy Resolution of Paths

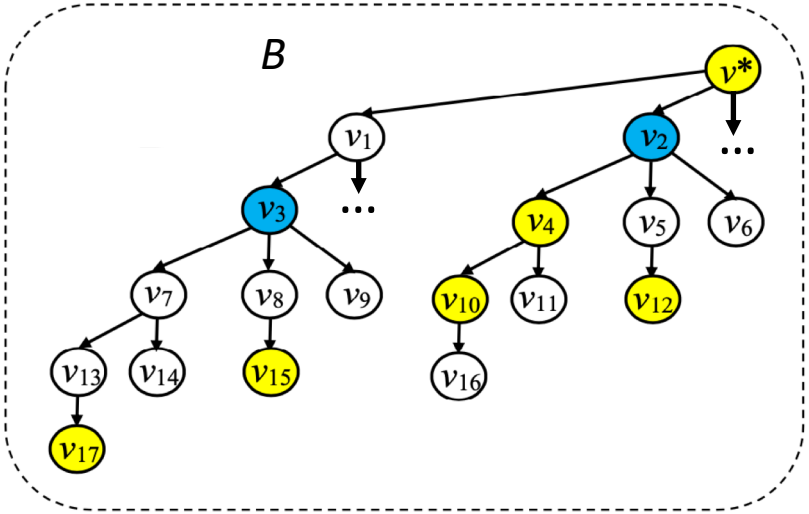
- An edge is **resolved** if it was replaced by a degree-2 path of its length.
- A V_F-vertex is **resolved** if all edges between it and its V_F-parent are resolved.

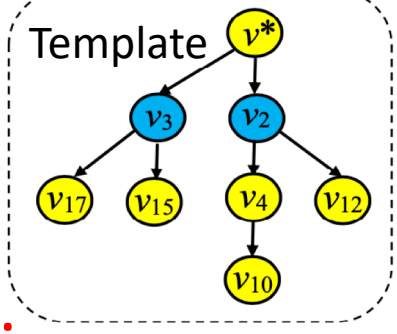
Alg. While there is an unresolved V_F-vertex:

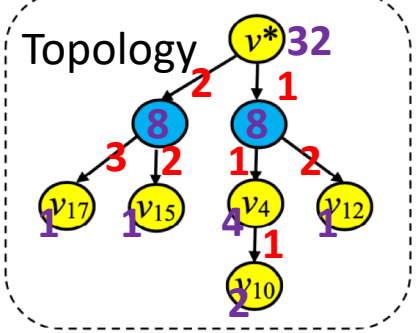
- ***v*:** Such a vertex highest in the topo. order of *D* with *F* reversed.
- ***u*:** The labeled ancestor of *v* that is closest to *v*.
- ***x*:** The sum of the lengths-1 of the edges between *u* and *v*.

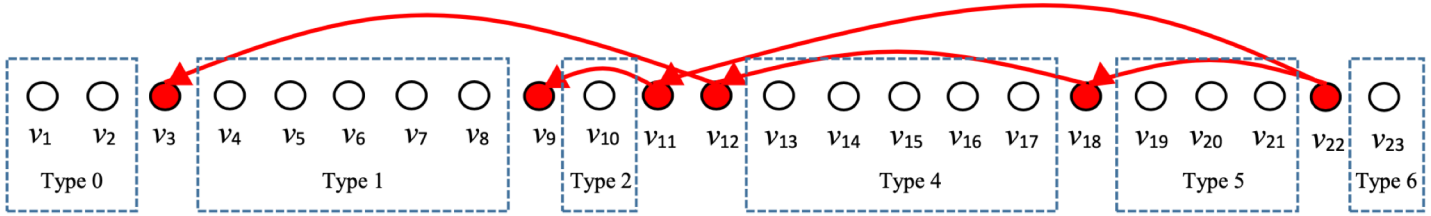
➢ **Resolution:** **Pick** the *x* ``free'' vertices between *u* and *v* in closest to *u* in the topo. order of *D* with *F* reversed.
Replace the edges between *u* and *v* by the unique path using these *x* vertices.

If we do not have enough vertices, answer No.









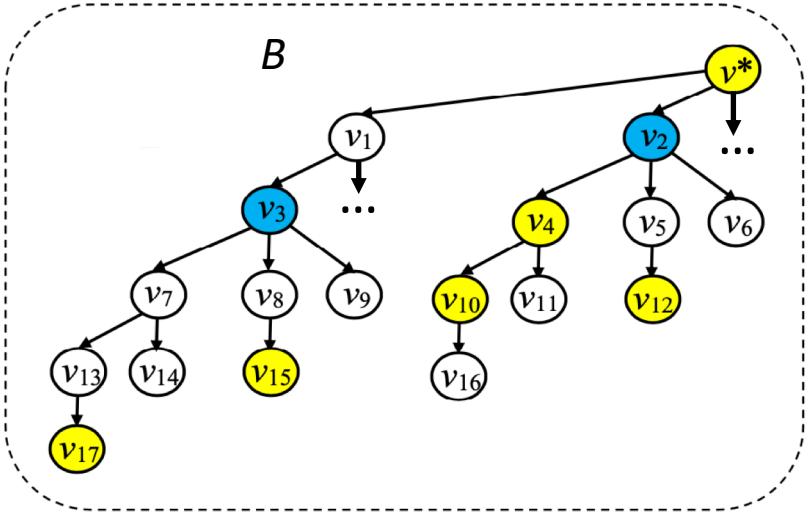
Phase III: Greedy Resolution of Paths

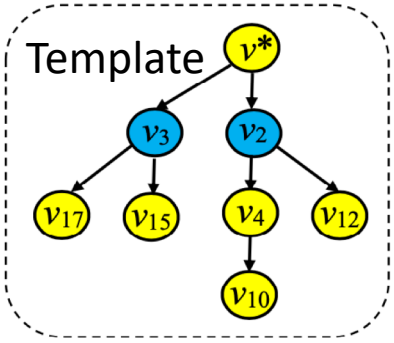
- An edge is **resolved** if it was replaced by a degree-2 path of its length.
- A V_F-vertex is **resolved** if all edges between it and its V_F-parent are resolved.

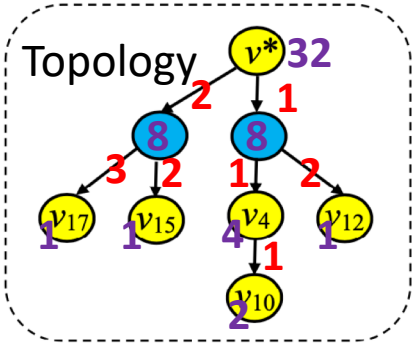
Alg. While there is an unresolved V_F-vertex:

- ***v*:** Such a vertex highest in the topo. order of D with F reversed.
- ***u*:** The labeled ancestor of v that is closest to v.
- ***x*:** The sum of the lengths-1 of the edges between u and v.

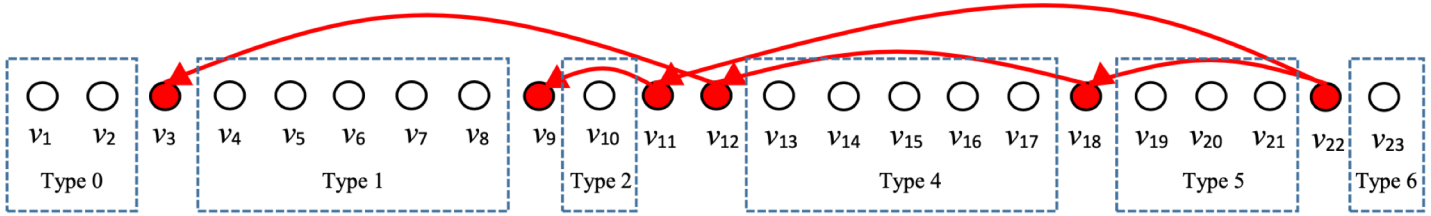
➢ **Resolution:** Pick the x ``free'' vertices between u and v in closest to u in the topo. order of D with F reversed.
Replace the edges between u and v by the unique path using these x vertices.





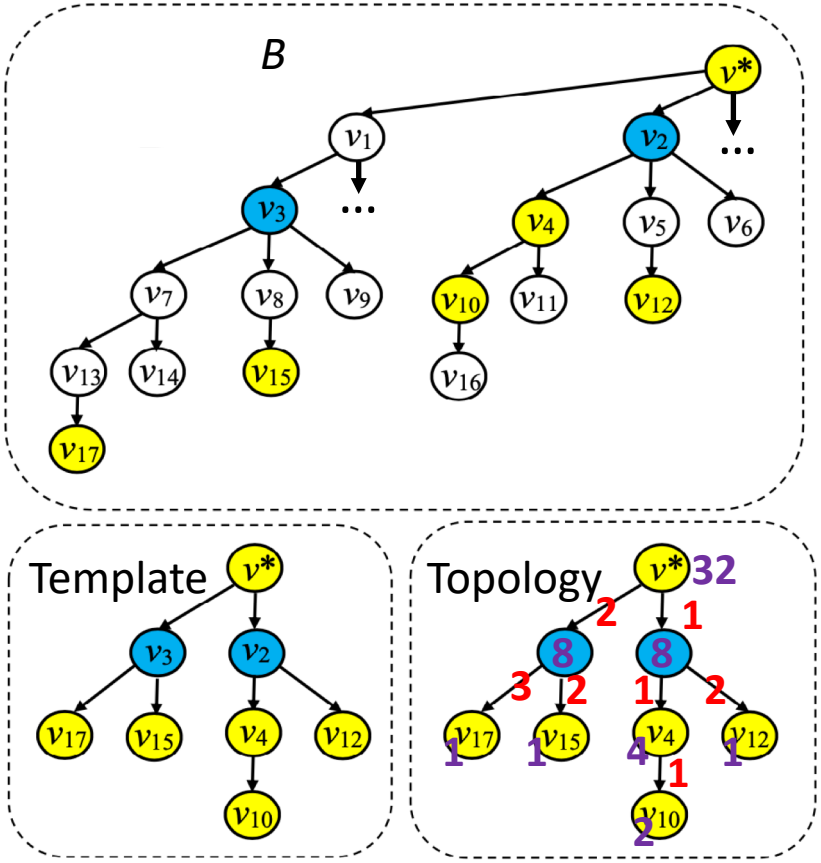


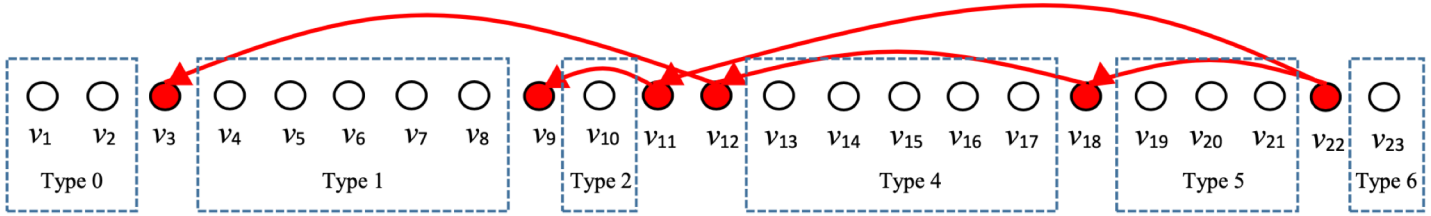
Intuition?



Phase IV: Greedy Resolution of Subtrees

- The **private vertices** of a V_F-vertex are the vertices that belong to its subtree and to none of the subtrees of its V_F-descendants.
- A V_F-vertex is **resolved** if it was assigned its private vertices.

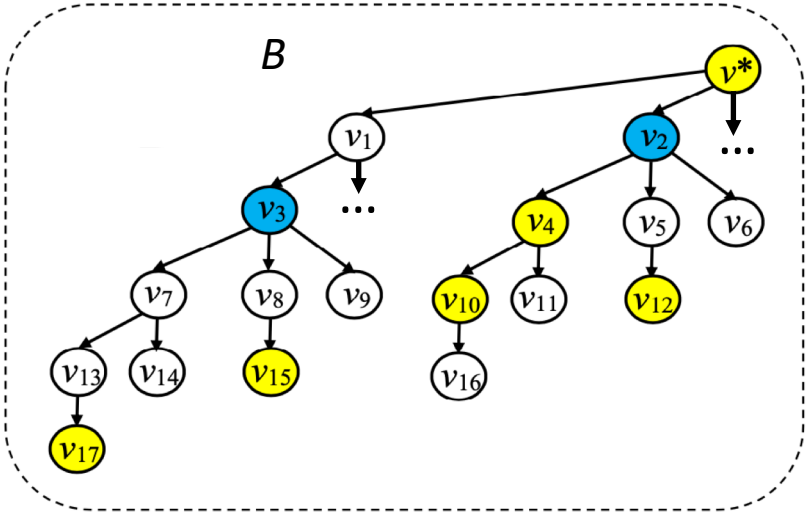


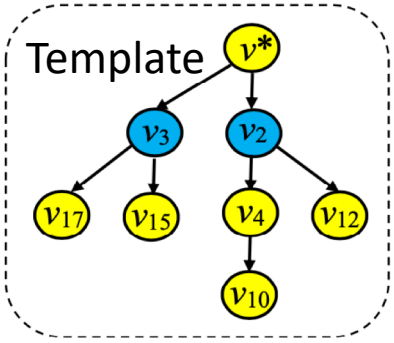


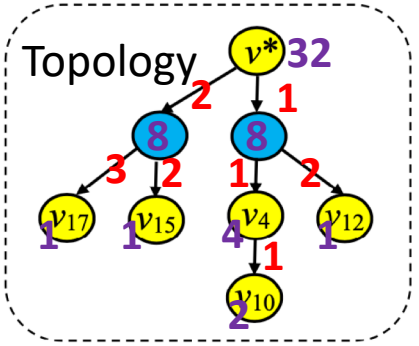
Phase IV: Greedy Resolution of Subtrees

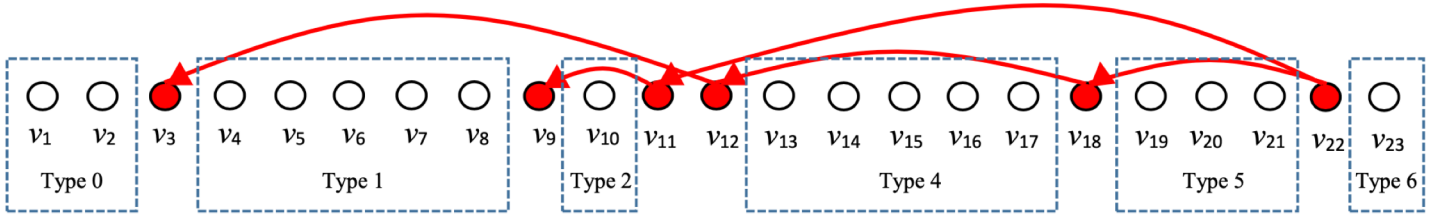
- The **private vertices** of a V_F-vertex are the vertices that belong to its subtree and to none of the subtrees of its V_F-descendants.
- A V_F-vertex is **resolved** if it was assigned its private vertices.

Alg. While there is an unresolved V_F-vertex:







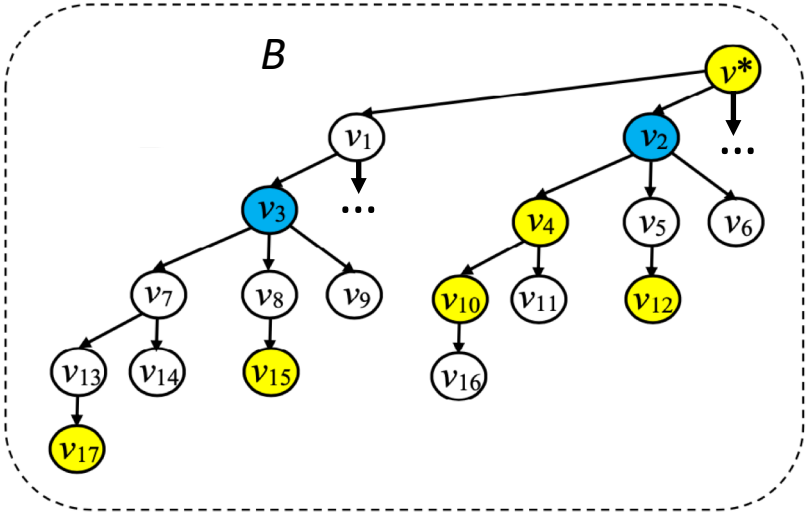


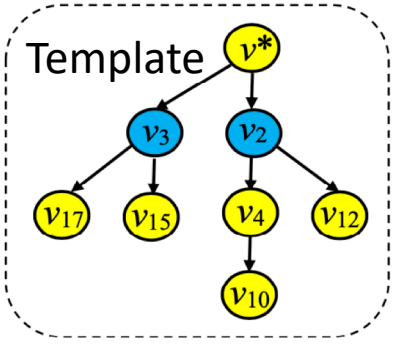
Phase IV: Greedy Resolution of Subtrees

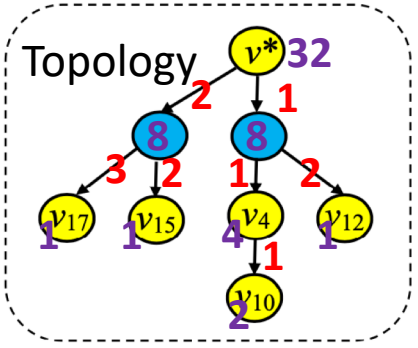
- The **private vertices** of a V_F-vertex are the vertices that belong to its subtree and to none of the subtrees of its V_F-descendants.
- A V_F-vertex is **resolved** if it was assigned its private vertices.

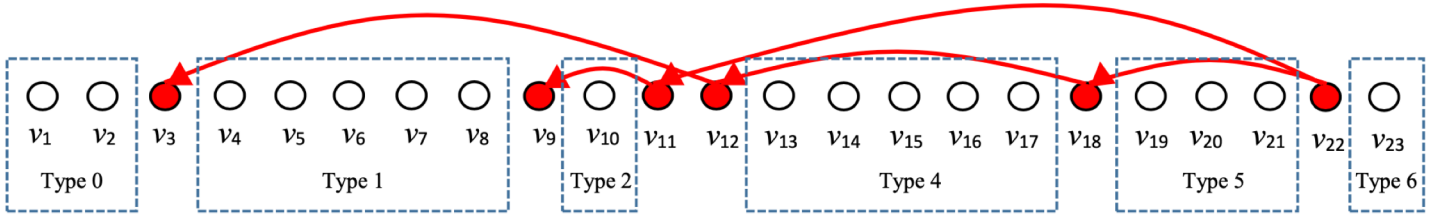
Alg. While there is an unresolved V_F-vertex:

- ***v*:** Such a vertex highest in the topo. order of D with F reversed.
- ***x*:** # of its ``empty'' private vertices.









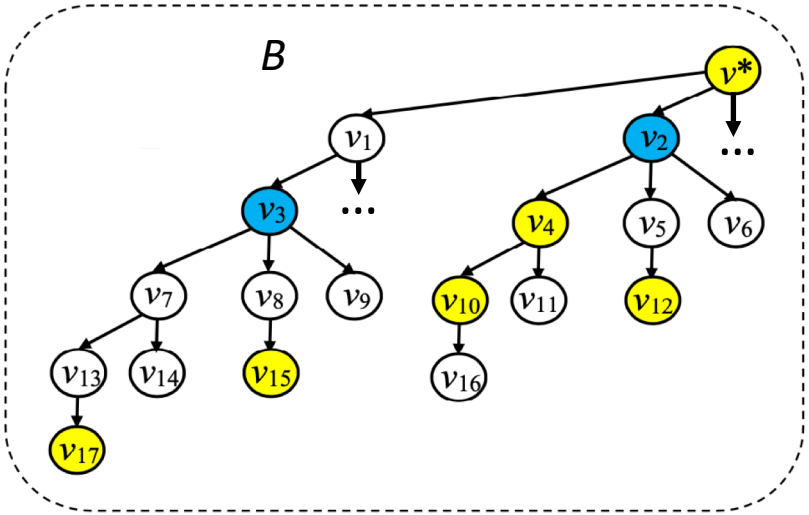
Phase IV: Greedy Resolution of Subtrees

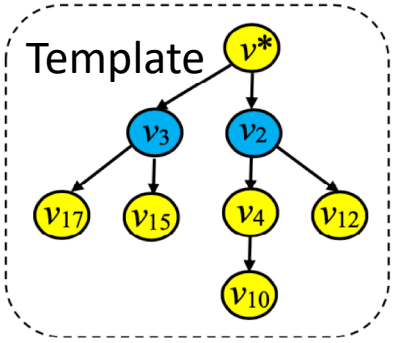
- The **private vertices** of a V_F-vertex are the vertices that belong to its subtree and to none of the subtrees of its V_F-descendants.
- A V_F-vertex is **resolved** if it was assigned its private vertices.

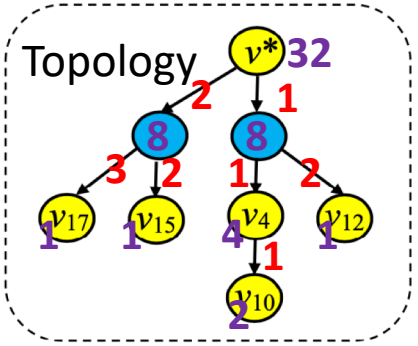
Alg. While there is an unresolved V_F-vertex:

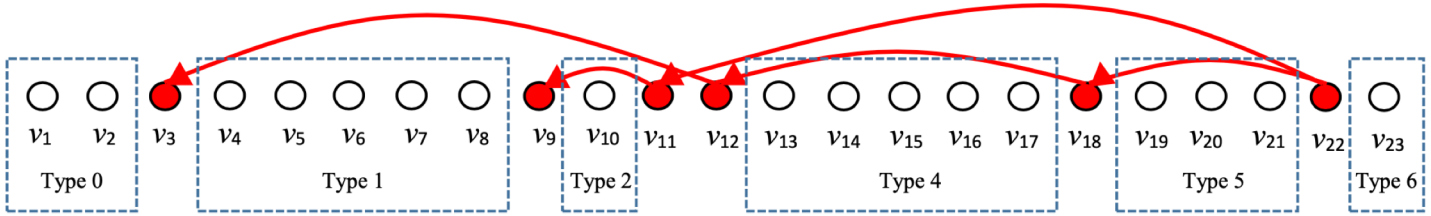
- ***v*:** Such a vertex highest in the topo. order of D with F reversed.
- ***x*:** # of its ``empty'' private vertices.

➢ **Resolution:** Pick the x ``free'' vertices lower than (but closest to) v in the topo. order of D with F reversed. Assign these vertices to v.









Phase IV: Greedy Resolution of Subtrees

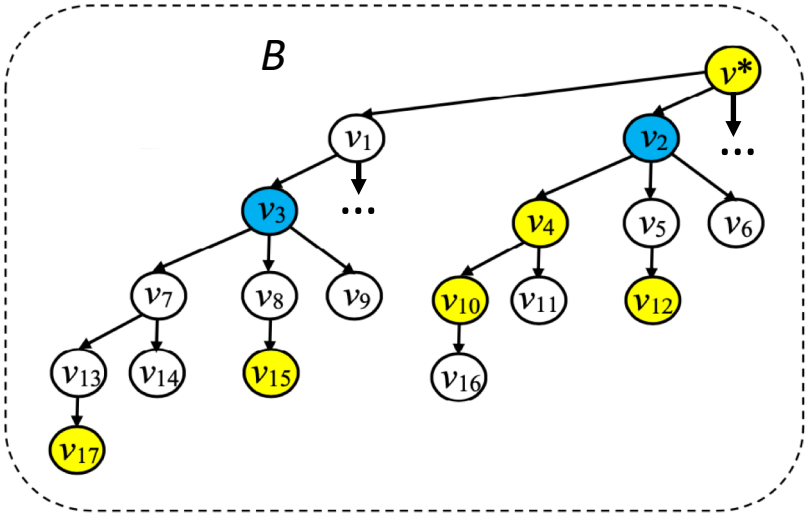
- The **private vertices** of a V_F-vertex are the vertices that belong to its subtree and to none of the subtrees of its V_F-descendants.
- A V_F-vertex is **resolved** if it was assigned its private vertices.

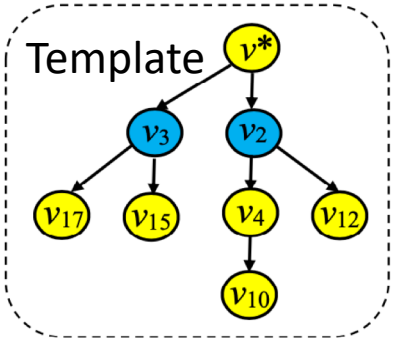
Alg. While there is an unresolved V_F-vertex:

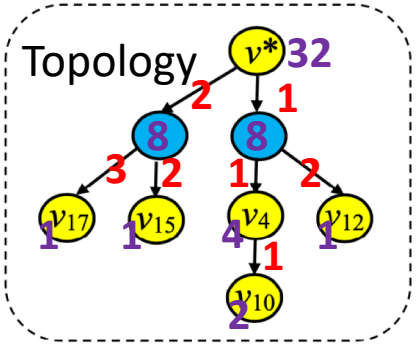
- ***v*:** Such a vertex highest in the topo. order of D with F reversed.
- ***x*:** # of its ``empty'' private vertices.

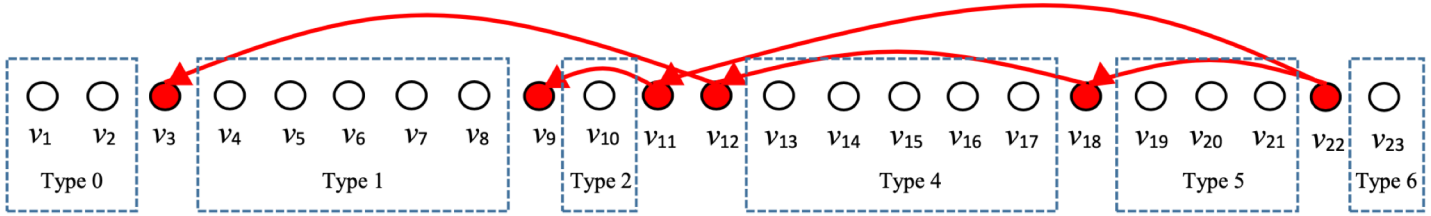
➢ **Resolution:** **Pick** the x ``free'' vertices lower than (but closest to) v in the topo. order of D with F reversed. Assign these vertices to v.

If we do not have enough vertices, answer No.









Phase IV: Greedy Resolution of Subtrees

- The **private vertices** of a V_F-vertex are the vertices that belong to its subtree and to none of the subtrees of its V_F-descendants.
- A V_F-vertex is **resolved** if it was assigned its private vertices.

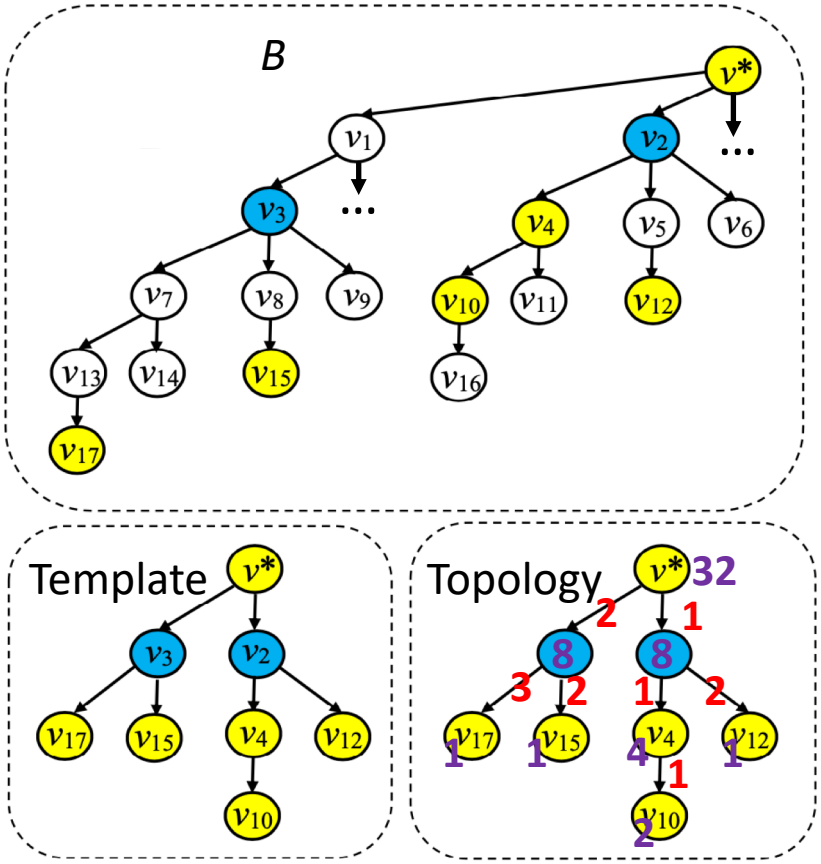
Alg. While there is an unresolved V_F-vertex:

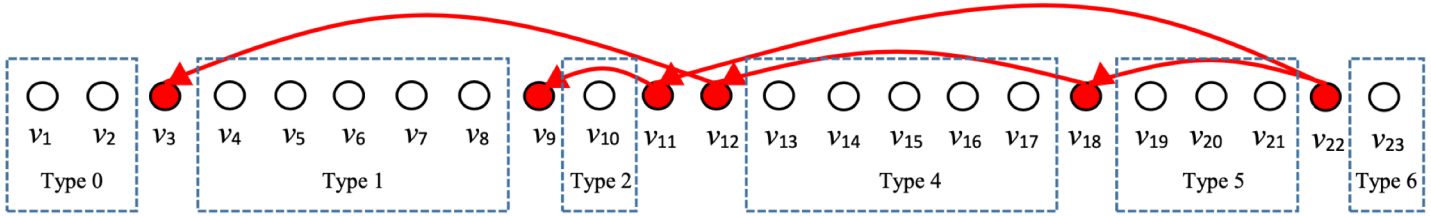
- ***v*:** Such a vertex highest in the topo. order of D with F reversed.
- ***x*:** # of its ``empty'' private vertices.

➢ **Resolution:** **Pick** the x ``free'' vertices lower than (but closest to) v in the topo. order of D with F reversed. Assign these vertices to v.

If we do not have enough vertices, answer No.

Else, when the algorithm terminates, answer Yes.





Phase IV: Greedy Resolution of Subtrees

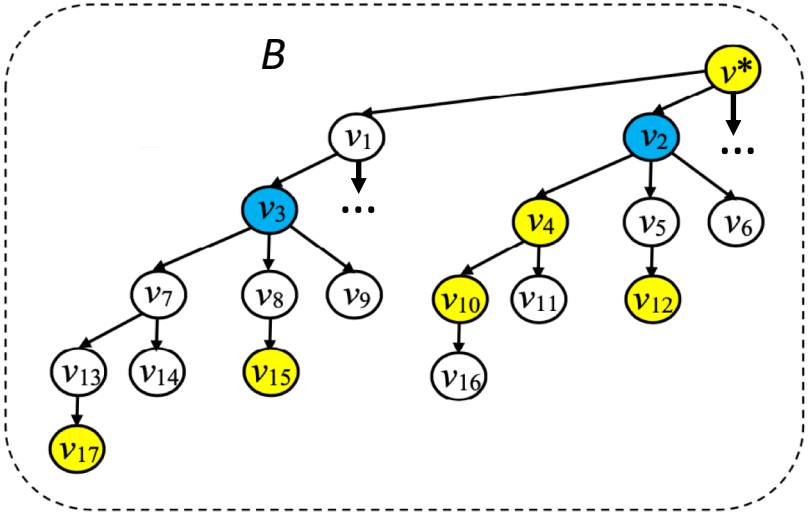
- The **private vertices** of a V_F-vertex are the vertices that belong to its subtree and to none of the subtrees of its V_F-descendants.
- A V_F-vertex is **resolved** if it was assigned its private vertices.

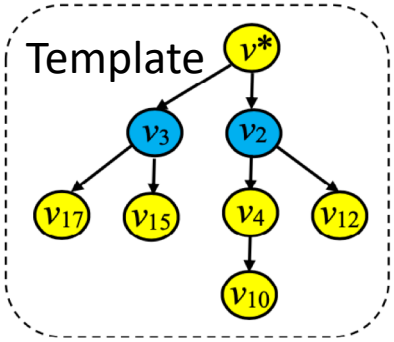
Alg. While there is an unresolved V_F-vertex:

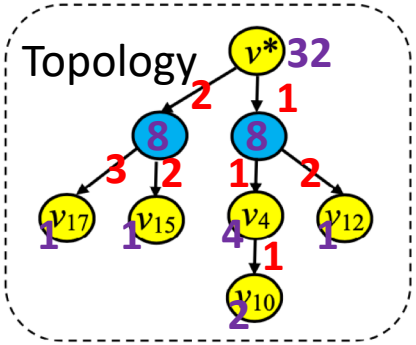
- ***v*:** Such a vertex highest in the topo. order of D with F reversed.
- ***x*:** # of its ``empty'' private vertices.

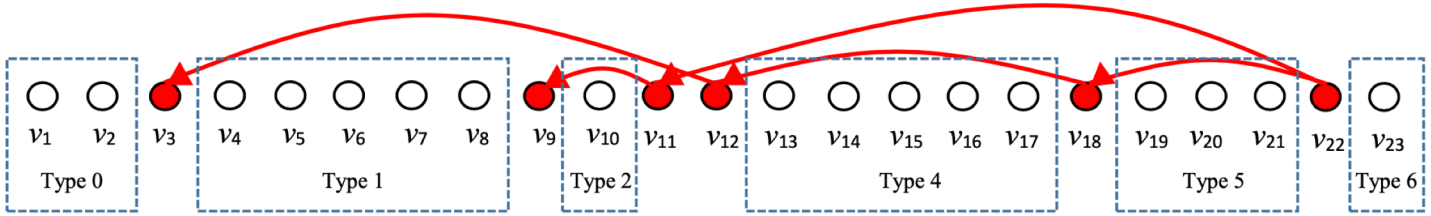
➢ **Resolution:** Pick the x ``free'' vertices lower than (but closest to) v in the topo. order of D with F reversed. Assign these vertices to v.

Intuition?









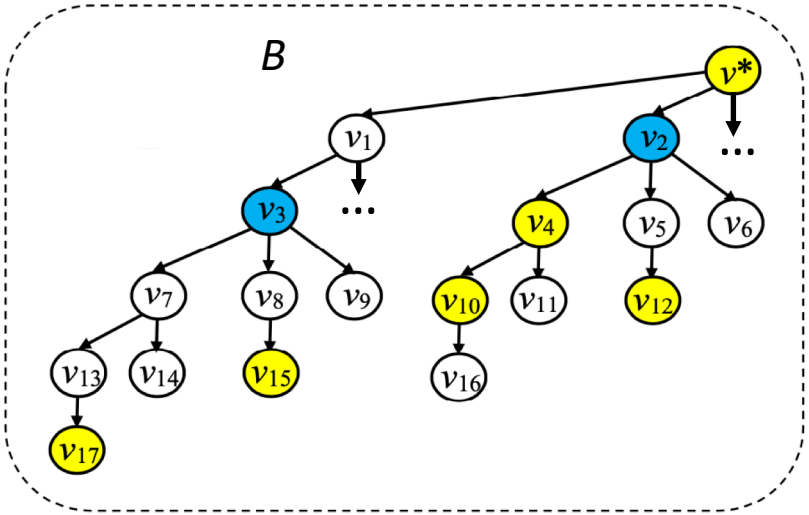
Phase IV: Greedy Resolution of Subtrees

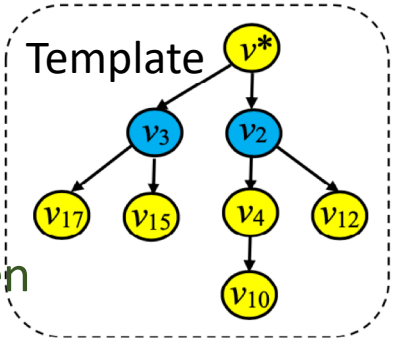
- The **private vertices** of a V_F-vertex are the vertices that belong to its subtree and to none of the subtrees of its V_F-descendants.
- A V_F-vertex is **resolved** if it was assigned its private vertices.

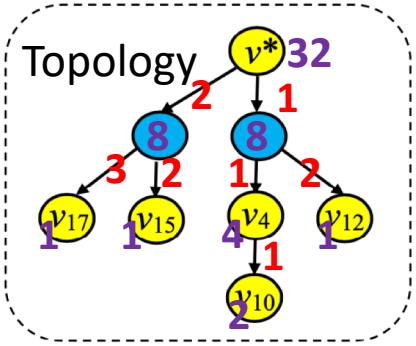
Alg. While there is an unresolved V_F-vertex:

- ***v*:** Such a vertex highest in the topo. order of D with F reversed.
- ***x*:** # of its ``empty'' private vertices.

➢ **Resolution:** Pick the x ``free'' vertices lower than (but closest to) v in the topo. order of D with F reversed. Assign these vertices to v.







Intuition?

In particular, to argue that if we answer Yes, then it is correct: Use Phase II (Verify Realizability).

Outline

Background

FPT algorithm for the TF problem

Kernel for the TF problem

Kernel for TFP

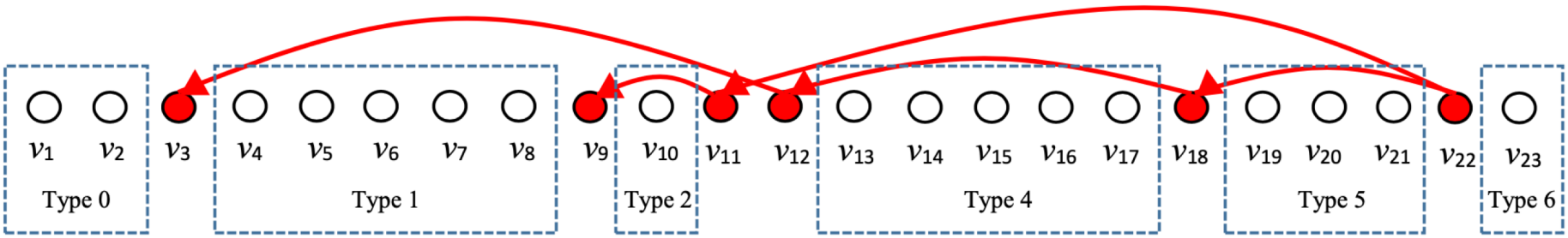
Outline

I. Trivial polynomial compression where witnesses have exponential size.
II. Reinterpretation of the problem as a “packing problem”.
III. Encoding the packing problem using SAT. (Most of the work.)
IV. Sequence of known reductions:
SAT → 3SAT → 3SAT where every literal appears at most twice → TFP.

Kernel for TFP

Trivial Compression

Compute V_F (use an approximation algorithm).



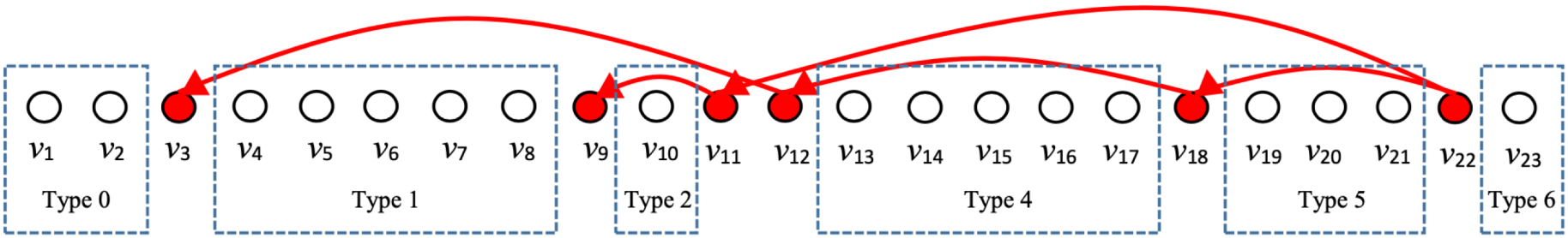
Kernel for TFP

Trivial Compression

Compute V_F (use an approximation algorithm).

Encoding:

- Encode V_F and F.
- Encode the order of V_F. (So, we know the subtournament on V_F.)
- For each type, encode the number of vertices of that type.



Kernel for TFP

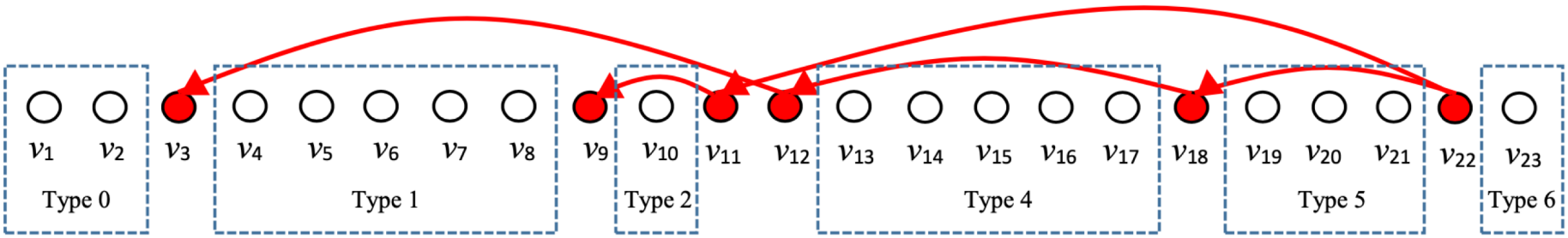
Trivial Compression

Compute V_F (use an approximation algorithm).

Encoding:

- Encode V_F and F.
- Encode the order of V_F. (So, we know the subtournament on V_F.)
- For each type, encode the number of vertices of that type.

Size: Assume log$n$ < $k$log$k$, else the problem is solvable in polynomial-time.
So, we the encoding takes $O(k^2 \log k)$ bits).



Kernel for TFP

Trivial Compression

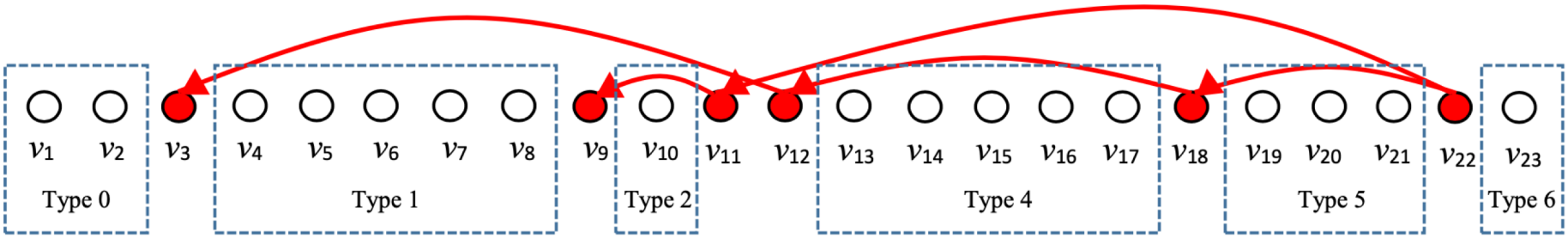
Compute V_F (use an approximation algorithm).

Encoding:

- Encode V_F and F.
- Encode the order of V_F. (So, we know the subtournament on V_F.)
- For each type, encode the number of vertices of that type.

Size: Assume log$n$ < $k$log$k$, else the problem is solvable in polynomial-time.
So, we the encoding takes $O(k^2 \log k)$ bits).

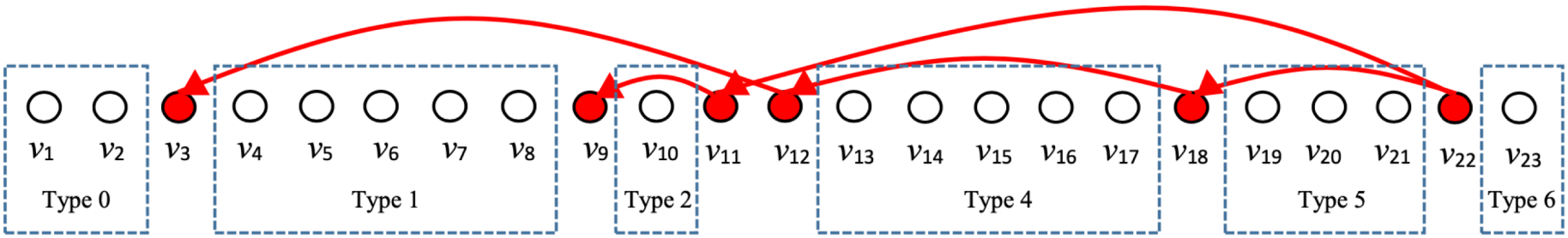
Witness: Single-elimination tournament / SBA: Space $\Omega(n)$ (can be $\Omega(2^{k \log k})$).



Kernel for TFP

Packing-Based Reinterpretation

Guess. Tuple (T, lengths, sizes).

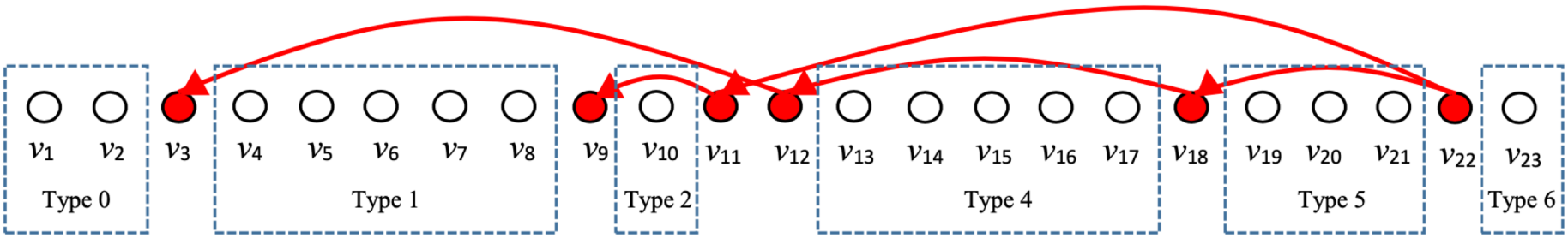


Kernel for TFP

Packing-Based Reinterpretation

Guess. Tuple (T, lengths, sizes).

Valid guess. A guess (T, lengths, size) such that:



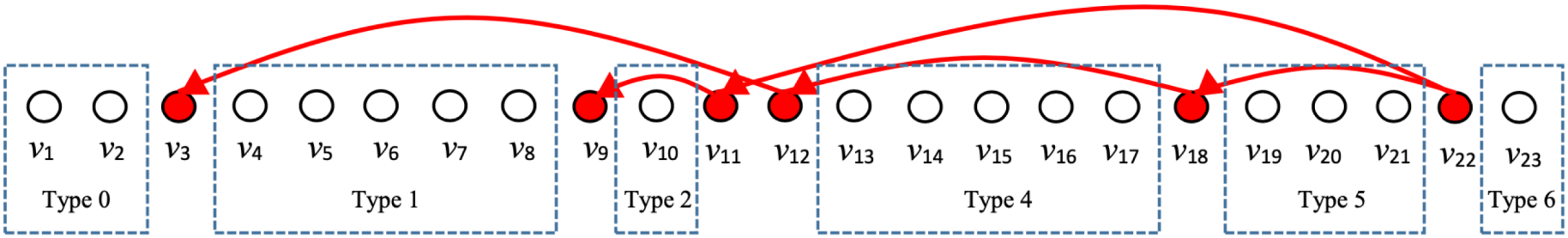
Kernel for TFP

Packing-Based Reinterpretation

Guess. Tuple (T, lengths, sizes).

Valid guess. A guess (T, lengths, size) such that:

- There exists an assignment from E(T) to tuples in $\{0,1,\ldots,\log n\}^{\text{types}}$ such that: (i) Compliance with lengths. (ii) Beaten by tail and beat head.



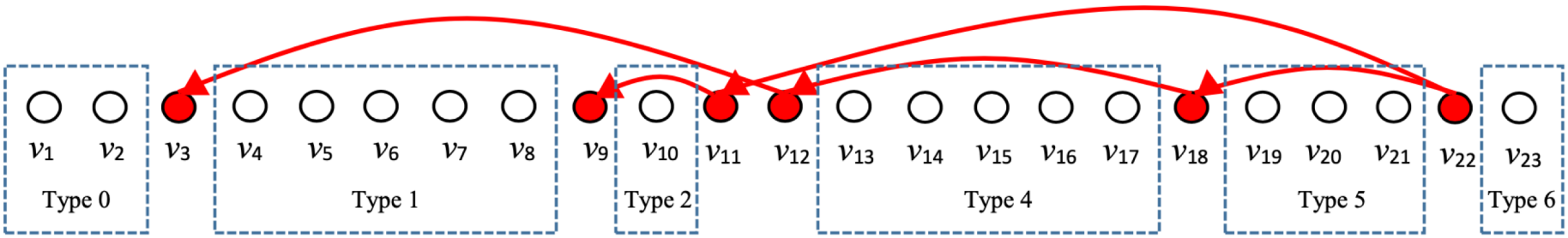
Kernel for TFP

Packing-Based Reinterpretation

Guess. Tuple (T, lengths, sizes).

Valid guess. A guess (T, lengths, size) such that:

- There exists an assignment from E(T) to tuples in $\{0,1,\ldots,\log n\}^{\text{types}}$ such that: (i) Compliance with lengths. (ii) Beaten by tail and beat head.
- There exists an assignment from $V(T)$ to tuple in $\{0,1,\ldots,n\}^{\text{types}}$ such that: (i) Compliance with #private vertices (implied by sizes). (ii) Beaten by the vertex.



Kernel for TFP

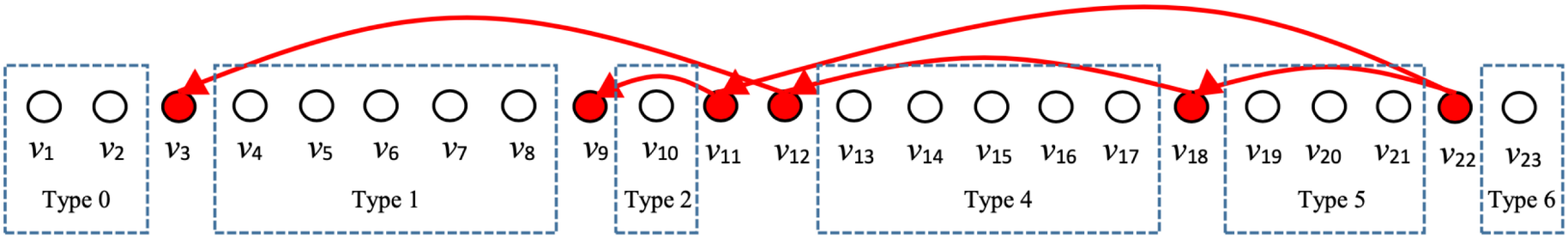
Packing-Based Reinterpretation

Guess. Tuple (T, lengths, sizes).

Valid guess. A guess (T, lengths, size) such that:

- There exists an assignment from E(T) to tuples in $\{0,1,\ldots,\log n\}^{\text{types}}$ such that: (i) Compliance with lengths. (ii) Beaten by tail and beat head.
- There exists an assignment from $V(T)$ to tuple in $\{0,1,\ldots,n\}^{\text{types}}$ such that: (i) Compliance with #private vertices (implied by sizes). (ii) Beaten by the vertex.

And: The number of times each type is ``used'' equals the number of vertices of that type.



Kernel for TFP

Packing-Based Reinterpretation

Guess. Tuple (T, lengths, sizes).

Valid guess. A guess (T, lengths, size) such that:

- There exists an assignment from E(T) to tuples in $\{0,1,\ldots,\log n\}^{\text{types}}$ such that: (i) Compliance with lengths. (ii) Beaten by tail and beat head.
- There exists an assignment from $V(T)$ to tuple in $\{0,1,\ldots,n\}^{\text{types}}$ such that: (i) Compliance with #private vertices (implied by sizes). (ii) Beaten by the vertex.

And: The number of times each type is ``used'' equals the number of vertices of that type.

Realizable guess: A valid guess such that there exists an UBA of size n that ``complies'' with it.

Kernel for TFP

Packing-Based Reinterpretation

Guess. Tuple (T, lengths, sizes).

Valid guess. A guess (T, lengths, size) such that:

- There exists an assignment from E(T) to tuples in $\{0,1,\ldots,\log n\}^{\text{types}}$ such that: (i) Compliance with lengths. (ii) Beaten by tail and beat head.
- There exists an assignment from $V(T)$ to tuple in $\{0,1,\ldots,n\}^{\text{types}}$ such that: (i) Compliance with #private vertices (implied by sizes). (ii) Beaten by the vertex.

And: The number of times each type is ``used'' equals the number of vertices of that type.

Realizable guess: A valid guess such that there exists an UBA of size n that ``complies'' with it.

Still: The ``UBA-part'' of the witness can be of exponential size.

Kernel for TFP

Main Step: SAT-Encoding

We show how to encode the existence of a realizable guess using SAT-constraints.

Kernel for TFP

Main Step: SAT-Encoding

We show how to encode the existence of a realizable guess using SAT-constraints.

Question: ``Direct'' polynomial kernel (where the polynomial is of small degree)?

Summary

Background

FPT algorithm for the TF problem

Kernel for the TF problem

Summary

Background

FPT algorithm for the TF problem

Kernel for the TF problem

Thank you! Questions?