A Near-Optimal Distributed Fully Dynamic Algorithm for Maintaining Sparse Spanners

Michael Elkin

Ben-Gurion University
The Message-Passing Model

- \( n \) processors reside in vertices of an unweighted undirected graph \( G = (V, E) \). Each processor has a unique id.

- Interconnected via links of \( E \).

- *Short* messages \( (O(\log n) \text{ bits}) \).

- Unlimited computational power. Local computation requires zero time.
The Message-Passing Model (Cont.)

Synchronous setting (for this talk).

- Communication in \textit{discrete} rounds.

- Messages sent in the beginning of a round $R$, arrive before the round $R + 1$ starts.

Running Time $= \#\text{rounds}$.

Message Complexity $= \#\text{messages}$.
Dynamic Model

Edges and vertices may appear or crash at will.

Motivation for the dynamic model: real-life networks, modern ad-hoc, sensor, wireless networks.

Require *simple* algorithms!

- Endpoints of a crashing edge are notified by a link-level protocol.

- Message is lost only if its edge crashes.
Quiescence Complexity; Spanners

Topology updates cease occurring at time $\alpha$. $\beta$ is the time when all vertices stop processing updates. At this point the algorithm maintains a correct structure.

Quiescence time $= \max\{\beta - \alpha\}$.
Quiescence message $= \#$ messages sent within $[\alpha, \beta]$.

$G' = (V, H)$ is a $t$-spanner of $G = (V, E)$, $H \subseteq E$, if $\forall u, w \in V$,

$$\text{dist}_{G'}(u, w) \leq t \cdot \text{dist}_G(u, w).$$
Applications of Spanners

Underlying construct for many distributed algorithms.

• Synchronization.
  [Peleg,Ullman,89],
  [Awerbuch,Peleg,90]

• Routing.
  [Hassin,Peleg,99]

• Approximate Distances and Shortest Paths Computation.
  [Awerbuch,Berger,Cowen,Peleg,93],
  [Elkin,01]

• Broadcast.
  [Awerbuch,Goldreich,Peleg,Vainish,89],
  [Awerbuch,Baratz,Peleg,92]
Distributed Spanners

State-of-the-art distributed static algorithm.

[Baswana, Sen, 03],
[Baswana, Kavitha, Mehlhorn, Pettie, 05]

For \( t = 1, 2, \ldots \), and \( n \)-vertex \( G \), constructs \((2t - 1)\)-spanner with expected \( O(t \cdot n^{1+1/t}) \) edges.

Time: \( O(t) \).
Message: \( O(|E| \cdot t) \).
Space: \( O(deg(v) \cdot \log n) \).

Near-optimal tradeoff.
Dynamic State-of-the-Art

[Baswana, Sen] composed with the simulation technique of [Awerbuch, Patt-Shamir, Peleg, Saks, 92]:

$(2t - 1)$-spanner of expected size $O(t \cdot n^{1 + 1/t})$,
Quiescence time: $O(t \cdot \log^3 n)$.
Quiescence message: $O(t \cdot |E| \cdot \log^3 n)$.
Space: $O(\deg(v) \cdot \log^4 n)$.

Drawbacks of APSPS simulation technique:

Extremely complex (a reset procedure, neighborhood covers, a bootstrap technique, a local rollback).

Heavy local computations - unsuitable for simple devices.
Our Result

$(2t - 1)$-spanner of expected size $O(t \cdot n^{1+1/t})$.

Quiescence time: $3t$ instead of $O(t \cdot \log^3 n)$. 
Note: $t \leq \log n$.

Quiescence message: worst-case $O(|E| \cdot t)$, expected $O(|E|)$.
Space: $O(\deg(v) \cdot \log n)$.
Expected local processing per edge: $O(1)$.

Lower bound: $2t/3$.
$t - 1$ under Erdos girth conjecture.

Better performance in purely incremental and purely decremental settings.

In both algorithms: non-adaptive adversary, oblivious to coin tosses.
Memoryless Dynamic Algorithm

**Standard approach:** maintain *history* of communication, undo operations based on the history.

Very expensive in terms of *local computation*. *Unfeasible* in wireless, sensor, ad-hoc networks.

**Our approach:** No history stored!

Look for a “replacement” for crashing edges.

Undo operations, but the list-to-undo is deduced from the current state of affairs.

Reminiscent of *online* algorithms.
The Incremental Variant: Initialization

For this talk: only incremental algorithm.

Set a parameter $p \approx n^{-1/t}$.

Each $v$ picks a radius $r(v)$ from the truncated geometric distribution

$$\text{IP}(r = k) = p^k \cdot (1 - p), \text{ for } k \in [0, \ldots, t - 2],$$
and $$\text{IP}(r = t - 1) = p^{t-1}.$$

Memoryless distribution

$$\text{IP}(r \geq k + 1 \mid r \geq k) = p$$
for $k \in [0, 1, \ldots, t - 2]$.

[Linial,Saks,92],[Bartal,96]
Labels

Each $v$ has a unique id $I(v)$, and a label $P(v)$.

Initially, $P(v) \leftarrow I(v)$ ($P(v) \leftarrow (I(v), 0)$).

$P = (B(P), L(P))$, or $P = n \cdot L(P) + B(P)$.

Implicitly, the algorithm maintains a tree cover.

$B(P)$ - the id of a tree $\tau$ to which the vertex $v$ labeled by $P$ currently belongs.

$L(P)$ - the distance between $v$ and the root of $\tau$.

The vertex $w = w_P$ s.t. $I(w) = B(P)$ is the base vertex of $P$.

$w_P$ is the root of the tree $B(P)$. 
$r(w_P)$ - maximum distance to which $B(P) = I(w_P)$ is allowed to propagate. The tree $B(P)$ cannot be deeper than $r(w_P)$.

⇒ For each label $P$, $L(P) \leq r(w_P)$.

A label $P$ is selected if $L(P) < r(w_P)$. In this case $v$ may be an internal vertex of the tree $B(P)$. 
Vertices *adopt* labels from their neighbors. When $v$ adopts a label from $u$, it becomes its child in the tree $B(P)$, $P = P(u)$. When a label $P$ is adopted, $L(P)$ is incremented, but $B(P)$ stays unchanged.

![Diagram](image)
Every $v$ maintains an edge set $Sp(v)$.

Initially, $Sp(v) = \emptyset$.

$Sp(v)$ grows monotonely.

$Sp(v) = T(v) \cup X(v)$.

$T(v)$ - the tree edges of $v$.

$X(v)$ - the cross edges of $v$.

An implicit construction of a tree cover. Edges of the tree cover are stored in $T(v)$’s.

The spanner also has edges connecting different trees. Those are edges of $X(v)$’s.
For each vertex $v$, the algorithm maintains a table $M(v)$. Initially, $M(v) = \emptyset$.

$M(v)$ is the set of trees to which $v$ is already connected in the spanner.

$e' = (v,z) \in X(v) \implies B(P(z) \in M(v)$

\[ B(P(z)) = B(P(u)) \]

\[ \downarrow \downarrow \]

\[ e \text{ can be dropped!} \]
The Algorithm

For $2t$ rounds from the beginning or after detecting a new edge do

Go over all received messages and do
while $\exists$ message $P(u)$ with $P(u) \succ P(v)$

if $u$ is selected

$B(P(v)) \leftarrow B(P(u))$;
$L(P(v)) \leftarrow L(P(u)) + 1$;
$Sp(v) \leftarrow Sp(v) \cup \{e\}$ // add $e$ to $T(v)$

else if $B(P(u)) \notin M(v)$

$M(v) \leftarrow M(v) \cup \{B(P(u))\}$;
$Sp(v) \leftarrow Sp(v) \cup \{e\}$ // add $e$ to $X(v)$

end-if
end-while

Send to all neighbors the message $P(v)$. 
The Algorithm: Discussion

Very simple:

1. One type of messages.

2. The same behavior on each round.

3. A handful of local variables.

4. A basic data structure.
Summary

• Optimal solution for the dynamic distributed spanner problem.

• Memoryless paradigm for devising dynamic distributed algorithms.

• Lower bound of $\Omega(t)$.

• Applications for streaming and dynamic centralized models.
Open Questions

• Applications for the dynamic spanners. Synchronization (?), Routing (?), Online load balancing (?).

• Applications for the memoryless paradigm.

• Achieve spanner size of $O(n^{1+1/t})$ instead of $O(t \cdot n^{1+1/t})$.

• Derandomize.
  Less challenging - devise algorithm for an adaptive adversary.