1 Distributed Optimization

Assume we have network of computational nodes, which can send a message along each edge in each time step. Typically in this case, computation is considered free, and the bottle neck is communication between nodes.

1.1 Trivial bounds on computation time

A trivial lower bound on time needed to solve a global problem (one requiring information from potentially distant parts of the network) is $\Omega(D)$, where $D$ is the diameter of the network. This is because we need to make sure there is at least enough time for a given piece of information to get anywhere in the network.

A trivial upper bound is $O(m)$, which is the amount of time required to send a given piece of information anywhere in the network.

1.2 Congest Minimum Spanning Tree Problem

The minimum spanning tree (MST) problem is a representative global problem. We want to compute a minimum spanning tree on a graph. Each edge has a cost, which it can communicate to its neighbors in a given time step.

1.2.1 Boruvka’s algorithm for determining edges $T$ in the MST:

- In the first round, each node adds it’s cheapest edge $T$. This yields a set of connected components.
- In the next round, the cheapest edge out of each component is added to $T$.
- repeat previous step until $T$ is an MST (see fig. 1).

Round-complexity of Boruvka’s algorithm is $\log(n)O(D_{comp})$, where $D_{comp}$ is the component diameter. The $\log(n)$ comes from the fact that the number of components goes down by at least a factor of 2 in each round. The $D_{comp}$ comes from the fact that it takes $D_{comp}$ to find the cheapest edge out of a component, found by flooding the component with the cheapest edge.

Note that $D_{comp}$ can be greater than $D$, the diameter of the entire graph, and is at most $n$. Therefore, the round-complexity is $\log(n)O(n)$.

1.2.2 More bounds on MST problem:

- KP’95: $\tilde{O}(D + \sqrt{n})$
- RP’99, E’04, DHK+’11: Strong $\tilde{\Omega}(\sqrt{n})$ lower bound

However, we should expect to do better than this for nicer network topologies.
Figure 1: Successive iterations of Boruvka’s MST algorithm. Thicker lines indicate edges that have been determined to be in the MST.
1.3 Communication Complexity

Consider 2 parties, Alice and Bob. Each gets a bit string, \( x \in \{0,1\}^n \), and \( y \in \{0,1\}^n \). Say we want to compute some function \( f(x, y) \) (see fig. 2).

Communication complexity is concerned with the number of bits we need to send between Alice and Bob to compute this function. In general, we need to communicate \( n \) bits to do this deterministically. However, if we use randomization, this number can be reduced. For example, say we set \( f \) to be the equality function:

\[
  f(x, y) = EQ(x, y) = 1_{x = y}
\]

We might compute this by hashing \( x \) and \( y \), and comparing the hashes.

A common problem is disjointness:

\[
  \text{Disjoint}(x, y) = 1_{\exists i: x_i = y_i = 1}
\]

Communication Complexity of the disjoint function is \( \Omega(n) \).

1.3.1 Reduction of the MST problem

We can consider a reduction of the MST problem to utilize this result from communication complexity. Consider a graph with the following structure. There exist \( \sqrt{n} \) cheap paths, each of length \( \sqrt{n} \). Node Alice is connected to the left side of some set of the paths, and node Bob is connected to the right side of some set of the paths. Additionally, there is an expensive path that goes from Alice to Bob. But we would like to use the cheap paths for our MST. Note that there is a cheap path between Alice and Bob along one of the \( \sqrt{n} \) paths iff the set of paths Alice and Bob connect to are not disjoint.

1.3.2 Example demonstrating \( \tilde{\Omega}(D + \sqrt{n}) \) bound

Consider a graph structure the same as the last example, but in which all nodes in the paths are connected to the bottom of a binary tree (see fig. 3). This makes the graph diameter logarithmic. If we send information along the paths, it will take \( \sqrt{n} \) time (long). On the other hand, we can take a shortcut by going up the tree, but to significantly decrease the length of the path we need to go high up the tree. This results in congestion, as the messages become concentrated over an ever smaller number of edges. The additional slow-down due to congestion means it will still take \( \sqrt{n} \) time steps to get all the messages across.
Figure 3: Binary tree shortcuts. Alice can route her messages up the binary tree to obtain a shortcut, however this increases congestion as there are fewer paths that go higher in the tree.

1.4 Low-Congestion Shortcuts

Motivation: We have an "optimal" algorithm in terms of n and D, however we KP algorithm is always $\Omega(\sqrt{n})$ slow even on much nicer networks. We want an algorithm that can run faster on such nicer networks.

1.4.1 Partwise aggregation

Given connected node-disjoint parts $S_1, \ldots, S_t$, we want to compute a simple partwise aggregate of numbers given to nodes

Informal Claim: This problem completely captures the crux of most distributed optimization problems. I.e., how fast this partwise information aggregation can be solved in a given topology determines approximately how efficient any optimization algorithm can be.

Idea: Instead of only communication within each part, allow each part to use some shortcut edges and nodes for its communication.

1.4.2 Definition: Low-congestion ($\delta$-dilation $\gamma$-congestion) shortcut

$H_1, \ldots, H_T \subset E$ are low-congestion shortcuts if:

- $\forall i, G[S_i] + E_i$ has small diameter $\delta$
- Each edge should be in fewer than $\gamma$ H-sets to avoid congestion

There are some trivial shortcuts we can take:

- Give all parts the complete graph: $\forall i, E_i = E$.
  - Yields $\delta = D$ and $\gamma = t < n$
- Don’t give parts any additional edges: $\forall i, E_i = \emptyset$. 

8-4
Yields \( \delta = n \) and \( \gamma = 1 \)

- Give complete graph to parts with diameter > \( \sqrt{n} \) (semi-trivial shortcut)
  - Yields \( \delta = \sqrt{n} + D \) and \( \gamma = \sqrt{n} \)

1.4.3 Boruvka’s MST algorithm with Shortcuts

- Start with \( T = \emptyset \)
- Repeat \( \log(n) \) times:
  - Compute a low-congestion shortcut for the connected components
  - Using the random delay routing, compute the cheapest outgoing edge for each component, add these edges to \( T \)

Running time: \( O((\sqrt{n} + D) \log^2 n) \) Generally no better shortcuts or algorithms are possible, but this algorithm becomes faster for non-pathological networks with better shortcut constructions.

2 Acknowledgments

These notes are based on a presentation and lecture by Prof. Bernhard Haeupler.