Resource Allocation in Communication Networks Using Abstraction and Constraint Satisfaction

Christian Frei, Boi Faltings, and Mounir Hamdi, *Member, IEEE*

***Abstract*—The fundamental issue of quality-of-service (QoS) routing has triggered a lot of research during the last few years. However, the proposed algorithms attempt to route communication demands only on a call by call basis, without taking into account future traffic. There are nonetheless cases where the traffic profile is known. In this paper, we address this related problem to QoS routing, more specifically, the off-line planning of bandwidth allocation to demands known in advance. 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. In addition, this abstraction technique also allows to explain during search why some allocation problems are indeed infeasible. Then, the network regions between which bandwidth must be added are then identified.**

***Index Terms*—Abstraction, constraint satisfaction, constraint-based routing, quality-of-service (QoS) routing, resource allocation.**

I. Introduction

A CENTRAL problem in the field of communications is the automatic routing of traffic through a network. Currently, shortest-path (SP) routing, according to some metric, is most often used to route traffic across a network on a call by call basis. However, high-speed networks can offer a wide range of services to an increasing number of users, with a diverse range of quality-of-service (QoS) requirements. This has raised the issue of QoS routing of communication demands: the goal is now not only to achieve global efficiency in network resource utilization, but also to satisfy the QoS requirements of each admitted connection.

The proposed routing algorithms for this network environment do not make any assumption about future traffic. However, there are cases in which the traffic profile is already known. As a result, using *global information*, including not only the available link capacities but also the expected traffic profile for the period in question, can lead to routing strategies designed to minimize congestion and make better use of available network resources. 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., asynchronous transfer mode (ATM) and wavelength-division multiplexing (WDM)]; planning the routing of virtual path connections (VPCs) in an ATM network; planning the routing of virtual channel connections (VCCs) in the VPC network of an ATM backbone; or optimizing the routing tables of an Internet protocol (IP) network.

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. We define the problem of *resource allocation* in *networks* (RAIN) as follows.

- Given a network composed of nodes (e.g., switches, routers) and bidirectional links (e.g., copper wires, optical fibers), where each link has a given bandwidth availability, and a set of communication demands to allocate (the QoS metric that we consider in this paper is bandwidth), where each demand is defined by a triplet: (*source node, destination node,* and *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. (Assigning a route to a demand and reserving the resources needed is called establishing a *connection*.)

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. (Nonetheless, when there are several demands between the same endpoints, each of these demands can be allocated over a different route.)

With this restriction, the RAIN problem is NP-complete in the number of demands [1]. The RAIN problem is combinatorial due mainly to the exponential number of routes in a network. Suppose the network is simple but complete (this is not even the worst case, since a communication network is a multigraph: 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 (and distinct) nodes, the number of routes of length j is

Manuscript received December 1, 2003; revised May 15, 2004. This work was supported in part by the CTI Project 2831.1 in collaboration with Swisscom, and is partly a result of the Integrated Management for Multimedia Networks (IMMuNe) Project, supported in part by the Swiss National Science Foundation (FNRS) under Grant 5003-045 311. This paper was presented in part at INFOCOM 2000, Tel-Aviv, Israel, March 2000. (We note that a patent for the methods given herein is pending.)

C. Frei is with the ABB Corporate Research Center, CH-5405 Baden, Switzerland (e-mail: cfrei@acm.org).

B. Faltings is with the School of Computer and Communication Sciences, Ecole Polytechnique Federale de Lausanne (EPFL), CH-1015 Lausanne, Switzerland (e-mail: Boi.Faltings@epfl.ch).

M. Hamdi is with the Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong (e-mail: hamdi@cs.ust.hk).

Digital Object Identifier 10.1109/JSAC.2004.839377

$(n-2)!/(n-j-1)!$. The total number of routes between two nodes is, therefore, equal to $\sum_{i=1}^{n-1}(n-2)!/(n-i-1)!$. For instance, in a complete graph with ten nodes, there are 69 281 routes between any pair of nodes.

Currently, most network or service providers use SP-based algorithms, without any backtracking on routing decisions, 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. Although the use of SP routing for each single demand ensures the best possible route for that particular request, Wang and Crowcroft [2] have shown that it can lead to suboptimal routing or even highly congested network routing solutions when one considers the network utilization as a whole.

In order to do better, we must allow: 1) other routes than the shortest paths and 2) *backtracking* to previous allocations in order to squeeze in more demands. Basically, two types of algorithms can be used to solve the RAIN problem: incomplete and complete methods. *Incomplete* algorithms, such as the SP method, explore only partially the search space, i.e., they only consider a subset of possible routes. These algorithms are generally fast, however, they are not guaranteed to find a solution if there is one. On the other hand, *complete* algorithms perform an exhaustive search, and thereby do always find a solution if one exists. However, due to the huge search space, their worst case behavior is exponential. There is, therefore, a need for heuristics to guide the search, in order to solve most problem instances in a reasonable amount of time. To achieve that, we make use of constraint satisfaction techniques in conjunction with novel abstraction methods to solve the RAIN problem using complete search algorithms.

Constraint satisfaction [3] is an artificial intelligence technique which has been shown to work well for solving a variety of NP-hard problems [4]. A *constraint satisfaction problem* (CSP) is defined by a triple (X, D, C), where $X = \{x_1, \ldots, x_n\}$ is a set of *variables*, $D = \{D_1, \ldots, D_n\}$ a set of finite *domains* associated with the variables, and $C = \{C_1, \ldots, C_n\}$ 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. Indeed, 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, because of the exponential number of routes. It is thereby too expensive to compute, represent, and store the domain of a variable, i.e., all the routes that join the endpoints of a demand.

In this connection, we show how an abstraction of the network called *blocking islands* (BIs), create a compact representation of the domains which allows the application of 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 present the BI paradigm and outline its major properties. Section IV illustrates how BIs 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. Section VI examines how BI abstractions allow to prove in some cases that the problem is infeasible by identifying global bottlenecks in the network, or to identify culprit assignments of routes to demands that prevent the allocation of another demand. Empirical results are summarized in Section VII. The BI paradigm is generalized to multiple link constraints in Section VIII. Applying the abstraction technique to the routing and wavelength assignment problem in optical networks is explored in Section IX. Section X concludes the paper.

II. Related Work

Surprisingly, there has been little published research on the RAIN problem defined in this paper. Currently, most network providers use (incomplete) SP methods, as described in Section I, due to the complexity of the problem.

Operation research (OR) techniques are also applied to the RAIN problem. Most often, a fixed number of shortest paths for each demand are precomputed, and the problem is solved using linear programming with very large constraint systems of equations [5]–[7]. However, since 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.

Mann and Smith [8] search for routing strategies that attempt to ensure that no link is overutilized (hard constraint) and, if possible, that all links are evenly loaded (below a fixed target utilization), for the predicted traffic profile. Then, 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 multicriteria objective function they use cannot ensure that the hard constraint, i.e., no link is overutilized, 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.

To the best of our knowledge, the closest published work to ours is the CANPC framework [9]. It is based on the successive allocations of shortest routes to 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. Results will demonstrate that the methods we propose clearly outperform theirs.

Wang and Crowcroft [10] proved that the allocation of every single demand is NP-hard by itself, when demands are subject to multiple additive or multiplicative QoS criteria (such as delay and loss probability). This triggered much research during the last years and the proposed QoS routing algorithms are mainly variations of the SP method [11].

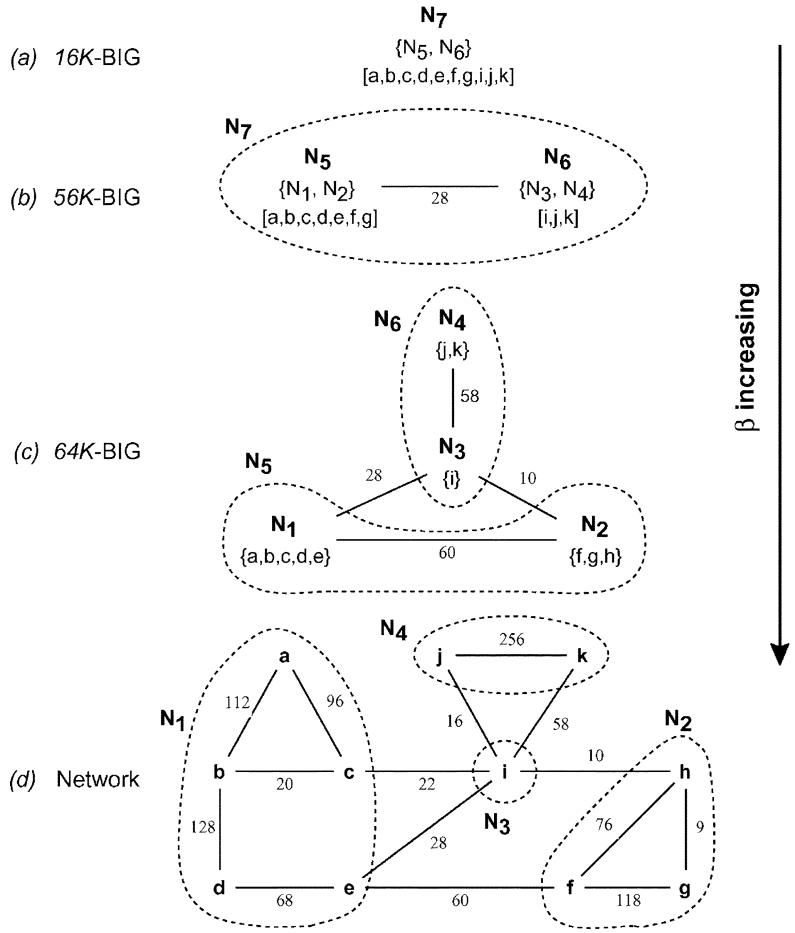


Fig. 1. BIH for resource requirements $\{64, 56, 16\}$. The weights on the links are their available bandwidth. Abstract nodes' description include only their node children and network node children in brackets. Link children (of BIs and abstract links) are omitted for more clarity, and the 0-BI is not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

Vedantham and Iyengar [12] 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.

Abstraction and reformulation techniques have already been applied to permit more efficient solution of a CSP (not specifically applied to communication networks). Choueiry and Faltings [13] relate interchangeability to abstraction in the context of a decomposition heuristic for resource allocation. Weigel and Faltings [14] 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 [15] present abstraction and reformulation techniques based on interchangeability to improve solving CSPs. A recent collection of papers addressing abstraction, reformulation, and approximation techniques can be found in [16].

III. Blocking Island (BI) Paradigm

Before the presentation of our proposed method to the RAIN problem, we first overview the BI paradigm which is key to our solution to this problem. Frei and Faltings [17] introduced a clustering scheme based on BIs, which can be used to represent network 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 64-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 64-BI N_1 but has less than 64 available resources. However, there are at least 64 available resources along route $\{(b, a), (a, c)\}$.

β-BIs have some fundamental properties, discussed in detail in [1]. Given any resource requirement, BIs partition the network into equivalence classes of nodes. The BIs are *unique*, and *identify global bottlenecks*, that is, inter-BI links. If inter-BI links are links with low remaining resources, as some links inside BIs may be, inter-BI 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

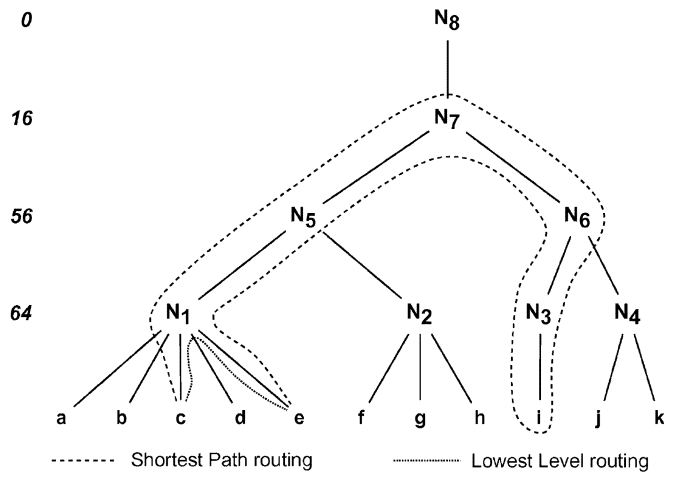


Fig. 2. Abstraction tree of the BIH of Fig. 1 (links are omitted for clarity), as well as the mapping the SP and the LL routes onto the abstraction tree.

demand $d_u = (x, y, \beta_u)$ if and only if its endpoints x and y are in the same β_u-BI. Furthermore, all links that could form part of such a route lie inside this BI.

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.

BIs 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 BI's 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: $\beta_1 > \beta_2 > \cdots > \beta_b$. This layered structure of BIGs is a blocking island hierarchy (BIH). The lowest level (LL) of the BIH 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). Fig. 1 shows such a BIH for resource requirements $\{64, 56, 16\}$. 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 cannot 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 the BIH in Fig. 1.

The β-BI S for a given node x of a network graph can be obtained by a simple greedy algorithm, with 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 $O(bm)$. 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 contains at most $bn + 1$ BIs: 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 the same BIs are clustered into a single abstract link. The memory storage requirement of a BIH is, thus, bound by $O(bn^2)$.

A BIH summarizes the available resources given the currently established connections at time t. As demands are allocated or deallocated, available resources on the links change and the BIH may need to be modified to reflect this. The changes can be carried out incrementally, only affecting the BIs which participate in the demand which is being allocated or deallocated, again with complexity $O(bm)$ [1]. They mainly amount to *splitting* or *merging* BIs. These incremental algorithms may also be applied to update the BIH in case of link failure, changes in link properties, introduction of new bandwidth requirements or removal of existing ones, or even network topology modifications.

IV. Routing From a BI Perspective

Consider the problem of routing a single demand $d_u = (c, e, 16\text{ K})$ in the network of Fig. 1(d). Since c and e are clustered in the same 16 K-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 in this example is 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 64 K-BI's N_1 and N_2 in the 64 K-BIG of Fig. 1(c). After that allocation, no other demand requiring 16 K (or more) between any of the nodes clustered by 56 K-BI N_5 and one of the nodes inside 56 K-BI N_6 can be allocated anymore. For instance, a demand $(c, i, 16\text{ K})$ 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 LL in the BIH, that is in 64 K-BI N_1, and no critical links (that is inter-BI links).

r_L is a route that satisfies the 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. 2: r_S is by far then the longest route, since its mapping traverses BI's 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 than r_L, but also many more BIs, and its allocation may cause to split each one of them. This observation (also) justifies the LL heuristic.

Even better, a BIH also gives 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 BIs in the BIH: obviously, the more splittings, the more links become 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 routing. For instance, Wang and Crowcroft [10] 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 WP), and when there are more than one WP, choose the one with shortest propagation delay (in our case the number of hops). In routing the same demand d_u as above, WP would select 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 the case of QoS routing (Section VII-C).

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) [3], a systematic search process. Such an algorithm maintains a partial solution (initially empty) which satisfies all the constraints and attempts to extend it to a full solution. Its basic operation is to pick one variable (demand) at a time, assign it a value (route) from its domain, 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 (the exponential number of routes for a demand—see Section I). Nonetheless, BIs 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 nonempty. 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.

We use this mapping of routes onto the BIH to formulate a forward checking criterion, as well as dynamic value ordering and dynamic variable ordering heuristics.

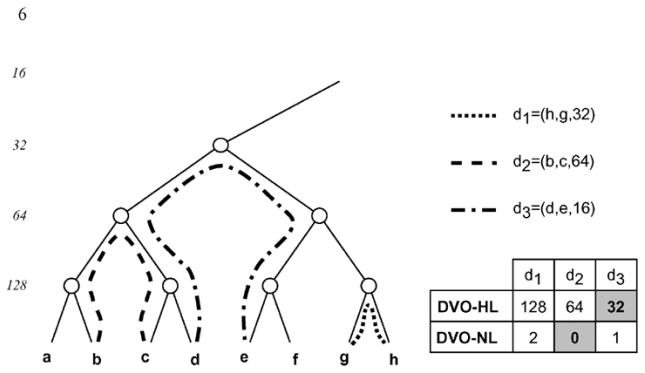
A. Forward Checking (FC)

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 is, therefore, 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 the "most beneficial" order, so as 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 SP order, 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 LL heuristic. Applied to the RAIN problem, it amounts to considering first, in shortest order, the routes in the lowest BI (in the BIH). Apart from attempting to preserve bandwidth connectivity, the LL heuristic allows to



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

Fig. 3. Selecting the next demand to allocate using DVO-HL and DVO-NL. The DVO-HL line shows the value of 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.

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. Moreover, the LL heuristic is especially effective during the early stages of the search, since it allows to make better decisions and, therefore, has a greater pruning effect on the search tree, as shown by the results in Section VII.

C. Variable Ordering

The selection of the next variable to assign may have a strong effect on search efficiency, as shown by Haralick and Elliot [18]. 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 idea 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-H Highest Level: First choose the demand whose lowest common father of its endpoints is the highest in the BIH (remember 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 is, 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-N 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. 3. In the implementation, both 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 LL 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.

```
Function FCRAIN (D, Γ, H)
  if D = ∅ then
    /* all demands allocated: 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. 4. 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 BIH. It returns either a set of connections Γ (a solution) or $\emptyset$ (in which case there is no solution).

There are numerous other dynamic variable ordering (DVO) heuristics that can be derived from analyzing the BIH. For instance, let β_d-BI be the blocking island clustering the endpoints of a demand d at its bandwidth requirement level β_d. Then, select first the demand d for which the β_d-BI contains the fewest nodes. This heuristic is called DVO-N. It follows the fail-first principle, since one can assume that the fewer nodes are, the fewer routes connect the endpoints of the demand. Two similar heuristics, with same justification, are to select first the demand d for which the β_d-BI contains the fewest links (DVO-L) or the lowest link density (DVO-D). An evaluation of these heuristics (and others) can be found in [1].

D. FC Algorithm for the RAIN Problem

Now that the different parts of the RAIN problem have been examined, we can put everything together into a systematic search algorithm. Fig. 4 shows a pseudocode for a recursive FC backtracking algorithm FCRAIN. FCRAIN can be explained 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 FC: 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. In order to be able to compute the next best route, the state of the route generators (one for each demand already allocated) is saved, so that when backtracking occurs, the search process can be resumed to get to the next best solution. Obviously, the route generator of a demand from which we backtrack is discarded: when that demand has to allocated again, a new route generator is started for that demand.

For the algorithm in Fig. 4, the worst case complexity is exponential in the number of demands, since we perform a backtrack search over them. However, the average case complexity is much lower, as has been shown in general for the backtrack search using CSP heuristics in constraint satisfaction problems.

VI. Conflict Identification and Resolution

BIs, and especially BIGs, identify *global bottlenecks* in the network, as shown for instance in Fig. 1(c) for 64 demands. The abstraction of the network into BIs allow to take measures when a demand cannot be allocated (because its endpoints are not clustered in the same BI at its bandwidth requirement level).

Suppose a set of demands $\mathcal{D}$ were allocated in the network and that the network's available resources is given by Fig. 1(d). Now, we are to allocate a new demand $d_n = (c, h, 64)$. Since c and h are not clustered in the same 64-BI, it is impossible to satisfy d_n without rerouting some previously allocated demands. There are two cases.

Case 1) It may be that the problem of allocating $\mathcal{D} \cup \{d_n\}$ is in fact unsolvable.

Case 2) Rerouting one or more already allocated demands may resolve the problem.

A. Unresolvable Problem Identification

Given the situation explained above where d_n cannot be allocated, it is possible, in some cases, to prove that the RAIN problem is unsolvable by approximating it with a network multiflow problem. If there is any cut in the network whose capacity is smaller than the sum of the bandwidth requirement of the demands that have one endpoint on each side of the cut. However, there is an exponential number of possible cuts in a network and examining them all is intractable. Nonetheless, BIs identify critical parts of the network and, therefore, it is a good idea to restrict the search for infeasibility to only the cocycle[1] of some BIs.

We call primary blocking islands (PBI) of a demand that cannot be allocated the two BIs of its endpoints at its bandwidth requirement level. For d_n, the PBIs are N_1 and N_2. If for any of the two PBIs the sum of the bandwidth requirements of the demands that have one and only one endpoint inside the PBI is higher than the capacity of the links of the PBI's cocycle, then it is obvious that the problem is infeasible. We call this unresolvable conflict detection (UCD). Search can then be aborted, thereby saving a lot effort. Note that if a problem is infeasible, it does not mean that unsolvability can be always be proven that way, because the RAIN problem is not a network flow problem. By examining only the PBIs of demands that cannot be allocated, we overcome the intractability issue raised above, while still being able to find such a cut with high probability.

Moreover, when the problem can be proven infeasible, the BIH helps the human operator in deciding where to add bandwidth resources, since global bottlenecks are identified. In the above case, in order to be able to satisfy d_n, we have to add bandwidth resources between the two PBIs, that is between one node of N_1 and one node of N_2, since routing inside the BIs of the bandwidth requirement level does not pose any problem. For instance, suppose the network is the one of a service provider. Depending on link prices, the service provider may buy some more resources from a network operator either between e and f, or directly between c and h (link addition), or even along a longer path, for instance between a and j, and i and f. The BI then provide meaningful abstractions of the network by hiding "easy" parts of the network and only showing to a human user the real problems.

B. Culprit Assignment Identification

In a classical backtracking algorithm, when a dead-end is reached, the most recent assignment is undone and an alternate value for the current variable is tried. In case all values have been unsuccessfully assigned, backtracking occurs to the preceding allocation. However, if we have the means to identify a culprit assignment of a past variable, we can directly *backjump* to it (the intermediate assignments are obviously undone), thereby possibly drastically speeding up the search process [19].

In case infeasibility cannot be proven when a dead-end is reached while solving a RAIN problem, analyzing the situation on the links of the PBI's cocycle indeed helps to identify a *culprit assignment*. In this case, we either find that one or more demands were at fault by being routed through at least one link on the BI's cocycle, or that there is in fact no solution. The latter cannot be established, however, a culprit assignment can be identified by analyzing the demands routed over the cocycle of the PBI. There are two possibilities.

1) The sum of the bandwidth requirements of all unallocated demands that have one and only one endpoint in the PBI is less than the sum of the available bandwidth on the links of the PBI's cocycle. This means that there is at least one already allocated demand that is routed over more than one link of the PBI's cocycle, thereby using up many critical resources. We, therefore, have to backjump to the point in the search where the total available bandwidth on the cocycle was enough to support all unallocated demands that have one and only one endpoint in the PBI.
2) Reallocating some of the demands that are routed over the PBI's cocycle over different routes may suffice to solve the problem. Therefore, the most recent culprit assignment is the latest demand that is routed over the PBI's cocycle.

[1]The cocycle of a subset of nodes A is the set of all links that have one and only one endpoint in A.

VII. Empirical Results

We report here some empirical results on solving feasible and impossible RAIN problems, as well as in a centralized on-line QoS routing scenario. We further introduce an *a priori* order parameter that attempts to capture the difficulty of solving a RAIN problem. All results were obtained on a Sun Ultra 60 with a LISP program.

A. Solving Feasible RAIN Problems

In practice, the RAIN problem poses itself in the following way. A network or service provider receives a request from the customer to allocate a number of demands, and must decide within a certain time decision threshold whether and how the demands can be accepted. The tradeoff we must consider is then completeness (that is satisfying all users if possible in order to ensure customer satisfaction) versus time efficiency (customers require an answer in a reasonable amount of time), and hereby complete versus incomplete algorithms. A meaningful analysis of the performance of the heuristics we propose would, thus, analyze the probability of finding a solution within a given time limit, and compare this with the performance that can be obtained using conventional methods, in particular SP 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 defined as 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. 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 23 000 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 assume 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 six different strategies: *basic-SP* performs a search using the SP heuristic, without any backtracking on decisions; *BT-SP* incorporates backtracking 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. *BI-BJ-LL-HL* and *BI-BJ-LL-NL* differ from the previous ones in the use of backjumping to culprit decisions, as described in Section VI-B.

Fig. 5(a) gives the probability of finding a solution to a problem in less than 1 s, given the tightness of the problems. 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. Backjumping methods show slightly better performance over their purely backtracking counterparts. The benefits of backjumping seem to be somewhat canceled by the computing overhead associated with calculating the culprit assignment, at least on these small problems. On problems of much larger size, we noticed that BJ algorithms do perform much better in average than their nonbackjumping counterparts. 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 are similar for larger time limits.

Fig. 5(b) provides the probability of solving a problem according to runtime. Two conclusions can be derived. First, BI methods curves continue to grow with time, albeit slowly, which is not the case for basic-SP (and obviously BT-SP). Second, maintaining the BIH on these small problems does not affect the BI algorithms very much. The quality of the solutions, in terms of network resource utilization, were about the same for all methods. However, when the solutions were different, bandwidth connectivity was generally better on those provided by BI methods.

The experimental results allow us to quantify 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 in Fig. 5, we can see that for the SP methods, they allow 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 QoS provided to the customer!

In addition, the relative performance can be expected to scale in the same way to large networks. This is corroborated by the results on larger problems. We generated 6000 different RAIN problems with 38 nodes, 107 links, and 1200 demands. The probability of solving such a problem according to runtime is given in Fig. 6. Here, we see that the maintenance of the BIH has a larger effect on solving "easy" problems. However, after 20 s of runtime, the BI methods clearly are more efficient than the non-BI techniques. The facts noticed for the smaller problems (Fig. 5) remain valid: NL outperforms HL, even if it requires slightly more runtime, and backjumping brings only a small advantage over their nonbackjumping counterparts. Further, as an aside result, BI-BJ-LL-NL solved a much larger RAIN problem (50 nodes, 171 links, and 3000 demands) in 4.5 min. BT-SP was not able to solve it within 12 h.

The advantages of the BI methods over naive SP 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).

B. Performance Evaluation on Infeasible RAIN Problems

In order to test the capability of identifying in some cases that a RAIN problem is infeasible, we generated problems using an existing network topology, and randomly generated

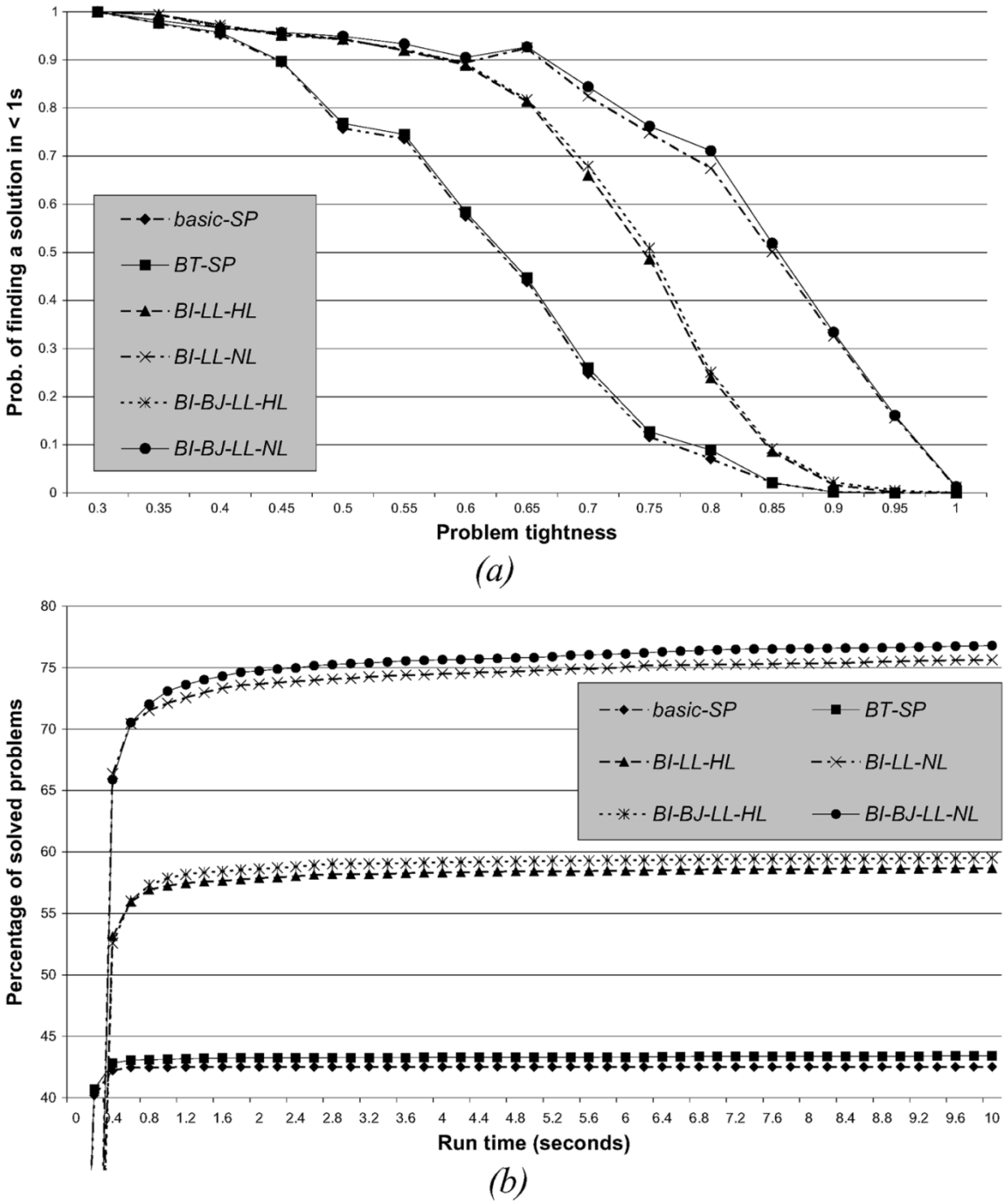


Fig. 5. Statistics on solving 23 000 randomly generated solvable problems with 20 nodes, 38 links, and 80 demands each. (a) Probability of finding a solution within 1 s, given the tightness of the problems. (b) Probability of solving the problems according to runtime.

demands until we could prove that it was infeasible (within 20 min of runtime), either by identifying an unresolvable conflict (as described in Section VI-A), or by having to explore the whole search space. The network topology used is the FUDI subnetwork.[2] covering the main cities in Europe. It comprises 12 nodes and 14 links, each link with a bandwidth capacity between 6 and 34 Mb/s.

We generated two sets of problems based on the FUDI subnetwork.[3]

- For the 300 problems of the *FUDI 1 set*, we randomly generated demands with relatively high bandwidth requirements with respect to the link capacities. The bandwidth requirement of a demand was uniformly distributed in $\{5, 6, 7, 8, 9, 10\}$ (Mb/s). For each problem, there are between 20 and 24 demands. The maximal bandwidth a de-

[2]See http://www.caida.org/Tools/Mapnet/Backbones.

[3]There are no "trivial" unsolvable problems in these sets, such as a node for which the total bandwidth capacity of the links adjacent to the node exceeds the total requirement of the demands of which the node is an endpoint.

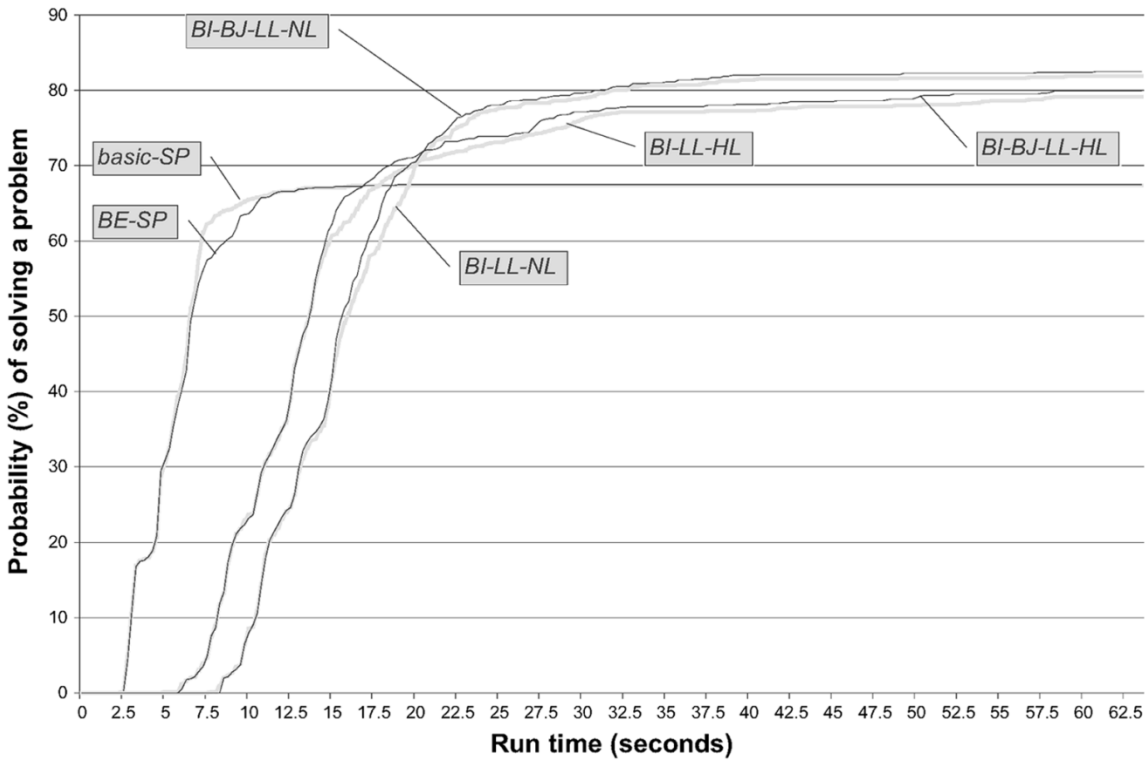


Fig. 6. Probability of solving a problem according to runtime (6000 problems, with 38 nodes, 107 links, and 1200 demands).

mand could ask for is, therefore, higher than the lowest link capacity. The goal was to make it easier for the BT-SP algorithm to identify the problems as impossible. These problems can be considered as "easy," since their search space is small.

- The *FUDI 2 set* of 300 problems is intended to be more difficult to solve. The bandwidth requirements of the demands are much smaller compared with the link capacities. The bandwidth requirement of a demand is uniformly distributed in $\{0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$ (Mb/s). There are between 78 and 164 demands for each problem. The search space of these problems is, therefore, much bigger than for those of the first set.

We compare the same algorithms as in Section VII-A, but instead of BI-LL-HL and BI-LL-NL, we use *BI-U-LL-HL* and *BI-U-LL-NL*, which are the same as BI-LL-HL and BI-LL-NL, respectively, however, incorporating unresolvable conflict detection (UCD). Note that BJ methods incorporate UCD. Basic SP is not taken into account here, since it does not perform a complete search and, therefore, cannot prove that a problem is unsolvable. Recall that BT-SP must perform an exhaustive search in order to conclude that the problem has no solution.

The percentage of problems identified infeasible according to runtime is given in Fig. 7. Fig. 7(a) displays the statistics for FUDI 1 set, and Fig. 7(b) displays the results for FUDI 2 set. For the set of easy problems [FUDI 1 set—see Fig. 7(a)], we also display the percentage of problems identified as impossible thanks to the UCD mechanism. Since these statistics are very close for nonbackjumping and backjumping algorithms, only those for BI-U-LL-HL and BI-U-LL-NL are depicted. The UCD

TABLE I
COMPARISON IN TERMS OF RUNTIME(S), GENERATED ROUTES, AND BACKTRACKS OF A BRUTE-FORCE BACKTRACKING METHOD TO BI-BASED SEARCH METHODS WHEN FINDING ALL SOLUTIONS (6504) ON A SMALL RAIN PROBLEM (A LEASED-LINE NETWORK), WITH 8 NODES, 9 LINKS, AND 18 DEMANDS

Search method	Time (sec)	Routes	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

mechanism performs only slightly better with the backjumping algorithms (BI-BJ-LL-HL and BI-BJ-LL-NL). The difference between the UCD curve and the curve of the associated solving method identify the percentage of problems proven infeasible by exhaustive search.

BI-based algorithms with UCD logically outperform BT-SP, which has to perform an exhaustive search before declaring a problem unsolvable. The performance of BT-SP is only acceptable on the easy problems (FUDI 1 set), where the search space is reasonable in size. On the second set of problems (FUDI 2 set), the search space gets too big to be manageable by BT-SP despite the small network size. Even after 6 mins, not a single problem was declared unsolvable by this method. The size of the search space for the problems of FUDI 2 set also prevents the exhaustive search by Bl-based algorithms within 6 min. Bl-based algorithms without UCD (which are not displayed in Fig. 7) are only slightly better than BT-SP on the FUDI 1 set, albeit slower in their increase. This is due to the computing overhead when maintaining the BIH. On the FUDI 2 set, no Bl-based algorithm without UCD was able to determine a problem as unsolvable.

The UCD mechanism is very fast. On the FUDI 1 set, most problems declared impossible by means of UCD are within 8 s.

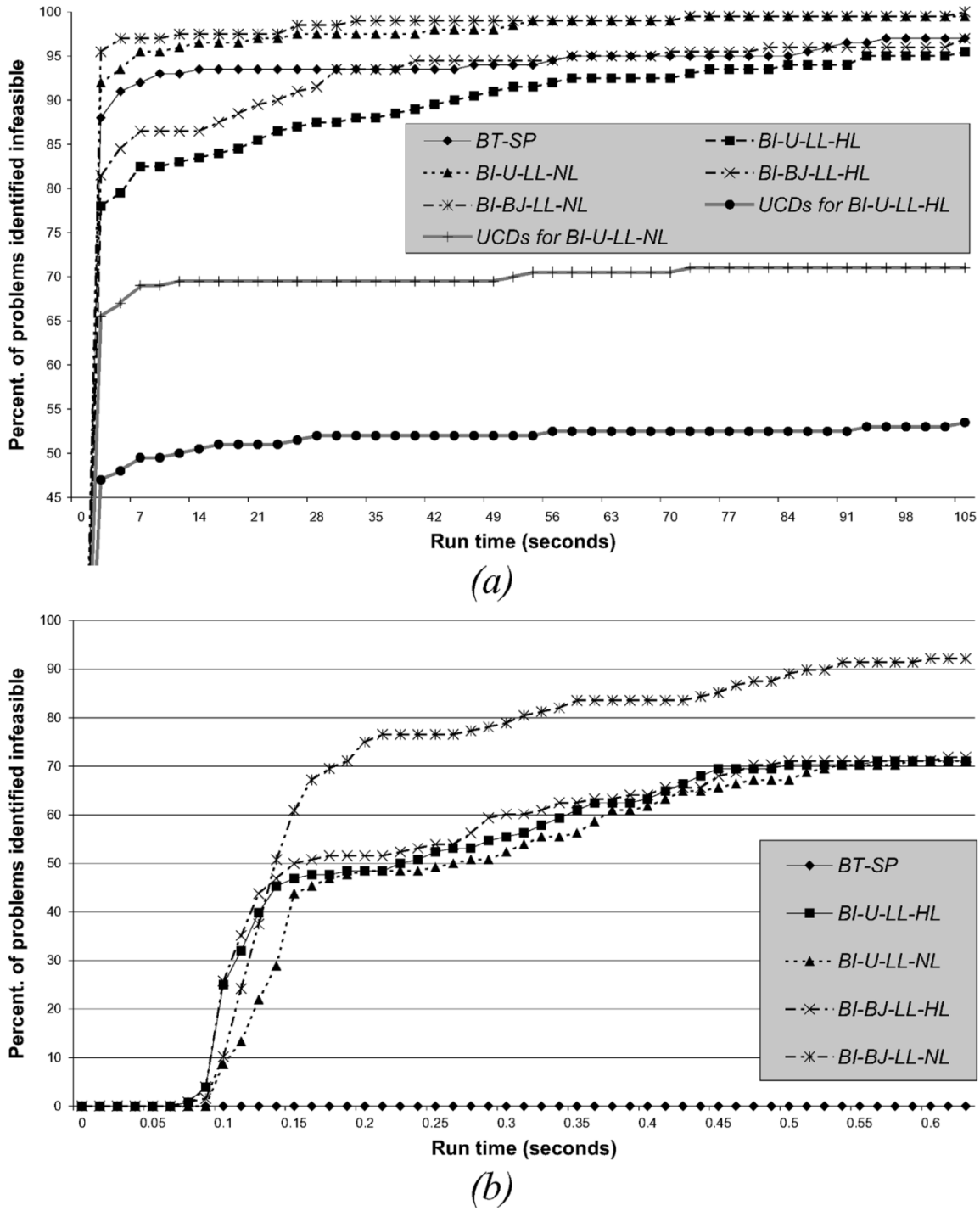


Fig. 7. Percentage of problems identified as infeasible according to runtime by different search methods. (a) The 300 problems of the FUDI 1 set; the percentage of problems identified as unsolvable thanks to the UCD mechanism is also depicted for BI-U-LL-HL and BI-U-LL-NL. (b) The 300 problems of the FUDI 2 set; every problem that was declared infeasible was also thanks to the UCD mechanism.

From then on, UCD then grows only very slowly. On the FUDI 2 set, UCD happens within the first second. It then increases only slightly (0.8%) for HL methods until 9 s elapsed. After that, no increase is established until the runtime limit of 360 s.

The NL demand ordering again outperforms HL, mainly because NL is able to locate critical parts of the network (i.e., identify an UCD) much better and faster. In Fig. 7(a), around 70% of the problems are declared unsolvable by the UCD mechanism for NL, and only slightly more than 50% for HL. On the easy problems, BI-BJ-LL-NL proved that all problems are unsolvable after 104 s, BI-BJ-LL-HL after 205 s. After 360 s, BT-SP is still not able to characterize 2% of the problems. This means that the UCD mechanism does perform well also on problem instances that require a lot of search to be proven infeasible, and not only on "easy" cases, and that search space exploration is most efficient for BI-BJ-LL-NL.

If we compare the search statistics on the problems identified as impossible by all methods, we can clearly establish that the

TABLE II
COMPARISON OF ROUTING HEURISTICS IN A CENTRALIZED QoS ROUTING SCENARIO

	Time [s]	Solved problems	Allocated demands	Bw. connectivity	Bw. usage
23,000 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%

number of generated routes and backtracks are significantly better for Bl-based algorithms, especially for those incorporating the UCD mechanism. For instance, after a runtime of 30 s on the FUDI 1 set problems solved by all methods, BI-BJ-LL-NL backtracked about 40 times fewer (this includes backjumps) and generated over 25 times fewer routes than BT-SP. These values are even higher for lower time thresholds, and only very slightly decrease with time (37 and 24, respectively, after 240 s).

The benefit of backjumping algorithms is greater for identifying impossible problems than for solving feasible ones. This is especially the case for BI-BJ-LL-NL applied to the FUDI 2 set of problems [Fig. 7(b)].

C. QoS Routing

We also evaluated the route ordering heuristics in an *online* QoS routing scenario. In this case, the demands are not known in advance and are 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 23 000 problems and 6000 larger problems, as in Section VII.

These results show that LL performs very similarly to SP: for the set of 23 000 problems, despite completely solving fewer problems, LL allocated more demands in average than SP, had a better remaining bandwidth connectivity, and used fewer bandwidth resources. However, the differences are extremely slight. The same can be observed on the 6000 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 runtime: 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 and 2) LL knows before computing a route if one exists, thereby saving time if a demand cannot be allocated, where 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 runtime 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 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 similar to SP in the QoS routing scenario, it combines better with these DVO heuristics.

D. A Priori Order Parameter Identification

The order parameter we previously used, the tightness as defined in Section VII-A, is a parameter that can only be computed when the problem is solved. We tried to identify other types of tightness measures that can be computed on an unsolved problem in order to estimate the difficulty of solving it.

A natural *a priori* order parameter is the *min-required* tightness. It is based on the minimal demand load (MDL). The MDL for a demand is the minimal amount of resources required to satisfy it, that is its bandwidth requirement times its minimal hop count. If we sum the MDL for all demands and divide the result by the total bandwidth capacity, we get the min-required tightness of the problem. Plots for this order parameter are given in Fig. 8. Again, the curves display a clear phase transition behavior.[4] The min-required tightness is, however, only a very raw characterization, and we are looking into other types of measures that better capture the idea of tightness.

VIII. GENERALIZING THE BI 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 by 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 $Q_u = [q_u^1, q_u^2, \ldots, q_u^z]$ 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

[4]Noteworthy, we see that after a min-required tightness of 0.9, the probability drastically increases. This is normal, since in these regions, a solution can almost only contain shortest routes, by definition of the order parameter.

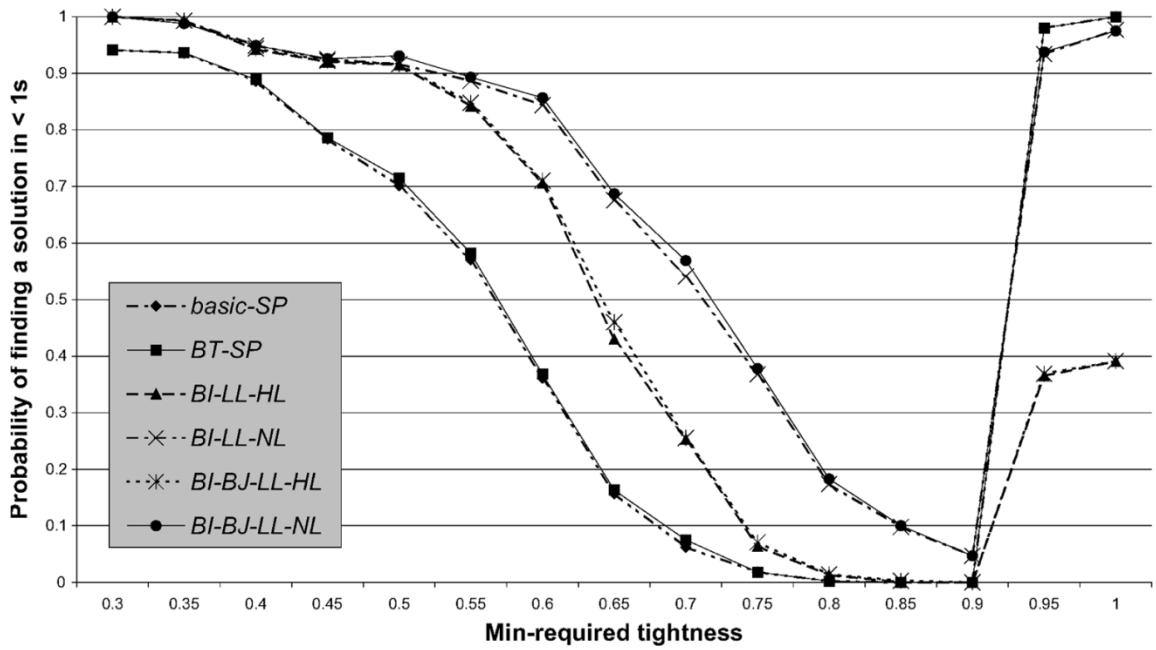


Fig. 8. Probability of finding a solution within 1 s, given the min-required tightness of the problems (23 000 random problems with 20 nodes, 38 links, and 80 demands).

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 $Q_3 < Q_2$ (i.d.).

Given a QoS requirement $Q_u = [q_u^1, q_u^2, \ldots, q_u^z]$, 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 (CCG) for a set of requirements Q_u (Q_u-CCG) is defined and built as the BIG. The connectivity cluster hierarchy (CCH) for a set of concave requirements sets is, however, a generalized version of the BIH. Since there is only a partial order on the possible requirements sets Q_u, father–child relations build a lattice instead of a tree. Fig. 9 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. 9 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. 9. 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} route through

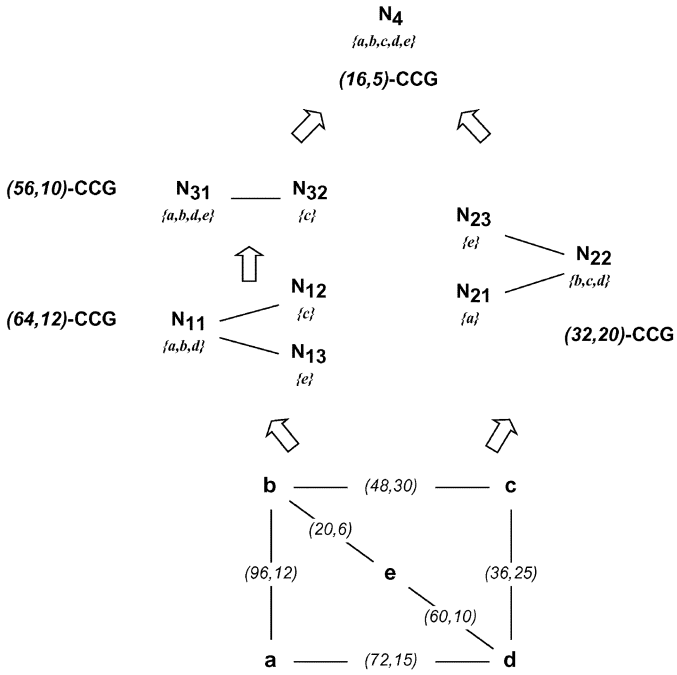


Fig. 9. Connectivity cluster hierarchy of a small network with possible QoS requirements $\{[64, 12], [56, 10], [32, 20], [16, 5]\}$.

c would be chosen. There are several solutions to this problem, for instance metric serialization or combination. *Metric serialization* amounts to imposing 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 selecting 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.

Please note that the application scenario to multiple concave metrics could be: links have certain levels of trust (security) and demands have to be routed only over links of a certain minimum trust. Furthermore, bandwidth is another consideration.

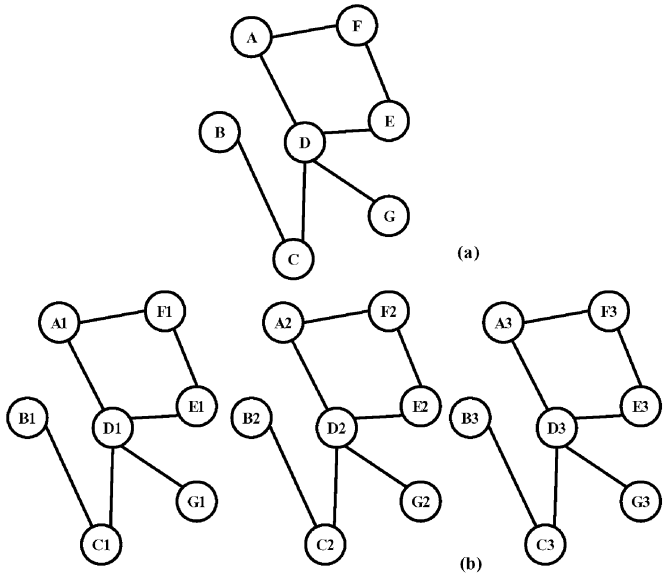


Fig. 10. (a) Network topology G with three wavelengths on each fiber link. (b) Corresponding BIGs are each composed of one BI, inside which all links have a capacity of 1.

IX. Application of BI to Optical Networks

In this section, we briefly illustrate the application of the BI paradigm to wavelength routed WDM networks which are extensively used as the backbone of the Internet. In a wavelength routed WDM network, a lightpath (e.g., wavelength continuous path without processing in the intermediate nodes) is first established between two network nodes before communication takes place. A lightpath may span multiple fiber links and must occupy the same wavelength on all the fiber links it traverses if there are no wavelength converters. This property is known as the wavelength continuity constraint. In order to satisfy a lightpath request in a wavelength routed WDM network, we not only need to consider routing but the wavelength selection as well. Given a set of connection requests, the problem of setting up a lightpath by routing and assigning a wavelength to each connection is called the routing and wavelength assignment (RWA) problem. In this section, we propose a new RWA algorithm using the BI paradigm. The algorithm proposed can be applied to any WDM network with an arbitrary topology.

Define a network topology $G(V, L, W)$ for a given WDM optical network, where V is the set of nodes, L is the set of bidirectional links, and W is the set of wavelengths per fiber link. In particular, the number of wavelengths corresponds to the bandwidth of the optical link, and we will use the term wavelength instead of bandwidth in the remainder of this section. Assume this is a single fiber network without wavelength converters, then the set of wavelengths on each fiber link is the same. Each connection request needs to be allocated over a route and assigned one wavelength. In particular, the network can be abstracted into $|W|$ BIGs. Each BIG starts with one BI representing one of the $|W|$ wavelengths and having the same topology as the original WDM optical network. Hence, the BIG network model $\mathrm{BIG}(mi_1, m_2, \ldots, m_{|W|})$ can be obtained from a given network topology G as follows. The topology of G is replicated $|W|$ times denoted by $m_1, m_2, \ldots, m_{|W|}$. Each BIG m_i is composed of one BI representing a wavelength and the link capacity is 1. An example is shown in Fig. 10.

The RWA problem can be solved based on the BIG model under different networking parameters, namely, static or dynamic traffic, single or multiple fiber links between node pairs, with or without wavelength converters. Briefly, we first transform the network topology into a BIG network model. Then, we use the *route existence property* to decide if the request(s) can be satisfied by checking if the two end nodes are in the same BI of at least one BIG. If they are satisfied, in each BI, we do the routing and wavelength selection and several heuristics are employed to get the "best" routes and wavelengths. In the static traffic case, a *backtracking scheme* is added to the algorithm to get a more optimized result. A formal description of the proposed algorithm is given below. We first assume the traffic is static. So, we know all the connection requests *a priori*. Our goal is to maximize the number of accepted requests given a fixed number of wavelengths per fiber link.

- Step 1) Transform the network topology into a BIG network model.
- Step 2) Order all the connection requests in decreasing length of their minimum number of hops (MNHs) distance (MNH distance is calculated using any SP algorithm, e.g., Dijkstra). Paths with equal lengths are ordered randomly.
- Step 3) Select an unallocated request $d, D = D - d$. If the request set D is empty, then go to Step 8).
- Step 4) Use the Route existence property to check if all the requests in D can be satisfied individually in any BIG. If it can, assign the request d to each possible wavelength BI and calculate K alternate shortest paths. If it cannot, go to Step 8).
- Step 5) Route and Wavelength Selection: Now, we have a set of candidate routes in each possible wavelength BIG. Compute the splitting numbers if the corresponding route is selected and removed from the

BIs. Find the route(s) with the minimum splitting numbers. The splitting number for a route means that how many more BIs will be generated if the route is removed from the current BIs. For example, if a route is removed and the number of BIs is the same as before, we say the splitting number for the route is 0. If the number of BIs is one more than before, we say the splitting number for the route is 1.

If there are several routes, pick up one with the shortest MNH distance.

If there are still a few routes, choose one whose congestion of the most loaded link (most number of wavelengths in the link has been used) is lower than the congestion of the most loaded link in other routes. Then, if there is more than one route left, randomly select one.

Step 6) Allocate the route in corresponding BI. Reconstruct the BIGs.

Step 7) Add the route and corresponding wavelength to the result set R, go to Step 3).

Step 8) If the request set is empty, output the result set. Otherwise, output "requests cannot be satisfied."

In order to get a more optimal result, a *backtracking scheme* is added to the algorithm as long as time is allowed. In Step 4), if not all the requests can be satisfied individually, instead of going to Step 8), we backtrack to the previous request and try another one of K alternate routes. We repeat the process until we succeed or all the alternate routes of previous requests are tried. In Step 4), the route existence property checks whether the current available resources can satisfy all single requests in D. In the backtracking process, it provides an early warning for selecting the wrong route and wavelength.

If the connection requests arrive dynamically, we need to do a few modifications to the original algorithm. For example, we cannot order the requests and the backtracking scheme is impossible. The steps of the algorithm are given next.

Step 1) Transform the network topology into a BIG network model.

Step 2) When a connection request d arrives, based on the arriving time of the request, reconstruct the BIGs.

Step 3) Check the request d using the route existence property in each BIG. If it exists in any possible wavelength BIG, calculate K alternate shortest paths; if it does not exist in any BIG, it is blocked.

Step 4) Route and wavelength selection (the same as the static case).

Step 5) Allocate the request and go to Step 2).

Because of the dynamic nature of the traffic and the different holding times of each session, we need to update the BIGs every time a new request arrives in Step 2).

At each step, updating the BIG has a worst case complexity $O(bm)$, where $b =$ number of levels and $m =$ number of links in the network. This is the worst case when the BIH is completely transformed (splitting number $= n$). It does not seem that we can have a tighter bound even when we restrict the splitting number, because even with 2 BIs, it is still possible that all links go between these two islands. Thus, when we apply the minimum splitting heuristic, the complexity for each demand would be $O(bmk)$ with k being the number of paths tested (a parameter of the algorithms in Section IX), and the complexity for all demands is $O(bmkd)$, where d is the number of demands.

If we add backtracking, then the worst case complexity becomes exponential in the number of demands, with the average case complexity being much lower.

Please note that our choice of a single-level BIHs is related to the fact that our heuristic algorithms attempt to choose a path that will avoid future conflicts.

TABLE III
SIMULATION RESULT N IS THE NUMBER OF NODES IN THE NETWORK. L IS THE NUMBER OF LINKS IN THE NETWORK. WLL REPRESENTS THE LOWER LIMIT ON THE NUMBER OF WAVELENGTHS. REFERENCE [20] IS THE RESULT OF OUR IMPLEMENTATION OF THE ALGORITHM IN [20]

Network	N	L	WLL	[20]	BI_RWA
ARPANet	20	31	33	34	34
UKNet	21	39	19	22	20
EON	20	39	18	18	18
NSFNet	14	21	13	13	13

For static traffic, one of the most important goals is to minimize the number of wavelengths needed to accommodate the given requests. The proposed *BI_RW A* algorithm **with backtracking** is applied to several existing or planned network topologies to verify its efficiency. We assume the incoming traffic is uniform. The networks considered are the ARPANet [21], NSFNet [22], the European Optical Network (EON) proposed in [23], and a hypothetical U.K. topology reflecting the current BT-networks [24]. Noted, those topologies are also evaluated in [20] and the results in [20] are near optimal. Our simulation results are shown in Table III indicating the minimum number of wavelengths needed for a given traffic using different algorithms. As can be seen, the application of the BI paradigm can lead to optimal or "near-optimal" results, and sometimes compare favorably to those in [20]. Reference [20] also proposes a method called WLL, based on *bisection bandwidth*, to calculate the lower limit on the number of wavelengths.

We also evaluated the performance of our proposed dynamic *BI_RW A* algorithm on NSFNet, which has 14 nodes and 21 links. The heuristic dynamic RAW algorithms used in the simulation are fixed routing with first-fit wavelength assignment (FR/FF), fixed routing with most used/pack wavelength assignment (FR/MU), alternate routing with most used/pack wavelength assignment (AR/MU), and alternate routing with random wavelength assignment (AR/RAN). These algorithms are among the most reputed algorithms for dynamic traffic allocation in WDM networks [22]. Figs. 11 and 12 compare the call blocking probabilities for dynamic *BI_RW A*, FR/FF, FR/MU, AR/MU, and AR/RAN using the NSFNet network. In the alternate routing algorithm, all the SPs are calculated for each node pair. In Fig. 11, the number of wavelengths on each link is four. In Fig. 12, the number of wavelengths on each link is eight. Again, the simulation results favorably compare our proposed algorithm to the related algorithms. This is a further confirmation of the usefulness of the BI paradigm to various network problems.

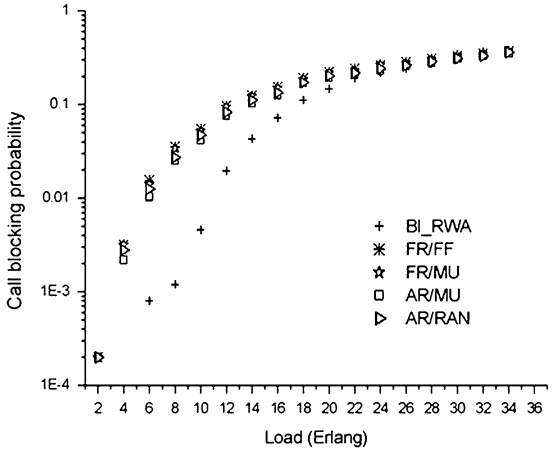


Fig. 11. Blocking probabilities for the NSFNet with four wavelengths.

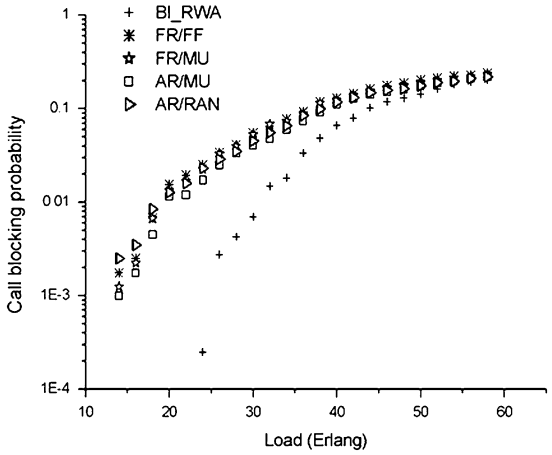


Fig. 12. Blocking probabilities for the NSFNet with eight wavelengths.

X. Conclusion

In this paper, we have shown that using BI abstractions coupled with CSP exhaustive search mechanisms and heuristics, it is possible to solve the network bandwidth allocation problem for many problem instances in reasonable time with a complete search algorithm. We have also demonstrated that this method gives better solutions than SP 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. The BI paradigm also proves to be very efficient in identifying infeasible problems very quickly, and thereby also constitutes a powerful aid to the network operator, since the UCD mechanism highlights the regions in the network between which bandwidth must be added in order to be able to satisfy all demands. Moreover, a generalization of the BI paradigm, the connectivity clusters, to take into account multiple concave QoS metrics has been proposed. Finally, we have illustrated the usefulness of the BI paradigm to the routing and wavelength assignment problem in optical networks under a variety of networking scenarios.

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 LL for route generation or the number of levels for demand selection, to multipoint demands is straightforward. In [1], we show additional applications of the BI paradigm, such as a new graphical display of the network's state that highlight bottlenecks for improved human-machine interaction, techniques for tuning network topology according to a traffic profile based on explanation of allocation failure, on-line routing by intelligent agents based on the BIH, faithful and concise topology aggregation for hierarchical networks, and a novel pricing scheme that can be extracted from the BIH.

We are also extending the application of the BI paradigm into two directions, and the results will be presented in future papers. First, we are looking into how to apply the BI techniques in connection-less networks (such as IP). Second, we are investigating how to incorporate nonconcave QoS metrics such as delay, delay jitter, and packet loss probability into the BI paradigm.

Acknowledgment

The authors wish to express their thanks to D. Allemang, S. Willmott, and M. Calisti for their invaluable comments, suggestions, and encouragements during the last years. The authors extend their gratitude to Prof. M. Golumbic for his proof of the NP-completeness of the RAIN problem.

References

[1] C. Frei, "Abstraction Techniques for Resource Allocation in Communication Networks," Ph.D. dissertation, Swiss Fed. Inst. Technol. (EPFL), Lausanne, Switzerland, 2000.

[2] Z. Wang and J. Crowcroft, "Analysis of shortest-path routing algorithms in a dynamic network environment," in *Proc. ACM SIGCOMM CCR*, vol. 22, 1992, pp. 63–71.

[3] E. Tsang, *Foundations of Constraint Satisfaction*, London, U.K.: Academic, 1993.

[4] R. J. Wallace, "Practical applications of constraint programming," *Constraints An Int. J.*, pp. 139–168, 1996.

[5] G. R. Ash, *Dynamic Routing in Telecommunications Networks*. New York: McGraw-Hill, 1998.

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

[7] M. Herzberg, D. J. Wells, and A. Herschtal, "Optimal resource allocation for path restoration in mesh-type self-healing networks," *Int. Teletraffic Congr. (ITC)*, vol. 15, pp. 351–360, 1997.

[8] J. W. Mann and G. D. Smith, "A comparison of heuristics for telecommunications traffic routing," in *Modern Heuristic Search Methods*. New York: Wiley, 1996, pp. 235–254.

[9] B. T. Messmer, "A framework for the development of telecommunications network planning, design and optimization applications," Tech. Rep. FE520.020 78.00 F, 1997. Swisscom.

[10] Z. Wang and J. Crowcroft, "Quality-of-service routing for supporting multimedia applications," *IEEE J. Sel. Areas Commun.*, vol. 14, no. 7, pp. 1228–1234, 1996.

[11] S. Chen and K. Nahrstedt, "An overview of quality of service routing for next-generation high-speed networks: Problems and solutions," *IEEE Netw.*, pp. 64–79, Nov./Dec. 1998.

[12] S. Vedantham and S. S. Iyengar, "The bandwidth allocation problem in the ATM network model is NP-complete," *Inform. Process. Lett.*, vol. 65, pp. 179–182, 1998.

[13] B. Y. Choueiry and B. Faltings, "A decomposition heuristic for resource allocation," in *Proc. 11th Eur. Conf. Artif. Intell.*, 1994, pp. 585–589.

[14] R. Weigel and B. Faltings, "Structuring techniques for constraint satisfaction problems," in *Proc. 15th Int. Joint Conf. Artif. Intell.*, 1997, pp. 418–423.

[15] E. C. Freuder and D. Sabin, "Interchangeability supports abstraction and reformulation for multidimensional constraint satisfaction," in *Proc. 15th Nat. Conf. Artif. Intell.*, 1997, pp. 191–196.
[16] B. Y. Choueiry and T. Walsh, Eds., *Lecture Notes in Artificial Intelligence 1864*. New York: Springer-Verlag, Jul. 2000, Proc. 4th Int. Symp. Abstraction, Reformulation and Approximation.
[17] C. Frei and B. Faltings, "A dynamic hierarchy of intelligent agents for network management," in *Lecture Notes in Artificial Intelligence 1437*. New York: Springer-Verlag, 1998, Proc. 2nd Int. Workshop Intell. Agents Telecommun. Applicat., pp. 1–16.
[18] R. M. Haralick and G. L. Elliott, "Increasing tree search efficiency for constraint satisfaction problems," *Artif. Intell.*, vol. 14, pp. 263–313, 1980.
[19] P. Prosser, "Hybrid algorithms for the constraint satisfaction problem," *Comput. Intell.*, vol. 9, no. 3, pp. 268–299, 1993.
[20] S. Baroni and P. Bayvel, "Wavelength requirements in arbitrarily connected wavelength-routed optical networks," *IEEE/OSA J. Lightwave Technol.*, vol. 15, no. 2, pp. 242–251, Feb. 1997.
[21] R. Ramaswami and K. N. Sivarajan, "Routing and wavelength assignment in all-optical networks," *IEEE/ACM Trans. Netw.*, vol. 3, pp. 489–500, Oct. 1995.
[22] T. M. Tsang and K. Bala, *Multiwavelength Optical Networks*. Reading, MA: Addison-Wesley, 1999.
[23] M. J. O'Mahony, D. Simoneonidou, A. Yu, and J. Zhou, "The design of a European optical network," *IEEE/OSA J. Lightwave Technol.*, vol. 13, pp. 817–828, May 1995.
[24] S. Appleby and S. Steward, "Mobile software agents for control in telecommunications networks," *British Telecom. Technol. J.*, vol. 12, no. 2, pp. 104–113, Apr. 1994.

Christian Frei received the Diploma degree in computer science and the Ph.D. degree from the Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland.

He is now a Scientist at the ABB Corporate Research Center, Baden, Switzerland. His research interests include constraint satisfaction, abstraction and problem reformulation techniques, graph theory, and resource allocation problems in general.

Dr. Frei received the 2000 ECCAI Artificial Intelligence Dissertation Award for his Ph.D. dissertation on abstraction techniques for constraint satisfaction, in particular, for applications in communication networks.

Boi Faltings received the Diploma degree in electrical engineering from Swiss Federal Institute of Technology (ETH) Zurich, Switzerland, and the Ph.D. degree in computer science from the University of Illinois at Urbana–Champaign.

He founded the Artificial Intelligence Laboratory, School of Computer and Communication Sciences, Ecole Polytechnique Federale de Lausanne (EPFL), Lausanne, Switzerland, in 1987 and from 1996 to 1998, he was Head of the Computer Science Department, EPFL. He is a Professor of Computer Science and Director of the Artificial Intelligence Laboratory. His research interests are in constraint satisfaction, case- and model-based reasoning, as well as applications in engineering and e-commerce.

Mounir Hamdi (S'89–M'90) received the B.S. degree in computer engineering (with distinction) from the University of Louisiana, Lafayette, in 1985 and the M.S. and the Ph.D. degrees in electrical engineering from the University of Pittsburgh, Pittsburgh, PA, in 1987 and 1991, respectively.

He has been a Faculty Member in the Department of Computer Science, Hong Kong University of Science and Technology since 1991, where he is now a Professor of Computer Science and the Director of the Computer Engineering Program that has around 350 undergraduate students. From 1999 to 2000, he held Visiting Professor positions at Stanford University, Stanford, CA, and the Swiss Federal Institute of Technology, Lausanne, Switzerland. His general areas of research are in high-speed packet switches/routers and all-optical networks, in which he has published more than 200 research publications, received numerous research grants, supervised some 20 postgraduate students, and for which he has served as consultant to various international companies. Currently, he is working on high-speed networks including the design, analysis, scheduling, and management of high-speed switches/routers, wavelength-division multiplexing (WDM) networks/switches, and wireless networks. He is currently leading a team that is designing one the highest capacity chip sets for terabit switches/routers. This chip set is targeted toward a 256×256 OC-192 switch, and includes a crossbar fabric chip, a scheduler/arbiter chip, and traffic management chip. In addition to his commitment to research and professional service, he is also a dedicated teacher.

Dr. Hamdi is a member of the Association for Computing Machinery (ACM). He received the Best Paper Award at the International Conference on Information and Networking in 1998 out of 152 papers. He received the Best Ten Lecturers Award (through university-wide students voting for all university faculty held once a year) and the Distinguished Teaching Award from the Hong Kong University of Science and Technology. He is/was on the Editorial Boards of the IEEE TRANSACTIONS ON COMMUNICATIONS, the *IEEE Communication Magazine*, *Computer Networks*, *Wireless Communications and Mobile Computing*, and *Parallel Computing*. He has been on the program committees of more than 50 international conferences and workshops. He was a Guest Editor of the *IEEE Communications Magazine*, the IEEE JOURNAL ON SELECTED AREAS OF COMMUNICATIONS, and *Optical Networks Magazine*, and has Chaired more than five international conferences and workshops including the IEEE GLOBECOM/ICC Optical Networking Workshop, the IEEE ICC High-Speed Access Workshop, and the IEEE IPPS HiNets Workshop. He is the Chair of IEEE Communications Society Technical Committee on Transmissions, Access, and Optical Systems, and Vice Chair of the Optical Networking Technical Committee, as well as ComSoc Technical Activities Council.

.