236602: Mathematical Techniques in the Theory of Distributed Computing

Spring 2013
Graph spanners

Given a graph $G=(V,E)$, a **t-spanner** is a subgraph $G’=(V,E’) \subseteq E$ such that

- for every: $(u,v) \in E$
- we have: $d_{G’}(u,v) \leq t \cdot d_G(u,v)$

We call $t$ the **stretch** of the spanner $G’$

The stretch holds for every path in $G$
Example

Stretch of the orange spanner is 4
Why Spanners?

Clearly, every graph is a 1-spanner of itself

We want a *sparse* spanner
  – As few edges as possible

Motivation:
  – Almost shortest paths
  – Routing schemes
  – Synchronizers
Synchronizers

Introduced by Awerbuch [1985]
Design algorithms for synchronous systems but run them in asynchronous systems
A synchronizer is a simulation scheme wrapped around the (synchronous designed) algorithm
Simple synchronizer?

Simple synchronizer: Each node $v$ generates pulse $p$ messages and sends them to neighbors. Moves to $p+1$ after receiving $p$-messages from all neighbors.

Problem: What if a neighbor $u$ never sent a $p$-message to $v$? Node $v$ will wait forever.
The problem

Solution 1:  
**u** sends an empty message

But this increases the message complexity compared to the synchronous case

Define \( C(s), T(s) \) to be the number of messages and time added by the synchronizer \( s \) per pulse
Simple Synchronizer

Solution 2:

1. Send an ACK upon receiving a message
   A node who receives all ACKs is called **safe**
2. Tell all your neighbors you are safe
3. Move to \( p+1 \) when all your neighbors are safe

Intuition: Moves responsibility to potential sender \( u \) instead of receiver \( v \)
Synchronizer

v is safe
Synchronizer

Why does this work?
• If \( v \) gets SAFE messages from all of its neighbors, then they are all safe.
• This means that they got ACKS for their \( p \)-message from everyone they sent it to.
• This implies that \( v \) got all \( p \)-messages sent to it.
Synchronizer

Number of messages is $C = O(|E|)$ per pulse, and time is $T = O(1)$ (time of slowest message)

Same as Solution 1...
Synchronizers

Can we do better than $C = O(|E|)$?

Assume we have a spanning tree

Send your SAFE messages towards the root if your sub-tree nodes are all safe

The root sends an ALLSAFE message down the tree
Synchronizers

Root of spanning tree
Synchronizers

Why does this work?
• If a node receives an ALL_SAFE message, then the root got SAFE messages from all of its neighbors in the spanning tree
• Implies that all nodes are safe
  – Proof by induction
Synchronizers

$C = \mathcal{O}(|V|)$, $T = \mathcal{O}(H)$, where $H$, the height of the tree, can be $\mathcal{O}(|V|)$ in the worst case.

So, we decreased $C$ but increased $T$. 

Root of spanning tree

ALL_SAFE
Synchronizers

Hybrid approach: assume a spanning tree of clusters. Within each cluster be safe if all neighbors are safe, between clusters wait for ALL_SAFE message

Complexity depends on sizes of clusters
Synchronizers vs. Spanners

**Theorem 1**: If G has a $t$-spanner with $m$ edges then it has a synchronizer $s$ with $C=O(tm)$, $T=O(t)$

**Theorem 2**: (Lower bound) If G does not have a $t$-spanner with at most $m$ edges then every synchronizer has either $C \geq m+1$ or $T \geq t+1$

So, we need $t$-spanners with small $t$, which are sparse, i.e., have small $m$
Spanner stretches

Another type of spanners: also an additive stretch

Given a graph $G=(V,E)$, an $(\alpha,\beta)$-spanner is a subgraph $G'=(V,E')$ such that

- for every: $u,v \in V$
- we have: $d_{G'}(u,v) \leq \alpha \cdot d_G(u,v) + \beta$
Almost shortest paths

Given a set $S$ of $s$ sources in $V$
compute shortest paths for every $(u,v) \in S \times V$

An algorithm is an $\alpha, \beta$-approximation if it
outputs a path of length at most $\alpha \cdot d_G(u,v) + \beta$
for every pair $(u,v) \in S \times V$

An $\alpha, \beta$-spanner would do this
Almost shortest paths

Solution: compute $s$ BFS trees

- Time: $O(s |E|)$

This gives exact shortest paths (not almost)

If we can compute an $(\alpha, \beta)$-spanner with $m$ edges in $O(T)$ time then we can compute the shortest paths in time $O(T+sm)$

This would be an $(\alpha, \beta)$-approximation
Routing Schemes

Nodes receive messages with destinations and need to choose which neighbor to forward to.

The **stretch** of a routing scheme is the worst ratio between a length of a routing path and the shortest path in the graph.

Sounds familiar?

Can also be weighted.
Spanners

[Peleg and Schäffer 1989]
Here: based on Peleg’s book

We will first see how to construct a “good” partition
And then use it to build a spanner

Sequentially!
A **cluster** is a subset $S$ of $V$ such that the induced subgraph $G(S)$ is connected.

$G(S) = (S, E')$ where $E'$ includes all edges with both endpoints in $S$.
A **cover** of G is a collection of clusters $P = \{S_1, \ldots, S_m\}$ such that $\bigcup_{1 \leq i \leq m} S_i = V$.
Partial Partition

A **partial partition** of $G$ is a collection of disjoint clusters $P=\{S_1, ..., S_m\}$, i.e. $S_i \cap S_j = \emptyset$ for every $i \neq j$ between 1 and $m$. 
Partition

A **partition** of G is a collection of clusters P that is both a cover and a partial partition of G
Locality

Our goal is to capture locality, so not every partition is going to be useful.

Consider a partition into long paths.
Radius of Clusters

\[ \text{Rad}(v,S) = \max_{w \in S} \{d_{G(S)}(w,v)\} \]

The radius of a cluster is the minimum of the above over all nodes in \( S \)

\[ \text{Rad}(S) = \min_{v \in S} \{\text{Rad}(v,S)\} \]

A center of \( S \) is any vertex \( v \) for which \( \text{Rad}(v,S) = \text{Rad}(S) \)

Notice that \( \text{Rad}(S) \leq \text{Diam}(S) \leq 2\text{Rad}(S) \)

We would like clusters with small radius

– Easy, take singletons
The Cluster Graph

But that is not enough

– Same for all graphs with n nodes...

Another measure of a partition is its **sparsity**

The cluster graph of a partition $P$ is $G^*(P) = (P, E^*)$

– Each cluster is a node

– There is an edge in $E^*$ for every $S, S'$ in $P$ that have an edge between them in $G$
  • These are called **intercluster** edges
Partition Sparsity

The other measure of a partition we will be interested (in addition to clusters of small radius) is the number of intercluster edges

These two measures impose a tradeoff

– If a partition has small clusters it may have many intercluster edges
  • Partition into singletons
– If a partition has only few intercluster edges it may have clusters with large radius
  • Partition into one cluster which is the entire graph
Algorithm: Partition(k)

- \( P = \emptyset \)
- While \( V \neq \emptyset \)
  - Select an arbitrary vertex \( v \in V \)
  - \( S = \{v\} \)
  - While \( |N(S)| > n^{1/k} |S| \) do
    - \( S = N(S) \)
    - Set \( P = P \cup \{S\} \) and \( V = V \setminus S \)
- Output \( P \)
Partition

**Theorem**: the algorithm Partition(k) constructs a partition \( P \) such that:
- For every \( S \in P: \text{Radius}(S) \leq k-1 \)
- The cluster graph \( G^*(P) \) has at most \( n^{1+1/k} \) intercluster edges