Algorithms for computing Maximally Redundant Trees for IP/LDP Fast-Reroute
draft-enyedi-rtgwg-mrt-frr-algorithm-00

Gábor Sándor Enyedi
egboeny@ericsson.com

Alia Atlas
akatlas@juniper.net

András Császár
eandcss@ericsson.com
• Maximally Redundant Trees
  – A pair of directed spanning trees
  – The common root is reachable along both of them
  – The two paths along the two trees are maximally disjoint
Why do we need this draft?

• We need a pair of MRTs rooted at each node
  – All the nodes should compute the same!
  – We will need standardization for MRT computation (algorithm) or results of that computation.
Principles

Partial order
ADAG
Blocks and GADAG
Partial order

• Partial order of a set (e.g. set of nodes)
  – A relation like a normal set
  – Except: not all the elements can be compared
    • For some $a$ and $b$ neither $a<b$ nor $a>b$

• Graph representation:
  – Directed Acyclic Graph (DAG)

```
min < a < b < f < c < e < max
• a < d < e
```
Finding node-disjoint paths

• Suppose that
  – We have a partial order of nodes
  – Exactly one min and max
  – Each node (except min and max) has a lower and greater neighbor

• Walk down and up
  – Min and max are reached
  – The two paths are node-disjoint!

min

max

• min<a<b<f<e<c<max
• a<d<e
Two paths to the same node

- DAG is not enough
  - Let min and max be the same node!
- Resulting graph is an Almost DAG (ADAG)
  - There is a single node, the root, such that without the root it is a DAG

\[
\begin{align*}
\text{min} & \rightarrow a \rightarrow b \\
\text{root} & \rightarrow d \rightarrow c \rightarrow f \\
\text{max} & \rightarrow e
\end{align*}
\]

- root \(a\rightarrow b\rightarrow f\rightarrow c\rightarrow e\rightarrow\text{root}\)
- \(a\prec d\prec e\)
Redundant paths to the root

- **Blue path**: Nodes must increase
- **Red path**: Nodes must decrease
- Load sharing is possible
Finding an ADAG (2-connected networks)

- Phase 1 – basic partial ADAG
  - Find a partial ADAG for a cycle containing the root
    - Use either direction
  - Extend partial ADAG into all nodes

![Diagram of a network with a root and nodes a, b, c, d, e, f connected by directed edges. The root is highlighted with a green outline.](image-url)
Finding an ADAG (2-connected networks)

• Phase 2 – extending
  – Find a path from one “ready” node to the another
  – Nodes along the path must not be ready (except the endpoints)
  – Add the path to the ADAG in a “proper” direction
Adding not used links

• Some links may be out of the ADAG

![Diagram of a directed acyclic graph (DAG)]
How can ordering be kept up?

- **ADAG is almost a DAG**
  - Let *root* be now only the smallest one
  - Now, it’s a DAG, create a topological sort
  - This is a total order
    - Add extra links with respect to this

Add back links to root and we have an ADAG using all links.
What if the network is not 2-connected?

- We need to split the graph into blocks
  - Block:
    - Maximally 2-connected subgraph
    - Two connected nodes
    - (Isolated node)
  - Each block has its local-root
    - That is the cut-vertex towards the root
  - Compute an ADAG in all the blocks
  - This is a Generalized ADAG
Generalized ADAG

- Block 1: root, a, b, c, d, e, f
- Block 2: f, g
- Block 3: g, h, i, j, k
The algorithm

MRTs in a block
MRTs in the whole network
How to Find MRTs

• If it is complex, then we break the problem down
  – Transform network into its blocks
  – Find ADAGs in each block
  – Connect up the ADAGs to make a GADAG
  – Add all the other links in – with the proper directionality

• From a GADAG, compute your next-hops to each destination
  – First for those in the same block
  – Destinations outside the block inherit their next-hops from a proxy in the block
How to Find MRTs

• If it is complex, then we break the problem down
  – Transform network into its blocks
  – Find ADAGs in each block
  – Connect up the ADAGs to make a GADAG
  – Add all the other links in – with the proper directionality

• From a GADAG, compute your next-hops to each destination
  – First for those in the same block
  – Destinations outside the block inherit their next-hops from a proxy in the block
MRTs in a single block

• As the computing router S: From the GADAG, can use SPF and reverse SPF to find next-hops to all destinations in the same block
  – SPF gives nodes definitely greater
  – rSPF gives nodes definitely lesser
  – Remaining nodes are not ordered

• Then use some simple rules
MRTs in a single block: source perspective

- Find greater and lesser nodes
- Rules
  1. If $S < D$ – increase to $D$ decrease to root
  2. If $S > D$ – increase to root decrease to $D$
  3. No order – decrease to root increase to root
  4. If $D=\text{root}$ – increase to root decrease to root
  5. If $S=\text{root}$ – increase to $D$ decrease to $D$

Routing table of node $c$ ($S = c$):

<table>
<thead>
<tr>
<th>Dest (D)</th>
<th>Rule Used</th>
<th>Blue Next-hop</th>
<th>Red Next-hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2</td>
<td>e</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>e</td>
<td>b</td>
</tr>
<tr>
<td>d</td>
<td>3</td>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>e</td>
<td>1</td>
<td>e</td>
<td>b</td>
</tr>
<tr>
<td>f</td>
<td>2</td>
<td>e</td>
<td>f</td>
</tr>
<tr>
<td>root</td>
<td>4</td>
<td>e</td>
<td>b</td>
</tr>
</tbody>
</table>
MRTs in a single block: destination perspective

- Find greater and lesser nodes
- Rules
  1. If $S < D$ – increase to $D$ decrease to root
  2. If $S > D$ – increase to root decrease to $D$
  3. No order – decrease to root increase to root
  4. If $d=root$ – increase to root decrease to root
  5. If $s=root$ – increase to $D$ decrease to $D$

Destination: node $c$ ($D = c$)

<table>
<thead>
<tr>
<th>Src (S)</th>
<th>Rule Used</th>
<th>Blue Next-hop</th>
<th>Red Next-hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>b</td>
<td>root</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>c</td>
<td>a</td>
</tr>
<tr>
<td>d</td>
<td>2</td>
<td>a</td>
<td>e</td>
</tr>
<tr>
<td>e</td>
<td>3</td>
<td>root</td>
<td>c</td>
</tr>
<tr>
<td>f</td>
<td>2</td>
<td>c</td>
<td>e</td>
</tr>
<tr>
<td>root</td>
<td>1</td>
<td>a</td>
<td>e</td>
</tr>
</tbody>
</table>
How to Find MRTs

• If it is complex, then we break the problem down
  – Transform network into its blocks
  – Find ADAGs in each block
  – Connect up the ADAGs to make a GADAG
  – Add all the other links in – with the proper directionality

• From a GADAG, compute your next-hops to each destination
  – First for those in the same block
  – Destinations outside the block inherit their next-hops from a proxy in the block
Inter-block MRTs

Proxy node: the last vertex in the block to the destination (along *any* path)

<table>
<thead>
<tr>
<th>Dest (D)</th>
<th>Rule Used</th>
<th>Blue Next-hop</th>
<th>Red Next-hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2</td>
<td>e</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>e</td>
<td>b</td>
</tr>
<tr>
<td>d</td>
<td>3</td>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>e</td>
<td>1</td>
<td>e</td>
<td>b</td>
</tr>
<tr>
<td>f</td>
<td>2</td>
<td>e</td>
<td>f</td>
</tr>
<tr>
<td>root</td>
<td>4</td>
<td>e</td>
<td>b</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dest (D)</th>
<th>Proxy</th>
<th>Blue Next-hop</th>
<th>Red Next-hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>f</td>
<td>e</td>
<td>f</td>
</tr>
<tr>
<td>h</td>
<td>f</td>
<td>e</td>
<td>f</td>
</tr>
<tr>
<td>i</td>
<td>f</td>
<td>e</td>
<td>f</td>
</tr>
<tr>
<td>j</td>
<td>f</td>
<td>e</td>
<td>f</td>
</tr>
<tr>
<td>k</td>
<td>f</td>
<td>e</td>
<td>f</td>
</tr>
</tbody>
</table>

![Diagram of a network with nodes labeled a, b, c, d, e, f, g, h, i, j, k, and root, and arrows indicating the next hops based on the rules specified in the table.](image)
Example – destination is node C

- **Block1:** root, a, b, c, d, e
- **Block2:** e, f
- **Block3:** f, g, h, i, j
Summary

• Algorithm
  – Find GADAG
    • ADAG in each block
    • Add not used links
  – Find next-hops along the MRTs
    • Do an SPF and an rSPF to find ordered nodes
    • Use rules to find NHs your block
    • Find proxy nodes
Thanks for the attention