# Bandwidth allocation heuristics in communication networks

**Christian Frei** and **Boi Faltings**
Artificial Intelligence Laboratory
Swiss Federal Institute of Technology (EPFL)
CH-1015 Lausanne, Switzerland
Voice: +41-21-693-6682 Fax: +41-21-693-5225
frei/faltings@lia.di.epfl.ch

## 1 Introduction

With the liberalization of the telecommunications industry, networks are being used in more and more flexible ways. Using technologies such as ATM, service providers create virtual networks on top of the physical networks. This new variability means that physical network resources must frequently be allocated to these virtual networks. Formally, 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 bandwidth 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, it is impossible to divide demands among multiple routes. 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 [10] 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. Shortest path routing can lead to poor network utilization and even congestion.

Constraint satisfaction [9] 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 computationally 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 (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 69281 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 abstraction of the network creates 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.

## 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. Bandwidth auctioning through a multi-agent system, such as in [8], was explored; however, this work is still at an early stage.

To our knowledge, the closest published work to ours is the CANPC framework [7]. 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.

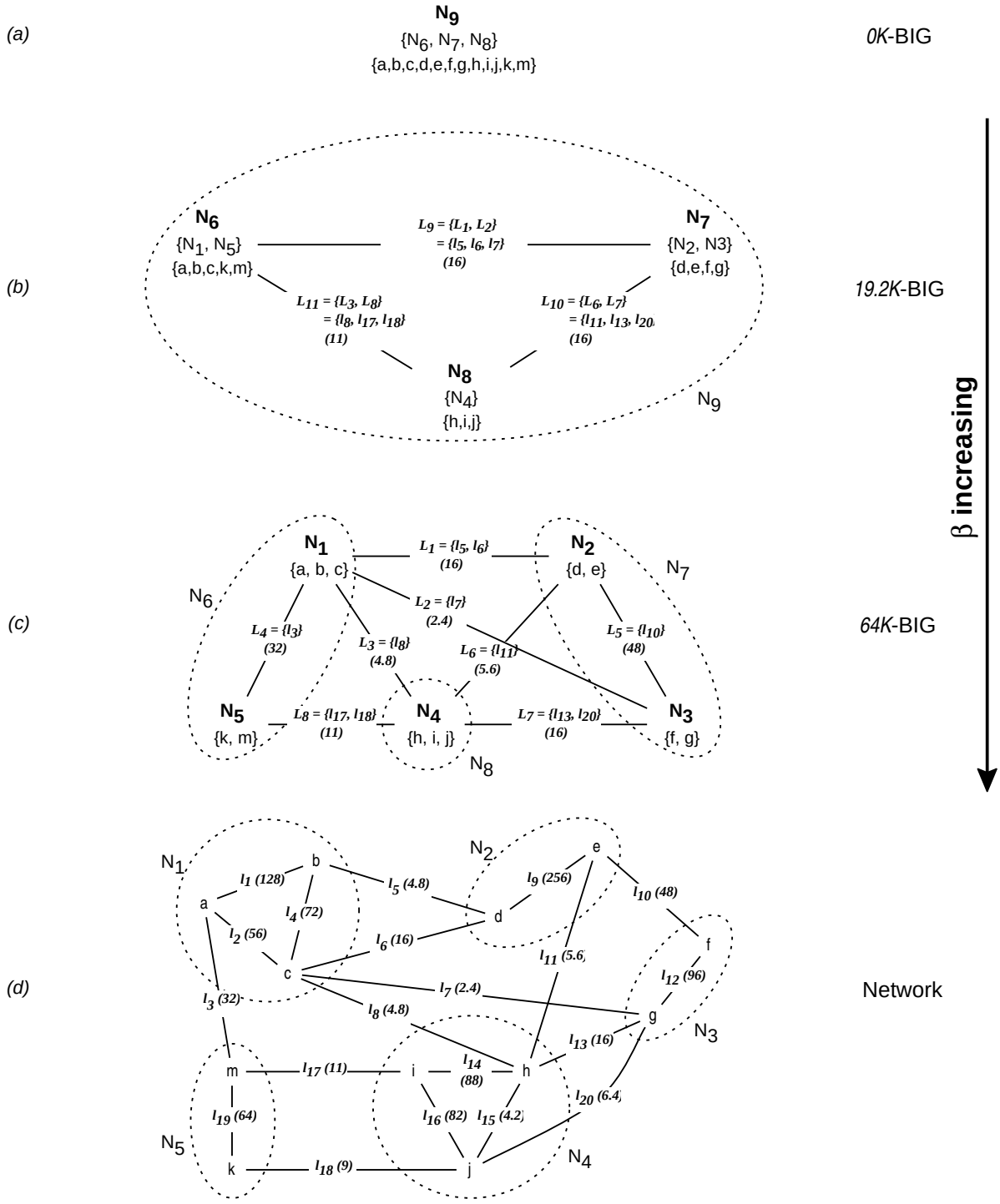


Figure 1: The blocking island hierarchy for resource requirements $\{64K, 19.2K\}$. The weights (in parenthesis) on the links are the available resources. Abstract links' description include their children and network children in brackets. Abstract nodes' description include only their node children and network node children in brackets: link children are omitted for more clarity. (a) the 0-BIG. (b) the $19.2K$-BIG.(c) the $64K$-BIG. (d) the network graph.

Mann and Smith [6] 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. [2] relate interchangeability to abstraction in the context of a decomposition heuristic for resource allocation. [11] 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. [4] present abstraction and reformulation techniques based on interchangeability to improve solving CSPs. [1] is a recent collection of papers addressing abstraction, reformulation, and abstraction techniques in a variety of AI techniques.

## 3 The Blocking Island paradigm

[3] introduce a clustering scheme based on Blocking Islands, which can be used to represent bandwidth availability at different levels of abstraction, as a basis for distributed problem solving. A $\beta$-blocking island ($\beta$-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 $\beta$ available resources, including $x$. Figure 1 (d) shows all $64K$-BIs for a network. Note that some links inside a $\beta$-BI, i.e., the links that have both endpoints in the $\beta$-BI, may have less than $\beta$ available resources. In such a case, it simply means that there is another route with $\beta$ available resources between the link's endpoints. As a matter of fact, link $l_2$ has both endpoints in the $64K$-BI $N_1$ but has less than 64 available resources. However, there are at least 64 available resources along route $\{l_1, l_4\}$ between $l_2$'s endpoints.

$\beta$-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 $\beta_u$-BI. Furthermore, all links that could form part of such a route lie inside this blocking island.*

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

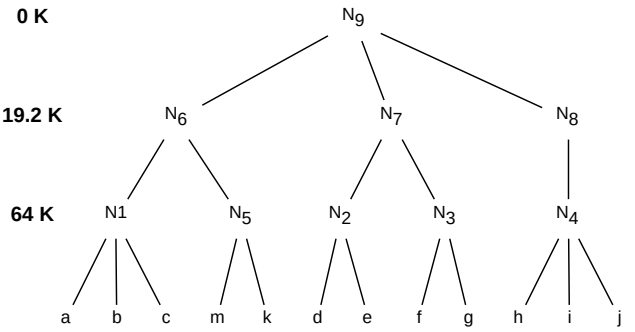


Figure 2: The abstraction tree of the BIH of Figure 1 (links are omitted for clarity).

In order to identify bottlenecks for different $\beta$s, e.g., for typical possible bandwidth requirements, we build a recursive decomposition of BIGs in decreasing order of the requirements: $\beta_1 > \beta_2 > ... > \beta_b$. This layered structure of BIGs is a *Blocking Island Hierarchy* (BIH). The lowest level of the blocking island hierarchy is the $\beta_1$-BIG of the network graph. The second layer is then the $\beta_2$-BIG of the first level, i.e., $\beta_1$-BIG, the third layer the $\beta_3$-BIG of the second, and so on. On top of the hierarchy there is a 0-BIG abstracting the smallest resource requirement $\beta_b$. 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 $\{64K, 19.2K\}$. 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 $\beta$-BIGs, but also as an *abstraction tree* when considering the father-child relations. In an 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 the BIH of Figure 1.

The $\beta$-BI $S$ for a given node $x$ of a network graph can be obtained by a simple greedy algorithm, with a linear complexity of $\mathcal{O}(m)$, where $m$ is the number of links.

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

The construction of a $\beta$-BIG is straightforward from its definition and is also linear in $\mathcal{O}(m)$. 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 $\mathcal{O}(bm)$, 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 $\mathcal{O}(bm)$ algorithms.

## 4 Routing as a CSP

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) [9]. 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 $\beta$-BI of its endpoints, where $\beta$ 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 2, it is easy to see that there is no route between $a$ and $m$ with 64 available resources, since any path in the tree must at least cross BIs at level 19.2. This mapping of routes onto the BIH is used to formulate dynamic variable and value ordering heuristics.

**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 $\beta$-BI, where $\beta$ 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 is thus implicit while maintaining the BIH.

**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, as shown by Haralick [5] and others. A widely used variable ordering technique is based on the "fail-first" principle (FFP): "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 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): 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.

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

**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.

However, 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

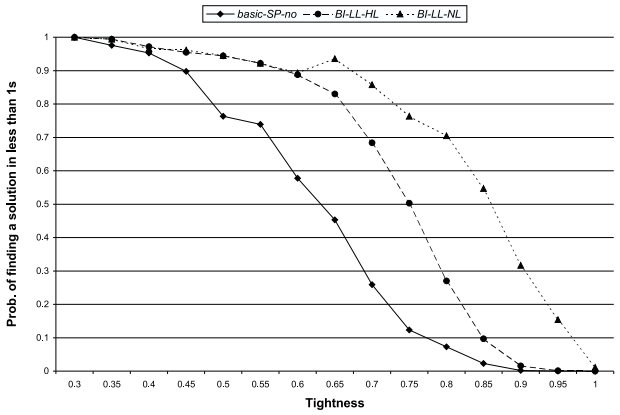


Figure 3: The probability of finding a solution within 1 second, given the tightness of the problems.

of future allocation failures. We call this special kind of load-balancing *preserving bandwidth connectivity*. 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 (as long as QoS is limited to bandwidth constraints).

## 5 Results and conclusion

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.

Figure 3 provides the probability of finding a solution to a problem in less than 1 second, given the tightness of the problems. The statistics were obtained from 22'000 randomly generated problems, each with 20 nodes, 38 links and 80 demands, and each having at least one solution. Each problem as a randomly generated network topology. Three curves are displayed: *basic-SP-no* performs a search using the shortest path heuristic common in the telecom world today; *BI-LL-HL* uses the LL heuristic for route generation and HL for DVO, whereas *BI-LL-NL* differs from the latter in using NL for DVO. As shown in Figure 3, both BI search methods are much better than brute-force, even on these small problems, where heuristic computation 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 on the search tree. The shape of the curves are similar for larger time scales. 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.

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!

## References

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

[2] Berthe Y. Choueiry and Boi Faltings. A Decomposition Heuristic for Resource Allocation. In *Proc. of the 11 th ECAI*, pages 585–589, Amsterdam, The Netherlands, 1994.

[3] 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.

[4] E.C. Freuder and D. Sabin. Interchangeability supports abstraction and reformulation for multi-dimensional constraint satisfaction. In *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI-97/IAAI-97)*, pages 191–196, July27–31 1997.

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

[6] Jason W. Mann and George D. Smith. A Comparison of Heuristics for Telecommunications Traffic Routing. In *Modern Heuristic Search Methods*, pages 235–254. John Wiley & Sons Ltd, 1996.

[7] 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.

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

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

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

[11] 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, August23–29 1997. Morgan Kaufmann Publishers.