Locally Self-Adjusting Tree Networks

Chen Avin (BGU)
Bernhard Häupler (MIT)
Zvi Lotker (BGU)
Christian Scheideler (U. Paderborn)

Stefan Schmid (T-Labs)
From “Optimal” Networks to Self-Adjusting Networks

- Networks become more and more dynamic (e.g., flexible SDN control)
- Vision: go beyond classic “optimal” static networks
- Example (of this paper): Peer-to-peer

**Chord, Pastry, SHELL**
- Hypercubic
- Log diameter
- Log degree
- Log routing

**Koorde, ...**
- Constant degree
- Log routing

**Pancake**
- Log/loglog degree and log/loglog routing
From “Optimal” Networks to Self-Adjusting Networks

- Networks become more and more dynamic (e.g., flexible SDN control)
- Vision: go beyond classic “optimal” static networks
- Example: Peer-to-peer Chord, Pastry, SHELL, Koorde, Pancake
- Hypercubic
- Log diameter
- Log degree
- Log routing
- Constant degree
- Log/loglog degree and log/loglog routing

What if networks could self-adjust depending on communication pattern?

Stefan Schmid (T-Labs)
An Old Concept: Move-to-front, Splay Trees, ...

- Classic data structures: lists, trees
- Linked list: move frequently accessed elements to front!

![Linked list diagram]

- Trees: move frequently accessed elements closer to root

![Tree diagram]
An Old Concept: Move-to-front, Splay Trees, ...

- Classic data structures: lists, trees

- Linked list: move frequently accessed elements to front!

- Trees: move frequently accessed elements closer to root
An Old Concept: Move-to-front, Splay Trees, …

- Classic data structures: lists, trees
- Linked list: move frequently accessed elements to front!
- Trees: move frequently accessed elements closer to root
An Old Concept: Move-to-front, Splay Trees, …

- Classic data structures: lists, trees
- Linked list: move frequently accessed elements to front!
- Trees: move frequently accessed elements closer to root
An Old Concept: Move-to-front, Splay Trees, …

- Classic data structures: lists, trees
- Linked list: move frequently accessed elements to front!
- Trees: move frequently accessed elements closer to root
An Old Concept: Move-to-front, Splay Trees, …

- Classic data structures: lists, trees
- Linked list: move frequently accessed elements to front!
- Trees: move frequently accessed elements closer to root

Splay Trees!
The Vision: Splay Networks ("Distributed Splay Trees")

- Most simple self-adjusting tree network: Binary Search Tree (BST)
The Vision: Splay Networks ("Distributed Splay Trees")

- Most simple self-adjusting tree network: Binary Search Tree (BST)
The Vision: Splay Networks ("Distributed Splay Trees")

- Most simple self-adjusting tree network: Binary Search Tree (BST)
The Vision: Splay Networks ("Distributed Splay Trees")

- Most simple self-adjusting tree network: Binary Search Tree (BST)

Why BST?!
- Most simple generalization of classic data structure
- Allows for local routing!
- Allows for algebraic gossip
Model: Self-Adjusting SplayNets

Input:
  - communication pattern: (static or dynamic) graph

Output:
  - sequence of network adjustments

Cost metric:
  - expected path length
  - # (local) network updates
Our Contribution

SplayNets

- “Online algorithm” for self-adjusting distributed trees
- Optimal offline algorithm (polynomial time, for large class of graphs!)

Performance evaluation:

- General bounds on amortized costs
- Lower bounds (empirical entropy)
- Analysis of convergence times for important static comm. patterns
- Optimality of online algorithm for special patterns (e.g., matchings)
- Simulation study (Facebook data)
The Optimal Offline Solution

Dynamic program
- Binary search: decouple left from right!
- Polynomial time (unlike MLA!)
- So: solved M”BST”A

See also:
- Related problem of phylogenetic trees
The Online SplayNets Algorithm

From Splay tree to SplayNet:

**Algorithm 1** Splay Tree Algorithm \( ST \)

1: (* upon lookup \((u)\) *)
2: **splay** \(u\) to root of \(T\)

**Algorithm 2** Double Splay Algorithm \( DS \)

1: (* upon request \((u, v)\) in \(T\) *)
2: \(w := \alpha_T(u, v)\)
3: \(T' := \text{splay } u\) to root of \(T(w)\)
4: \(\text{splay} \ v\) to the child of \(T'(u)\)
The Online SplayNets Algorithm

From Splay tree to SplayNet:

**Algorithm 1** Splay Tree Algorithm ST

1: (* upon lookup \( (u) \) *)
2: **splay** \( u \) to root of \( T \)
The Online SplayNets Algorithm

From Splay tree to SplayNet:

Algorithm 2 Double Splay Algorithm DS
1: (* upon request \((u, v)\) in \(T\) *)
2: \(w := \alpha_T(u, v)\)
3: \(T' := \text{splay } u\) to root of \(T(w)\)
4: \(\text{splay } v\) to the child of \(T'(u)\)

Least Common Ancestor
Analysis: Basic Lower and Upper Bounds

**Upper Bound**

\[ \text{A-Cost} < H(X) + H(Y) \]

where \( H(X) \) and \( H(Y) \) are empirical entropies of sources resp. destinations

Adaption of Tarjan&Sleator

**Lower Bound**

\[ \text{A-Cost} > H(X|Y) + H(Y|X) \]

where \( H(\cdot|\cdot) \) are conditional entropies.

Assuming that each node is the root for “its tree”

Therefore, our algorithm is optimal, e.g., if communication pattern describes a product distribution!
Nodes communicate within local clusters only!

Over time, nodes will form clusters in BST! No paths “outside”.
Properties: Optimal Solutions

Laminated scenario:

Will converge to optimum:
Amortized costs 1.

Non-crossing matching (= “no polygamy”) scenario:

Will converge to optimum:
Amortized costs 1.
Properties: Optimal Solutions

Multicast scenario (BST): Example

Invariant over “stable” subtrees (from right):
Improved Lower Bounds (and More Optimality)

Via interval cuts or conductance entropy:

Cut of interval: entropy yields amortized costs!

Grid:
Simulation Results

- Facebook component with 63k nodes and 800k edges
- SplayNet exploit random walk locality, to less extent also matching
Conclusion

- Vision: self-adjusting networks
- Interesting generalization of Splay trees
  - SplayNets
    - Formal analysis reveals nice properties
    - Amortized costs good: but tight?
    - Competitive ratio remains open
- Future work? Yes 😊
Thank you! Questions?

Algorithm 2 Double Splay Algorithm (DS)

1: (* upon request \((u,v)\) in \(T\) *)
2: \(w := \alpha_T(u,v)\)
3: \(T' := \text{splay } u\) to root of \(T(w)\)
4: \(\text{splay } v\) to the child of \(T'(u)\)

“Host Graph”

“Guest Graph”