Distributed Algorithms for Finding 2-Edge-Connected Subgraphs

Tim Nonner
Albert Ludwigs University Freiburg
Department of Computer Science

December 7th 2007
Outline

- Motivation
- Model
- 2-Edge Connectivity
- Results
- Preliminaries
- Weighted 2-ECSS
Motivation

Broadcasting in networks
Motivation(2)

Broadcasting in Networks
Motivation(3)

Reduce number of edges
Bad resilience!
Goal: distributively find 2-Edge-Connected Spanning Subgraph (2ECSS)

NP-hard: reduction of Hamiltonian Path, APX-hard
Model

CONQUEST Model:
• Graph $G=(V,E)$, $n=|V|$, $m=|E|$
• Vertex = processor with unlimited power
• Edge = communication link
• Communication in rounds
• Small messages (bits $O(\log n)$)

• Time Complexity: number of rounds
• Communication Complexity: number of messages

• Generalization of communication complexity model
• Small messages simulate bandwidth limitation
Trivial Algorithm: Collect the network topology in one vertex and solve the problem locally

$\Omega(\log n)$ bits to represent an edge $\Rightarrow$

- Trivial: $\Omega(n \cdot m)$
- Nontrivial: $O(n \log^c n) = \tilde{O}(n)$
- Sublinear: $O(D + n^c), c < 1$
- Local: $O(\log n)$
Unweighted 2-Edge Connectivity

- Folklore: 2-approximation
  - Any tree $\rightarrow$ MST

  - $O(n \log n) \rightarrow \tilde{O}(D + n^{0.614}) \rightarrow \tilde{O}(D + \sqrt{n})$

- Khuller et al. (1994): 3/2-approximation
  - DFS-tree $\rightarrow$ Augmentation

- Jothi et al. (2003): 5/4-approximation
  - More sophisticated
Weighted 2-Edge Connectivity

• Friedrickson et al. (1981): 3-approximation
  -> MST -> minimum directed spanning tree

• Khuller et al. (1994): 2-approximation
  -> minimum k-edge disjoint rooted subgraph

Distributed:
• Geometric graph: topology control ......
• Triangle Ineq.: Hajiaghayi et al. (2003): const. apx.

Distributed = Simple?
## Results for distributed 2-ECSS

<table>
<thead>
<tr>
<th></th>
<th>apx.factor</th>
<th>time</th>
<th>communic.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unweighted:</strong></td>
<td>$\frac{3}{2}$</td>
<td>$O(n)$</td>
<td>$O(m + n^2)$</td>
</tr>
<tr>
<td><strong>Weighted:</strong></td>
<td>3</td>
<td>$O(n \log n)$</td>
<td>$O(n \log^2 n + m)$</td>
</tr>
</tbody>
</table>
Preliminaries

Both algorithms use the general strategy:
• Calculate spanning subtree (tree edges)
• Augment this tree (back edges)

A back edge covers a tree edge if
Weighted 2-ECSS

Use strategy of Friedrickson:

• Calculate MST
• Calculate a 2-approximation of an optimal augmentation

Khuller et al. (1993): 2nd step -> minimal directed spanning subtree -> infeasible in distributed context: best time $O(n^2)$
Weighted 2-ECSS: Chain Case

MST is a chain: optimal augmentation corresponds to shortest path

Distributed Bellman-Ford in time/comm.\[O(n)/O(n \cdot m)\]

Problem: can not be extended to general trees
Weighted 2-ECSS: Chain Case

MST is a chain: optimal augmentation corresponds to shortest path

Distributed Bellman-Ford in time/comm.
\( O(n)/O(n \cdot m) \)

Problem: can not be extended to general trees
Weighted 2-ECSS: Chain Case(2)

Use hierarchical fragmentation of tree to iteratively calculate shortest path via an inorder-traversal
Weighted 2-ECSS: Chain Case(2)

Use hierarchical fragmentation of tree to iteratively calculate shortest path via an inorder-traversal.
Weighted 2-ECSS: Chain Case(2)

Use hierarchical fragmentation of tree to iteratively calculate shortest path via an inorder-traversal.
Weighted 2-ECSS: Chain Case(2)

Use hierarchical fragmentation of tree to iteratively calculate shortest path via an inorder-traversal.
Weighted 2-ECSS: Chain Case(2)

Use hierarchical fragmentation of tree to iteratively calculate shortest path via an inorder-traversal
Weighted 2-ECSS: Chain Case(2)

Use hierarchical fragmentation of tree to iteratively calculate shortest path via an inorder-traversal.
Emulate **inorder traversal** via sending of **token**

$\rightarrow$ **time/comm.** $O(n \log n + m)$
Weighted 2-ECSS: General Case

- Decompose a MST in heavy paths
Weighted 2-ECSS: General Case

- Decompose a MST in heavy paths
- Bottom-Up, for each heavy path, use the Chain Case algorithm: use the projection of each edge
Weighted 2-ECSS: General Case

- Decompose a MST in heavy paths
- Bottom-Up, for each heavy path, use the Chain Case algorithm: use the projection of each edge,...

... and use metric that incorporates additional benefit
Weighted 2-ECSS: General Case

- Decompose a MST in heavy paths
- Bottom-Up, for each heavy path, use the Chain Case algorithm: use the projection of each edge,...
  ... and use metric that incorporates additional benefit
- Top-Down, add „shortest“ paths
Weighted 2-ECSS: General Case(2)

3-approximation algorithm:

1 for MST, 2 for augmentation
time/comm. $O(n \log n) / O(n \log^2 n + m)$

Sketch of augmentation proof......
Weighted 2-ECSS: General Case(3)

1st case: \( w_i(e) = w_e(e') + d_e \cdot \text{length}(HP_e) \)
Weighted 2-ECSS: General Case(4)

- 2nd case: $w_i(e) = w(e')$
Weighted 2-ECSS: General Case (5)

• Run the algorithm with the same edge weights but restricted to a set $E' \subseteq E \rightarrow$ backedges $A(E')$
• Let $A_1, A_2, ..., A_k$ be the calculated "shortest" paths
• Split the edges of a path $A_i$ in first case and second case edges

\[
\begin{align*}
w_i(A_i) & \leq \sum_{e \in SC} w(e) + \sum_{e \in FC} (w_e(A_e) - \text{length}(HP_e)) \\
& = \text{ALG}(E')_i,
\end{align*}
\]
\[
\sum_i w(\text{ALG}(E')_i) \geq \sum_i \text{length}(P_i) = \sum_i w(\text{ALG}(E)_i)
\]
Weighted 2-ECSS: General Case(6)

• Let $E'$ be an optimal augmentation -> $ALG(E') = E'$

$$w(A(E)) \leq \sum_i w_i(A(E)_i) \leq \sum_i w_i(A(E')_i) \leq 2 \cdot w(A(E'))$$

-> 2-approximation
Conclusion/Open Problems

- Lower bounds
- Higher connectivity
- Randomization / statistical arguments
- Connectivity checks
- Node connectivity

The combination of connectivity and distribution makes sense
Questions?