IconoNET: a Tool for Automated Bandwidth Allocation Planning

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

G. Melissargos, P. Pu
Research group of ergonomics of intelligent systems
ISR/DMT
Swiss Federal Institute of Technology (EPFL)
CH-1015 Lausanne
Switzerland
{George.Melissargos, Pearl.Pu}@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. In this paper, 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 problem.

1 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 global network 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 the 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 links capacities but also the expected traffic profile, in an off-line manner. The traffic profile may be given, as when setting up virtual private networks in an ATM network of a provider, or estimated by objective traffic measurements (which almost every network operator carries out).

In this paper, we introduce the techniques behind an innovative tool we developed, IconoNET (Figure 1). The tool allows modeling a network and demands, and provides decision aids and automatic functions for allocating these demands to network resources. The basic problem IconoNET solves is resource allocation in networks (RAIN):

> **Given** a network composed of nodes and bidirectional links, each link with a given resource capacity, and a set of communication demands to allocate, each demand defined by a triple: *(source node, destination node, requested bandwidth)*
>
> **Find** one route for each demand so that the bandwidth requirements of the demands are 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-hard 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.

The RAIN problem occurs as a basic problem in a large variety of network performance management settings. Most obvious is its application in routing, but it is also a key component of network dimensioning, designing network topologies, and analyzing robustness to link and node faults. However, it has to be embedded as a decision aid and be highly efficient to be useful for such tasks. IconoNET uses a novel abstraction scheme, which allows the RAIN problem to be solved more efficiently than traditional *k*-shortest path methods.

IconoNET uses techniques of *constraint programming* to solve the allocation problem. With respect to other optimization techniques, constraint programming allows us to represent arbitrary constraints on QoS or routing. This makes possible for users to interact with the allocation procedure without having to understand

abstract representations. For example, when allocation fails it is possible to provide explanations of where network capacity is insufficient. Graphical representations allow rapid understanding of bottlenecks of problems with robustness.

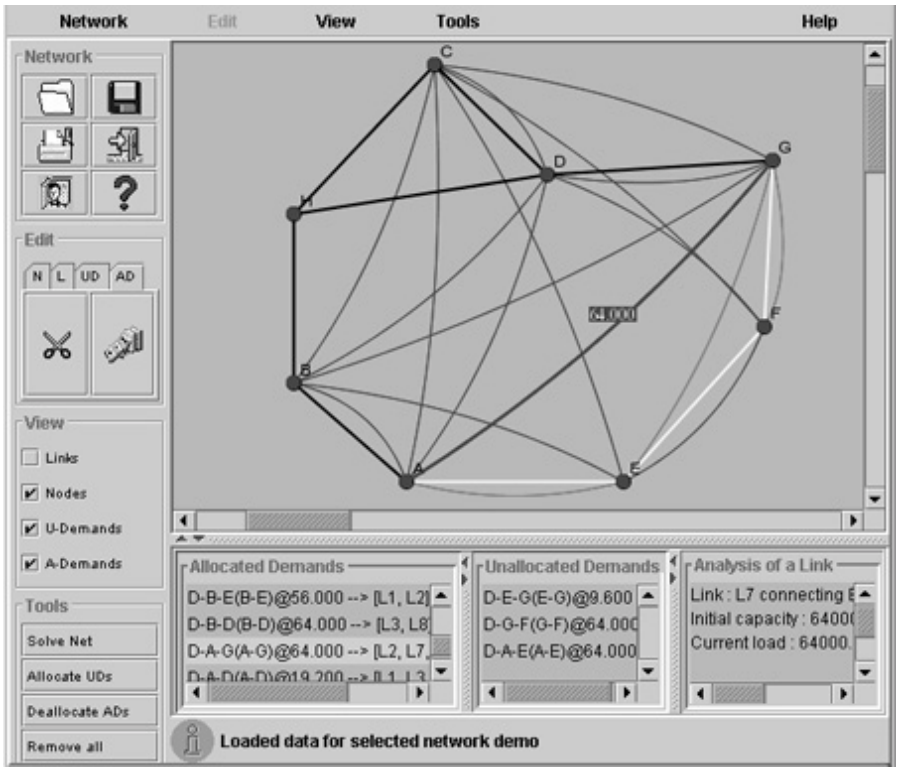


Figure 1: The graphical user interface of IconoNET.

Constraint satisfaction [2] is a technique, which has been shown to work well for solving certain NP-hard problems. 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*.

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. It is too expensive to compute, represent, and store the domains of the variables, i.e., all the routes that join the endpoints of each demand. Suppose the network is simple but complete with *n* nodes (this is not even the worst case, since a communication network is a multi-graph: it allows multiple links between same endpoints). 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 *(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 pair of 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.

In this paper, we show how abstractions of the network, called Blocking Islands, create a compact representation of the problem which allows the application of well-known CSP techniques such as forward checking, variable and value ordering to the RAIN problem, with manageable complexity.

2 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. Operation Research 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 [3]. However, because only a given number of routes are considered, these techniques are not guaranteed to find a solution if one exists.

Mann and Smith [4] 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, the routing assignment attempts 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 easier, without having to recompute a complete solution.

To our knowledge, the closest published work to ours is the CANPC framework [5]. 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, preliminary results show that the methods we propose clearly outperform theirs.

3 THE BLOCKING ISLAND PARADIGM

[6] introduces 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*. Figure 2 (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 *(a,b)* 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 *{(a,c), (c,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*-BI. Furthermore, all links that could form part of such a route lie inside this blocking island.*

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. Figure 2 (c) is the 64-BIG of the network of Figure 2 (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.

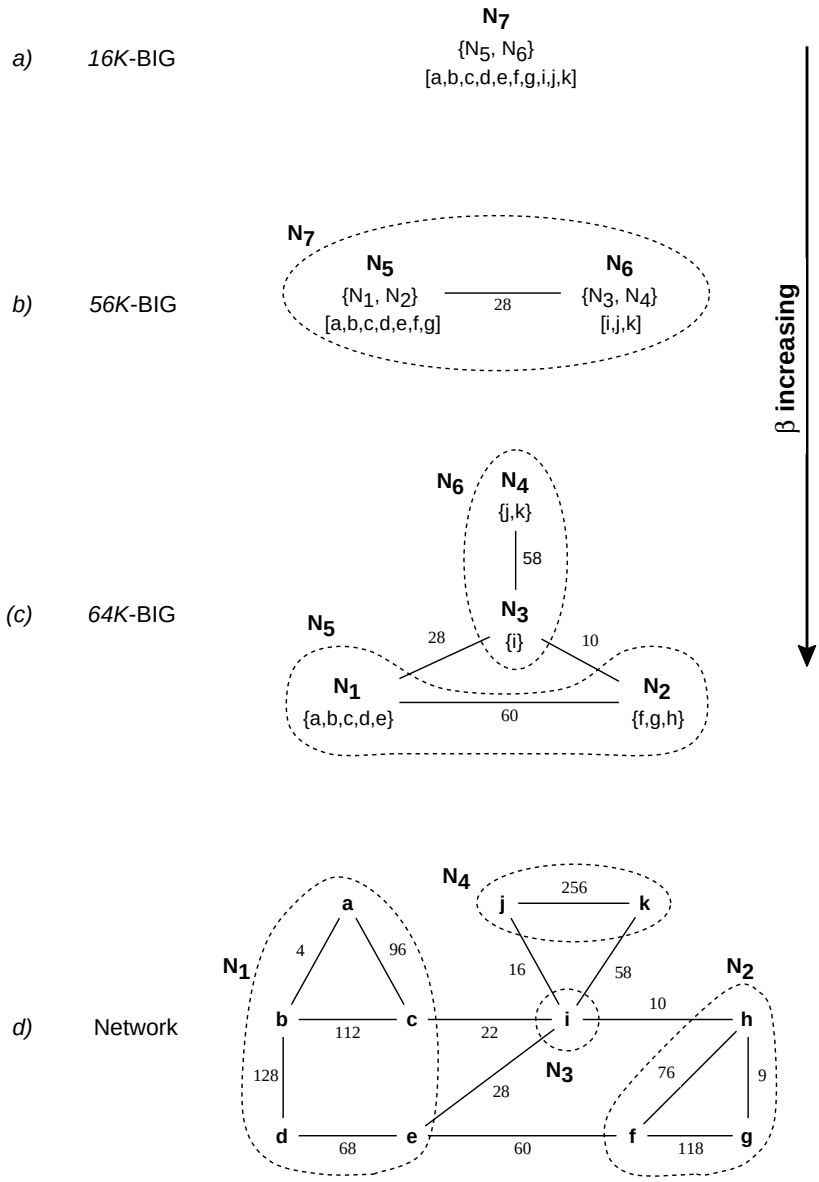


Figure 2: The blocking island hierarchy for resource requirements {64K, 56K, 16K}. The weights on the links are their available bandwidth. Abstract nodes' description includes only their node children and network node children in brackets. Link children (of BIs and abstract links) are omitted for more 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.

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. Figure 2 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. Figure 3 is the abstraction tree of Figure 2.

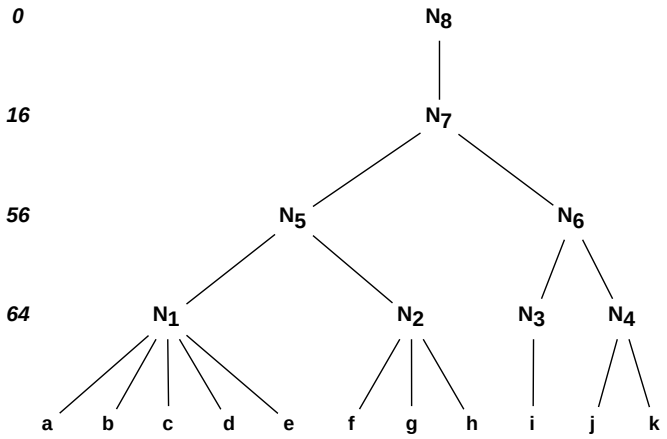


Figure 3: The abstraction tree of the BIH of Figure 2 (links are omitted for clarity).

The blocking island hierarchy summarizes the available bandwidth given the currently allocated demands at a time *t*. As demands are allocated or deallocated, available bandwidth changes on the communication links and the BIH may need to be modified to reflect this. The changes can be carried out incrementally, only affecting the blocking islands which participate in the demand which is being allocated or deallocated:

- When a new demand is allocated along a particular route, the bandwidth of each link decreases. If it falls below the bandwidth β of its blocking island, and no alternative route exists with available resources ≥ β within the BI, it

causes a split of the BI into two parts. Furthermore, this split must be propagated downwards to all BI in the hierarchy with a higher β.

- When a demand is deallocated, bandwidth across each link increases. If it thus becomes higher than the β of the next higher level in the hierarchy, it will cause two disjoint blocking islands to merge into a single one. This merge must be propagated upwards to all levels with a lower β.

[6] presents in more details how the incremental adaptation of a BIH is performed, not only when new connections are established or existing ones deallocated, but also in case of link failure, link properties alteration or even network topology changes (such as link or node addition/removal).

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 by *O(bm)* algorithms. Therefore, since the number of possible bandwidth requirements (*b*) is constant, all BI algorithms are linear in the number of links of the network.

4 AUTOMATICALLY SOLVING A RAIN PROBLEM

Solving a RAIN problem amounts to solving the CSP introduced in Section 1. 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 1). 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). In fact, there is a mapping between each route that can be assigned to a demand and the BIH: a route can been 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 Figure 3, it is easy to see that there is no route between *a* and *g*

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 dynamic variable and value ordering heuristics. (We note that a patent for the methods given below is pending.)

4.1 Forward Checking

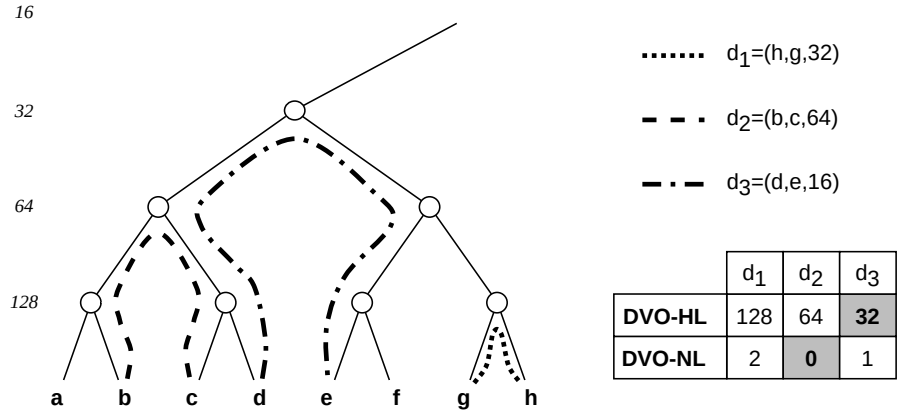
Forward checking is a technique to improve backtracking algorithms. Its idea is to propagate value assignments to unallocated variables along the constraints in order to detect a dead-end earlier, thereby increasing search efficiency. Moreover, decisions regarding which variable to select next and what 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 does not hold, another route must be tried. Domain pruning (i.e., constraint propagation) is thus implicit while maintaining the BIH.

4.2 Variable Ordering

A backtracking algorithm involves two types of choices: the next variable to assign, and the value to assign to it. The selection of the next variable to assign may have a non-negligible effect on search efficiency. A widely used variable ordering technique in CSP is based on the "fail-first" principle (FFP): "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 choose first the demand that requires the most resources. BIs allow to dynamically (during search) approximate the difficulty of allocating a demand more subtly, according to the current state of the network:

- **DVO-HL** (Highest Level): choose first 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, the more allocating the demand may restrict the routing of the remaining demands (FFP), since it will use resources on more critical links.
- **DVO-NL** (Number of Levels): choose first 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.



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

Figure 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.

Figure 4 illustrates these two heuristics.

4.3 Value Ordering

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 order* (SP), since the shorter the route, the fewer resources will be used to satisfy a demand. Consider the problem of routing a single demand *d=(c,e,16)* in the network of Figure 2 (d). The shortest route satisfying the demand is $c \rightarrow i \rightarrow e$. However, allocating this route to *d* is here not a good idea, since it uses resources on two critical links in terms of available bandwidth: *(c,i)* and *(i,e)*. Allocating this route causes a split of N_7, thereby preventing the allocation of a demand requiring 16 (or more) between any of the nodes of N_5 and N_6.

We can do better with a kind of min-conflict heuristic, based on the BIH, called *lowest level* (LL) heuristic. It considers first (in shortest order) the routes in the lowest blocking island (in the BIH), i.e., the blocking island for the highest resource requirement clustering the endpoints of the demand. 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 overall load-balancing effect is achieved, therefore reducing the risk of future allocation failures. Moreover, the lower a BI is, the smaller it is in terms of nodes and links, thus reducing even more the search space when looking for the first routes, and thereby achieving a computational gain during the early stages of the search.

Generating one route with the LL heuristic can be done in linear time in the number of links. For the same demand d, LL selects route $c \rightarrow b \rightarrow d \rightarrow e$, and the allocation of that route does not change the bandwidth connectivity in the network.

5 RESULTS

In practice, the RAIN problem poses itself in the following way: a service provider receives a request from the customer to allocate a number of demands, and must decide within a certain decision threshold (for example, 5 seconds), whether and how the 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 telecom 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 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 22'000 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.

Figure 5 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 those provided by BI methods.

These 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.

All results were computed on a Sun Sparc 60 with a LISP program.

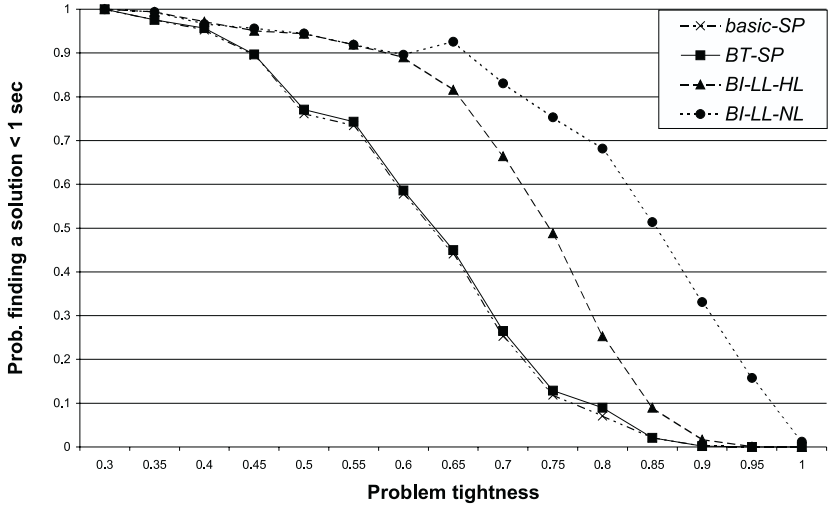


Figure 5: The probability of finding a solution within 1 second, given the tightness of the problems (22'000 random problems with 20 nodes, 38 links, 80 demands).

6 CONCLUSION

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 search mechanisms and heuristics, it is possible to solve plan bandwidth allocation in reasonable time, and to get better solutions than shortest path routing algorithms, especially 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 than before. An on-line demo of IconoNET illustrating these features is available on the WWW at `http://www.iconomic.com`.

In this paper, we restricted demands to point-to-point traffic. However, this need not be. 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) are straightforwardly generalized to multipoint demands. Moreover, quality of service was expressed in terms of bandwidth only. It is however easy to incorporate other QoS parameters (such as maximal delay, maximal loss ratio, or node capacity) or cost into the technique by checking that the generated routes do verify those additional constraints. 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, which do not allow the integration of new constraints in such a straightforward manner.

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 then in the route generation process, since IP uses hop-by-hop routing tables, and the generated routes must respect that additional constraint.

REFERENCES

[1] Zheng Wang and Jon Crowcroft. Quality-of-Service Routing for Supporting Multimedia Applications. *IEEE Journal on Selected Areas in Communications*, 14(7):1228-1234, September 1996.

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

[3] Gerald R. Ash. *Dynamic Routing in Telecommunications Networks*. Mc-Graw Hill, 1998.

[4] Jason W. Mann and George D. Smith. A Comparison of Heuristics for Telecommunications Traffic Routing. In V. J. Rayward-Smith, I. H. Osman, C.

R. Reeves and G. D. Smith, editors, *Modern Heuristic Search Methods*, pages 235-254. John Wiley & Sons Ltd., 1996.

[5] Bruno 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.

[6] Christian Frei and Boi Faltings. A Dynamic Hierarchy of Intelligent Agents for Network Management. In *2nd International Workshop on Intelligent Agents for Telecommunications Applications, IATA'98*, pages 1-16, Paris, France, July 1998. Lecture Notes in Artificial Intelligence, Vol. 1437, Springer-Verlag.

BIOGRAPHY

Christian Frei was born in Kampala, Uganda, in 1967. He received a diploma in Computer Science from the Swiss Federal Institute of Technology in Lausanne (EPFL), Switzerland, in March 1994. Since July 1994, he is a research assistant at the Artificial Intelligence Laboratory of the Swiss Federal Institute of Technology in Lausanne. He has developed abstraction techniques for constraint satisfaction, in particular for applications in communication networks. His research interests include constraint satisfaction techniques, graph theory, and resource allocation problems in general.

Boi Faltings is professor of computer science and director of the Artificial Intelligence Laboratory at the Swiss Federal Institute of Technology, Lausanne (EPFL). He holds a diploma in Electrical Engineering from ETH Zurich and a Ph.D. in Computer Science from the University of Illinois. He founded the EPFL Artificial Intelligence Laboratory in 1987, and has been head of the EPFL Computer Science department from 1996 to 1998. His research interests are in constraint satisfaction, case- and model-based reasoning, as well as applications in engineering and e-commerce.

George Melissargos was born 20/08/67 in Athens, Greece. After 4 years of studies in Computer Systems Engineering at the Technological Educational Institute of Piraeus he continued at the State University of New York at Binghamton where he earned the BS and MS degrees in Computer Science. After a short stay at the CN division at CERN he moved to the Swiss Federal Institute of Technology at Lausanne. There, as a research assistant, he works for the last 4 years towards the Ph.D. degree in the Department of Microengineering. His interests lie in the areas of Human Computer Interaction, Networks and Information Visualization.

Pearl Pu is a senior researcher at the Swiss Institute of Technology in Lausanne. She heads the research group on ergonomics of intelligent systems. She obtained her Ph.D. from the University of Pennsylvania in 1988. Her research interests include information visualization, artificial intelligence, information retrieval, and electronic catalogs.