Abstraction and Constraint Satisfaction Techniques for Planning Bandwidth Allocation

Christian Frei and Boi Faltings
Artificial Intelligence Laboratory
Swiss Federal Institute of Technology (EPFL)
CH-1015 Lausanne, Switzerland
Christian.Frei@epfl.ch

Abstract— **Communication networks are expected to offer a wide range of services to an increasingly large number of users, with a diverse range of quality of service. This calls for efficient control and management of these networks. We address the problem of quality-of-service routing, more specifically the planning of bandwidth allocation to communication demands. Shortest path routing is the traditional technique applied to this problem. However, this can lead to poor network utilization and even congestion. We show how an abstraction technique combined with systematic search algorithms and heuristics derived from Artificial Intelligence make it possible to solve this problem more efficiently and in much tighter networks, in terms of bandwidth usage.**

Keywords— **Quality of Service routing, constraint-based routing, resource allocation planning, abstraction, constraint satisfaction.**

I. Introduction

THE communication networks of the next millennium are expected to offer a wide range of services to an increasingly large number of users, with a diverse range of Quality of Service (QoS) requirements. This calls for efficient control and management of these high-speed networks. A central problem is the automatic routing of traffic through the network. Routing must be a very fast process, in order to guarantee customer satisfaction. Currently, shortest path routing is most often used to route traffic across a network. Although this ensures the best possible route for each particular demand, it can lead to ineffective use of the network as a whole and even congestion, especially in highly loaded networks.

From the routing point of view, the key resource to manage in networks is bandwidth. Therefore, in order to make better use of available network resources, there is a need for planning bandwidth allocation to communication demands, in order to set up routing tables (or any other route selection criterion) more purposefully. This can be achieved by the use of *global information*, including not only the available link capacities but also the expected traffic profile. In this paper, we consider the problem of allocating in an *off-line* manner a set of demands known in advance within the resource capacities of a communication network. This situation may arise for instance when setting up virtual private networks in a connection-oriented network (e.g., ATM, TDM) of a provider; planning the routing of virtual path connections (VPC) in an ATM network; planning the routing of virtual channel connections (VCC) in the VPC network of an ATM backbone; or optimizing the routing tables of an IP network (demands are then estimated from objective traffic measurements).

Formally, we define the problem of resource allocation in networks (RAIN) as follows:

Given a network composed of nodes and bidirectional links, where each link has a given bandwidth capacity, and a set of communication demands to allocate, where each demand is defined by a triple:

(source node, destination node, requested bandwidth)

Find one and only one route for each demand so that the bandwidth requirements of the demands are simultaneously satisfied within the resource capacities of the links.

It is important to note that because of technological limitations (for ATM typically) and/or performance reasons, it is impossible to divide demands among multiple routes. However, there may be several demands between same endpoints. With this restriction, the RAIN problem is NP-hard in the number of demands. When demands are subject to multiple additive or multiplicative quality of service (QoS) criteria, then Wang and Crowcroft [1] have shown that the allocation of every single demand is NP-complete by itself. This creates a new situation for the networking community, as traditional routing algorithms such as shortest paths do not perform very well on this problem.

In practice, the RAIN problem poses itself in the following way: a network or service provider receives a set of requests from some customers to allocate a number of demands, and must decide within a certain time decision threshold whether and how the demands can be accepted.

Constraint satisfaction [2] is a technique which has been shown to work well for solving certain NP-hard problems, and has been applied to a variety of domains [3]. A *Constraint Satisfaction Problem* (CSP) is defined by a triple (X, D, C), where $X = \{x_1, ..., x_n\}$ is a set of *variables*, $D = \{D_1, ..., D_n\}$ a set of finite *domains* associated with the variables and $C = \{C_1, ..., C_m\}$ a set of *constraints*. The domain of a variable is the set of all values that can be assigned to that variable. A constraint between variables restricts the combinations of values that can be assigned to those variables. Solving a CSP amounts to finding a value for each variable so that all constraints are satisfied. This may be done with a *backtracking algorithm*.

The RAIN problem is easily formulated as a CSP in the following way: variables are demands, the domain of each variable is the set of all routes between the endpoints of the demand, and constraints on each link must ensure that the resource capacity is not exceeded by the demands routed through it. A solution is a set of routes, one for each demand, respecting the capacities of the links. However, this formulation presents severe complexity problems. It is too expensive to compute, represent, and store the domain of a variable, i.e., all the routes that join the end-

points of a demand. Suppose the network is simple but complete (this is not even the worst case, since a communication network is a multi-graph: it allows multiple links between same endpoints) with n nodes. A route is a simple path, its length in number of links is therefore bounded by $n-1$. Since a route of length j has $j-1$ intermediate (distinct) nodes, the number of routes of length j is $(n-2)!/(n-j-1)!$. The total number of routes between two nodes is therefore equal to $\sum_{i=1}^{n-1}(n-2)!/(n-i-1)!$. Storing all routes between a pair of nodes would require exponential space. For instance, in a complete graph with 10 nodes, there are 69'281 routes between any two nodes. Since methods such as forward checking or dynamic variable ordering require explicit representation of domains, they would be very inefficient on a problem of realistic size.

It has long been observed that the complexity of solving a problem can depend heavily on how it is formulated. Giunchiglia and Walsh define abstraction as follows in [4]: *"Abstraction is the mapping of a problem representation into a simpler one that satisfies some desirable properties in order to reduce the complexity of reasoning. The problem is solved in the abstract space and the solution is then mapped back to the more complex ground space."* The ground space refers to the original problem representation. Abstractions are naturally used my humans to solve problems. Reducing problem complexity is a major reason for using abstraction techniques. A recent collection of papers addressing abstraction, reformulation, and approximation techniques in a variety of AI domains can be found in [5].

In this paper, we show how an abstraction of the network called Blocking Islands, create a compact representation of the domains which allows the application of well-known CSP techniques such as forward checking, variable and value ordering to the RAIN problem with manageable complexity. In the following section, we review some of the related work. In Section III, we briefly recall the Blocking Island paradigm and outline its major properties. Section IV illustrates how blocking islands help to route a single demand while attempting to preserve bandwidth connectivity inside the network. In Section V, we present a generic algorithm and some heuristics to solve the RAIN problem. Empirical results are summarized in Section VI. The blocking island paradigm is generalized to multiple link constraints in Section VII. Finally, we conclude by some future work directions.

II. Related Work

Surprisingly, there has been little published research on the RAIN problem. Currently, most network providers use some kind of best effort algorithm, without any backtracking due to the complexity of the problem: given an order of the demands, each demand is assigned the shortest possible route supporting it, or just skipped if there is no such route.

Operations Research (OR) techniques are also applied to the RAIN problem. Most often, a fixed number of shortest paths for each demand are pre-computed, and the problem is solved using linear programming with very large constraint systems of equations [6], [7], [8], [9]. However, because only a given number of routes are considered, these techniques are not guaranteed to find a solution if one exists. Moreover, OR techniques are not as flexible as CSP-based methods (see Section VIII).

Mann and Smith [10] search for routing strategies that attempt to ensure that no link is over-utilized (hard constraint) and, if possible, that all links are evenly loaded (below a fixed target utilization), for the predicted traffic profile. Finally, they attempt to minimize the communication costs. Genetic algorithms and simulated annealing approaches were used to develop such strategies. However, their methods do not apply well, if not at all, to highly loaded networks, mainly because the multi-criteria objective function they use cannot ensure that the hard constraint, i.e., no link is over-utilized, is respected in every case. Moreover, we think that load balancing should be viewed in terms of bandwidth connectivity and not the even distribution of the load among the links, especially in highly loaded networks, since high bandwidth connectivity allows to route additional demands without having to recompute a complete solution.

Vedantham and Iyengar [11] prove that the problem of effective bandwidth utilization in the ATM network model is NP-complete. In the situation where there are more incoming calls than available bandwidth, they also propose the use of Genetic Algorithms for maximizing the revenue.

Bandwidth auctionning through a multi-agent system is being explored [12]; however, this work is still at an early stage.

To our knowledge, the closest published work to ours is the CANPC framework [13]. It is based on the successive allocations of shortest routes to the demands, without any backtracking when an assignment fails. They propose several heuristics to order the demands (such as bandwidth ordering) to provide better solutions, i.e., to route more demands. They are currently developing an optimization tool that takes the partial solution as input to try to allocate all demands. However, results show that the methods we propose clearly outperform theirs.

Abstraction and reformulation techniques have already been applied to permit more efficient solution of a CSP. Choueiry and Faltings [14] relate interchangeability to abstraction in the context of a decomposition heuristic for resource allocation. Weigel and Faltings [15] cluster variables to build abstraction hierarchies for configuration problems viewed as CSPs, and then use interchangeability to merge values on each level of the hierarchy. Freuder and Sabin [16] present abstraction and reformulation techniques based on interchangeability to improve solving CSPs.

The phenomenon of phase transitions occurring in many types of problems as a control parameter is varied has been recognized and studied extensively in recent years. Cheeseman *et al.* [17] first reported a phase transition between a region where almost all problems have many solutions and are relatively easy to solve, and a region where almost all problems have no solution and their insolubility is relatively easy to prove. In this intervening region, the probability of problem solubility falls from close to 1 to 0, and the cost of searching these problems is highest. Cheeseman suggests the following conjecture: *"All NP-complete problems have at least one order parameter and the hard to solve problems are around a critical value of this order parameter"*. The critical value (or a range) of the order parameter is where phase transition occurs. The value of the order parameter for a problem instance is often called the *tightness* of the problem instance. Noteworthy, Gent *et al.* [18] observed that the relative behavior of algorithms on large and

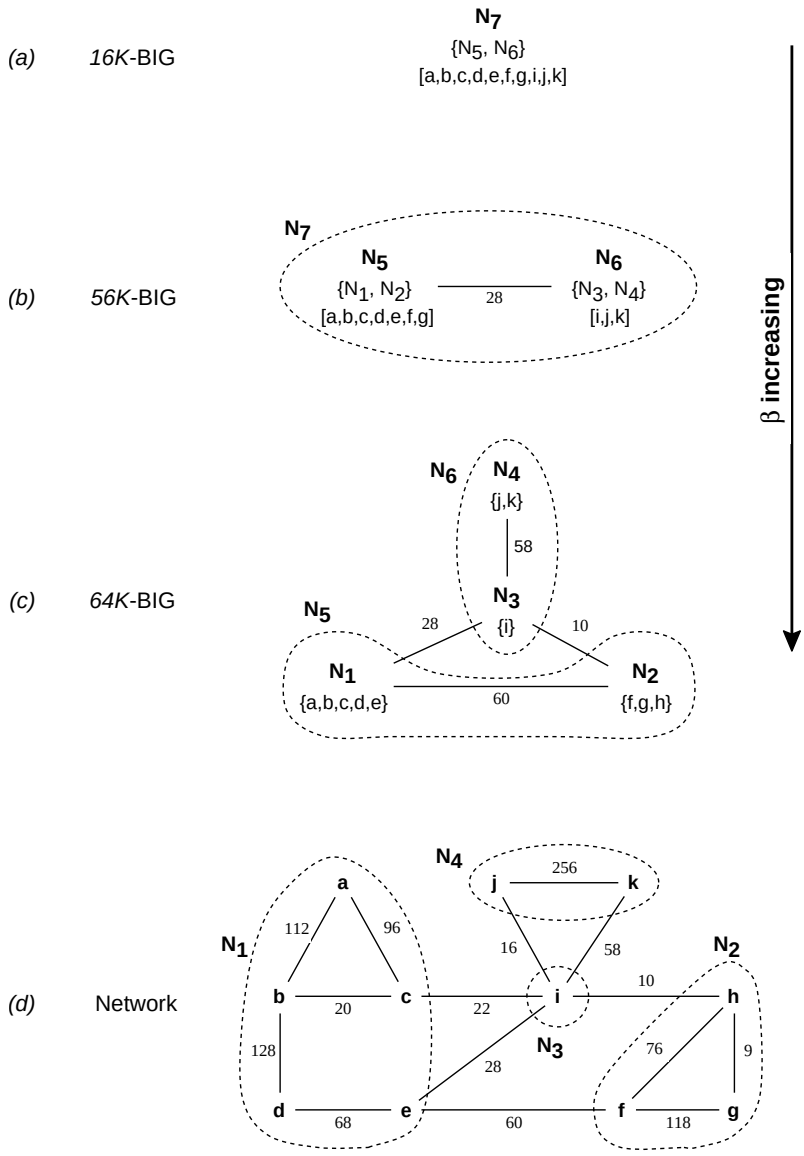


Fig. 1. The blocking island hierarchy for resource requirements $\{64K, 56K, 16K\}$. The weights on the links are their available bandwidth. Abstract nodes' description include only their node children and network node children in brackets. Link children (of BIs and abstract links) are omitted for more clarity, and the 0-BI is not displayed since equal to N_7. (a) the 16-BIG. (b) the $56K$-BIG.(c) the $64K$-BIG. (d) the network.

small problems is the same when plotted against this parameter. Comparison of different algorithms can therefore be performed on small problems, and results can be expected to scale to larger problems. Phase transition behavior has been reported in an increasing number of NP-complete problems [19].

III. The Blocking Island paradigm

Frei and Faltings [20] introduce a clustering scheme based on Blocking Islands (BI), which can be used to represent bandwidth availability at different levels of abstraction, as a basis for distributed problem solving. A β-blocking island (β-BI) for a node x is the set of all nodes of the network that can be reached from x using links with at least β available resources, including x. Fig. 1 (d) shows all $64K$-BIs for a network. Note that some links inside a β-BI, i.e., the links that have both endpoints in the β-BI, may have less than β available resources. In such a case, it simply means that there is another route with β available resources between the link's endpoints. As a matter of fact, link (b, c) has both endpoints in $64K$-BI N_1 but has less than $64K$ available resources. However, there are at least $64K$ available resources along route $\{(c, a), (a, b)\}$.

β-BIs have some fundamental properties. Given any resource requirement, blocking islands partition the network into equivalence classes of nodes. The BIs are *unique*, and *identify global bottlenecks*, that is, inter-blocking island links. If inter-blocking island links are links with low remaining resources, as some links inside blocking islands may be, inter-blocking island links are links for which there is no alternative route with the desired resource requirement. Moreover, BIs highlight the *existence* and *location* of routes at a given bandwidth level:

Proposition 1 (Route Existence Property) There is at least one route satisfying the resource requirement of an unallocated demand $d_u = (x, y, \beta_u)$ if and only if its endpoints x and y are in the same β_u-blocking island. Furthermore, all links that could

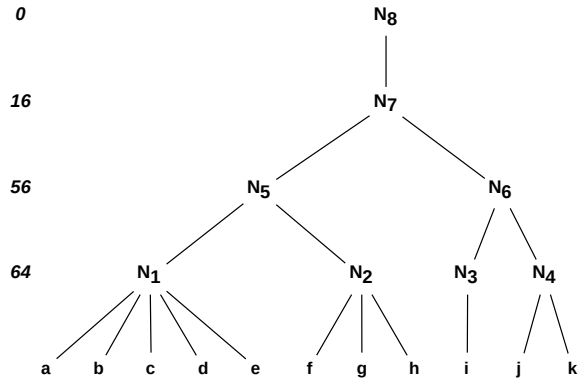


Fig. 2. The abstraction tree of the BIH of Fig. 1 (links are omitted for clarity).

form part of such a route lie inside this blocking island.

Finally, the *inclusion property* states that for any $\beta_i < \beta_j$, the β_j-BI for a node is a subset of the β_i-BI for the same node.

Blocking islands are used to build the β-blocking island graph (β-BIG), a simple graph representing an *abstract* view of the available resources: each β-BI is clustered into a single node and there is an abstract link between two of these nodes if there is a link in the network joining them. Fig. 1 (c) is the 64-BIG of the network of Fig. 1 (d). An abstract link between two BIs clusters all links that join the two BIs, and the abstract link's available resources is equal to the maximum of the available resources of the links it clusters (since a demand can only be allocated over one route). These abstract links denote the critical links, since their available resources do not suffice to support a demand requiring β resources.

In order to identify bottlenecks for different βs, e.g., for typical possible bandwidth requirements, we build a recursive decomposition of BIGs in decreasing order of the requirements: $\beta_1 > \beta_2 > ... > \beta_b$. This layered structure of BIGs is a *Blocking Island Hierarchy* (BIH). The lowest level of the blocking island hierarchy is the β_1-BIG of the network graph. The second layer is then the β_2-BIG of the first level, i.e., β_1-BIG, the third layer the β_3-BIG of the second, and so on. On top of the hierarchy there is a 0-BIG abstracting the smallest resource requirement β_b. The abstract graph of this top layer is reduced to a single abstract node (the 0-BI), since the network graph is supposed connected. Fig. 1 shows such a BIH for resource requirements $\{64K, 56K, 16K\}$. The graphical representation shows that each BIG is an abstraction of the BIG at the level just below (the next biggest resource requirement), and therefore for all lower layers (all larger resource requirements).

A BIH can not only be viewed as a layered structure of β-BIGs, but also as an *abstraction tree* when considering the father-child relations. In the abstraction tree, the leaves are network elements (nodes and links), the intermediate vertices either abstract nodes or abstract links and the root vertex the 0-BI of the top level in the corresponding BIH. Fig. 2 is the abstraction tree of Fig. 1.

The β-BI S for a given node x of a network graph can be obtained by a simple greedy algorithm: starting with an initial set $S = \{x\}$, we recursively add every node to S that can be reached by a link adjacent to a node of S, and that has at least β available bandwidth. When no more new nodes can be added, S is the β-BI sought. This algorithm has a linear complexity of $O(m)$, where m is the number of links. The construction of a β-BIG is straightforward from its definition and is also linear in $O(m)$. A BIH for a set of constant resource requirements ordered decreasingly is easily obtained by recursive calls to the BIG computation algorithm. Its complexity is bound by $O(bm)$, where b is the number of different resource requirements. The adaptation of a BIH when demands are allocated or deallocated can be carried out incrementally with complexity $O(bm)$ (see [20] for more details). Therefore, since the number of possible bandwidth requirements (b) is constant, all BI algorithms are linear in the number of links of the network.

A BIH contain at most $bn+1$ BIs, that is, one BI for each node at each bandwidth requirement level, plus the 0-BI. In that worst case, there are $\min\{m, n(n-1)/2\}$ links at each bandwidth level, since multiple links between same BIs are clustered into a single abstract link. Therefore, the memory storage requirement of a BIH is bound by $O(bn^2)$.

IV. Routing from the BIH perspective

Consider the problem of routing a single demand $d_u = (c, e, 16K)$ in the network of Fig. 1 (d). Since c and e are clustered in the same $16K$-BI (N_7), we know that at least one route satisfying d_u exists. Classical wisdom would select the shortest route, that is the route r_S: $c \to i \to e$. However, allocating this route to d_u is here not a good idea, since it uses resources on two critical links in terms of available bandwidth, that is (c, i) and (i, e): these two links join $64K$-BIs N_1 and N_2 in the $64K$-BIG of Fig. 1 (c). After that allocation, no other demand requiring 16K (or more) between any of the nodes clustered by $56K$-BI N_5 and one of the nodes inside $56K$-BI N_6 can be allocated anymore. For instance, a demand $(c, i, 16K)$ is then impossible to allocate. A better way to route d_u is r_L: $c \to b \to d \to e$, since r_L uses only links that are clustered at the lowest level in the BIH, that is in $64K$-BI N_1, and no critical links (that is inter-BI links).

r_L is a route that satisfies the *lowest level* (LL) heuristic. Its principle is to route a demand along links clustered in the lowest BI clustering the endpoints of the demand, i.e., the BI for the highest bandwidth requirement containing the endpoints. This heuristic is based on the following observation: the lower a BI is in the BIH, the less critical are the links clustered in the BI. By assigning a route in a lower BI, a better *bandwidth connectivity* preservation effect is achieved, therefore reducing the risk of future allocation failures. Bandwidth connectivity can therefore be viewed as a kind of *overall load-balancing*.

Another way to see the criticalness of a route is to consider the *mapping* of the route onto the abstraction tree of Fig. 3: r_S is by far then the longest route, since its mapping traverses BIs N_1, N_5, N_7, N_6, N_3, and then back; r_L traverses only BI N_1. r_S therefore affects not only critical links at higher level then r_L, but also many more BIs, and its allocation may cause to split each of them. This observation (also) justifies the LL heuristic.

Even better, a BIH gives also the means to compare *a priori* equivalent routes in order to decide for the "best" one, besides the length criterion. The *minimal splitting* (MS) heuristic selects the route that causes the fewest splittings of blocking islands in the BIH: obviously, the more splittings, the more links become

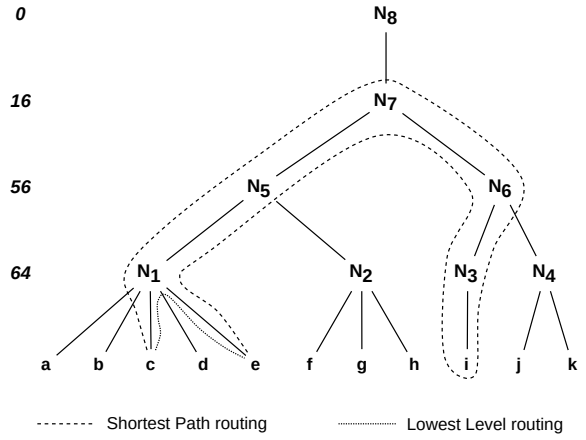


Fig. 3. Mapping the shortest path (SP) and the lowest level (LL) routes onto the abstraction tree.

critical, leading to more allocation failures of demands. MS has therefore an even greater bandwidth connectivity preservation effect than LL. Unfortunately, only an approximation of the MS heuristic can effectively be used in practice, since an exact implementation of MS requires to compute all routes beforehand in order to compare them. A possible implementation is to compute a given number of routes using LL, and then to order them according to the MS heuristic to select a route. The evaluation of the MS heuristic is left for a later paper.

Widest path routing has been proposed as an alternative to SP. For instance, Wang and Crowcroft [1] advocate the use of *shortest-widest path* (WP) for hop-by-hop routing algorithms. This strategy is to find a route with maximum bottleneck bandwidth (a widest path), and when there are more than one widest path, choose the one with shortest propagation delay (in our case the number of hops). In routing the same demand d_u as above, WP selects the route r_W: $c \to a \to b \to d \to e$, a route longer than r_S or r_L. Therefore, even if it attempts to distribute the load by avoiding as much as possible bottleneck links, WP may select a very long route, thereby using a lot of resources globally. However, WP performed very poorly in our experiments, as expected, and we will not report it on solving the RAIN problem. We show nonetheless its behavior in case of QoS-routing (Section VI-B).

Because of its characteristics, LL can be viewed as a mixture of SP and WP.

V. Automatically solving a RAIN problem

Solving a RAIN problem amounts to solving the CSP introduced in Section I. This can be done using a *backtracking algorithm* with *forward checking* (FC) [2]. Its basic operation is to pick one variable (demand) at a time, assign it a value (route) of its domain that is compatible with the values of all instantiated variables so far, and propagate the effect of this assignment (using the constraints) to the future variables by removing any inconsistent values from their domain. If the domain of a future variable becomes empty, the current assignment is undone, the previous state of the domains is restored, and an alternative assignment, when available, is tried. If all possible instantiations fail, backtracking to the previous past variable occurs. FC proceeds in this fashion until a complete solution is found or all possible assignments have been tried unsuccessfully, in which case there is no solution to the problem.

The formulation of the CSP presents severe complexity problems (see Section I). Nonetheless, blocking islands provide an abstraction of the domain of each demand, since any route satisfying a demand lies within the β-BI of its endpoints, where β is the resource requirement of the demand (Proposition 1). Therefore, if the endpoints of a demand are clustered in the same β-BI, there is at least one route satisfying the demand. We do not know what the domain of the variable is *explicitly*, i.e., we do not know the set of routes that can satisfy the demand; however we know it is non-empty. In fact, there is a mapping between each route that can be assigned to a demand and the BIH: a route can be seen as a path in the abstraction tree of the BIH. Thus, there is a route satisfying a demand if and only if there is a path in the abstraction tree that does not traverse BIs of a higher level than its resource requirement. For instance, from the abstraction tree of Fig. 2, it is easy to see that there is no route between a and f with 64 available resources, since any path in the tree must at least cross BIs at level 56.

This mapping of routes onto the BIH is used to formulate a forward checking criterion, as well as dynamic value ordering and dynamic variable ordering heuristics.

A. Forward Checking

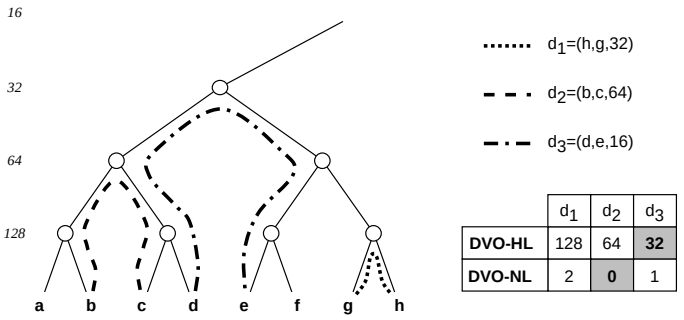
Forward checking is a technique to improve backtracking algorithms. Its idea is to propagate value assignments to unallocated variables along the constraints in order to detect a dead-end earlier, thereby increasing search efficiency. Moreover, decisions regarding which variable to select next and what value of the selected variable to try next can then be done in a more informed way.

Thanks to the route existence property, we know at any point in the search if it is still possible to allocate a demand, without having to compute a route: if the endpoints of the demand are clustered in the same β-BI, where β is the resource requirement of the demand, there is at least one, i.e., the domain of the variable (demand) is not empty, even if not explicitly known.

Therefore, after allocating a demand, forward checking is performed first by updating the BIH, and then by checking that the route existence property holds for all uninstantiated demands. If the latter property does not hold at least once, another route must be tried for the current demand. Domain pruning (i.e., the update of the domain of the demands by propagation of the allocation decisions) is thus implicit while maintaining the BIH.

B. Value Ordering

A backtracking algorithm involves two types of choices: the next variable to assign (see Section V-C), and the value to assign to it. As illustrated above, the domains of the demands are too big to be computed beforehand. Instead, we compute the routes as they are required. In order to reduce the search effort, routes should be generated in "most interesting" order, so to increase the efficiency of the search, that is: try to allocate the route that will less likely prevent the allocation of the remaining demands. A natural heuristic is to generate the routes in *shortest path* order



	d_1	d_2	d_3
DVO-HL	128	64	**32**
DVO-NL	2	**0**	1

Fig. 4. Selecting the next demand to allocate using DVO-HL and DVO-NL. The DVO-HL line shows the value for the lowest common father level for each demand. The DVO-NL line shows the number of levels between the lowest common father and the bandwidth requirement level. The selected demand by each heuristic is shaded in gray.

(SP), since the shorter the route, the fewer resources will be used to satisfy a demand.

However, Section IV shows how to do better using a kind of min-conflict heuristic, the lowest level heuristic. Applied to the RAIN problem, it amounts to considering first, in shortest order, the routes in the lowest blocking island (in the BIH). Apart from attempting to preserve bandwidth connectivity, the LL heuristic allows to achieve a computational gain: the lower a BI is, the smaller it is in terms of nodes and links, thereby reducing the search space to explore. Generating one route with the LL heuristic can be done in linear time in the number of links (as long as QoS is limited to bandwidth constraints).

C. Variable Ordering

The selection of the next variable to assign may have a strong effect on search efficiency, as shown by Haralick [21] and others. A widely used variable ordering technique is based on the "fail-first" principle: *"To succeed, try first where you are most likely to fail"*. The rationale is to minimize the size of the search tree and to ensure that any branch that does not lead to a solution is pruned as early as possible when choosing a variable.

There are some natural static variable ordering (SVO) techniques for the RAIN problem, such as first choose the demand that requires the most resources. Nonetheless, BIs allow dynamic (that is during search) approximation of the difficulty of allocating a demand in more subtle ways by using the abstraction tree of the BIH:

DVO-HL (Highest Level): first choose the demand whose lowest common father of its endpoints is the highest in the BIH (recall that high in the BIH means low in resources requirements). The intuition behind DVO-HL is that the higher the lowest common father of the demand's endpoints is, the more constrained (in terms of number of routes) the demand is. Moreover, the higher the lowest common father, the more allocating the demand may restrict the routing of the remaining demands (fail first principle), since it will use resources on more critical links.

DVO-NL (Number of Levels): first choose the demand for which the difference in number of levels (in the BIH) between the lowest common father of its endpoints and its resources requirements is lowest. The justification of DVO-NL is similar to DVO-HL.

The behavior of DVO-HL and DVO-NL are illustrated in Fig. 4. In the implementation, both latter heuristics use the required bandwidth as a secondary criterion to break ties: in case two or more demands have the same value for the criterion, the one with highest requirement is preferred. Besides obeying the fail-first principle, this secondary criterion attempts to minimize one side effect of the lowest level heuristic for route selection: by avoiding the routing through critical links, LL may cause the split of BIs at very low levels in the hierarchy, i.e., for high bandwidth requirements, thereby preventing the allocation of demands with high requirements.

There are numerous other *Dynamic Variable Ordering* (DVO) heuristics that can be derived from analyzing the BIH, and their presentation and evaluation is left for a later paper.

```
Function FCRAIN (D, Γ, H)
  if D = ∅ then
    (* no more demands to allocate: solution found *)
    return Γ
  else
    d = (x, y, β) ← pick a demand of D
    r ← NextBestRoute (d, H)
    repeat (* Try connection (d, r) *)
      γ ← (d, r)
      UpdateConnectionAddition (H, Γ, γ)
      (* Forward checking: verify that (d, r) does *)
      (* not prevent the remaining allocations *)
      if ExistsRouteForAll (D − {d}, H) then
        Res ← FCRAIN (D − {d}, Γ ∪ {γ}, H)
        if Res ≠ ∅ then
          return Res (* We have a solution *)
      end if
      UpdateConnectionRemoval (H, Γ, γ)
      r ← NextBestRoute (d, H)
    until r = ∅
    return ∅ (* Backtrack *)
  end if
end FCRAIN
```

Fig. 5. The forward checking algorithm *FCRAIN* for the RAIN problem. Its input is threefold: $\mathcal{D}$ is the set of still unallocated demands, Γ the set of established connections, and $\mathcal{H}$ the current blocking island hierarchy. It returns either a set of connections Γ (a solution) or $\emptyset$ (in which case there is no solution).

D. A forward-checking algorithm for the RAIN problem

Now that the different parts have been examined, we can put everything together into a systematic search algorithm. Fig. 5 shows a pseudo-code for a recursive forward-checking backtracking algorithm *FCRAIN*. FCRAIN can be read as follows: if there are still unallocated demands, it selects the next demand to allocate using a dynamic demand ordering heuristic (Section V-C). *NextBestRoute* computes the best route according to the dynamic route ordering heuristic (Section V-B). FCRAIN then performs forward-checking: it verifies that all remaining unallocated demands can still be allocated, using generic function *ExistsRouteForAll*. The latter checks that all unallocated demands have both endpoints in the same BI at their bandwidth

requirement level (as explained in Section V-A). If it is the case, it recursively allocates the next demand. Otherwise, the current allocation is undone, and the best next route is tried. If all routes have been tried unsuccessfully, backtracking to the previously allocated demand occurs. This amounts to deallocating the current demand and the previously allocated demand, and selecting for the latter the next best route.

VI. Empirical Results

A. Results on the RAIN problem

Recall that in the typical scenario presented in Section I, a network/service provider must decide within a certain time decision threshold whether and how a set of demands could be accepted. A meaningful analysis of the performance of the heuristics we proposed would thus analyze the probability of finding a solution within the given time limit, and compare this with the performance that can be obtained using common methods of the networking world, in particular shortest-path algorithms.

For comparing the efficiency of different constraint solving heuristics, it is useful to plot their performance for problems of different tightness. In the RAIN problem, tightness is the ratio of resources required for the best possible allocation (in terms of used bandwidth) divided by the total amount of resources available in the network. This is an approximation of the "constraint tightness" in the CSP. Since it is very hard to compute the best possible allocation, we use an approximation, the best allocation found among the methods being compared.

We generated 22'000 instances of RAIN problems, each with at least one solution. Each problem has a randomly generated network topology of 20 nodes and 38 links, and a random set of 80 demands, each demand characterized by two endpoints and a bandwidth constraint. A solution must allocate all demands within the bandwidth capacities of the links. No other restriction was imposed on the routes. We especially supposed no hop-by-hop routing table constraints for instance. A solution is thus applicable to a connection-oriented network such as ATM. The problems were solved with four different strategies: *basic-SP* performs a search using the shortest path heuristic common in the networking world today, without any backtracking on decisions; *BT-SP* incorporates backtracking to the previous in order to be able to undo "bad" allocations. The next search methods make use of the information derived from the BIH: *BI-LL-HL* uses the LL heuristic for route generation and DVO-HL for dynamic demand selection, whereas *BI-LL-NL* differs from the latter in using DVO-NL for choosing the next demand to allocate.

Fig. 6 provides the probability of finding a solution to a problem in less than 1 second, given the tightness of the problems (as defined above). Both BI search methods prove to perform much better than brute-force, even on these small problems, where heuristic computation (and BIH maintenance) may proportionally use up a lot of time. Noteworthy, NL outperforms HL: NL is better at deciding which demand is the most difficult to assign, and therefore achieves a greater pruning effect. The shape of the curves is similar for larger time scales. The quality of the solutions, in terms of network resource utilization, was about the same for all methods. However, when the solutions were different, bandwidth connectivity was generally better on

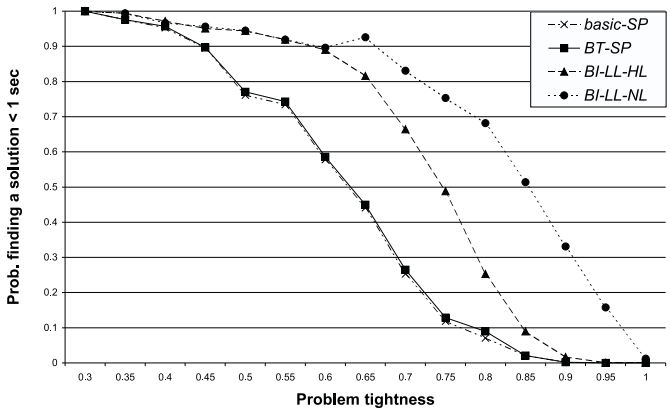


Fig. 6. The probability of finding a solution within 1 second, given the tightness of the problems (22'000 random problems with 20 nodes, 38 links, 80 demands).

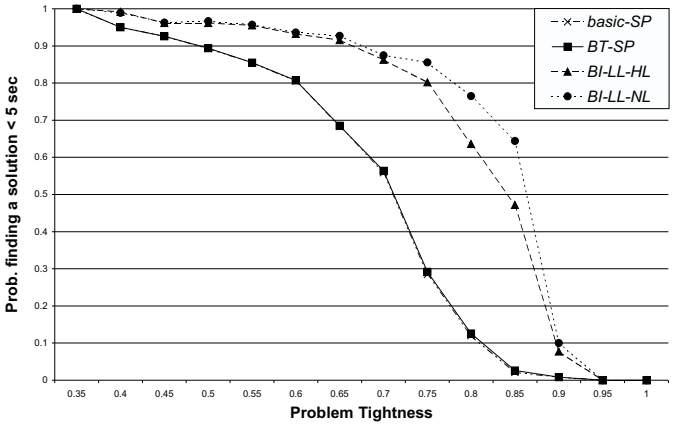


Fig. 7. The probability of finding a solution within 5 seconds, given the tightness of the problems (6'000 random problems with 20 nodes, 38 links, 200 demands).

those provided by BI methods.

Note that the experimental results allow quantifying the gain obtained by using our methods. If an operator wants to ensure high customer satisfaction, demands have to be accepted with high probability. This means that the network can be loaded up to the point where the allocation mechanism finds a solution with probability close to 1. From the curves, we can see that for the shortest-path methods, this is the case up to a load of about 40% with a probability of 0.9, whereas the NL heuristic allows a load of up to about 55%. Using this technique, an operator can thus reduce the capacity of the network by an expected 27% without a decrease in the quality of service provided to the customer! Moreover, according to phase transition theory, relative performance can be expected to scale in the same way to large networks. The latter is corroborated by another result on a series of larger problems, see Fig. 7. Noteworthy, BI-LL-NL solved a larger RAIN problem (50 nodes, 171 links, and 3'000 demands) in less than 6 minutes, whereas BT-SP was not able to solve it within 12 hours.

The advantages of the BI methods over naive shortest path

TABLE I

The comparison of a brute-force backtracking method to BI-based search methods when finding all solutions (6'504) on a small RAIN problem (a leased-line network), with 8 nodes, 9 links, and 18 demands.

Search method	Runtime [s]	Routes generated	Backtracks
BT-SP	191.140	795'425	788'921
BI-LL-HL	10.523	16'668	10'164
BI-LL-NL	8.466	10'694	4'190

allocation are best illustrated when searching all solutions of a RAIN problem, as shown in the comparisons of Table I for a small problem. Thanks to their much better pruning power and more purposeful search guidance heuristics, they are much faster (about 20 times), generate fewer routes (between 48 and 74 times) and backtrack even less (between 78 and 188 times).

All results were computed on a Sun Sparc 60.

B. QoS-routing

We also evaluated the route ordering heuristics in an *on-line* QoS-routing scenario. In this case, the demands are not known in advance and allocated one after the other (if possible) by a centralized state-based algorithm. Demand ordering heuristics and backtracking algorithms are then not applicable. We compared the three routing heuristics presented in Section IV (SP, LL, WP), and the results are summarized in Table II for the same 22'000 problems and 6'000 larger problems as in Section VI.

These results show that LL performs very similarly to SP: for the set of 22'000 problems, despite completely solving fewer problems (i.e., allocating all demands of a problem), LL allocated more demands in average than SP, had a better remaining bandwidth connectivity (the probability being able to allocate an additional new demand), and used fewer bandwidth resources. However, the differences are extremely slight. The same can be observed on the 6'000 larger problems: the only change is that LL solves more problems and uses more bandwidth, however still has a better bandwidth connectivity. The major difference between the two is run time: LL is more than twice faster than SP, despite the overhead of maintaining the BIH. We see two explanations for this: (1) if a demand can be allocated, LL allows to find a route faster because it limits the search space to the lowest BI clustering the endpoints of the demand. (2) LL knows before computing a route if one exists (route existence property), thereby saving time if a demand cannot be allocated, when SP has to explore the network graph before asserting that a demand cannot be allocated.

WP is clearly outperformed by both LL and SP in all domains. While allocating fewer demands (and solving just half of the problems that LL and SP solved), WP still uses more bandwidth resources. The run time is about twice longer than SP, which is easy to explain: routes are more expensive to compute for WP than for SP, because WP routes cannot be shorter than SP routes by definition. In fact, WP routes are often much longer, and therefore a bigger part of the network needs to be explored.

These experiments show that the DVO heuristics are very efficient for the RAIN problem, and that they are mainly responsible for the effectiveness of the proposed algorithms over existing methods. And even though LL performs very similarly to SP in the QoS-routing scenario, it combines better with these DVO heuristics.

TABLE II

Comparison of routing heuristics in a centralized QoS-routing scenario.

QoS method	Time [s]	Solved problems	Allocated demands	Bw. connectivity	Bw. usage
22'000 random problems with 20 nodes, 38 links, and 80 demands					
SP	0.292	39.40%	97.90%	81.80%	59.04%
LL	0.125	39.20%	97.93%	82.42%	58.99%
WP	0.414	20.90%	96.15%	80.95%	64.75%
6'000 random problems with 20 nodes, 38 links, and 200 demands					
SP	0.719	60.08%	99.49%	91.82%	59.11%
LL	0.307	60.71%	99.51%	92.05%	59.15%
WP	1.553	43.13%	98.55%	87.84%	71.74%

VII. Generalizing the Blocking Island paradigm to multiple concave metrics

A resource metric μ is said to be *concave* if for any path p, $\mu(p) = \min_{l \in p} \mu(l)$, where $\mu(l)$ is the metric for link l. Bandwidth is typically a concave resource, however there are others: for instance, the number of connections routed over a link may be limited, due to various factors, such as the number of allowed identifiers of connections over a link (e.g., virtual path/connection identifiers in ATM). The links of a communication network may belong to different operators. A demand may require to use links under contract of one or a set of operators. This requirement corresponds also to a concave metric.

When there are multiple concave metrics to take into account, it is possible to build a BIH for each metric and apply the presented techniques on one or the other BIH, or both. However, the BI paradigm is straightforwardly generalized to integrate multiple concave metrics into a connectivity cluster. In this context, a demand is defined by a triple $d_u = (x_u, y_u, Q_u)$, where $Q_u = [q_u^1, q_u^2, .., q_u^z]$ is an array of z concave QoS requirements. For a given metric, we say that $q_i^k < q_j^k$ if q_j^k is harder to satisfy than q_i^k, i.e., if a link supports a demand requiring q_j^k resources of the k-th metric, than it can support a demand requiring q_i^k. There is therefore a partial ordering on the QoS requirements. For instance, suppose we have two metrics for which both a higher value means a harder constraint (e.g., bandwidth). $Q_1 = [64, 12]$ and $Q_2 = [32, 20]$ are incomparable requirements, since a link with $[96, 12]$ available resources may accommodate a demand with QoS Q_1, but not Q_2, and a link with $[48, 30]$ free resources supports a demand requiring Q_2 but not Q_1. However, for a requirement $Q_3 = [16, 5]$, we have $Q_3 < Q_1$ (since $16 < 64$ and $5 < 12$), and $Q_3 < Q_2$ (id.).

Given a quality of service requirement $Q_u = [q_u^1, q_u^2, .., q_u^z]$, of which each corresponds to a concave metric, after a set of demands has been allocated, we call a *Connectivity Cluster* (CC) for a node x under Q_u, or the Q_u-CC for x, the set of all nodes of the network that can be reached from x through links respecting the QoS constraints Q_u, including x. A CC restricted to one concave metric is then a BI. CCs have the same properties as BIs (see Section III), such as unicity, route existence and location. A *Connectivity Cluster Graph* for a set of requirements Q_u (Q_u-CCG) is defined and built as the BIG. The

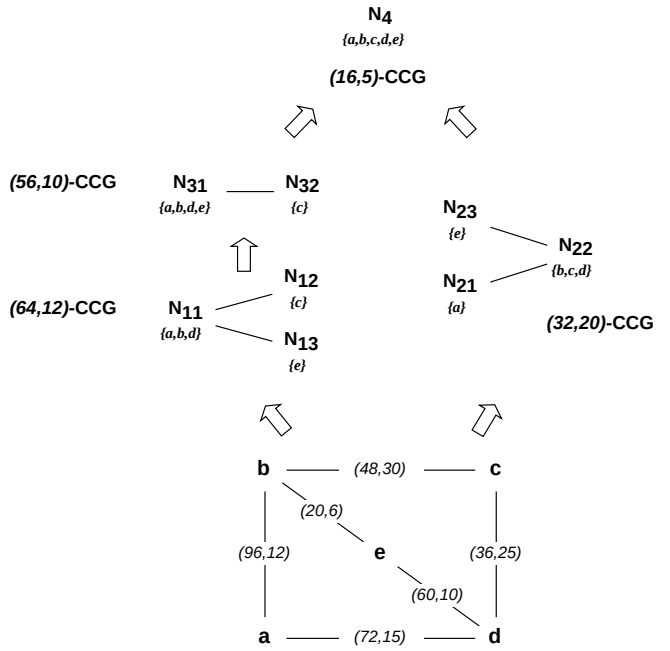


Fig. 8. The Connectivity Cluster Hierarchy of a small network with possible QoS requirements $\{[64, 12], [56, 10], [32, 20], [16, 5]\}$.

Connectivity Cluster Hierarchy (CCH) for a set of concave requirements sets is a generalized version of the BIH. Since there is only a partial order on the possible requirements sets Q_u, father-child relations define a lattice instead of a tree. Fig. 8 shows the CCH of a simple network for possible requirements $\{[64, 12], [56, 10], [32, 20], [16, 5]\}$. Two concave metrics are thus taken into account. Each node of the network has two fathers, one a $[64, 12]$-CC and one a $[32, 20]$-CC. As for the BIH, there is a null resource requirement $Q_0 = [0, 0]$ clustering the whole network (it is not displayed in Fig. 8 since equal to N_4).

The construction and maintenance of a CCH is more complex than for a BIH. Nevertheless, it allows to abstract resource availability for several resource requirements at the same time, and allows to use the heuristics for route and demand selection, with some adaptations. For instance, the LL heuristics selects the shortest route within the lowest BI clustering the endpoints of the demand. However, the lowest CC clustering the endpoints of a demand is not always univocally defined because of the partial order on the QoS requirements. For instance, suppose a demand $d = (b, d, [16, 5])$ in Fig. 8. Since its endpoints are clustered in the same $[16, 5]$-CC, we know there is at least one route satisfying d_u. However, which is the lowest CC, $[64, 12]$-CC N_{11} or $[32, 20]$-CC N_{22}? They both contain d's endpoints, but are in incomparable levels in the CCH. LL applied to N_{11} selects the route through a, whereas in N_{22} a route through c would be chosen. There are several solutions to this problem, for instance metric serialization or combination. *Metric serialization* amounts to impose a precedence over the metrics. If the first metric is preferred to the second, because it is considered more important to maintain the connectivity for it in the network, then LL applies to N_{11} because its value for the first metric is higher. Typically, if one metric is sharable (a resource is sharable if it can be simultaneously allocated to multiple consumers, e.g., the operator constraint) and the other not (e.g., bandwidth), metric serialization should be conducted according to the second metric, since it makes no sense to do load-balancing on sharable resources. *Metric combination* amounts to select the shortest route within the subgraph composed of the children of both candidates, in this case the subgraph restricted to nodes $\{a, b, c, d\}$. The same techniques can be applied for DVO-HL and DVO-NL.

VIII. Conclusion and future work

The current technique for routing communication demands in a network is to select the shortest route for each particular demand. However, this strategy can lead to suboptimal routing or even highly congested network utilization as a whole. Information about the expected traffic allows to make better use of network resources. However, on-line routing processes cannot make use of this knowledge since they must be very fast to ensure customer satisfaction. Instead, bandwidth allocation can be planned in an off-line manner with this information, thanks to a systematic search algorithm that is capable of backtracking to faulty routing decisions in order to satisfy all demands. However, search must be guided carefully since the search space to explore is exponentially large.

We have shown that using blocking island abstractions coupled with CSP exhaustive search mechanisms and heuristics, it is possible to solve the bandwidth allocation planning problem in reasonable time, in particular when using the lowest level heuristic for route generation and the number of levels heuristic for demand selection. We also get better solutions than shortest path routing algorithms, markedly in terms of the remaining bandwidth connectivity in the network after all demands have been allocated. This is especially useful when another unexpected demand needs to be routed, since the likelihood of being able to route it without recomputing a full solution from scratch is higher. Network operators (or service providers) can now plan the allocation of bandwidth in much tighter networks and more often than before. Noteworthy, it is possible to simulate link and node failures, and, using the same technique, compute alternative routing plans for these situations. An online demo illustrating these features is available on the WWW at `http://www.iconomic.com`. Moreover, we have proposed a generalization of the BI paradigm, connectivity clusters, to take into account multiple concave QoS metrics. We note that a patent for the methods described in this paper is pending.

In this paper, the performance of the proposed heuristics where only analyzed on solvable problems. When the allocation problem is not feasible, the potential problem is that search can take a very long time without producing an answer. To avoid this, the blocking island paradigm allows detecting unsolvability by comparing total demand to total capacity available in and out of a blocking island. When such unsolvability is detected, it also pinpoints where to either remove demands or add capacity to make the problem solvable again. Preliminary results show that this technique works very well. The evaluation of the proposed search algorithm on infeasible problems, however, lies outside the scope of this paper and is left for a later article.

The presented techniques are not only applicable to connection-oriented networks (such as ATM), but also to connection-less networks (such as IP). In a connection-less network, demands can be derived from traffic statistics between

nodes. If a solution can be found, applying it will prevent congestion in the network, or at least reduce its probability in case of unexpected traffic. The only difference to connection-oriented networks is in the route generation process, since IP uses hop-by-hop routing tables. The additional constraint of routing being implementable by hop-by-hop routing tables can be formulated in CSP and taken into account during search. Even if the route existence property does not hold in this context, its natural counterpart, the route *inexistence* property does. Nonetheless, we are confident that solving a RAIN problem for an IP network will be as efficient as for the connection-oriented case.

In this paper, we restricted demands to point-to-point traffic. However, the same techniques can be applied for multipoint demands: routes are then trees instead of simple paths. Generalizing the presented heuristics, such as the lowest level (LL) for route generation or the number of levels for demand selection (DVO-NL), to multipoint demands is straightforward.

Current and future work is conducted along five axis:

1. We are concerned with the explanation of allocation failures during search (dead-end), in order to determine the culprit assignment. This can be realized by examining the demands routed over the cocycles of the blocking islands (the cocycle of a subset of nodes A is the set of all links that have one and only one endpoint in A). When such a culprit can be identified, we are then able to directly *backjump* to the cause of the failure, without having to explore (pruning) parts of the search space that do not contain any solution, thereby increasing search efficiency. Backjumping algorithms [22] are widely used in the CSP community nowadays.
2. We want to develop more sophisticated heuristics for route generation and demand selection during search, using more information that can be derived from the BIH, again to increase search efficiency.
3. We are developing new types of hierarchies that decompose even better the network according to resource availability. One major goal is here to get rid of the fixed levels in the hierarchy that depend on the possible QoS requirements of the demands.
4. Taking into account other QoS constraints of demands (such as delay or loss probability) or network element limitations (e.g., node buffer capacity) is also one of our main concerns. CSP modeling has the facility to easily take such additional restrictions into account, by just adding these additional constraints "as is". CSPs have in this case a major advantage over Operations Research techniques, that do not allow the integration of new constraints in such a straightforward manner. Another approach is the connectivity cluster paradigm, a generalization of blocking island abstractions to multiple concave metrics into a single and concise representation of resource availability. Even if not concave, delay can be approximated as a concave metric in case there are large variations from one link to another: if the delay varies a lot from one link to another (for instance a satellite link vs. an optical fiber), the delay of a route is close to the delay of the slowest link (the satellite link).
5. We intend to adapt the techniques to perform on-line routing with intelligent agents, as presented in [20]. The idea is to assign one agent to each BI, each agent then being responsible for the allocation of demands inside its domain, thereby possibly cooperating with agents below it in the hierarchy.

ACKNOWLEDGMENTS

This work is partly a result of the IMMuNe (Integrated Management for Multimedia Networks) Project, supported by the Swiss National Science Foundation (FNRS) under grant 5003-045311. The authors are grateful to Dean Allemang, Beat Liver, Steven Willmott, and Monique Calisti for their invaluable comments, suggestions, and encouragements during the last years. The authors also wish to thank George Melissargos and Pearl Pu for their work on the GUI of the developed tool.

REFERENCES

[1] Z. Wang and J. Crowcroft, "Quality-of-Service Routing for Supporting Multimedia Applications," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 7, pp. 1228–1234, 1996.

[2] E. Tsang, *Foundations of Constraint Satisfaction*, Academic Press, London, UK, 1993.

[3] R. J. Wallace, "Practical Applications of Constraint Programming," *Constraints. An International Journal*, pp. 139–168, 1996.

[4] F. Giunchiglia and T. Walsh, "A Theory of Abstraction," *Artificial Intelligence*, vol. 57, pp. 323–389, 1992.

[5] "Symposium on Abstraction, Reformulation and Approximation (SARA-98)," Supported in Part by AAAI, Asilomar Conference Center, Pacific Grove, California, May 1998.

[6] G. R. Ash, *Dynamic Routing in Telecommunications Networks*, McGraw Hill, 1998.

[7] S. Cosares and I. Saniee, "An optimization problem related to balancing loads on SONET rings," *Telecommunication Systems*, vol. 3, pp. 165–181, 1994.

[8] M. Herzberg, D. J. Wells, and A.Herschtal, "Optimal Resource Allocation for Path Restoration in Mesh-Type Self-Healing Networks," *International Teletraffic Congress (ITC)*, vol. 15, pp. 351–360, 1997.

[9] T.-H. Wu, *Fiber Network Service Survivability*, Artech House, Boston - London, 1992.

[10] J. W. Mann and G. D. Smith, "A Comparison of Heuristics for Telecommunications Traffic Routing," in *Modern Heuristic Search Methods*. 1996, pp. 235–254, John Wiley & Sons Ltd.

[11] S. Vedantham and S. S. Iyengar, "The Bandwidth Allocation Problem in the ATM network model is NP-Complete," *Information Processing Letters*, vol. 65, pp. 179–182, 1998.

[12] M. S. Miller, D. Krieger, and N. Hardy, "An Automated Auction in ATM Network Bandwidth," in *Market-Based Control: A Paradigm for Distributed Resource Allocation*, pp. 96–125. World Scientific, 1996.

[13] B. T. Messmer, "A framework for the development of telecommunications network planning, design and optimization applications," Technical Report FE520.02078.00 F, Swisscom, Bern, Switzerland, 1997.

[14] B. Y. Choueiry and B. Faltings, "A Decomposition Heuristic for Resource Allocation," in *Proceedings of the 11 th European Conference on Artificial Intelligence, ECAI-94*, 1994, pp. 585–589.

[15] R. Weigel and B. Faltings, "Structuring techniques for constraint satisfaction problems," in *Proc. of the 15 th International Joint Conference on Artificial Intelligence, IJCAI-97*, 1997, pp. 418–423.

[16] E. C. Freuder and D. Sabin, "Interchangeability Supports Abstraction and Reformulation for Multi-Dimensional Constraint Satisfaction," in *Proc. of the 15 th National Conference on Artificial Intelligence, AAAI-97*, 1997, pp. 191–196.

[17] P. Cheeseman, B. Kanefsky, and W. M. Taylor, "Where the Really Hard Problems Are," in *Proceedings of the 12 th International Joint Conference on Artificial Intelligence, IJCAI-91*, 1991, pp. 331–337.

[18] I. P. Gent, E. MacIntyre, P. Prosser, and T. Walsh, "Scaling Effects in the CSP Phase Transition," in *First International Conference on Principles and Practice of Constraint Programming (CP'95)*, 1995, pp. 70–87.

[19] I. P. Gent and T. Walsh, "Computational Phase Transitions from Real Problems," *In Proceedings ISAI-95*, pp. 356–364, 1995.

[20] C. Frei and B. Faltings, "A Dynamic Hierarchy of Intelligent Agents for Network Management," in *2 nd International Workshop on Intelligent Agents for Telecommunications Applications, IATA'98*. 1998, pp. 1–16, LNAI 1437, Springer-Verlag.

[21] R. M. Haralick and G. L. Elliott, "Increasing Tree Search Efficiency for Constraint Satisfaction Problems," *Artificial Intelligence*, vol. 14, pp. 263–313, 1980.

[22] P. Prosser, "Hybrid Algorithms for the Constraint Satisfaction Problem," *Computational Intelligence*, vol. 9, no. 3, pp. 268–299, 1993.