

A Hybrid CP-LP Method for a Shortest Path Routing Problem

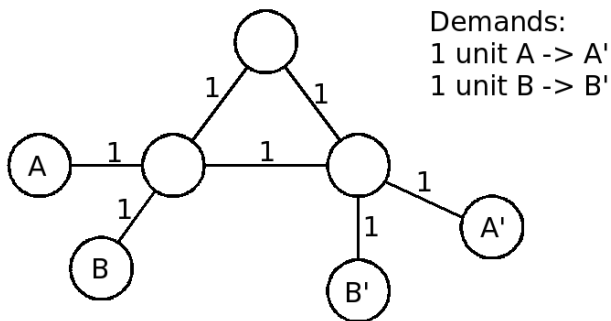
M.P. Petterson, R. Szymanek, K. Kuchcinski

February 5, 2007

- 1 The Problem
- 2 Solution Approach
- 3 The CP Model
- 4 Search
- 5 Experiments
- 6 Future Work

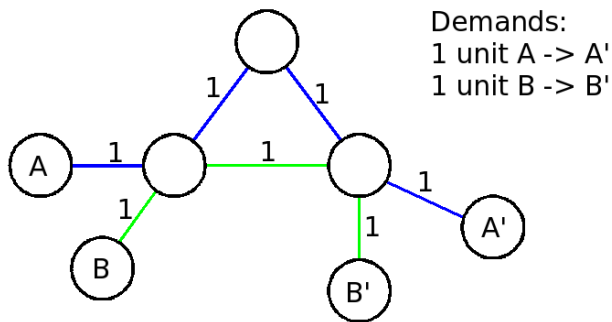
- 1 The Problem
- 2 Solution Approach
- 3 The CP Model
- 4 Search
- 5 Experiments
- 6 Future Work

A Simple Routing Problem with no Routing Restrictions



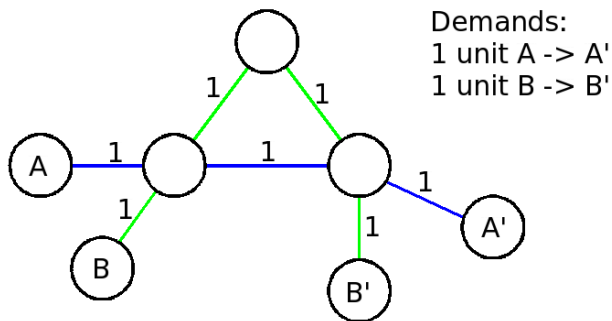
- Bandwidth demanded between source and target node pairs.
- Limited capacity on edges.

A Simple Routing Problem with no Routing Restrictions



- Bandwidth demanded between source and target node pairs.
- Limited capacity on edges.

A Simple Routing Problem with no Routing Restrictions



- Bandwidth demanded between source and target node pairs.
- Limited capacity on edges.

Problems with Unrestricted Routing

Unrestricted routing is nice, but there are problems:

- There are many decisions to make in a large network with many demands.
- What happens if a link fails?
- Much work to do in routers.

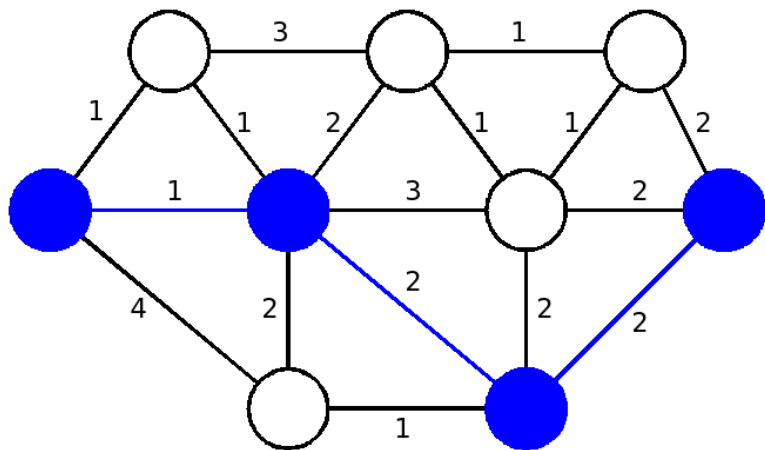
Shortest Path Routing

In shortest path routing, every edge is assigned a positive integer weight by the network administrator. This information is broadcasted to each router in the network.

Traffic is routed on shortest paths as defined by these weights. Routers can compute these easily.

Typically administrators use very simple methods to set the weights.

An Example Network with a Shortest Path



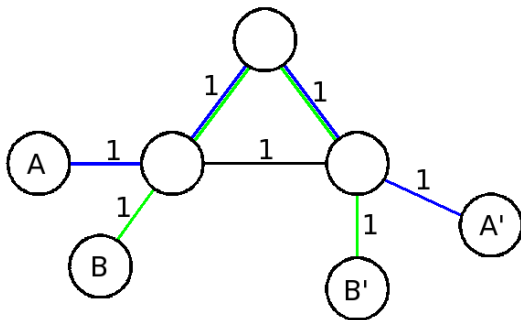
Shortest Path Routing

Advantages with shortest path routing:

- Simple. Routers only need to know about the weights.
- Fast. Routers only need to look at the destination of each packet.
- Flexible. “Automatic” rerouting when links go down.

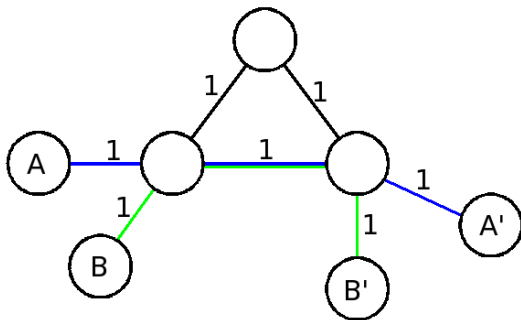
Shortest Path Routing

Disadvantage: Not all routings are possible, so it may not be possible to use the network optimally.



Shortest Path Routing

Disadvantage: Not all routings are possible, so it may not be possible to use the network optimally.



Two Further Restrictions

We consider shortest path routing with two further restrictions.

- Shortest paths are unique, i.e., there is only one shortest path per demand.
- Traffic is symmetric, i.e., the path from s to t is “the same” as the path from t to s .

Symmetry allows us to consider only undirected graphs.

Some Terminology

A *weight system* assigns a weight to each edge.

A *path system* specifies a unique simple path between each node pair.

A path system is defined by a weight system if all the paths are unique shortest paths w.r.t. the weights.

A path system is *feasible* if it is defined by some weight system.

A Unique Shortest Path Routing Problem :

An instance of the problem is defined by:

- An undirected graph, (V, E) .
- A link capacity, $c_{\{u,v\}}$ for each undirected edge $\{u, v\}$.
- A bandwidth demand, $d_{\{s,t\}}$, for each node pair $\{s, t\}$.

A solution is defined by a weight system s.t.:

- The weight system uniquely defines a path system.
- Using this path system to route the demands will not lead to exceeding the capacity on any edge.

An Example Real Network



- 1 The Problem
- 2 Solution Approach**
- 3 The CP Model
- 4 Search
- 5 Experiments
- 6 Future Work

Branching on the weight system

First idea:

- One weight variable per edge.
- Branch on weight variables.
- Propagate to variables representing the resulting path system and flows on edges

This does not work very well.

- Many weights need to be set before we can infer shortest paths.
- Many weight systems define the same path system.
- We do not know how large weights we may need.

Branching on the path system

Instead:

- Branch on variables defining the path system.
- Propagate to variables representing flows on edges.
- No weight variables!
- Use constraints that restrict the set of allowed path systems to those that can be defined by a weight system.
- Compute weights when we have a final path system.

- 1 The Problem
- 2 Solution Approach
- 3 The CP Model**
- 4 Search
- 5 Experiments
- 6 Future Work

For each node pair, $\{s, t\}$, we model its path in two ways:

- As a set of nodes. One binary *node path variable*, $n_{\{s,t\}u}$, per node $u \in V$. $n_{\{s,t\}u} = 1$ if the node u is included in the path between s and t .
- As a set of edges. One binary *edge path variable*, $e_{\{s,t\}\{u,v\}}$, per edge $\{u, v\} \in E$. $e_{\{s,t\}\{u,v\}} = 1$ if the edge $\{u, v\}$ is included in the path between s and t .

Either way is sufficient, but some constraints are more conveniently expressed on node path variables, some on edge path variables.

- For each edge, $\{u, v\}$, we have a *flow variable*, $f_{\{u,v\}}$, equal to the total use of bandwidth on that edge.

Five Basic Types of Constraints

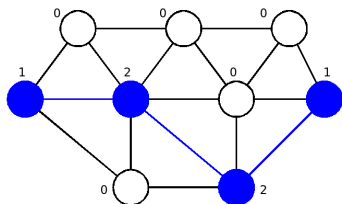
- Channelling constraints for node and edge path variables.
- Path constraints.
- Feasibility constraints.
- Constraints linking path and flow variables.
- Capacity constraints.

Channelling constraints link node path and edge path variables:

$$e_{\{s,t\}\{u,v\}} = 1 \Leftrightarrow n_{\{s,t\}u} = 1 \wedge n_{\{s,t\}v} = 1 \\ \wedge \forall w \neq u, v. n_{\{u,v\}w} = 0$$

This is checked every time a path variable is assigned.

Path Constraints

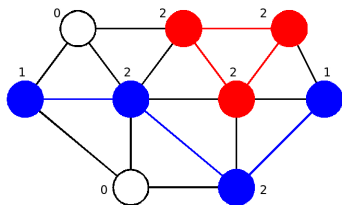


For each node pair, $\{s, t\}$, its edge path variables should define a unique simple path. This requires that for each node, v :

- If $v = s$ or $v = t$, v should have exactly one adjacent edge.
- Otherwise, v should have either two or zero adjacent edges, depending on whether it is included in the path or not.

This is checked every time an edge path variable is assigned.

Path Constraints



We also need to rule out disconnected loops. As soon as there is a path connecting s and t , we check that all remaining edge path variables can be set to zero.

Constraints Linking Path and Flow Variables

The flow on an edge is defined by the using demands.

$$f_{uv} = \sum_{\{s,t\}} d_{\{s,t\}} \cdot e_{\{s,t\}\{u,v\}}, \forall \{u, v\} \in E$$

No link can have a flow exceeding its capacity:

$$f_{\{u,v\}} \leq c_{\{u,v\}}, \forall \{u, v\} \in E$$

If we remove all routing restrictions, allowing flow between node pairs to spread freely in the graph, we have the *multicommodity flow problem*.

- A relaxation of our problem.
- Possible to express as a linear program.
- Gives us more global reasoning about capacity bottlenecks.

For partially defined path systems, we can include some path decisions in the LP model.

Given a path system, the problem of determining whether it is feasible, the *inverse shortest paths problem*, can be solved using linear programming.

However, this is costly, and is only done when a path system is fully defined.

Feasibility, 3-Node Constraint

Consider any three nodes in the network, a , b , and c . The shortest path routing restriction constrains the possible combinations of values for the three node path variables defined on these nodes. Out of the eight possible assignments, only four are actually possible. The constraint can be expressed easily:

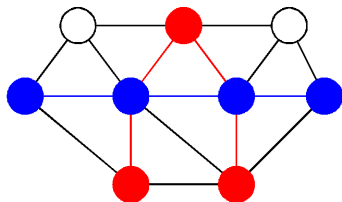
$$n_{\{a,b\}c} + n_{\{a,c\}b} + n_{\{b,c\}a} \leq 1$$

This is only a necessary condition for feasibility. It is checked every time a node path variable is assigned.

Feasibility, 4-Node Constraint

We can make the same kind of consideration for all groups of four nodes. For each group, there are twelve variables involved. Out of the 4096 assignments, only 53 are actually consistent with the weight requirement. This is only a necessary condition for feasibility. It is checked every time a node path variable is assigned.

Graph Partition Conditions



Sometimes, removing the nodes in a path will divide the graph into disconnected components. Due to the shortest path routing, paths between nodes in one of the component can never use nodes in the other component.

Most common in planar graphs.

- 1 The Problem
- 2 Solution Approach
- 3 The CP Model
- 4 Search**
- 5 Experiments
- 6 Future Work

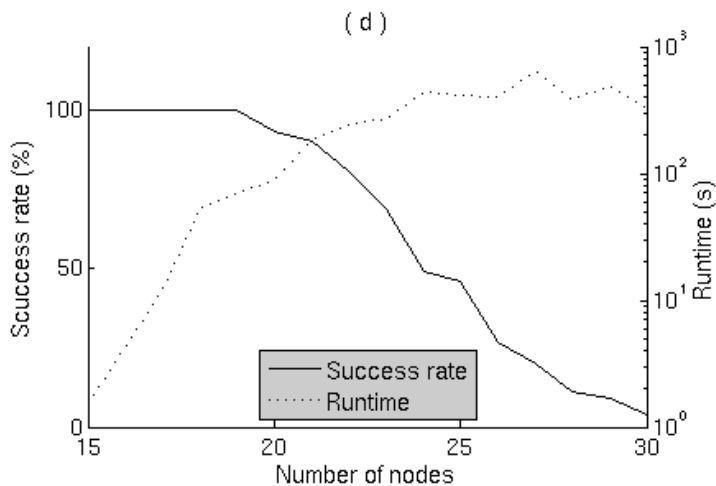
Our branching policy tries to construct paths between distant node pairs.

- We choose the node pair, $\{s, t\}$, that requires the highest number of edges for its path.
- Search will, recursively through backtracking, try to build a path from s to t .
- When building the path, we prefer to include edges that minimize the number of edges in the path.

- 1 The Problem
- 2 Solution Approach
- 3 The CP Model
- 4 Search
- 5 Experiments**
- 6 Future Work

We generate our own test data. There are no big public datasets available.

- Build a random topology. A node has three neighbors on average.
- Choose random weights for the edges until they define a unique path system.
- Choose random bandwidth requirements for demands.
- Compute minimum required bandwidth for each edge. Increase by some percentage (10%).



- 1 The Problem
- 2 Solution Approach
- 3 The CP Model
- 4 Search
- 5 Experiments
- 6 Future Work**

Some extensions have been implemented:

- Optimization.
- Random restarts search. The problem seems to be heavy-tailed.

Other:

- Directed flows.
- New necessary conditions for path system feasibility.
- Network design with shortest path routing.
- Fault-tolerant shortest path routing.