A Local Search/Constraint Propagation Hybrid for a Network Routing Problem

Jonathan Lever

IC-Parc
Imperial College
London, UK
SW7 2AZ
j.lever@imperial.ac.uk

Abstract

This paper presents a hybrid algorithm that combines local search and constraint programming techniques to solve a network routing problem. The problem considered is that of routing traffic demands from a set of requests over a network with limited capacity so as to minimise the cost of any unrouted demands. The hybridisation is twofold: pure local search is used to find a good cost bound for a subsequent branch-and-bound optimisation phase, with local search again applied at the nodes of the branch-and-bound search tree. Constraint propagation occurs in the search tree to reduce the domains of the decision variables, using a set of constraints that are independent of the action of local search at the nodes. In contrast to previous constraint programming/local search hybridisations, here local search is used to satisfy the *hard* problem constraints, while optimisation is handled in the framework of constraint programming. The resulting algorithm is incomplete, but is shown to be compare favourably with a complete approach to this problem.

Introduction

This paper presents a novel hybridisation of local search and constraint programming techniques that was developed while studying traffic placement problems in network routing. The specific optimisation problem we consider has the following form: assuming a network containing already routed traffic demands (a demand being a requirement for a connection of some specified bandwidth from a source to a destination), and a set of new demands to be routed, find routes for the new demands such that the “cost” of the new demands which cannot be admitted onto the network is minimised. In particular, we are interested in tightly-constrained or over-constrained problems, where it is difficult or impossible to route all the requested demands without exceeding link capacities. If a solution is found which routes all demands without exceeding link capacities, the algorithm does not perform any additional optimisation on, for example worst-case or average case link utilisation.

Apart from its application to the specific problem considered here, the algorithm is interesting in that it proposes a new way to hybridise local search and constraint propagation techniques. Previously, the relative strengths of local search in the area of optimisation with few hard constraints, and constraint-based techniques for problems in which hard constraints can provide significant reductions of the search-space through propagation, has suggested hybrid algorithms in which local search handles the optimisation component, while constraint propagation provides pruning using the hard problem constraints. Examples can be found in (Casseau and Laburthe, 1999), (Genreau and Pesant, 1999), (Jussien and Lhomme, 2000), (Kamarainen and El-Sakkout, 2002), (Schaerf, 1997), (Shaw, 1998) and (Zhang and Zhang, 1996).

In the hybrid form presented here, satisfaction of the hard constraints – the capacity of links for traffic being routed – is addressed by the local search component, while constraint propagation is applied in a branch-and-bound search tree that aims to minimise the cost of unrouted demands. The local search component is, by its nature, not a complete algorithm for solving the hard constraints, and this incompleteness is inherited by the hybrid algorithm. However, the complex nature of the application itself precludes the development of a complete algorithm that is effective for problems of significant scale.

The paper is organised as follows: after introducing the problem, discussing possible approaches and motivating the strategy adopted here, we outline the three phases of the algorithm, and the local search procedure for restoration of consistency. Before concluding, we report experimental results on data derived from a large-scale real network.

The Demand Admission Problem

The demand admission problem considered in this paper can be stated as follows: given a network made up of nodes and links, each link having an available capacity, and a set of point-to-point traffic demands, each with a bandwidth requirement and a cost incurred if that demand is not placed on the network, find routes for demands that satisfy the link capacities, while minimising the cost of any unrouted demands.

For the purposes of this presentation, we assume that demands already routed on the network have their paths fixed, and their bandwidth requirements have been taken into account when specifying the available link capacities. In principle, there is no problem in allowing existing demands to be re-routable and/or "droppable" i.e. removable from the network in favour of new demands. In practice, this increases the problem scale, which is essentially determined by two factors: the number of demands whose paths are moveable, and the number of demands that need not be routed. Also, we assume that a *single* path is required for each demand.

The algorithm we present is also able to solve problems in which demands have a maximum propagation delay, which the sum of the propagation delays of the links a path for that demand must not exceed. However, the implementation of this feature is not core to the algorithm and to simplify the presentation we will not discuss it further in this paper.

Solution Strategies

At the simplest level, heuristic algorithms based on the use of the constrained shortest path first (CSPF) algorithm (Lee, Hluchyi, and Humblet 1995) can be applied. Demands are considered in a certain order, and an attempt is made to route each one, given the current available link capacities. If this is successful, the demand is placed on the network and its bandwidth requirement subtracted from the available capacity of the links on its path. If not, the demand is rejected. Some improvements may be gained by considering various different orderings of the demands, but the approach remains a heuristic one and does not perform search in any deep way.

There have been a number of papers presenting algorithms for a related problem - that of optimising network utilisation under the assumption that all demands are placed. Methods proposed for this problem often apply mathematical programming techniques based on multi-commodity flow formulations, and include (Barnhart, Hane and Vance, 2000) and (Wang and Wang, 1999). When moving to over-constrained demand admission problems, another level of choices is added to the demand routing problem, namely, which demands are placed and which are not. Introducing choices at this level presents challenges for mathematical programming models from the point of view of scalability. An alternative strategy in which it is first assumed that all demands are placed (allowing capacity violations), then one algorithm is applied to reduce capacity violations and a second applied to remove demands so as to satisfy the capacity constraints, would scale better. However, this imposes a separation between routing and demand selection which can lead to solutions that are significantly sub-optimal.

In (Liatsos, Novello, and El-Sakkout, 2003), a technique is presented in which a search tree is constructed, using Mixed Integer Programming to reroute a single demand at the nodes of the tree. Finite domain constraints based on minimal cuts for subsets of demands are used to propagate information concerning the routing of demands over links. It is assumed that all demands are placed, and the optimisation component is on the maximum utilised link in the network. An unpublished variant of their technique that is able to make choices about which demands are placed in over-constrained problems has been developed, and is used for comparison in the experimental results section of this paper.

The strategy adopted in the algorithm we propose here is as follows:

1) obtain an initial solution, using CSPF methods
2) improve the initial solution incrementally by adding in demands not yet placed and using local search to restore consistency with respect to the link capacity constraints
3) execute a hybrid branch-and-bound tree search in which alternative combinations of placed and unplaced demands are considered, with local search again used to restore consistency

While investigating the problem, local search proved to be an effective technique for restoring consistency in a "slightly" inconsistent network by re-routing demands. The degree of inconsistency introduced by adding a single demand is generally small, and a short run of local search can often restore consistency when this is possible. Capacity violations are handled through the optimisation function. Although penalisation of hard constraint violations in this way can be problematic when it mixes constraint violation penalties with "real" optimisation criteria, here, the hard constraint violations completely determine the solution cost.

The scope of decisions handled by local search is restricted to routing decisions - while inclusion of the potential to remove, as well as reroute, placed demands in the neighbourhood operator of local search is a possibility, it did not appear a promising direction. However, as the local search procedure is applied many times within the context of the incremental improvement and tree search phases – which address the selection of placed demands - the two levels of decision-making remain closely integrated in the algorithm.

The Initial CSPF Solution

The initial solution is constructed using the CSPF method, applied to each demand in turn, after ordering them in order of cost, highest first. The path must satisfy the bandwidth requirement of the demand without violating available link capacities. The metric used is "inverse free capacity" i.e. the reciprocal of the free capacity remaining on the link if the demand were routed over it. The metric value of a path defined as the sum of the metrics of its links. This metric was chosen as it tends to give good average-case link utilisation, which benefits later phases of the algorithm.

If a valid path is found for a demand, it is placed on the network i.e. its required bandwidth is subtracted from the free capacity of the links in its path. If no valid path is found, the demand remains unplaced and will be passed into later phases of the algorithm for routing.

The Incremental Improvement Phase

The incremental improvement phase seeks to extend the set of demands placed in the initial solution by means of a loop which proceeds as follows:

1) choose an unplaced demand
2) place it on the network, allowing violations of the link capacity constraints
3) call a local search procedure that attempts to restore consistency by rerouting demands

If the local search procedure is successful in restoring consistency, the demand has been successfully placed, and will remain placed for the remainder of the incremental improvement phase – though its route may change. In this way, the local search procedure does not need to consider dropping previously placed demands as a possible means of restoring consistency. However, the price to pay for this is relatively high, as the order in which demands are considered strongly determines the solutions that can be found. It is for this reason that the tree search phase that follows, in which all combinations of placed/unplaced demands are potentially considered, was introduced.

The unplaced demands are considered in order of cost, highest first. When an unplaced demand is initially placed on the network, its path is computed using the CSPF algorithm, with link capacity violation as metric. The local search procedure used to restore consistency is also used in the tree search phase of the algorithm, and will be described in detail later.

As there is a random element in the local search procedure, and as the routing of demands changes as demands are placed, it can be beneficial to make a number of passes through the set of unplaced demands: demands that at first failed to be placed may be successfully placed when reconsidered. This helps to establish a stronger bound for the tree search phase that follows.

Hybrid Tree Search

The Hybrid Tree Search phase of the algorithm constructs a branch-and-bound search tree with the aim of optimising the total cost of the unplaced demands. Each demand D_i is associated with a boolean variable B_i, which has the value 1 if-and-only-if the associated demand is routed. The total cost is thus given by

$$\sum_i (1 - B_i) * \text{cost}\ (D_i)$$

Starting from a network in which none of the demands are placed, finite domain constraints on the demand booleans are generated (described in more detail below), and a search tree is constructed as follows:

i) select the next demand for labelling
ii) label the demand boolean to 1, leaving a choice point. On backtracking to this choice point, the boolean will be labelled to 0
iii) if the boolean was labelled to 1, then it is placed on the network, allowing link capacity violations. If there are violations, local search is called with the objective of restoring consistency: if consistency is restored, search continues on that branch, otherwise, backtracking follows

The overall structure is therefore a binary tree with the demand booleans as decision variables. Local search is called at the nodes of the tree that follow a boolean being labelled to 1. Two factors may cause backtracking: constraint propagation following the labelling of a demand boolean, or failure of local search to restore consistency. Note that as the local search procedure executed at the nodes is incomplete, so is the overall search.

The demands are ordered by cost, largest first, prior to tree search, but there is a dynamic aspect to the order in which demands are labelled: constraint propagation may instantiate demand booleans to 1 before the corresponding demand has been selected for labelling, indicating that the must be placed, and such demands are labelled before the remaining demands, again in order of cost, largest first.

When a demand is to be routed, its initial route is calculated using CSPF, with link capacity violation as metric. The local search phase that follows is the same as that applied during the preceding incremental improvement phase, and is described in the next section. In particular, it has the freedom to reroute any routed demands in order to restore consistency of the network with respect to the link capacity constraints. The number of moves that can be made by local search is kept fairly small: as in the incremental improvement phase, the hope is that routing a single demand on a consistent network can create relatively few capacity violations, and if consistency can in theory be restored, the local search algorithm should be effective enough to do so within a reasonably small number of moves.

The hybrid tree search is executed in the context of finite domain constraints. These constraints can act through propagation to instantiate the demand booleans, which can influence the order in which demands are labelled, or can cause a branch to fail early. In the design of the algorithm, the routing of demands is achieved using methods other than constraint programming, and in particular, in contrast to network flow models there are no boolean variables that reflect the presence of a particular demand on a particular link. The routes of placed demands are changed freely by the local search algorithm at the nodes of the search tree without affecting the values of any variables. It is in general possible to derive constraints that relate demand booleans and link capacities from information about the necessary presence of demands on links, regardless of the specific routes taken by the demands. For example, a link is a *necessary link* for a demand if whenever that demand is routed, it must pass over that link. This may happen because of topological or capacity considerations, or because of a maximum propagation delay constraint on the path of the demand. More generally, it is possible to compute *cut-sets* for a demand - sets of links such that, if the demand is placed, it must pass through one member of the cut-set.

Necessary links can be computed easily – by, for example, finding a path using CSPF, then prohibiting the individual links of that path and calling CSPF a second time: if a path cannot then be found, the prohibited link is necessary. Certain other cut-sets are also easy to derive: the outgoing links from the source of the demand is a cut-set, as is the set of outgoing links from the destination of a necessary link, and similarly for incoming links to the demand's destination, and to the source of a necessary link. While these are the means used in the current implementation, it could be fruitful to explore work in graph and network theory to add further forms of cuts.

For each link in a cut-set, a boolean variable is created that represents the presence of the demand on that link. The sum of these variables is constrained to be equal to the demand boolean. After generating cut-set booleans and recording them on their links, a constraint is asserted for each link that the weighted sum of the cut-set booleans for that link – the weights being the bandwidths of the associated demands – is within the link capacity.

Since the tree is constructed in the context of branch-and-bound optimisation, there is a cost-bound constraint on the total cost of unrouted demands, which is initially bounded to be less than the cost of the best solution found prior to tree search, and is updated as the branch-and-bound search proceeds in the usual way. If the incremental search phase that precedes tree search was restricted to one pass through the unplaced demands of the initial CSPF solution, it could be seen as the left-most branch of the tree, rather than as a search phase in its own right. However, multiple passes have proved effective in providing a better bound for tree search. Also, it is possible for the incremental search phase to solve the problem completely (i.e. place all demands) prior to tree search, without requiring generation of cut-set constraints (or constraint propagation).

The Local Search Procedure for Restoring Consistency

The local search procedure for restoring consistency in an inconsistent demand placement is a key component of both the Incremental Improvement and Hybrid Tree Search phases of the algorithm. Essentially, local search algorithms maintain a current search state, and proceed by moving from one state to another by computing a "neighbourhood" of a state, and selecting a neighbouring state to move to. In the present context, a state is defined by the routes of all routed demands. The state may have a number of links in which the capacity is exceeded – if there are no such links, consistency has been restored and search terminates with success.

The neighbourhood of a state is defined as all states that can result from rerouting a demand whose route includes the link whose capacity is exceeded maximally so that the demand's route no longer includes that link. This neighbourhood is typically rather large, and is certainly expensive to compute in full due to the routing calculations involved. Consequently, a 'lazy' approach is taken: a subset of demands on the maximally violated link are passed forward for rerouting, and a single reroute is generated for each such demand, using CSPF with link capacity violation as metric. Each such reroute is given a rating, based on a function of the violations it improves or introduces and those it worsens or removes, and one of the better reroutes is chosen – with a degree of randomness to discourage the search from falling into repetitive loops.

Associated with a state is a cost, i.e. the value of an objective function that the search is aiming to minimise. Here, the aim of search is to remove link capacity violations and the cost of a state is defined by a term whose arguments are the number of links whose capacity is exceeded in the current state, and the sum of the bandwidth violations, respectively. One cost term is better than another if it is lexicographically smaller.

Within local search techniques there are a number of attitudes towards neighbours that worsen the objective function. In hill-climbing, such neighbours are prohibited, while in simulated annealing (Kirkpatrick, Gelatt Jr. and Vecchi 1983), they may be accepted with a probability that diminishes as search proceeds. The approach taken here is to allow moves to such neighbours, within a hard bound on the degree of worsening. This bound can mean that no demands on the link whose capacity is exceeded that were passed forward into rerouting produce an acceptable neighbour, in which case an alternative subset of demands routed on that link will be selected, until all such demands have been tried. If this point is reached, the "least-bad" neighbour will be chosen in order to allow search to proceed. The hard bound on worsening is set quite generously, and functions as a filter for very bad moves. This technique was chosen as it proved important to allow moves that worsen the objective function in order to escape from local optima, while the fact that the local search procedure is run repeatedly through the incremental improvement and hybrid tree search phases of the algorithm precluded application of simulated annealing, which typically needs fairly long runs to converge to good solutions.

Having allowed moves to worsen the objective function, it was necessary to introduce a mechanism to discourage search from going "too far in the wrong direction" from a promising state. This is implemented through a "fallback bound": the best state found so far is recorded, along with its cost, and if a state with better cost is not found within the number of moves specified by the fallback bound, the best found state is recalled, and search proceeds from it again. At the same time, the fallback bound is increased to give search greater freedom. The fallback bound is reset to its initial value each time a new best state is found. Given the large size of the neighbourhood, and the randomised nature of the selection of reroutes, search will typically go in different directions each time a fallback occurs.

Experimental Results

In this section we give experimental results for a variety of problems on a large scale, real-world network. Due to commercial sensitivities, exact details of the network cannot be provided, however, it has roughly 100 routers and 350 (directed) links. Demands were derived from actual demand data for the network, with scaling procedures applied to make the problems hard or over-constrained. The cost of not placing a demand is taken to be the bandwidth of the demand.

All components of the hybrid algorithm were implemented in the ECLiPSe constraint logic programming language, and experiments were run on a 2GHz Pentium 4 processor under Linux. Memory requirements are modest, with the largest problems requiring about 120mb RAM.

First we consider a set of 200 problems, which have between 12 and 324 demands to be routed on an already loaded network. The average number of new demands per problem was 118. The original demand profile was used to load the network, while the choice of new demands to be routed was governed by possible use-case scenarios, such as adding a new set of services to the network. Two loading factors were used to multiply demand bandwidths, in order to generate different degrees of overconstrained problems – there were 100 problems at each loading factor. The percentage of requested bandwidth unplaced in the CSPF solution was 3.27% for the lower loading factor and 7.47% for the higher. In order to understand the relative contributions of the different phases of the hybrid algorithm, we give results for the initial CSPF solution, the solution achieved after incremental improvement, and solution found by the algorithm as a whole, i.e. after running tree search. Results are given in Figures 1 & 2, where Total Cost is the sum of costs of all problems, while Percentage Cost Reduction is relative to the initial CSPF solution.

Algorithm	**Total Cost**	**% Cost Reduction**
CSPF	7837755	-
CSPF + Incremental Improvement	7745647	1.18
Full LS/CP Hybrid	7554663	3.61

Figure 1: Demand Admission, lower loading

Algorithm	**Total Cost**	**% Cost Reduction**
CSPF	19987968	-
CSPF + Incremental Improvement	19420280	2.84
Full LS/CP Hybrid	18685141	6.52

Figure 2: Demand Admission, higher loading

In order to get some measure of the incompleteness inherent in the LS/CP Hybrid, for the above demand admission problems we compare the deficiency cases between this algorithm and Probe Backtrack Search algorithm presented in (Liatsos, Novello, and El-Sakkout, 2003), extended by the authors of that paper to cover over-constrained problems. A problem instance counts as a deficiency for an algorithm if the alternate algorithm finds a better solution on that problem. Both algorithms were run with a timeout of 600 cpu-seconds.

Algorithm	**Deficiency Cases (lower loading)**	**Deficiency Cases (higher loading)**
LS/CP Hybrid	11	23
PBT Search	18	24

Figure 3: Demand Admission deficiency cases

This comparison indicates that the problems are indeed hard for both algorithms, as each beats the other in a significant number of cases. However, the local search/constraint programming hybrid achieves better results than the complete algorithm within the allowed cpu-time. The relative improvement for PBT in the higher loading case can perhaps be explained by the fact that it uses a more extensive set of constraints for propagation than the LS/CP hybrid, and in the higher loading case demand bandwidths are greater with respect to available capacities, giving more opportunity for pruning.

We continue by reporting some results on a large scale problem, in which all demands of the actual demand data were considered as new demands, and their bandwidths scaled in order to generate a tightly-constrained but solvable problem with approximately 1900 demands to be routed. The aim was to increase the scale factor as far as possible while still placing all demands. The results do not include the hybrid tree search phase, which did not succeed in placing all demands when the incremental improvement phase had failed to do so. These results appear in Figure 4, and show that the incremental improvement phase produced an increase of 2.92% over the bandwidth placed in the initial solution, within a relatively short run-time.

Algorithm	**Scale Factor**	**CPU time**
CSPF Initial Solution	2.74	4.05 seconds
CSPF + Incremental Improvement	2.82	109.42 seconds

Figure 4: Scaled Actual Demands

There are two further points to mention. First, the scale of demand admission problems is related to two aspects: the number of demands that may have their routes changed, and the numbers of demands that need not be routed i.e. may be rejected or dropped. The former influences that scale of the reroute problem, while the latter determines the potential scale of the branch-and-bound search tree. In situations when demands already on the network may be rerouted but not dropped, the scale of the reroute problem is increased independently of that of tree search. The performance on the large scale problem above demonstrates the effectiveness of local search procedures for the routing aspect of demand admission problems.

Secondly, for large scale problems, or problems in which there remain many unrouted demands following the incremental improvement phase, the search tree is potentially very large. In such cases, initial experiments indicate that it can be interesting to apply Limited Discrepancy Search (Harvey and Ginsberg 1995) to bound the degree to which solutions may differ from the best solution found by the earlier search phases. This keeps search within the vicinity of a good solution, while exploring changes to all assignments of that solution.

Conclusions and Further Work

We have presented a hybrid algorithm combining constraint propagation and local search in a novel way, with local search used to satisfy the hard problem constraints in the context of branch-and-bound optimisation in which constraint propagation also occurs. The algorithm was developed with the aim of solving complex, large scale demand admission problems, and has been shown to be effective for such problems. A potential contribution from Limited Discrepancy Search has been identified, and will be investigated further.

Acknowledgements

Thanks to my colleagues at IC-Parc and PARC Technologies Ltd: Farid Ajili, Vassilis Liatsos, Stefano Novello, Hani El-Sakkout and Josh Singer. This research has been supported by Parc-Technologies Ltd.

References

Barnhart, C., Hane, C. A., and Vance, P. H. 2000. Using branch-and-price-and cut to solve origin-destination integer multicommodity flow problems. *Operations Research 48(2)*:318-326.

Casseau, Y., and Laburthe, F. 1999. Heuristics for large constrained vehicle routing problems. *Journal of Heuristics*, 5(3):281-303. Kluwer.

Genreau, M., and Pesant, G. 1999. A constraint programming framework for local search methods. *Journal of Heuristics*, 5(3):255-279. Kluwer.

Harvey, W. D. and Ginsberg, M. L. 1995. Limited discrepancy search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, 607-615*. Los Angeles.: Morgan Kaufmann.

Jussien, N., and Lhomme, O. 2000. Local search with constraint propagation and conflict-based heuristics. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, 2000:169-174. AAAI Press.

Kamarainen, O., and El-Sakkout, H. 2002. Local probing applied to scheduling. *In Proceedings of the Eighth International Conference on the Principles and Practice of Constraint Programming*, 155-171. Springer-Verlag.

Kirkpatrick, S., Gelatt Jr., C, and Vecchi, M. 1983. Optimization by simulated annealing. *Science* 220:671-680.

Lee, W., Hluchyi, M., and Humblet, P. 1995. Routing subject to quality of service constraints in integrated communication networks. *IEEE Network*, July 1995: 46-55.

Liatsos, V., Novello, S., and El-Sakkout, H. 2003. A probe backtrack search algorithm for network routing. In *Proceedings of the CP2003 Third Workshop on Cooperating Solvers in Constraint Programming*, Springer Verlag.

Schaerf, A. 1997. Combining local search and look-ahead for scheduling and constraint satisfaction problems. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, 1254-1259*. Los Angeles.: Morgan Kaufmann.

Shaw, P. 1998. Using constraint programming and local search methods to solve vehicle routing problems. In *Proceedings of the Fourth International Conference on the Principles and Practice of Constraint Programming*, 417-431. Springer-Verlag.

Wang, Y., and Wang, Z. 1999. Explicit routing algorithms for internet traffic engineering. In *Proceedings of IEEE ICCCN'99*.

Zhang, J., and Zhang, H. 1996. Combining local search and backtracking techniques for constraint satisfaction. In *Proceedings of the American Association for Artificial Intelligence*, 1996:369-374. AAAI Press.