Maintaining a Kingdom in a Tournament

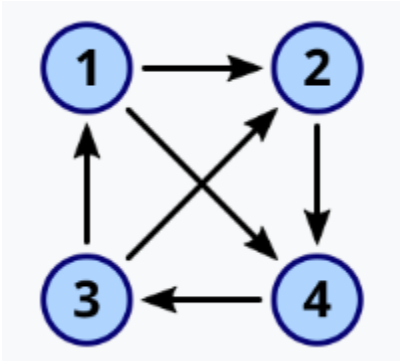
Oren Weimann & Raphael Yuster

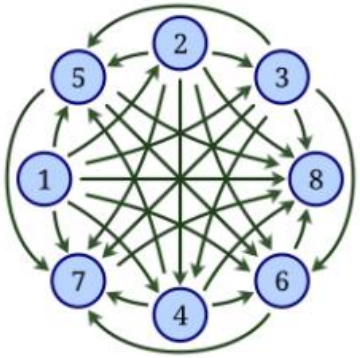
University of Haifa

SOFSEM - 2026

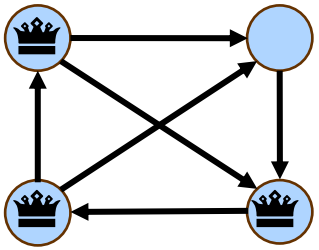
Tournaments & kings

- A **tournament** is an orientation of a complete graph:





- Tournaments are an important and widely researched topics in Combinatorics, TCS, Social Choice Theory and Economics.
- Many problems that are hard for general directed graphs are easier for tournaments, while many others remain hard (e.g., finding a maximum transitive sub-tournament).
- One appealing and important property is that every tournament has a **king**.
  This is a vertex that can reach any other vertex in at most two hops.
  - Exercise: every vertex with maximum out-degree must be a king.
  - So, we can find a king in linear (i.e., $O(n^2)$) time.



Tournaments & kings

- It is no surprise that the following natural problems have been looked at by researchers:
  - **finding a king**.
  - **Construct a kingdom** - a data structure holding a king and that given any vertex v, answers a query which returns a shortest path from the king to v in *constant time*.

One would like to find a king and construct a kingdom in **sublinear time.**

- Fastest (deterministic) algorithm - Shen, Sheng & Wu [SICOMP 2003]. It finds a king in $O(n^{3/2})$ time. They have also proved that no deterministic algorithm can be faster than $O(n^{4/3})$ time. Closing this gap is an intriguing (longstanding) open problem.
- Fastest randomized (Las Vegas) algorithm: Abboud, Grossman, Naor & Solomon [ESA 2024] finds a king in expected $O(n)$ time (so, there is a proven gap between the deterministic and the randomized solutions for this problem).
- Both algorithms also construct a **kingdom** at no additional cost.
  - Note: Since tournaments are complete graphs, the input to the algorithms is the adjacency matrix of the tournament (so there is $O(1)$ access to adjacency queries).

Dynamic algorithm for kings and kingdom

- As usual in graph algorithms, here we consider the **dynamic aspect** of the problem.
- The standard "classification" of dynamic algorithms in our setting:
  - Modify the kingdom after adding vertices.
  - Modify the kingdom after removing vertices.
  - Modify the kingdom after modifying (i.e. flipping) tournament edges.
  - Fault-tolerant (modify after a single / constant number) of changes.
- Of course, we would like to do that quickly: either in constant/logarithmic time per change or at least in small amortized time per change.

Example: single edge flip

Results

We shall state our results and then sketch a proof of one of them.

Theorem 1: A k-tolerant algorithm

Given an n-vertex tournament G, for every constant k there is an algorithm that:

1. Preprocesses G in $\tilde{O}(n^2)$ deterministic time or $\tilde{O}(n)$ expected time, so that:
2. Following any set of at most k vertex additions and removals, we can compute a kingdom for the new tournament deterministically in $O(\log n)$ time.
3. If only removals are present, the new kingdom is computed in $O(1)$ time.

Notice that in the randomized setting, the algorithm is essentially optimal both in preprocessing and in the dynamic part. In the deterministic setting, it is essentially optimal in the dynamic part, but perhaps the preprocessing can be improved?

Theorem 2: Dynamic insertion algorithm

Given an n-vertex tournament G, there is a dynamic algorithm that, after preprocessing G in $O(n^{3/2})$ deterministic time or $\tilde{O}(n)$ expected time, computes a kingdom in $O(n^{1/2})$ amortized time following every new vertex insertion.

Note that insertions can go on indefinitely (this is a dynamic algorithm…) and the price to pay after each insertion is small in average. E.g., if you insert n new vertices, you double the tournament size yet pay a total of $O(n^{3/2})$ while maintaining a kingdom after **each** insertion.

Results

Theorem 3: Dynamic edge changes (flips)

Given an n-vertex tournament G, there is a dynamic algorithm that, after preprocessing G in $\tilde{O}(n)$ expected time, computes a kingdom in $O(\log n)$ expected time following every edge-flip.

So as long as the tournament order is not changed but the tournament itself changes, we have an almost optimal dynamic randomized algorithm. (Of course, as usual, the algorithm is oblivious to the (possibly infinite) sequence of edge flips chosen by the adversary).

Theorem 4: Optimal single fault tolerant algorithm for edge changes

Given an n-vertex tournament G, we can preprocess G in $O(n^{3/2})$ time, so that following any **single edge-flip**, a kingdom of the resulting tournament can be computed in $O(1)$ time.

Clearly this is optimal (and deterministic). Note that we have no idea which of the $\binom{n}{2}$ edges the adversary may decide to flip, yet we can always fix the kingdom in constant time.

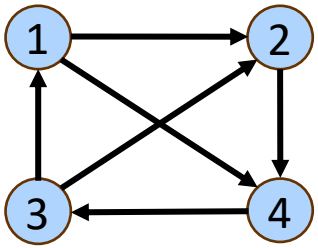
Sketch of Theorem 1 (removals only)

Theorem 1 (restated for removals only): A k-fault tolerant algorithm

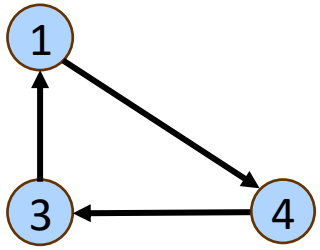
Given an n-vertex tournament G, for every constant k, there is an algorithm that:

1. Preprocesses G in $\tilde{O}(n^2)$ deterministic time or $\tilde{O}(n)$ expected time, so that:
2. Following any set of at most k vertex removals, we can compute a kingdom for the new tournament deterministically in $O(1)$ time.

Vertex removal:



	1	2	3	4
1	0	1	0	1
2	0	0	0	1
3	1	1	0	0
4	0	0	1	0



Kingdom Structure (KS): A triple (x, Y, S) such that

- x is a king of G,
- Y is a list of (not necessarily all) out-neighbors of x so that $x \cup Y$ is a **dominating star**,
- S is a data structure that is queried in ***constant time*** to return a path $q(v)$ from x to v.
  1. If $v = x$, then $q(v) = \{x\}$
  2. If $(x,v) \in E(G)$, then $q(v) = \{x, v\}$
  3. Otherwise, $q(v) = \{x, y, v\}$ for some $y \in Y$.

Preprocessing:

Lemma (Constructing a kingdom structure with small out-star)

Given an n-vertex tournament G, we can construct a KS (x, Y, S) for G with $|Y| = O(\log n)$ in $O(n^2)$ deterministic time or $O(n)$ expected time.

Proof:

- Initially, set:
  - All vertices of G as *unmarked*.
  - An array P indexed by $V(G)$ where $P(v)$ is either $\emptyset$ or $P(v) = u$ for some $(u,v) \in E(G)$.
  - Let Y be an (initially empty) list.
  - Let x be a variable that at the end will contain a king.

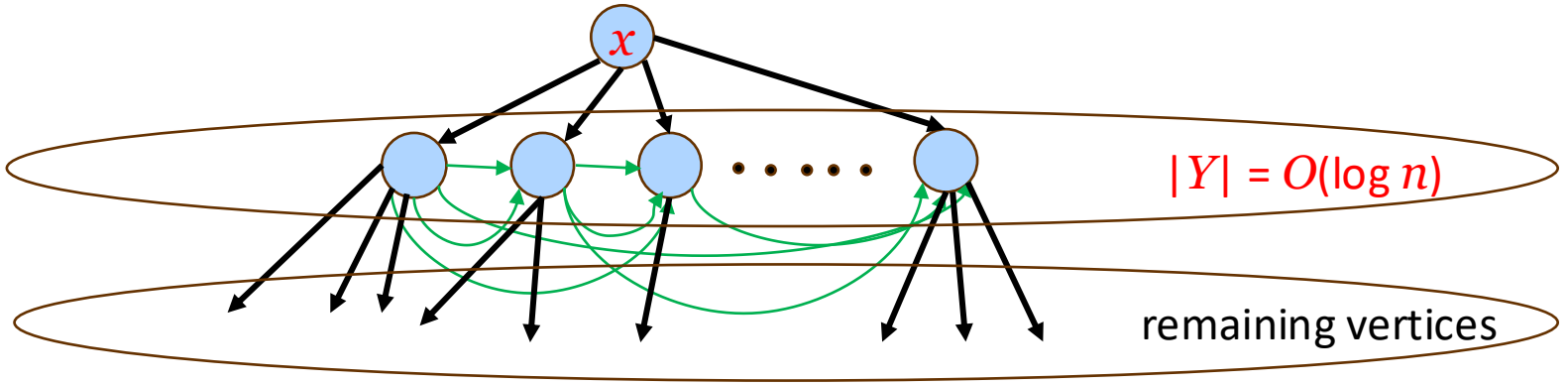
- At any given stage, compute a vertex y of maximum out-degree in the subtournament of G induced by the unmarked vertices in $O(nd)$ time where d = #of unmarked vertices. Alternatively, we can just pick a randomly chosen unmarked vertex y and it is not difficult to show that its *expected* out-degree is at least ½ of the maximum out-degree.

  - Set y marked.
  - For each unmarked out-neighbor v of y, set $P(v) = y$ and set v marked.
  - If all vertices $V(G)$ are now marked, set $x = y$ and halt.
  - Otherwise, add y to Y and repeat.

Preprocessing (cont.):

Proof of Lemma (cont):

- Notice that the number of rounds (i.e., choosing a vertex of maximum out-degree in the unmarked subtournament) is only $O(\log n)$ as in a tournament, a vertex with maximum outdegree dominates at least half of the vertices. So (as claimed):
  - The total runtime (deterministic) is $O(n^2) + O(n \cdot n/2) + O(n \cdot n/4) + \ldots = O(n^2)$
  - The total expected runtime (randomized) is $O(n) + O(n/2) + O(n/4) + \ldots = O(n)$
- Notice that each chosen y must dominate all previously chosen y's so the final chosen x is indeed a king and we obtain the claimed kingdom structure:



■

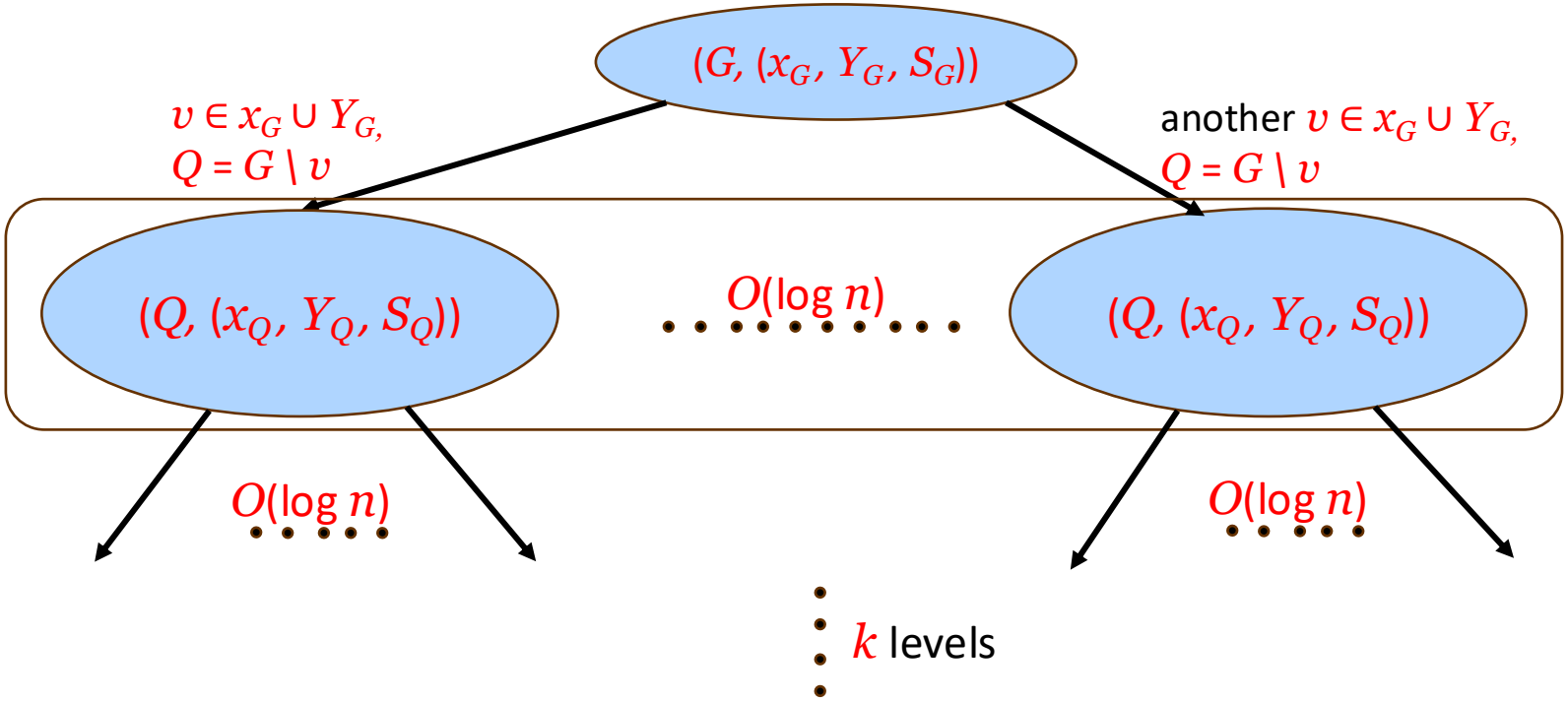
Preprocessing (cont.):

The idea is to repeatedly apply the lemma and store the information in a rooted tree T of depth k.

- The nodes of T correspond to subgraphs of G *obtained* by removing at most k vertices.
- The root of T corresponds to the entire tournament G, and stores a (x, Y, S) for G obtained using the lemma.
- The children of the root correspond to subgraphs of the form $G \setminus v$. The crucial observation is that we only need to consider vertices v that belong to $x \cup Y$ (other vertices do not affect the KS), and by the lemma we have $|Y| = O(\log n)$ i.e., the root has only $O(\log n)$ children. We repeat this process until depth k.
- Formally, the nodes of T are pairs of the form $(Q, (x_Q, Y_Q, S_Q))$ where Q is a subtournament of G and (x_Q, Y_Q, S_Q) is a KS for Q obtained using the lemma.
- The root of T is $(G, (x_G, Y_G, S_G))$. For a node $(Q, (x_Q, Y_Q, S_Q))$ of T, its children are defined as follows:
  - For each $v \in x_Q \cup Y_Q$ there is a child of the form $(Q^*, (x_{Q^*}, Y_{Q^*}, S_{Q^*}))$ where $Q^* = Q \setminus v$ is the tournament obtained from Q after removing v.
  - We construct T only until depth k (the depth of the root is 0).

Preprocessing (cont.):

- By the lemma, the number of children of a node of $\mathcal{T}$ at level r is $O(\log(n\text{-}r))$ as every tree vertex at level r corresponds to a tournament with $n\text{-}r$ vertices, whose KS is constructed in $O((n\text{-}r)^2) = O(n^2)$ deterministic time or $O(n)$ expected time.
- Hence, the entire tree $\mathcal{T}$ is of size $O(\log^k n)$ and is constructed in $O(n^2 \log^k n) = \tilde{O}(n^2)$ deterministic time or $\tilde{O}(n)$ expected time, as required.



Handling faults:

Recall: we must show that following any set of (at most) k vertex removals, we can compute a KS for the new tournament deterministically in $O(1)$ time.

- Let R denote the set of removed vertices.
- We locate in $\mathcal{T}$ a node $(Q, (x_Q, Y_Q, S_Q))$ where $V(G) \setminus V(Q) \subseteq R$ and $(x_Q \cup Y_Q) \cap R = \emptyset$. This can be done by traversing $\mathcal{T}$ from the root as we next describe:

  - Initialize $R^* = R$ and proceed as follows.
  - If the root $(G, (x_G, Y_G, S_G))$ already has the required property, we halt at the root.
  - Otherwise, let $v \in (x_G \cup Y_G) \cap R^*$. Set $R^* = R^* \setminus v$ and traverse to the child $(Q, (x_Q, Y_Q, S_Q))$ for which $V(G) \setminus V(Q) = \{v\}$.
  - Now continue in the same manner: Suppose we are at some tree node
    If $(x_Q \cup Y_Q) \cap R^* = \emptyset$ we halt at this tree node. Otherwise, let $v \in (x_Q \cup Y_Q) \cap R^*$. Set $R^* = R^* \setminus v$ and traverse to the corresponding child Q^* for which $V(Q) \setminus V(Q^*) = \{v\}$.

- Notice that since $|R| \leq k$ and since $|R^*|$ decreases by 1 as we traverse to the next child, the procedure must halt at some tree node $(Q, (x_Q, Y_Q, S_Q))$ after k steps as required, namely: $V(G) \setminus V(Q) \subseteq R$ and $(x_Q \cup Y_Q) \cap R = \emptyset$.

Handling faults (cont.):

- The time required to traverse the tree is $O(\log n)$ as each list Y_Q has $O(\log n)$ elements.
  - We may further reduce the traversal time to $O(1)$ as recall that in the KS of the Lemma, S_Q (the query d.s.) is just an array $P = P_Q$, where $P(v) = \emptyset$ if $v \in (x_Q \cup Y_Q)$.
  - So we just need to query $P(v)$ for every $v \in R^*$ and see if it is empty (and traverse to the corresponding child if this happens). As $|R^*| \leq k$ the total traversal time is time is $O(k^2) = O(1)$.

- Once we arrive at our desired tree node $(Q, (x_Q, Y_Q, S_Q))$ we have the property that $V(G) \setminus V(Q) \subseteq R$ and $(x_Q \cup Y_Q) \cap R = \emptyset$. Recall also that S_Q is just an array $P = P_Q$ such that $P(v) = \emptyset$ if $v \in (x_Q \cup Y_Q)$ and otherwise $P(v)$ is a vertex of $x_Q \cup Y_Q$ which dominates v.

- Let $G^* = G \setminus R$ (the subtournament obtained after removing R).
- Define $S_{G^*} = (P, R)$ (namely, the d.s., is the array P and the set R of removed vertices).
- We claim that $(G^*, (x_Q, Y_Q, S_{G^*}))$ is a KS for G^*.
- Indeed, given $v \in V(G)$ we wish to return (in $O(1)$ time) a path from the king to v.

  - If $v \in R$ then we return that the query is invalid (v is not a vertex of G^*).
  - Otherwise, if $v = x_Q$ we return the path $\{x_Q\}$,
  - Otherwise, if $(x_Q, v) \in E(G)$ then $(x_Q, v) \in E(G^*)$ and we return the path $\{x_Q, v\}$.
  - Otherwise, we return the path $\{x_Q, P(v), v\}$ (which is a valid path of length 2 in G^*).

- We have proved that a KS for G^* can be constructed in constant time, as required.

Thanks