A Hybrid solver for optimal routing of bandwidth-guaranteed traffic

Wided Ouaja, Barry Richards
IC-Parc, Imperial College London
Address : IC-Parc, Imperial College London, London SW7 2AZ, UK
Phone: +44 20759484 59/29, fax: +44 2075948432
Email: {wo1,ebr}@icparc.ic.ac.uk

Abstract

The dramatic explosion of Internet services requires that traffic flows across the network meet certain QoS parameters while efficiently utilizing network resources. MPLS, for example, provides the facilities to achieve this through explicit routing. Finding optimal paths for all the traffic demands which satisfy QoS requirements is a non-trivial task. Indeed, guaranteeing just bandwidth is known to be $\mathcal{NP}$-hard. In this paper we propose a new algorithm for solving this problem, which is a hybrid that tightly integrates Lagrangian optimization and Constraint Programming search. We evaluate its performance on a set of benchmark tests, based on a large commercial backbone topology. The tests involve demand sets of varying size, mostly between 100 to 600 demands. We compare the results with those achieved by several other well-known algorithms, some complete and some heuristic. This reveals that the hybrid algorithm typically yields the most informative results in the most effective way.

keywords: Routing, Traffic Engineering, MPLS, Operations Research, Constraint Programming.

1 Introduction

The dramatic explosion of Internet services requires that traffic flows across the network meet certain QoS parameters while efficiently utilizing network resources. MPLS, for example, provides the facilities to achieve this through explicit routing. The task, given a set of traffic demands, is to find a path for each demand such that they jointly optimize pre-defined criteria and satisfy a number of constraints, such as bandwidth availability [2, 6]. This is referred to generally as constraint-based routing (CR) [2, 6].

CR can be performed either off-line or on-line [20]. In the on-line case, traffic demands are placed dynamically, often on a first-come-first-serve basis. Routes are calculated one-by-one by some appropriate algorithm, *e.g.*, constrained shortest path first (CSPF) [6], shortest-distance path (SDP) [13], widest-shortest path (WSP) [13], or shortest-widest path (SWP) [13, 17]. As individual routes are computed *greedily*, meeting QoS for future demands can be a major problem. Some feel that this might be addressed with over-provisioning. Our results show, however, there can be problems even in very well provisioned networks.

An alternative strategy is to compute explicit routes for a *set* of traffic demands, with a view to optimizing the *global* impact of their placement on the network. This can be done in a centralized off-line manner [18, 20]. However, finding an *optimal* constrained routing of *multiple* demands can be difficult; indeed, the problem is $\mathcal{NP}$-hard with bandwidth as the only resource constraint [7]. This optimization problem, which we refer to as the traffic placement (TP) problem, is summarized as follows:

- *given* a network composed of nodes and links, each link with a maximum capacity, a set of traffic demands defined by their source/destination nodes and required bandwidth values respectively,
- *find* a *single* path for each demand such that all demands are satisfied, no link capacity is exceeded, while optimizing a pre-defined objective function.

The choice of objective function is crucial, but debatable. Based on a previous evaluation [14], we found that minimizing the average bandwidth utilization over all links yields the best balance between hop counts and efficient bandwidth utilization. We shall use this objective function here. The algorithm is, however, applicable to any linear function.

As for the choice of algorithm, a new paradigm is emerging in combinatorial optimization where the aim is to exploit *hybridizations* of Constraint Programming (CP) and Operations Research (OR) [12]. CP is distinctive in tackling the satisfaction of constraints by using local consistency techniques [16]. It assists search by *inferring*

new information about the decision variables (*e.g.*, tighter bounds, fixed variables). OR, on the other hand, offers powerful special purpose techniques for optimization of certain problem classes (*e.g.*, linear). The respective strengths of CP and OR (*satisfaction* vs *optimization*) suggest that they can be integrated to create efficient *hybrid* solvers for combinatorial optimization. The benefits have been seen in various combinatorial problems [12].

In this paper, we explore the benefits of a CP-OR hybridization for the TP problem. We present a *new complete* hybrid solver which tightly integrates Lagrangian relaxation and CP search. It is designed to prove optimality/infeasiblity, and since it is complete, it finds a solution if one exists, and proves infeasibility otherwise. One of its most distinctive features, mainly owing to CP, is that it finds solutions (to solvable problems) early in the search. These are often close to the optimum, and Lagrangian relaxation serves to indicate how close. As a result, it is sometimes not worth seeking the optimal solution, since the first one is sufficiently near.

The effectiveness of the proposed hybrid algorithm is measured in an experimental study, based on a major US backbone. We generate a large number of different sets of traffic demands to be placed on the network; these are designed to simulate "real" traffic matrices. We then compare the performance of the hybrid algorithm with several other algorithms, *viz.* CPLEX [11] for Mixed Integer Programming (MIP), Wang's method (LPF-RR) [18], and four shortest path algorithms (CSPF, SDP, WSP, and SWP). The results reveal the benefit that can be realized through explicit routing.

2 Problem Formulation

We model the network as a directed graph $G = (V, E)$, where V and E are the sets of nodes and directed links respectively. Bandwidth is modeled as a non-negative number measured in kbps. Each link (i, j) has a capacity b_{ij}. For the sake of convenience, and without loss of generality, we assume that the initial load of each link is zero. Let K be the given set of traffic demands. A demand $k \in K$ is defined by a tuple (s_k, t_k, d_k), where s_k, t_k, and d_k denote respectively the source node, destination node, and required bandwidth of demand k.

For each edge (i, j) and demand $k \in K$, we define a variable X_{ij}^k to represent the proportion of k's bandwidth that crosses (i, j). Since no flow splitting is allowed, each X_{ij}^k must take the value zero or one. We formulate the TP problem as an integer linear multicommodity flow problem using the following node-arc model (P):

$$z^* = \min \frac{1}{|E|} \sum_{k \in K} \sum_{(i,j) \in E} \frac{d_k}{b_{ij}} X_{ij}^k \tag{1}$$

$$\sum_{j:(i,j) \in E} X_{ij}^k - \sum_{j:(j,i) \in E} X_{ji}^k = \begin{cases} 1 & i = s_k \\ -1 & i = t_k \\ 0 & i \neq s_k, t_k \end{cases} \quad \forall\ i \in V,\ k \in K \tag{2}$$

$$\sum_{k \in K} d_k X_{ij}^k \leq b_{ij} \quad \forall (i, j) \in E \tag{3}$$

$$0 \leq X_{ij}^k \leq 1,\ \text{integer}(X_{ij}^k) \quad \forall\ (i, j) \in E,\ k \in K \tag{4}$$

The objective function (1) minimizes the total cost of accommodating all demands across the network (*i.e.*, the average link utilization). The flow constraints (2) model the flow of each demand $k \in K$ in the network. The capacity constraints (3) tie together the demands by restricting the total flow $\sum_{k \in K} d_k X_{ij}^k$ of all demands on each edge (i, j) to at most b_{ij}. In the constraints (4), all variables X_{ij}^k are declared as 0-1 decision variables to ensure that each traffic demand follows one path.

The complicating capacity constraints (3) make the TP problem $\mathcal{NP}$-hard [7]. When relaxed, the problem decomposes into $|K|$ *separate* shortest path subproblems, that can be efficiently solved using, *e.g.*, linear programming or Dijkstra's algorithm [1]. This observation lies at the heart of our approach, which is described next.

Related work on integer multicommodity flow problems can be found in [4, 5]. They do not, however, consider large-scale instances of the size we address.

3 Proposed hybrid algorithm

3.1 Lagrangian relaxation

The above remarks motivate the choice to apply Lagrangian relaxation [1] (LR) on P by dualizing the capacity constraints (3) into the objective function with associated vector of nonnegative *Lagrangian multipliers* λ. We refer to the resulting problem, P_λ, as the *Lagrangian subproblem* of P. For any fixed $\lambda \geq 0$, P_λ decomposes into $|K|$ separate *"easy"* shortest-path problems. Its optimal objective value is a lower bound on z^*. To obtain the sharpest bound L^* along with the optimal multipliers λ^*, we shall solve the *Lagrangian dual problem*, using, *e.g.*, subgradient

optimization (SG) [1, 8]. SG is a simple procedure that iteratively updates λ and solves the corresponding P_λ. SG stops when it converges, a specified limit of iterations is reached, or the stepsize becomes sufficiently small. We implement an enhanced SG variant similar to the one in [9]. Moreover, we test certain conditions *a priori* to reduce the number of unnecessary re-routings.

Unfortunately, pure LR is generally not sufficient to solve the TP problem. This is due to i) the existence of a duality gap (*i.e.*, $z^* - L^* > 0$) and ii) the inconsistency of the relaxed solution.

In case (i), since P is an integer problem, a duality gap exists in general [1]. Moreover, because P_λ has the integrality property, L^* equals the optimal objective value z° of the linear relaxation of P, thus the gap is $z^* - z^\circ$ [1].

As for case (ii), an LR solution is often not *primal* feasible (wrt P), because it might violate some capacity constraints. However, it tends to be near-feasible given the fact that LR encapsulates a resource view in the objective function. We designed *efficient* primal heuristics to adjust infeasible solutions into feasible ones.

Recent results in [3, 15] show that for *continuous* problems, a convex combination of all solutions generated by SG can be primal feasible after a finite number of iterations. In our *integer* context, if such a solution is found, it provides an upper bound on z°, thus on L^* and hence contributes to better convergence (given that it is crucial for computing the stepsize). We exploit this result in our approach.

3.2 Hybridization with CP

To close the duality gap and find an optimal primal solution, we shall combine LR with Constraint (Logic) Programming *inference* within branch-and-bound tree search. The resulting hybrid algorithm is referred to by HLR. It is briefly described following this general overview of CP inference.

3.2.1 Overview of CP inference

CP inference is supported by a finite domain constraint solver that relies on efficient local consistency techniques [16]. The constraints accumulated in the CP *constraint store* communicate with each other through their variable domains. Basically, inference on the constraints linking each variable removes domain values which cannot be part of any solution. The reduction of a variable's domain *triggers* the examination of other constraints, which in turn may reduce other domains. This recursive process, called *constraint propagation*, terminates when no domain can be further reduced or a domain becomes empty. In the latter case, we know that no solution satisfying the constraints exists, thus a failure is raised. Propagation is *automatically* triggered as soon as a constraint is added to the store or a variable's bound changes. Since all variables in our model are binary, a reduction of a domain's variable results in the assignment to the other domain value. Therefore, CP inference can result in fixing some variables to zero or one, which corresponds to forbidding or imposing certain edges for some demands.

3.2.2 Algorithm description

Before starting search, we attempt to find a feasible placement using CSPF. If no feasible one is found, we setup cutset constraints based on the max-flow-min-cut theorem [1] with the aim of detecting infeasibility. The minimal cutset is computed for each demand using Ford-Fulkerson max-flow-min-cut algorithm [1]. The cutset constraints assert that the aggregate bandwidth of all demands necessarily crossing the same cutset should be no greater than the total bandwidth of all links in the cutset. If violated, the problem is proved infeasible and no search is needed. Otherwise, HLR performs a depth-first traversal of a binary branch-and-bound search tree. Attached to every tree node are a Lagrangian subproblem P_λ and a constraint store CS sharing the problem variables; they are incrementally updated during search.

At the root node, CS consists of the flow and domain constraints ((2), (4)). Propagation of these constraints forbids for each demand k, links that cannot be part of any path from s_k to t_k. Analogously, it imposes for every demand k, links that are contained in any path from s_k to t_k. Propagation is enhanced by constraining the total outflow and inflow for every node $i \in V$ and demand $k \in K$ to be less or equal than one. Indeed, once an edge (i, j) is enforced for a demand k (*i.e.*, $X_{ij}^k = 1$), inference ensures that all other outgoing edges from i and incoming edges to j are forbidden from the kth route (*i.e.*, their corresponding flow variables are fixed to 0).

At every node, the LR solver operates on the local subproblem by performing a number of SG iterations. After each SG iteration, new constraints based on capacity violations and reduced costs are dynamically discovered and added to CS. Propagation is then performed on CS to (1) infer new fixed variables or (2) detect a failure. In case (1) the next iteration SG tackles a reduced easier problem, whereas in case (2) the node is pruned immediately; no need to execute further local SG iterations.

The generated constraints based on detected capacity violations are cover cuts [19] which assert that demands crossing an overloaded link can not simultaneously use that link in any solution. These are propagated and also dualized in the next-node subproblem to strengthen the relaxation.

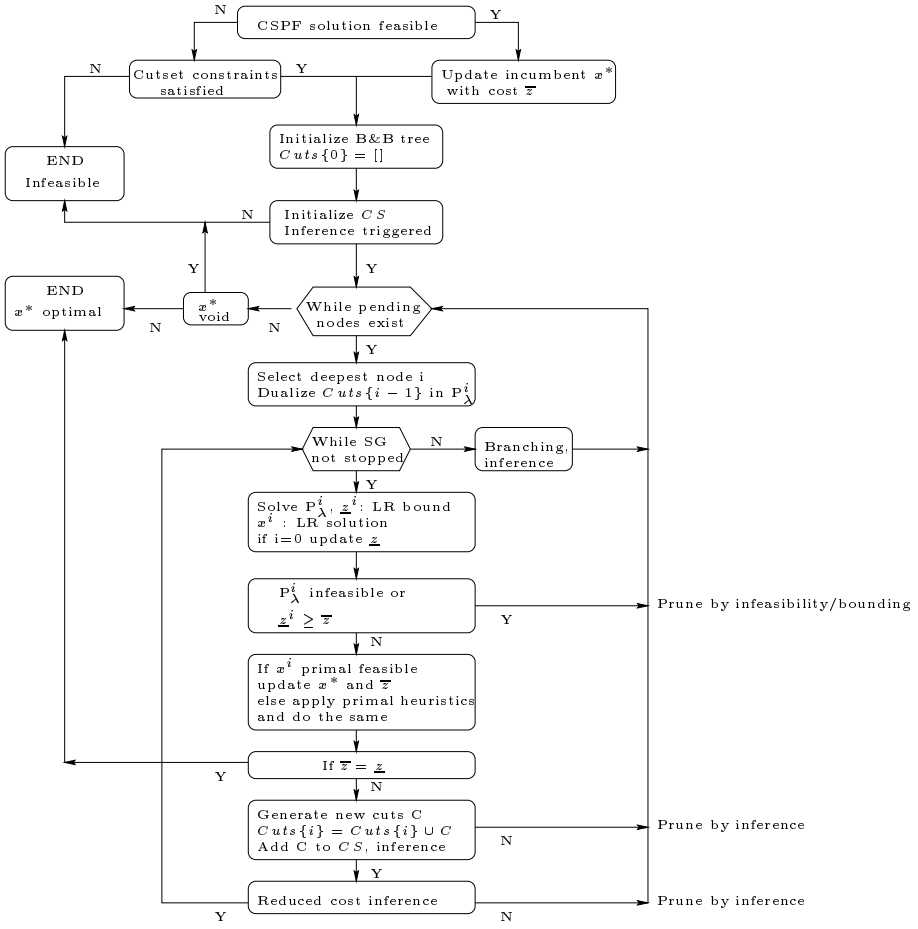

Figure 1: Algorithm flow chart.

As for reduced costs, they are deployed to remove values that can not participate in the optimal solution. Although this technique is widely used within the OR community, typically known as variable fixing, it shows a more substantial benefit when applied frequently within our CP context. It often results in a large number of variables being fixed explicitly or due to propagation.

An efficient primal heuristic is deployed following each SG iteration in an attempt to adjust an LR inconsistent solution into a feasible one. It basically re-routes, one-by-one, a subset of demands crossing the overloaded links, using an appropriate link metric.

When LR optimization stops, a Lagrangian solution is returned. This is a total assignment to all variables. It might violate some capacity constraints. Guided by this solution, the algorithm heuristically selects a branching decision $X^k_{ij} = 0$ and adds it to P_λ and CS. On backtracking, the decision is revoked and its negation $X^k_{ij} = 1$ is added instead. Posting the search decision triggers inference.

The main aspect that distinguishes HLR from traditional LR-based branch-and-bound algorithm is constraint propagation. This overlaps with LR optimization and search. It can fix a potentially large number of variables. This aids the LR solver in generating good solutions faster, and assists search since only a reduced number of decisions need to be examined. Additionally, propagation can detect inconsistency, and hence leads to early pruning of nodes that yield no feasible/optimal solution. Consequently, a smaller search tree is explored.

A flow chart of the algorithm is depicted in Figure 1. Note that search ends *without* having to explore pending nodes, when the incumbent cost ($\overline{z}$) equals a valid global lower bound ($\underline{z}$), here the best LR bound found at the root.

More details of the algorithm can be found in [14].

(a) Test-A

Alg	Opt	Inf	Sol	Resol
HLR	66.78	24.78	72.78	97.56
MIP	72.12	25.98	73.00	98.89
LPF-RR	29.67	25.45	45.45	70.90
CSPF	-	-	69.12	69.12
SDP	-	-	69.34	69.34
SWP	-	-	54.12	54.12
WSP	-	-	69.89	69.89

(b) Test-B

Alg	Opt	Inf	Sol	Resol
HLR	30.22	7.78	84.77	92.55
MIP	35.89	11.89	40.67	52.56
LPF-RR	11.22	11.89	33.11	45.00
CSPF	-	-	70.89	70.89
SDP	-	-	83.44	83.44
SWP	-	-	22.67	22.67
WSP	-	-	82.56	82.56

Table 1: Results over (a) Test-A and (b) Test-B.

4 Experimental study

In this section we compare the performance of HLR with other algorithms, *viz.* CPLEX [11], a variant of LPF-RR (adapted to use average link utilization instead of maximum link utilization as the objective function), and four shortest path algorithms (CSPF, SDP, SWP and WSP). LPF-RR solves the continuous version of the problem using Simplex then heuristically routes split demands one-by-one. For the shortest-path methods, we shall adopt the bin-backing heuristic of routing demands in decreasing order of bandwidth size. This is widely felt to be effective in delivering the best results. For CSPF we use a link metric that is inversely proportional to the link capacity.

With the exception of CPLEX (which is a commercial package), all the algorithms are implemented in the Constraint Logic Programming language ECL^{i}PS^e [10]. ECL^{i}PS^e is equipped with several constraint solvers, one of which is over finite domains. It also embeds an external linear solver, CPLEX, which we use to solve the Lagrangian subproblems.

4.1 Test sets

The test data were created using a generator provided by an industrial partner. They are based on a major US backbone, composed of 88 nodes and 336 directed links; the network is very generously provisioned. The generated traffic is designed to simulate "real" traffic matrices.

We generate two different benchmarks, Test-A and Test-B, with 900 test cases each. The test cases in Test-A range in size typically between 100 and 200 demands. The second benchmark, Test-B, consists of larger test cases, with the number of demands exceeding the number of links. Here the number of demands is evenly distributed over the range [300, 600].

4.2 Experimental results

The experiments were run on Pentium II 450 MHz processors, and for each test case there was a timeout of 1000 CPU seconds. The optimality tolerance was set to 0.0001. The results are summarized in Table 1, where the algorithms are compared on four dimensions, *viz.* on the percentage of cases in each test where the algorithms (1) proved optimality, (2) proved infeasibility, (3) found a solution (not necessarily the optimal one), and (4) resolved the problem, *i.e.*, either found a solution or proved infeasibility. These provide an interesting perspective both on the algorithms and on the TP problem itself. We discuss the results on each dimension, and provide a composite view as it emerges. This is where the insight lies.

4.2.1 Cases proved optimal

The second column in Table 1-(a) and -(b) shows that on both test benchmarks MIP finds roughly 6% more optimal solutions than does HLR (within the timeout). Note, however, that the performance of both algorithms drops sharply on the larger cases in Test-B.

Perhaps surprisingly, LPF-RR solves *some* tests optimally, although it is incomplete. In these cases, LPF-RR solutions are optimal because they coincide with the LP relaxed solutions which happened to be integral. However, LPF-RR is clearly not competitive with MIP or HLR. As for the four shortest-path algorithms, they are not designed to prove optimality and hence, a comparison here is irrelevant.

One can conjecture from these results that proving optimality is not in general a realistic goal.

4.2.2 Cases proved infeasible

The third column in Table 1-(a) and -(b) shows that MIP is slightly better than HLR in proving infeasibility. The difference is 1.1% on Test-A and 4.1% on Test-B.

The performance of LPF-RR is virtually the same as MIP on both test benchmarks. This indicates that for most instances proved infeasible, both the continuous and discrete variants of the TP problem are infeasible. As for the shortest-path algorithms, they are of course inappropriate to prove infeasibility.

The surprising result is the percentage of cases that are infeasible, especially since the network is so generously provisioned. Nearly 26% are infeasible in Test-A, and nearly 12% in Test-B.

4.2.3 Cases solved

Column four in Table 1-(a) and -(b) shows the total number of cases solved. On Test-A HLR solved almost as many cases as MIP (73%). LPF-RR, in contrast, found solutions in only 45.5% of the cases.

The picture changes dramatically on Test-B. Here MIP solved only 40.6% of the cases, compared to 84.% for HLR. On this dimension there is an enormous difference between MIP and HLR. Clearly, MIP is not well-suited to solving large-scale problems, hence the need for decomposition. The difference is even larger in the case of LPF-RR, which solved just over 33% of the cases.

Significantly, SDP and WSP approximated the number of cases solved by HLR. CSPF was much less effective, and SWP non-competitive. This raises an interesting question, what is the gain in using HLR over SDP or WSP? We discuss this below.

4.2.4 Total resolved cases

We now turn to the composite view of the results, captured in the last column of Table 1-(a) and -(b). Here we look at the number of cases resolved by each algorithm, i.e. those cases where the algorithm found an optimal or non-optimal solution, or proved infeasibility. This measures one aspect of the overall performance.

On Test-A MIP resolved 98.9% of the cases, compared to 97.6% for HLR. If we take solution quality to one side, there seems little to choose between these two algorithms. But Test-B reveals a significant "gap". MIP resolved only 52.5% of these cases, leaving a total of 47.5% unresolved. In contrast, HLR resolved 92.6% of the cases, timing out (without a solution) in only 7.4%.

As for SDP, WSP and CSPF, they failed to resolve just over 30% of the cases in Test-A. Most of these are infeasible (slightly less than 26% of the total), which leaves about 3% of the known solvable cases unsolved. On Test-B SDP and WSP left roughly 17% of the cases unresolved, about 5% might have a solution. CSPF left a lot more unresolved cases, about 39%.

So what does this mean? SDP and WSP approximate HLR in finding solutions on the two benchmark tests (when there are solutions); here the differences are not large. What is surprising is the percentage of known infeasible cases, nearly 26% on Test-A and 12% on Test-B. This seems very large given that the test network was thought to be well over-provisioned. SDP and WSP or any similar shortest path algorithm, cannot tell us anything about these cases. In effect, the cases they leave unresolved (30% on Test-A and 17% on Test-B) are undifferentiated; they might or might not all have a solution. This leaves a big question when routing with SDP or WSP: what percentage of the unplaced demands have a solution? This will vary widely depending upon the topology, and the number and size of the traffic demands. This can present a real challenge to keeping one's customers happy.

4.2.5 First solution found

We observed that on average the quality of the first solution found by HLR differed only slightly from the best solution found, or indeed from the optimal one. Here it is important to note that HLR provides an estimate of how far each solution found lies from the optimal. This is measured by the duality gap, which is the difference between the best global lower bound and the solution cost. Recall that the duality gap is in fact an overestimate of a solution's distance from the optimal.

In Test-A the average duality gap for first solutions is very small (0.015%) and the worst case is only half of one percent (0.52%). In Test-B, the average gap is slightly larger (0.3%), with the worst case at 2.3%. The message here seems clear. The first solutions found by HLR are near-optimal, and seeking to improve on the first solution will typically yield a very small return. Hence, if we increase the optimality tolerance, many more solutions would be considered optimal, thus narrowing the gap between the number of optimal cases and number of solved cases.

Moreover, the first solutions are found quickly. In Test-A it took on average 12 CPU secs to find the first solution, and in Test-B 20 secs. Curiously, the infeasible cases were proved on both tests in roughly the same time, 13 secs on Test-A and 14 on Test-B.

4.3 Summary

The experiments on Test-A and Test-B show the benefits that a CP-LR hybridization can achieve on the TP problem. HLR typically yields the most informative results in the most effective way. It resolved nearly 98% of the cases in Test-A, taking on average 13 CPU secs to do so. In Test-B it resolved just under 93% of the cases, taking on average 20 secs to find the first solution and 14 secs to prove infeasibility.

On Test-B MIP left almost half the cases unresolved, although it broadly resolved as many cases as did HLR on Test-A. SDP and WSP did much better on Test-B, leaving around 17% unresolved. Note, however, that it is impossible to estimate the quality of the solutions they found, unlike HLR. Leaving 17% of the cases unresolved leaves some important open questions.

Finally, these results seem particularly striking in view of the fact that the test network is very generously provisioned, some may think even over-provisioned. The average link utilization on all test cases is less than 26%. One might not expect there to be so many infeasible cases in this situation, or indeed so many unresolved cases. The situation can only deteriorate when the utilization rises. When this happens, HLR will become even more informative.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, "Network flows: theory, algorithms, and applications". Prentice-Hall, 1993.

[2] D. Awduche and K. Kompella, "Notes on path computation in constraint-based routing". draft-kompella-te-pathcomp-00.txt, 2000.

[3] F. Barahona and R. Anbil, "The volume algorithm: producing primal solutions with a subgradient method". Mathematical Programming, 385-399, 2000.

[4] C. Barnhart, C. A. Hane, and P. H. Vance, "Integer multicommodity flow problems". Proceedings of IPCO, 58-71, 1996.

[5] C. Barnhart, C. A. Hane, and P. H. Vance, "Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems". Operations Research, 48, 318-326, 2000.

[6] B. Davie and Y. Rekhter, "MPLS technonlogy and applications". Morgan Kaufmann Publishers, 2000.

[7] M. Garey and D. Johnson, "Computers and intractability". Freeman and Company, 1979.

[8] M. Held, P. Wolfe, and H. P. Crowder, "Validation of subgradient optimization". Mathematical Programming, 6, 62–88, 1974.

[9] K. Holmberg, D. Yuan, "A Lagrangian heuristic based branch-and-bound for the capacitated network design problem". Research Report LiTH-MAT-R-1996-23, Linkoping Institute of Technology, Sweden, 1996.

[10] IC-PARC, "ECL^{i}PS^e user manual". IC-PARC, Imperial College London, 2000.

[11] ILOG, "CPLEX Optimizer 6.5 user manual". http://www.cplex.com, 1999.

[12] N. Jussein and F. Laburthe, editors, "Proceedings of CP-AI-OR". Le Croisic, France, 2002.

[13] Q. Ma and P. Steenkiste, "On path selection for traffic with bandwidth guarantees". Proceedings of IEEE ICNP, 191-202, 1997.

[14] W. Ouaja and B. Richards, "A Hybrid multicommodity routing algorithm for traffic engineering". Technical Report IC-PARC-03-3, May 2003. Submitted for publication.

[15] H. D. Sherali and G. Choi, "Recovery of primal solutions when using subgradient optimization methods to solve lagrangian duals of linear programs". OR Letters, 19:105–113, 1996.

[16] E. Tsang, "Foundations of constraint satisfaction". Academic Press, 1993.

[17] Z. Wang and J. Crowcroft, "QoS routing for supporting multimedia applications". IEEE JSAC, 14(7), 1228–1234, 1996.

[18] Y. Wang and Z. Wang, "Explicit routing algorithms for internet traffic engineering". Proceedings of IEEE IC3N, 582–588, 1999.

[19] L. A. Wolsey, "Integer programming". Wiley Interscience, 1998.

[20] X. Xiao et al., "Traffic engineering with MPLS in the internet". IEEE Network magazine, 28-33, March 2000.