Resource Allocation in Networks Using Abstraction and Constraint Satisfaction Techniques

Christian Frei¹ and Boi Faltings¹

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

Abstract. Most work on constraint satisfaction problems (CSP) starts with a standard problem definition and focuses on algorithms for finding solutions. However, formulating a CSP so that it can be solved by such methods is often a difficult problem in itself. In this paper, we consider the problem of routing in networks, an important problem in communication networks. It is as an example of a problem where a CSP formulation would lead to unmanageable solution complexity. We show how an abstraction technique results in tractable formulations and makes the machinery of CSP applicable to this problem.

1 Introduction

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. Shortest path routing is the traditional technique applied to this problem. However, this can lead to poor network utilization 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 link capacities but also the expected traffic profile. This traffic profile may be given, as when setting up virtual private networks in an ATM backbone of a provider, or estimated by objective traffic measurements (which almost every network operator carries out).

A communication network is composed of nodes interconnected with communication links. We model it as a connected *network graph* $\mathcal{G} = (\mathcal{N}, \mathcal{L})$ an undirected multi-graph without loops, i.e., edges whose endpoints are the same vertex, see Fig. 1 (d). The set of nodes $\mathcal{N}$ are processing units, switches, routers, etc., and the links $\mathcal{L}$ correspond to bidirectional communication media, such as

optical fibers. Each link l is characterized by its *bandwidth capacity*, the (currently) *available bandwidth*. In Fig. 1.d, the weights on the links denote their available bandwidth. Our network must fulfill communication needs between pairs of nodes, or *demands*. A demand d_u is defined by a triple: $d_u = (x_u, y_u, \beta_u)$, where x_u and y_u are distinct nodes of the network graph $\mathcal{G}$ and define the nodes between which communication is required to take place; the demand's *endpoints*. Parameter β_u describes the demand's bandwidth requirement. A network $\mathcal{G}$ satisfies a set of demands by allocating a *connection* for each demand. A connection is a simple path in the network graph that satisfies the bandwidth requirement.

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

Given a network composed of nodes and links, each link with a given resource capacity, and a set of demands to allocate,

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

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

Constraint satisfaction [2] is a technique which has been shown to work well for solving certain NP-hard problems. 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 the domains of the variables, i.e., all the routes that join the endpoints of each demand. Suppose the network is simple but complete (note that this is not even the worst case, since a communication network is a multi-graph: it allows multiple links between same endpoints) with n nodes. A route is a simple path, its length in number of links is therefore bounded by $n - 1$. Since a route of length j has $j - 1$ intermediate (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 applying well-known CSP techniques such as forward checking, variable and value ordering to the RAIN problem, with manageable complexity. When a dead-end is reached during search, blocking island abstractions also 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. The latter feature is used as a backjumping criteria for improved search.

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. There are some proprietary tools for this, about which nothing much is known.

To our knowledge, the closest published work to ours is the CANPC framework [3]. 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.

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.

Abstraction and reformulation techniques have already been applied to permit more efficient solution of a CSP. [5] relate interchangeability to abstraction in the context of a decomposition heuristic for resource allocation. [6] 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. [7] present abstraction and reformulation techniques based on interchangeability to improve solving CSPs. [8] is a recent collection of papers addressing abstraction, reformulation, and abstraction techniques in a variety of AI techniques.

3 The Blocking Island Paradigm

[9] introduce a clustering scheme based on Blocking Islands (BI), which can be used to represent bandwidth availability at different levels of abstraction, as a basis for resource allocation by intelligent agents. 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 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 (a, b) 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 $\{(a, c), (c, b)\}$.

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 BIs 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 bandwidth 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 1 (c) is the 64-BIG of the network of Fig. 1 (d). An abstract link between two BIs clusters all links that join the two BIs, and the abstract link's available resources is equal to the maximum of the available resources of the links it clusters (because a demand can only be allocated over one route). These abstract links denote the critical links, since their available bandwidth do not suffice to support a demand requiring β resources.

In order to identify bottlenecks for different β s, e.g., for typical possible bandwidth requirements, we build a recursive decomposition of BIGs in decreasing order of the requirements: $\beta_1 > \beta_2 > \dots > \beta_b$. 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 1 shows such a BIH for resource requirements $\{64, 56, 16\}$. The graphical representation

Fig. 1. The blocking island hierarchy 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) the 16-BIG. (b) the 56-BIG. (c) the 64-BIG. (d) the network.

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

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 2 is the abstraction tree of Fig. 1.

The blocking island hierarchy summarizes the available bandwidth given the currently established connections at a time t . As connections 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, as explained in [9]:

- 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 capacity $\geq \beta$ within the BI, it causes a split of the BI. Furthermore, this split must be propagated 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 is propagated to all levels with a lower β .

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.

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

4 Routing from a BIH Perspective

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

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

Another way to see the criticalness of a route is to consider the *mapping* of the route onto the abstraction tree of Fig. 2: r_S is by far then the longest route, since its mapping traverses BIs N_1, N_5, N_7, N_6, N_3 , and then back; r_L traverses only BI N_1 . This observation (also) justifies the LL heuristic.

5 Solving a RAIN Problem

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

This mapping of routes onto the BIH is used to formulate a forward checking criterion, as well as dynamic value ordering and dynamic variable ordering heuristics. Some of these were briefly introduced in [10]. (We note that a patent for the methods given below is pending.)

5.1 Forward Checking

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 (for instance using the LL heuristic for computing a route – see Sect. 4), forward checking is performed first by updating the BIH, and then by checking that the route existence property holds for all uninstantiated demands in the updated BIH. If the latter property does not hold at least once, another route must be tried for the current demand. Domain pruning of open variables is therefore implicit while maintaining the BIH.

5.2 Value Ordering

A backtracking algorithm involves two types of choices: the next variable to assign (see Sect. 5.3), and the value to assign to it. As illustrated above, the

domains of the demands are too big to be computed beforehand. Instead, we compute the routes as they are required. In order to reduce the search effort, routes should be generated in “most interesting” order, so to increase the efficiency of the search, that is: try to allocate the route that will less likely prevent the allocation of the remaining demands. A natural heuristic is to generate the routes in *shortest path* order (SP), since the shorter the route, the fewer resources will be used to satisfy a demand.

However, Sect. 4 shows how to do better using a kind of min-conflict heuristic, the lowest level heuristic. Applied to the RAIN problem, it amounts to considering first, in shortest order, the routes in the lowest blocking island (in the BIH). Apart from attempting to preserve bandwidth connectivity, the LL heuristic allows to achieve a computational gain: the lower a BI is, the smaller it is in terms of nodes and links, thereby reducing the search space to explore. Moreover, the LL heuristic is especially effective during the early stages of the search, since it allows to take better decisions and therefore has a greater pruning effect on the search tree, as shown by the results in Sect. 7. Generating one route with the LL heuristic can be done in linear time in the number of links (as long as QoS is limited to bandwidth).

5.3 Variable Ordering

The selection of the next variable to assign may have a strong effect on search efficiency, as shown by Haralick [11] and others. A widely used variable ordering technique is based on the “fail-first” principle: “To succeed, try first where you are most likely to fail”. The 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-HL (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, the more allocating the demand may restrict the routing of the remaining demands (fail first principle), since it will use resources on more critical links.

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

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

6 Conflict Identification and Resolution

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 are able to directly *backjump* [12] to it (the intermediate assignments are obviously undone), thereby possibly drastically speeding up the search process.

When approximating the RAIN problem with a network multi-flow problem, BIs provide means to identify culprit allocations and, better, allow in some cases to prove that the problem is in fact unsolvable, since they highlight global bottlenecks in the network. Flow theory says that the maximal value of a flow from a node s to a node t never exceeds the capacity of any cut¹ separating s and t .

Suppose a set of demands were allocated in the network and that the network's available resources is given by Fig. 1 (d), and that we are now 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 , and a dead-end is reached. We call *primary blocking islands* (PBI) of a demand that cannot be allocated the two BIs of its endpoints at its bandwidth requirement level. The PBIs for d_n are N_1 and N_2 .

Given the latter definition and the result of flow theory, the following is easily established when a dead-end is reached during search:

- 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 the problem is infeasible obviously. Search can then be aborted, thereby saving us much effort².
- In case infeasibility cannot be proven, analyzing the situation on the links of the PBIs cocycle indeed helps to identify a *culprit assignment*. There are two cases:
 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

¹ A *cut* separating two nodes s and t is a set of links $\omega(A)$ (the cocycle of A), where A is a subset of nodes such that $s \in A$ and $t \notin A$.

² Note that if a problem is infeasible, it does not mean that unsolvability can be always be proven that way, firstly because not all cuts are being examined, and secondly because the RAIN problem is not a network flow problem.

- 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. Otherwise, re-allocating 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.

7 Empirical 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, 1 second), 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 networking world, in particular shortest-path algorithms.

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

We generated 23'000 RAIN problems in total, 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. The problems were solved with six different strategies on a Sun Ultra 60: *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. *BI-BJ-LL-HL* and *BI-BJ-LL-NL* differ from the previous in the use of backjumping to culprit decisions, as described in Sect. 6.

Figure 3 (a) gives the probability of finding a solution to a problem in less than 1 second, given the tightness of the problems (as defined above). 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

(a)

(b)

Fig. 3. Statistics on solving 23'000 randomly generated solvable problems with 20 nodes, 38 links, 80 demands each. (a) the probability of finding a solution within 1 second, given the tightness of the problems. (b) the probability of solving the problems according to run time.

much larger size, BJ algorithms do perform much better in average than their non-backjumping 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. Figure 3 (b) provides the probability of solving a problem according to run time. Two conclusions can be derived from the latter figure: first, BI methods curves continue to grow with time, albeit slowly, which is not the case for basic-SP and 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.

Note that the experimental results allow quantifying the gain obtained by using our methods. If an operator wants to ensure high customer satisfaction, demands have to be accepted with high probability. This means that the network can be loaded up to the point where the allocation mechanism finds a solution with probability close to 1. From the curves in Fig. 3 (a), we can see that for the shortest-path methods, this is the case up to a load of about 40% with a probability > 0.9 , whereas the NL heuristic allows a load of up to about 65%. Using this technique, an operator can thus reduce the capacity of the network by an expected 38% without a decrease in the quality of service provided to the customer!

According to phase transition theory [13], relative performance can be expected to scale in the same way to large networks. This is corroborated by first results on larger problems. We generated 800 different RAIN problems with 38 nodes, 107 links, and 1200 demands. The probability of solving such a problem according to run time is given in Fig. 4. Here, we see that the maintenance of the BIH has a larger effect on solving “easy” problems. However, after 20 seconds of run time the BI methods clearly are more efficient than the non-BI techniques. The facts noticed for the smaller problems (in Fig. 3) remain valid: NL outperforms HL, even if it requires slightly more run-time, and backjumping brings only a small advantage over their non-backjumping counterparts.

Further, as another result, BI-BJ-LL-NL solved a much larger RAIN problem (50 nodes, 171 links, and 3’000 demands) in 4.5 minutes. BT-SP was not able to solve it within 12 hours.

8 Conclusion

Much research in AI has focused on efficient algorithms to search for answers to problems posed in a particular way. However, intelligence not only involves answering questions, but also *asking* the right questions. It has long been observed that the complexity of finding a solution by search can depend heavily on how it is formulated. It is surprising how little attention has been paid to methods for putting search problems in a suitable form.

Fig. 4. The probability of solving a problem according to run time (800 problems, with 38 nodes, 107 links, 1200 demands).

In order to obtain general results in this direction, it is important to first consider techniques for *particular* difficult problems. In this paper, we have shown a dynamic abstraction technique for such a problem, resource allocation in networks (RAIN).

The next step will be to generalize this result to a larger class of problems with similar characteristics. For example, in techniques such as Graphplan [14], plans become paths in a graph. When many plans must be executed in parallel while sharing resources, for example in a factory, finding a suitable combination of plans again becomes a RAIN problem. Among other such problems are frequency allocation in radio networks, product configuration, and road traffic planning. We can hope to eventually develop quite general techniques for deciding when good abstractions are possible, and also for generating them.

Acknowledgments

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

of the developed tool. A special thank also to the reviewers for their help to improve this paper.

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] 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.
- [4] Jason W. Mann and George D. Smith. A Comparison of Heuristics for Telecommunications Traffic Routing. In *Modern Heuristic Search Methods*, John Wiley & Sons Ltd, 235–254, 1996.
- [5] Berthe Y. Choueiry and Boi Faltings. A Decomposition Heuristic for Resource Allocation. In *Proc. of the 11th ECAI*, pages 585–589, Amsterdam, The Netherlands, 1994.
- [6] Rainer Weigel and Boi V. Faltings. Structuring Techniques for Constraint Satisfaction Problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 418–423, San Francisco, August 1997, Morgan Kaufmann Publishers.
- [7] E.C. Freuder and D. Sabin. Interchangeability Supports Abstraction and Reformulation for Multi-Dimensional Constraint Satisfaction. In *Proc. of AAAI-97*, pages 191–196, Providence, Rhode Island, 1997.
- [8] Symposium on Abstraction, Reformulation and Approximation (SARA-98). Supported in Part by AAAI, Asilomar Conference Center, Pacific Grove, California, May 1998.
- [9] Christian Frei and Boi Faltings. A Dynamic Hierarchy of Intelligent Agents for Network Management. In *2nd Int. W. on Intelligent Agents for Telecommunications Applications, IATA'98*, pages 1–16, Paris, France, July 1998. Lecture Notes in Artificial Intelligence, Springer-Verlag.
- [10] Christian Frei and Boi Faltings. Bandwidth allocation heuristics in communication networks. In *1ères Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications, ALGOTEL'99*, pages 53–58, Roscoff, France, 1999.
- [11] R. M. Haralick and G. L. Elliott. Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Artificial Intelligence*, 14:263–313, 1980.
- [12] J. Gaschnig. Experimental case studies of Backtrack vs. Waltz-type new algorithms. In *Proceedings 2-nd Biennial Conf. Canadian Society for Computational Study of Intelligence*, Toronto, Ontario, July 1978.
- [13] Peter Cheeseman and Bob Kanefsky and William M. Taylor. Where the Really Hard Problems Are. In *Proc. of the 12th IJCAI*, pages 331–337, Sidney, Australia, 1991.
- [14] Avrim Blum and Merrick Furst. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence*, 90:281–300, 1997.