A CP-LP Hybrid Method for Unique Shortest Path Routing Optimization

Mats Petter Pettersson, *Department of Computer Science, Lund University, Sweden*
Radosław Szymanek, *Ecole Polytechnique Fédérale de Lausanne, Switzerland*
Krzysztof Kuchcinski, *Department of Computer Science, Lund University, Sweden*

Abstract

In this paper, we consider a routing problem related to the widely used Open Shortest Path First (OSPF) protocol. We address the special version of OSPF which requires unique and symmetrical paths. To solve this problem, we propose a hybrid approach which combines Constraint Programming (CP) and Linear Programming (LP). Our approach employs some new constraint with problem-specific filtering algorithms to efficiently remove inconsistent values from partial solutions. Moreover, we use two LP relaxations to indicate infeasible partial solutions due either to network capacity constraints, or to protocol-specific routing constraints. We show the efficiency of our complete approach on backbone networks with hundreds of different demands to route.

Keywords: shortest path routing, constraint programming, linear programming

1. Introduction

Telecommunication routing problems, where data traffic is to be routed between pairs of routers in a network, are naturally modelled as network flow problems, where the routers are represented by nodes and network links are represented by edges in a graph. Traffic is specified as point-to-point bandwidth demands for all pairs of routers. Finally, each link in the network has a limited bandwidth capacity. The objective of the routing problem is to route demands in a way that minimizes the utilization of the links, relative to their capacity.

In this work, we consider routing optimization for the Open Shortest Path First (OSPF) routing protocol, one of the most commonly used routing protocols in backbone networks. In OSPF, network operators control routing by assigning a weight to each link in the network. Routers will send traffic along the shortest paths defined by these weights. The version of OSPF routing where more than one shortest path is allowed between node pairs has been shown to be NP-hard [13].

We consider a version of OSPF with the further requirements that each demand is routed along a unique path, and that routing is symmetric, i.e., demands from s to t use exactly the same edges as demands from t to s. There are practical reasons for routing demands symmetrically on unique paths. It produces flows that are preferred by network operators, for example because packets always arrive in the correct order [3].

Previous approaches to this problem and similar problems have mainly applied mathematical programming [4, 14] and local search techniques [9]. One mathematical programming approach [1] has used some constraints to improve search performance. Our approach differs from this in fundamentally being a CP method, augmented with some LP relaxations. Other related work in the CP community has focused on network problems with single path routing but without the extra weight-based requirements on path systems introduced by OSPF [6, 11, 15, 16]. In [12], we presented a method for finding feasible solutions for this problem. The contribution of this work lies in applying an objective function, and using random restarts in a search scheme with two parallel searches, to make optimization more efficient while preserving the possibility to prove optimality.

This paper is organized in the following way. Section 2 defines the problem more formally. In Section 3, we introduce our modelling of the problem, both the CP part and the LP relaxations. Search control is discussed in Section 4. The experimental results are presented in Section 5, followed by conclusions in Section 6.

2. The problem

We are considering the unique shortest path problem with symmetric routing. Since we consider the symmetric case, we use an undirected graph, (V, E), to represent the network topology. We will use set notation, $\{u, v\}$, to denote the undirected edge that connects nodes u and v. For every edge $\{u, v\} \in E$, we have a link capacity, $c_{\{u,v\}}$, defined. Further, we let $b_{\{s,t\}}$ denote the sum of all bandwidth requirements for demands with s and t as endpoints. If there are no demands between s and t, $b_{\{s,t\}} = 0$. Finally, we let D denote the set of all unordered node pairs, i.e., $\{\{s, t\} \mid s \in V, t \in V, s \neq t\}$.

A feasible solution defines a positive integer link weight, $w_{\{u,v\}}$, for each $\{u, v\} \in E$, so that this weight system defines a unique shortest path between every node pair. Given such a weight system, we let $x_{\{s,t\}\{u,v\}} = 1$ if the edge $\{u, v\}$ is included in the shortest path between s and t and 0 otherwise. The objective to be minimized, z, is the maximum link utilization over all network links, (1). This objective is frequently used for network optimization problems, but alternatives exist [10].

$$z = \max_{\{u,v\} \in E} \left(\frac{1}{c_{\{u,v\}}} \sum_{\{s,t\} \in D} b_{\{s,t\}} x_{\{s,t\}\{u,v\}} \right) \tag{1}$$

3. Modelling

Our method is based on branch-and-bound search for a CP model of the problem. This is combined with two LP relaxations that can be used for early detection of branches without solutions. Every time an LP relaxation is used, at some search node, its formulation is tightened by inferences made in the CP model. In the rest of this section, we briefly introduce constraint programming, describe the CP model, and the two LP relaxations.

In CP, modelling is typically done with variables with finite discrete domains. Constraints on subsets of variables restrict what partial assignments are *consistent* (allowed) for those variables. The constraints collectively define the set of feasible solutions for the problem as a whole – full assignments consistent with every constraint. The set of constraints can be heterogeneous, unlike linear programming where only linear inequalities are allowed. Unlike LP, inference in CP is not global, not taking all constraints into account at once, but done by each constraint in isolation. Whenever a variable has its domain of possible values reduced, constraints involving that variable may imply domain reductions in other variables, which in turn may lead to further domain reductions in other variables. This process is known as *constraint propagation*, and will be triggered at each new search node by any domain reductions caused by the branching decision. Thus each constraint is equipped with a specific *filtering algorithm*, which is invoked whenever there is a domain reduction in any of the variables involving the constraint, and is responsible for the inference and enforcing of domain reductions in its other variables. When a domain has its domain emptied, we have discovered an inconsistency, and the search will backtrack.

We use two ways of modelling a shortest path for each node pair: as a set of edges and as a set of nodes. Binary *edge path variables*, $x_{\{s,t\}\{u,v\}}$, have value 1 iff the edge $\{u, v\}$ is used in the path between s and t. Binary *node path variables*, $y_{\{s,t\}u}$, have value 1 iff node u is used in the path. By definition, we let $y_{\{s,t\}s} = y_{\{s,t\}t} = 1$. For each edge, $\{u, v\}$, we have an integer *flow variable*, $f_{\{u,v\}}$, equal to the amount of bandwidth used on the edge. Finally, we have for each node pair, $\{s, t\}$, an integer *hop count variable*, $h_{\{s,t\}}$, equal to the number of edges used by the path between s and t. In addition, we will let z denote the current upper bound on the objective (1). Note that the CP model does not include any link weights. These will be computed by one of the LP relaxations, based on the paths defined by the CP model.

Below, we list the most important constraints that are used by our method. Filtering and detection of inconsistent assignments is done for each constraint. Note that even though some constraints are expressed as linear

inequalities, filtering for those constraints is done using CP-based interval reasoning methods, not by LP.

- Both edge path and node path variables are by themselves sufficient to model unique paths between all node pairs. However, in the CP part of the model, some constraints are more naturally expressed using the node path formulation, some using the edge path formulation. Therefore, we keep both, and use channelling constraints to keep the two representations consistent. For each edge path variable, the following constraint is imposed:

$$x_{\{s,t\}\{u,v\}} = 1 \leftrightarrow y_{\{s,t\}u} = 1 \wedge y_{\{s,t\}v} = 1 \wedge (\forall w.w \neq u \wedge w \neq v \rightarrow y_{\{u,v\}w} = 0) \tag{2}$$

- For each node pair $\{s,t\}$, the edge path variables $x_{\{s,t\}\{u,v\}}$ should define a unique simple path between s and t. This condition can be enforced partly by restricting the number of adjacent edges for each node. Nodes s and t have exactly one adjacent edge path variable set to one, and other nodes have either two adjacent edge path variables set to one (if they are included in the path), or none (if they are not included in the path). This condition will be enforced every time an edge path variable is set. However, it does not rule out disconnected cycles, so we will also need to enforce that no cycles are constructed. This is done whenever a path connecting s and t is finished, when we know that all edge path variables that are not included in the path should be set to zero.

- For each edge, we have a constraint, $f_{\{u,v\}} = \sum_{\{s,t\}\in D} b_{\{s,t\}} \cdot x_{\{s,t\}\{u,v\}}$, defining the flow on the edge. We also have a constraint, $f_{\{u,v\}} \leq z \cdot c_{\{u,v\}}$, involving the current upper bound on the objective.

- Each hop count variable is defined by a constraint: $h_{\{s,t\}} = \sum_{\{u,v\}\in E} x_{\{s,t\}\{u,v\}}$.

- Link weights are not included in the CP part of the model. Experiments we have made suggest that CP reasoning about explicit weights is not very effective. Instead we consider some other restrictions on possible sets of paths that unique shortest path routing implies. Consider any three nodes in the network, u, v, and w. The requirement that there should be a compatible weight system constrains the possible combinations of values for the three node path variables involving these nodes. Out of the eight possible assignments, only four are actually consistent. The constraint can be expressed compactly as $y_{\{u,v\}w} + y_{\{u,w\}v} + y_{\{v,w\}u} \leq 1$. We can make the same kind of consideration for all groups of four nodes. For four nodes, there are twelve node path variables that only involve these nodes, and out of the 4096 possible assignments, only 53 are consistent. We have implemented an efficient filtering method based on these conditions. Every time a node path variable is set it checks all possibilities to set other node path variables.

- Assume that the search has decided a path between a pair of nodes, and that the removal of the nodes in the path would divide the graph into two disconnected components. For a compatible weight system to exist, paths between nodes in the same component can never use nodes in other components, so corresponding node path variables can be set to zero.

The CP model still allows some infeasible solutions. The reason is that the constraints defined on node path variables for groups of three or four nodes do not guarantee that a path system has a compatible weight system. However, given a path system, we can formulate a linear program whose solution defines a set of compatible weights, or is infeasible if no compatible weights exist [2]. This problem is called the *inverse shortest paths problem*. The LP produces real-valued weights, but these can always be converted (through scaling up and rounding) to an integer weight system defining the same path system. Adding this LP makes our combined CP-LP model sound. The LP-model we use is a simplified version, which is possible to use since our 4-node constraints maintain a condition called suboptimality. We use w-variables for link weights, and l-variables for shortest path lengths. Note that the formulation depends on the current domains of edge path variables, denoted $dom(x_{\{s,t\}\{u,v\}})$, and will thus be different at different nodes.

$$w_{\{u,v\}} \geq 1 \qquad \forall \{u,v\} \in E \tag{3}$$

$$l_{\{u,v\}} \geq 0 \qquad \forall u \in V, \forall v \in V \tag{4}$$

$$l_{\{u,u\}} = 0 \qquad \forall u \in V \tag{5}$$

$$l_{\{u,v\}} + 1 \leq l_{\{u,w\}} + w_{\{w,v\}} \qquad \forall u \in V, \forall v \in V, \forall \{w,v\} \in E \text{ s.t. } dom(x_{\{u,v\}\{w,v\}}) = \{0\} \tag{6}$$

$$l_{\{u,v\}} \leq l_{\{u,w\}} + w_{\{w,v\}} \qquad \forall u \in V, \forall v \in V, \forall \{w,v\} \in E \text{ s.t. } dom(x_{\{u,v\}\{w,v\}}) = \{0,1\} \tag{7}$$

$$l_{\{u,v\}} = l_{\{u,w\}} + w_{\{w,v\}} \qquad \forall u \in V, \forall v \in V, \forall \{w,v\} \in E \text{ s.t. } dom(x_{\{u,v\}\{w,v\}}) = \{1\} \tag{8}$$

Our second LP relaxation is the well-known multicommodity flow problem, which can be modelled as a linear program [5]. We use the dual formulation which proves infeasibility if the minimum value of the objective function is negative. In (14), sup denotes the supremum (least upper bound on a set).

$$\pi_{\{u,v\}} \geq 0 \qquad \forall \{u,v\} \in E \tag{9}$$

$$\lambda_{\{u,v\}} \geq 0 \qquad \forall u \in V, \forall v \in V \tag{10}$$

$$\sum_{\{u,v\} \in E} \pi_{\{u,v\}} = 1 \tag{11}$$

$$\lambda_{\{u,u\}} = 0 \qquad \forall u \in V \tag{12}$$

$$\lambda_{\{u,v\}} \leq \lambda_{\{u,w\}} + \pi_{\{w,v\}} \qquad \forall u \in V, \forall v \in V, \forall \{w,v\} \in E \text{ s.t. } dom(x_{\{u,v\}\{v,w\}}) \neq \{0\} \tag{13}$$

$$minimize \sum_{\{u,v\} \in E} \sup(dom(f_{\{u,v\}})) \cdot \pi_{\{u,v\}} - \sum_{\{s,t\} \in D} b_{\{s,t\}} \cdot \lambda_{\{s,t\}} \tag{14}$$

4. Search

At each generated search node, we will start by running constraint propagation. If this does not detect inconsistency, we proceed by solving LP relaxations, taking into account the current domains of the variables. The multicommodity flow relaxation is solved at every search node, while the inverse shortest path relaxation is solved at each node with a full path system, to guarantee soundness, and for other nodes from time to time. The relaxations will possibly detect infeasibility, but will never infer any further domain reductions. Thus, the inference at each search node combines the strength of CP (the use of efficient filtering algorithms to prune away infeasible parts of the search space) with the strength of LP (global reasoning about the objective function and weight systems).

Our heuristic for choosing which variable to branch on at a given node is based on the intuition that fixing long paths restricts our remaining possibilities more than fixing short paths. When fixing a long path, we simultaneously fix many short subpaths. Our branching heuristic will choose the unfinished demand $\{s,t\}$, that seems to require the longest path, looking at the minimum remaining value in the domain for the hop count variables, breaking ties randomly. It will build a path for that demand, starting with one of the end nodes as the current node, and recursively choosing a next edge from the current node, backtracking at failure. Each branching decision either sets the edge path variable $x_{\{s,t\}\{u,v\}} = 1$ for the next edge $\{u,v\}$ from the current node u, or sets it to zero if the previous branching decision at the node was $x_{\{s,t\}\{u,v\}} = 1$ and that decision failed. When a path has been built successfully, a next demand to build a path for is chosen, based on the same criteria. Note that this is all done recursively, so if we fail to find a path for the next demand, through backtracking we will return and try different path choices for the previous demand.

A common way of solving optimization problems to optimality is by using standard branch-and-bound search. One problem here is that the search can spend much time in parts of the search tree where there is no improving solution. This can happen even with a current bound that is far from the optimal, which means that we get a low-quality solution if the search has to be interrupted because of time limitations. One possible way to handle these problems is to use random restarts [7], restarting the search after a limited number of backtracks

have been made. If the branching heuristic has a sufficiently random component, e.g., random tie-breaking with ties being sufficiently frequent, a restarted search is likely to explore different parts of the search tree. This works well for feasibility problems, if we are sure that there is a feasible solution, or for optimization problems, as long as we know that the current solution is not optimal. However, since exploration of the search tree is not made systematically, random restart search has problems showing infeasibility, or showing that a solution to an optimization problem cannot be improved.

To deal with this problem, we are using a search scheme running two searches in parallel. The searches will maintain common lower and upper bounds on the objective function. Initially the lower bound is set to zero and the upper bound to some large number. The *upper search* performs a standard branch-and-bound search, looking for a solution improving on the upper bound. Whenever such a solution is found, the upper bound is updated and the search continues. When the upper search has explored the entire search tree, the upper bound is the optimum value of the objective function. The *lower search* is introduced to deal with the problems with branch-and-bound search mentioned above. It runs a random restarts search, and at each restart it chooses a local upper bound, randomly from the interval between the lower bound and the upper bound, and looks for a solution improving this local upper bound. There are three possible outcomes of the lower search. If it finds an improving solution, we can update the common upper bound. The upper search will be immediately notified and use the new improved upper bound instead of the old one. If the lower search fails to find a solution without reaching its limit in number of backtracks, we can update the lower bound. If it finishes because it has used all its backtracks, we cannot infer anything about the bounds.

5. Experiments

We have generated random instances to evaluate our method. For each instance, we first generate a random network topology, using the Waxman model described in [17], with an average node degree of three, only accepting topologies that are connected and where every node has degree at least two. For every node pair we then set a demand with bandwidth requirement chosen randomly between 50 and 100. To set realistic link capacities, we randomly choose a path system, and set capacities at the level required to route the demands according to the path system with link utilization 1 for every link. The path system is chosen by randomly assigning real-valued link weights, between 1 and 3, and taking the shortest paths to define the path system (only accepted if unique). Note that the weights and the path system are not kept after the problem is generated.

We generated 100 instances for each network size between 20 and 30 nodes, and ran our method with a timeout of 30 minutes. The experiments were done on a Linux Intel 1.6 GHz, machine with 512 MB of RAM. For the implementation, we used the Java-based constraint solver JaCoP [8] and lp_solve for the linear relaxations. The results are given in figure 1. Figure 1a shows the percentage of runs that were solved to optimality within the 30 minute time limit. Figure 1b shows the average approximation guarantee, defined as the difference between the upper bound and the lower bound, divided by the lower bound. Figure 1c shows the average time used by runs that found optimal solutions.

Our complete method performs very well for networks of size 20. For larger sizes we start to observe timeouts, indicating that we would need significantly more time to solve networks of size 30 and above. When comparing this to related work, like [1], it should be noted that we are considering networks with demands between every node-pair, which means that we have a much higher number of demands to route for the same number of nodes.

6. Conclusions

We have presented a hybrid CP-LP method for the unique shortest path routing problem, and demonstrated its performance on a large set of generated test data. Our method combines CP with LP, using problem-specific

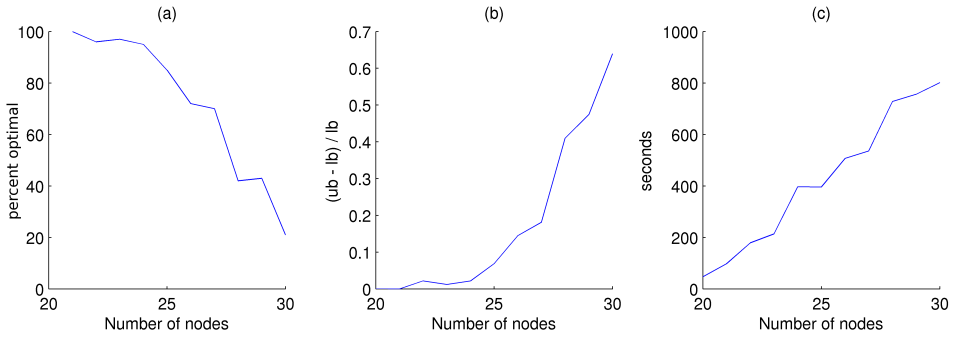

Figure 1: Test results for networks with 20 to 30 nodes, averaging over 100 problems for each size.

constraints based on necessary conditions for symmetric unique shortest path routing. The search method uses a version of random restarts which makes it possible to prove optimality, and give better approximation guarantees at timeout, due to tighter lower bounds.

Acknowledgements

The authors would like to thank Michał Pióro for introducing us to this problem, and for fruitful discussions during the work. This work was partly done during a visit to Cork Constraint Computation Centre by the first author, financed by the Swedish National Graduate School in Computer Science (CUGS).

References

[1] F. Ajili, R. Rodosek, and A. Eremin. A branch-price-and-propagate approach for optimizing igp weight setting subject to unique shortest paths. In H. Haddad, L. M. Liebrock, A. Omicini, and R. L. Wainwright, editors, *SAC*, pages 366–370. ACM, 2005.

[2] W. Ben-Ameur and E. Gourdin. Internet routing and related topology issues. *SIAM J. Discrete Math.*, 17(1):18–49, 2003.

[3] W. Ben-Ameur, N. Michel, B. Liau, and E. Gourdin. Routing strategies for ip networks. *Telektronikk Magazine*, (2/3):145–158, 2001.

[4] A. Bley. A lagrangian approach for integrated network design and routing in ip networks. In *INOC*, pages 107–113, 2003.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* The MIT Press, second edition, 2001.

[6] C. Frei and B. Faltings. Resource allocation and constraint satisfaction techniques. In *Principles and Practice of Constraint Programming*, pages 204–218, 1999.

[7] C. P. Gomes, B. Selman, and H. A. Kautz. Boosting combinatorial search through randomization. In *AAAI/IAAI*, pages 431–437, 1998.

[8] K. Kuchcinski. Constraints-driven scheduling and resource assignment. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 8(3):355–383, July 2003.

[9] D. Lucraft, A. Eremin, and F. Ajili. Enforcing path uniqueness in internet routing. In H. Haddad, editor, *SAC*, pages 399–403. ACM, 2006.

[10] W. Ouaja and B. Richards. A hybrid multicommodity routing algorithm for traffic engineering. *Networks*, 43(3):125–140, 2004.

[11] C. Le Pape, L. Perron, J.-C. Régin, and P. Shaw. Robust and parallel solving of a network design problem. In P. Van Hentenryck, editor, *CP*, volume 2470 of *Lecture Notes in Computer Science*, pages 633–648. Springer, 2002.

[12] M. P. Pettersson, R. Szymanek, and K. Kuchcinski. A cp-lp approach to network management in ospf routing. In *Proceedings of the 2007 ACM Symposium on Applied Computing (SAC), Seoul, Korea, March 11-15, 2007*, page to appear. ACM, 2007.

[13] M. Pióro and D. Medhi. *Routing, Flow, and Capacity Design in Communication and Computer Networks.* Morgan Kaufmann, 2004.

[14] M. Pióro, Á. Szentesi, J. Harmatos, A. Jüttner, P. Gajowniczek, and S. Kozdrowski. On open shortest path first related network optimisation problems. *Perform. Eval.*, 48(1/4):201–223, 2002.

[15] J.-C. Régin. Modeling problems in constraint programming. (tutorial). In *CP'04, Toronto, Canada, Sept 2004*.

[16] L. Ros, T. Creemers, E. Tourouta, and J. Riera. A global constraint model for integrated routeing and scheduling on a transmission network. In *Proc. of the 7th International Conference on Information Networks, Systems and Technologies.*, pages 40–47, 2001.

[17] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. How to model an internetwork. In *INFOCOM*, pages 594–602, 1996.