A Decomposition Heuristic for Resource Allocation

Berthe Y. Choueiry[1] **and Boi Faltings**[2]

Abstract. The allocation of resources to a set of tasks with fixed start and end times is known to be NP-complete. We propose an approximation based on identifying pools of interchangeable resources and tasks using a novel decomposition heuristic. The pools are abstractions which provide simple representations for making conflicts explicit. The method is an anytime algorithm that efficiently finds a sub-optimal solution which can then be improved by refining the underlying decomposition. This is more practical than constraint relaxation techniques traditionally used to deal with such problems.

1 Introduction

Planning, scheduling and resource allocation are closely related tasks. Because they intervene at different time scales, they are often decoupled. Consider, for instance, the problem of managing flights of an airline company. Purchasing airplanes, hiring personnel, setting flight timetables as well as making large investment decisions are *planned* for a relatively long period of time and are subject to general company policies. Regular maintenance work for airplanes and working time shifts for personnel are *scheduled* for a shorter time horizon, say on a monthly basis. On a much finer time scale, human operators must decide how to *allocate* the available resources, (e.g., airplanes, crew, maintenance teams, etc.) in order to execute the scheduled tasks for the next day or week. This resource allocation problem is the focus of this paper.
Because of the difficulty of correctly and completely formulating all constraints on this problem, fully automatic methods are usually not acceptable. We propose a division of labor between the user and the computer. The machine decouples a problem into abstract subcomponents which serve to simplify interactive human decision making.

2 Problem definition

In Fig. 1 (left), we show a simple example of a resource allocation problem. For each task, we define a set of resources that can execute the task. The problem is to assign one resource to each task such that no resource is allocated to more than one task at some time.
We express the resource allocation problem as a *discrete* Constraint Satisfaction Problem (CSP) [19]. A *constraint graph* is associated with the CSP where nodes (variables) represent tasks to be executed, labels (value sets) are resource sets, and arcs (constraints) link nodes that intersect in time and have at least one resource in common (Fig. 1 right). A single resource may not be allocated to two linked nodes (constraint of *mutual exclusion*).
This CSP is equivalent to a *list coloring*[3] problem in interval graphs, which is known to be NP-complete[4] [1].

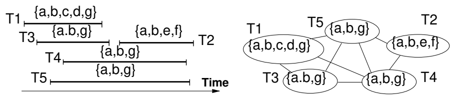

Figure 1. *Left:* A schedule of five tasks whose start and end time have been determined in the scheduling phase. For each task a set of possible resources is shown. *Right:* Constraint graph.

3 Problem decomposition heuristic

In this section, we introduce a heuristic for decomposing resource allocation problems and show how it applies to a simple example. The decomposition algorithms and termination criteria can be found in [5], and are applied to a real-world example in [6].
The decomposition heuristic, Value-Assignment Delay heuristic (VAD), operates on constraint graphs. In practical resource allocation problems, many resources are solicited by many tasks at the same time and the corresponding constraint graphs tend to be densely connected. The intuition behind the VAD heuristic is to limit the high degree of interdependency among tasks by hiding the shared resources. This operation, when applied repeatedly to the most solicited values, will result in progressive elimination of arcs in the constraint graph, gradually separating it into independent clusters.
The VAD heuristic is summarized as follows: *Delay the assignment of the most solicited values and try to solve the simplified problem without them. Distribute the delayed values to those variables that really cannot do without them.*
We show how this applies to the simple example of Fig. 2 (left). The initial constraint graph has five nodes, $N_1 \ldots N_5$, annotated with their labels. A backtrack-search procedure that starts by assigning value a to either N_1 or N_2 will not find

[1] This research is supported by the Swiss National Science Foundation Project#5003-034284.
[2] Artificial Intelligence Laboratory, Swiss Federal Institute of Technology (EPFL), INR-Ecublens, CH-1015 Lausanne, Switzerland

[3] List coloring differs from ordinary *graph coloring*, also called *minimal coloring*, in that the set of colors allowed for each node is restricted, and may be different for each node.
[4] Note that when all resources are equivalent, the problem reduces to graph coloring in interval graphs which can be solved in linear time [15, 14].

ECAI 94. *11th European Conference on Artificial Intelligence* Edited by A. Cohn
Published in 1994 by John Wiley & Sons, Ltd.

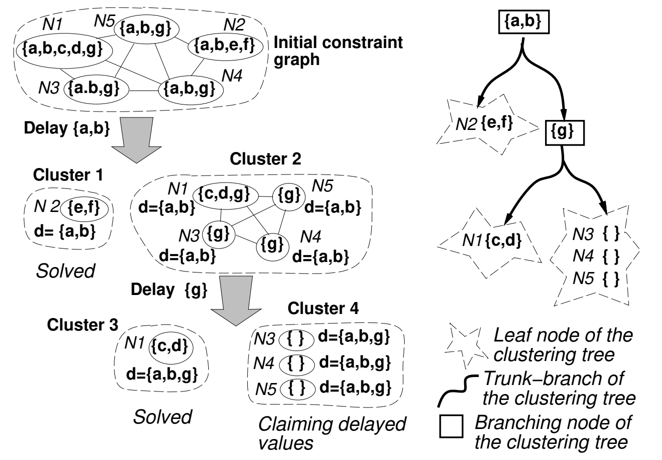

Figure 2. *Left:* Applying the Value-Assignment Delay heuristic to a simple resource allocation problem. *Right:* The resulting clustering tree.

a solution before going through a significant amount of backtracking. In Fig. 2, we apply iteratively the value-assignment delay heuristic to the initial constraint graph, and then to the generated clusters.

The set of most solicited values is $\{a, b\}$. The criterion for selecting this set is discussed in Section 4. The VAD delays the assignment of these values. The initial cluster splits into two parts because N_2 now shares no values with the rest of the graph. The values whose assignments are delayed are called *delayed values.* In $Cluster_1$, one can assign value e or f to N_2 without hesitation. In $Cluster_2$, the four nodes are still competing for common values, and each has, as a reserve, a set of delayed values, $d = \{a, b\}$. At the next step, $\{g\}$ is delayed, and unsolved $Cluster_2$ is split further into two parts $Cluster_3$ and $Cluster_4$. N_1 in $Cluster_3$ can be assigned either c or d without affecting the rest of the problem. The remaining three nodes are left with an empty set of possible values. Each of these nodes reclaim the delayed values $\{a, b\}$ and $\{g\}$.

The application of the VAD heuristic to the initial graph yields a tree structure, the *clustering tree*, in which the branching nodes are pools of delayed values (potential sources of conflicts), the leaves are aggregates of tasks and non-conflicting values (subproblems), and the branches show how these components relate together, Fig. 2 (right). Resources are said to be *upstream* from a collection of tasks if they are on the path leading from these tasks to the root of the tree and tasks are said to be located *downstream* from a given set of delayed values if they belong to the subtree rooted at this set. The structure of the tree shows how the isolated subproblems relate and will serve as a basis for conflict isolation. This feature is important for interactive problem solving.

Once the problem has been decomposed, the subproblems in the leaves are reduced in size by the application of an abstraction operator called *summarization*[5][6, 7], which will not be discussed here due to lack of space.

Discovering useful abstractions. The VAD heuristic is itself an *abstraction technique.* By grouping values on the trunk of the partitioning tree and by pushing variables into the leaves, it implicitly creates aggregates of objects: it abstracts out the features of each atomic object and retains the similarities and differences among these entities as far as value contention is concerned. For instance, in Fig. 2, the VAD discovered that c and d are equivalent with respect to N_1, .

In [7], we characterize these new components in terms of interchangeable sets [11]. We claim that the VAD decomposition scheme structures the solution space into *compact families*: many solutions can be generated simply by *switching* interchangeable values among tasks in the same leaf. Classical enumerative methods fail to organize the solution space in such a concise way and to identify the boundaries within which the effect of a change remains local.

Conflict detection. In the example of Fig. 2, $Cluster_1$ and $Cluster_3$ can be solved independently. $Cluster_4$ is claiming value sets $\{a, b\}$ and $\{g\}$, which were delayed higher up in the tree, and is not yet solved. For each set of delayed values claimed by one or more unsolved leaves we define a *conflict*, as show in Fig. 3.

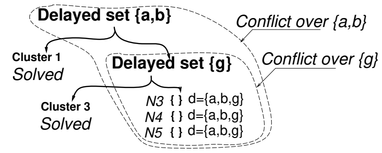

Figure 3. *Conflict isolation.*

Definition 1 *A* conflict *is defined by the set of delayed resources at a branching node of the clustering tree and the tasks downstream that solicit these resources.*[6]

Definition 2 *A* conflict resolution *is the decision to allocate a delayed resource to one of the tasks that solicits it.*

Overconstrained problems. Suppose, in Fig. 2, the value g did not appear in any of the labels. The decomposition by the VAD yields the tree shown in Fig. 4. The conflict detection procedure isolates the nodes which contend for the same resources, thus defining a *conflict set*.[7] Sources of conflicts can then be presented to the user in a compact manner. This is to be contrasted to the enumerated conflict sets returned by a dependency directed backtracking procedure or an assumption-based truth maintenance system. In [7], we propose to apply various abstraction operators to these sets in a concept formation procedure for assessing problem tightness and aiding in rescheduling.

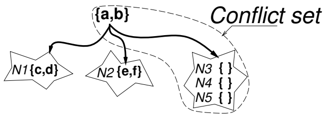

Figure 4. *Identifying overconstrained problems.*

[5] *Summarization* reduces the size of the leaf by merging tasks which do not overlap in time according to some criteria.

[6] Only the tasks downstream the set of values need to be considered because the partitioning of the initial problem implies that no other tasks need the delayed resources.

[7] We still have to check that such sets are minimal.

Broad Brush Effect. During the conflict resolution procedure, subproblems downstream of a given branching node may compete for values delayed at a higher level in the hierarchy. This does not imply that all tasks in different leaves *necessarily* compete for the values delayed, since their time intervals do not necessarily intersect. In the example of Fig. 5, T_2 and T_3 do not intersect in time and could be assigned the same resource a. However, since the decomposition places them in separates leaves, they appear to compete for the same value. This apparent competition among tasks belonging to separate leaves is caused by what we call the *broad brush* effect (BBE) of the decomposition procedure.

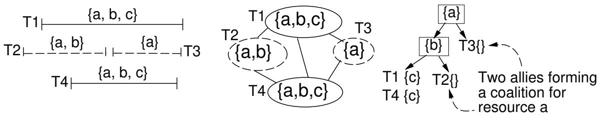

Figure 5. *Illustration of the broad brush effect: Left:* Four tasks and their possible resources. *Center:* Constraint graph. *Right:* Decomposition by the VAD.

When the tasks in two apparently competing leaves do not all intersect in time[8] the two leaves define a possible *alliance*. They can be merged and the non-intersecting tasks can be aggregated, thus reducing the overall need for resources.

One may say that the VAD heuristic paints the problem space with a *broad brush*. The problem space is quickly decomposed as if it was covered with paint by a broad brush. All the spots covered are correct, all the solutions proposed are sound. However, some spots "in the corners" may be missed: some opportunities for solution are overlooked.

This effect can show up in any decomposition strategy that considers the problem too globally[9] In order to take into account all the details, discriminative methods must enumerate and thus are inappropriate for interactive problem solving. All strategies that generate compact decompositions are likely to face a broad brush effect.

Dealing with the BBE in the VAD. When the initial constraint graph is complete, the BBE may not occur during the decomposition by the VAD because no alliance can be found among the generated subcomponents.

Lemma 1 *A complete graph is BBE-free when processed by the VAD heuristic.*

Corollary 1 *If, at any stage of the decomposition by the VAD heuristic, the constraint graph of a generated subproblem is complete, no alliances can be found among the offsprings generated starting from this node.*

Consequently, the goal is to produce graphs that remain as complete as possible to reduce the occurrences of the BBE. Since the start and end times of all intervals are fixed, and the labels (values) of each interval (node) are determined by the problem instance, the initial constraint graph cannot be manipulated to reduce the BBE. The only degree of freedom left is the *selection of the values to be delayed at each step* of the VAD heuristic. Thus, one should use a strategy for the selection and ordering of the values to be delayed in a list coloring problem that yields the greatest possible number of complete subproblems. One potential solution is to delay exactly those values that decompose the interval graph into its maximal cliques[1](Section 5). However, this approach yielded no change when applied to the examples in Section 6. Tests will be carried out on randomly generated cases.

Automatic conflict resolution. Although our goal is to allow the users to selectively participate in conflict resolution by viewing and manipulating conflicts and alliances determined in the previous steps, we also provide an automatic conflict resolution procedure. This procedure may integrate domain dependent knowledge or may be heuristic. For instance: one might prefer to allocate a delayed value to the tasks that participate in the least number of alliances, thus adopting a *least commitment strategy*, or one might select the alliance that most reduces the demand on delayed resources. A list of heuristic strategies for conflict resolution can be found in [5].

One may consider a complete backtrack search within the space of isolated conflicts and discovered alliances. Such backtracking would benefit from the grouping of resources and tasks into aggregates, and from the knowledge of possible alliances. We call such a procedure *informed backtracking*: backtracking iterates over sets of interchangeable objects rather than over atomic values and variables. We intend to further investigate this new backtracking scheme and compare it with the one proposed in [16].

4 Computing the resource set to delay

It is insufficient to simply delay those values that appear in the largest number of nodes. The values must belong to *competing* nodes (i.e. intervals that intersect). Therefore, we must compute *resource contention curves* over time, similar to those in [2, 20].

Fig. 6 shows the time representation of a set of tasks, $T_1 \dots T_{16}$, competing for a set of resources, $R_1 \dots R_4$. The contention curve of a given resource in a given cluster[2]is built by adding up the demands of all jobs in the cluster that can be executed by the resource. In order to determine the most solicited resource(s), a *cumulative demand list* is computed for each resource from its contention curve by counting the number of time units a given value appears in its contention curve and ordering these values in decreasing order. The cumulative demand lists of a cluster are then compared according to the *cumulative demand list* (CDL) *comparison criterion* shown in Fig. 7, and only the ones with the highest values in their lists are retained. R_2 is rejected in the example of Fig. 6.

If this procedure ends up with more than one resource, the remaining resources are discriminated further by retrieving the list of tasks to which they can be allocated. Those that are solicited by the same set of tasks are grouped into one

[8] In the general case of list coloring, the broad brush effect appears between nodes linked by no constraint in the initial graph.

[9] The BBE is somewhat different from the loss of information due the use of Theorem-Decreasing Abstractions [13]: problem decomposition does not modify the problem space or lose the details, it simply neglects them and they can be recovered whenever necessary.

[1] Note that those cliques usually *contain*, but are not necessarily equal to, the cliques of the constraint graph.

[2] A resource can figure independently in one or more clusters and have thus one or more cumulative contention curves.

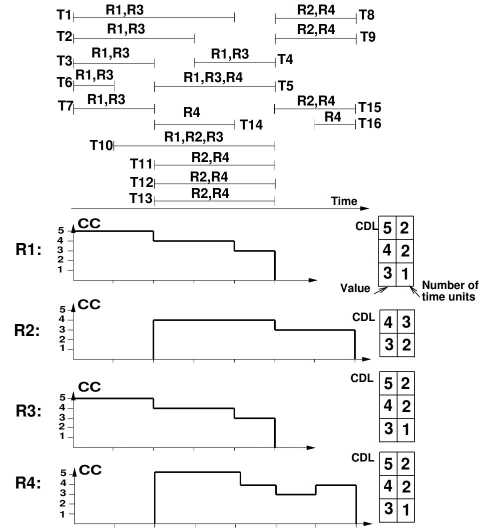

Figure 6. *Contention curves* (CC) *and cumulative demand lists* (CDL) *for* $R_1, \ldots, R_4$.

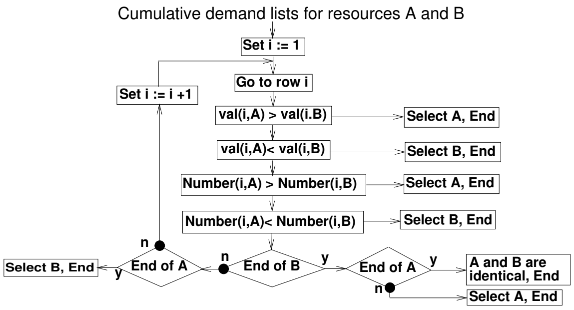

Figure 7. *Cumulative demand lists comparison criterion.*

set, S_{dri}, that constitutes a set of values that can be delayed simultaneously. In the example of Fig. 6, we have:

$R_1 : \{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_{10}\}$

$R_3 : \{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_{10}\}$

$R_4 : \{T_8, T_9, T_{11}, T_{12}, T_{13}, T_{14}, T_{15}, T_{16}\}$

Hence, $S_{dr1} = \{R_1, R_3\}$ and $S_{dr2} = \{R_4\}$.

A heuristic still has to be developed for discriminating among those sets of resources. One could select as the set of resources to be delayed: 1) the one with the greatest number of resources, S_{dri} of highest cardinality, 2) the one with the least number of resources, S_{dri} of lowest cardinality, 3) the one that is solicited by the greatest number of tasks, or 4) the one that is solicited by the least number of tasks. We have not yet studied the effects of these choices.

5 Related work

Least commitment strategies and opportunistic approaches to decision making are features common to many AI systems [12]. Our contribution is to provide an overall structure to guide the decision process by isolating easy problems, identifying conflicts, and organizing the solution space into families of solutions by discovering interchangeabilities. More discussion about relations to other work can be found in [6].

Problem reduction. The resource allocation problem is a list coloring problem in interval graphs (see Section 2). Reduction techniques by vertex elimination (node deletion) and vertex merging (node contraction) have been widely used in graph coloring. Nodes of degree less than the cardinality of their label (or less than the chromatic number) can be safely eliminated from the graph [17, 4]. Nodes which are similar according to some similarity measure can be merged [22, 3, 4]. One may also consider local consistency techniques in CSPs as a kind of problem reduction: some values, or combination of values, are removed according to a consistency criterion.

Both graph reduction and consistency enforcement techniques rule out values in nodes which, although they do not participate in a feasible global solution, may be a good backup choice when the user is willing to relax some constraints. We need to keep these alternatives visible to the user, as the VAD does in its generated tree structure.

For testing purposes, we adapted the two most common graph reduction operators used in graph coloring to the case of list coloring. The first operator removes a node when its label's cardinality is higher than its degree. The second operator is based on discovering *false twins* (also called even pairs) [3, 4]. Very few reductions were noticed on the examples taken from our case study. Also, the loss of information was judged too severe and unacceptable for an interactive problem resolution framework.

In the case of list coloring, it is obvious that arc consistency will be inoperative whenever each node has more than one value in its label. This makes us consider the local consistency of higher orders. In [8], Dechter investigates the relationship between the level of consistency needed for achieving global consistency, the number of discrete values in the graph and the arity of constraints. However, in practice, the number of values and the width of the graph are often too high to allow the use of such techniques.

Clustering by simplicial decomposition. The interval graph of the resource allocation problem as defined in Section 2 is triangulated [14]. This raises the question how the tree decomposition scheme proposed in [9] can be exploited. An interval graph can easily be decomposed into *a chain* of its maximal cliques[3] Each maximal clique would constitute a cluster as required in [9]. The subproblems in clusters are solved independently, the solutions obtained are considered as singleton variables, directional arc-consistency is performed on the element of the chain and the chain can finally be solved in a backtrack-free manner. The complexity predicted is of $O(nk^r)$ where n is the number of maximal cliques, k is the number of values for each variable and r is the size of the largest clique. If the interval graph admits a maximal clique partition [21], the maximal cliques are "linked" by articulation points and, according to Freuder [10], the problem can be solved in time exponential in r, where r is the size of the largest clique.

These techniques give a bound on the complexity of list coloring in interval graphs[4] However, they give no method other than backtrack search for solving the cliques. In cases where the cliques admit solutions (which they usually do in practical applications) but the problem globally does not (which is

[3] Maximal cliques of an interval graph can be found in $O(|V|+|E|)$ [18].

[4] Another bound is $O(n^{k+1})$, where n is the number of nodes and k is the number of values [1].

also frequently the case), these techniques fail to isolate the sources of conflicts and thus fail to propose amendments.

6 Implementation and results

An example of the application of our techniques to a real world problem as well as the description of the implementation can be found in [6]. The sample problems tested were taken from the daily archives of a large hospital: the allocation of operating rooms and of a variety of technicians to surgical operations. The CSP measures [16] on these tests are as follows[5] $\langle n, a, t, d \rangle = \langle 22 \ldots 72,\ 5 \ldots 31,\ 0.037 \ldots 0.108,\ 0.376 \ldots 0.570 \rangle$.

We compared our technique with a traditional backtrack search with forward checking, backjumping, and some variable ordering heuristics. For those cases where no solutions existed, the backtrack search thrashed forever and the users could not guess whether they should wait for an outcome, or if the problem was overconstrained and why. This is unacceptable in practice. In contrast, our algorithm always terminates in polynomial time, with correct solutions for underconstrained problems and indications of conflicts and missing resources for those that are overconstrained. In overconstrained cases, the conflict detection and problem decomposition proved to be valuable.

There were cases where the backtrack search found solutions that were missed by our technique. This is due to the simplification introduced by an abstraction operator, called *summarization* [6, 7] and also by the least-commitment strategy for automatic conflict resolution. However, often the solutions found by backtrack search were not acceptable to the human operator. For example, 1) a technician was allocated to distant operating rooms, 2) some technicians were not given any load at all while some others were overloaded and, not surprisingly, 3) some technicians were allocated to appropriate tasks although they do not get along well with the surgeon. These problems are avoided in our framework because the expert can intervene directly to force in subjective preferences.

All in all, results were positive. However, tests on randomly generated problems should be carried out before we can draw final conclusions. This is an on-going task.

7 Conclusions

Traditional techniques used for solving the resource allocation problem are based on variations of the backtrack search mechanism and are inevitably undermined by risks of combinatorial explosion. We have introduced a decomposition heuristic which structures the solution space of a resource allocation problem in a compact manner suitable for interactive problem solving. It separates easy and hard subproblems, shows how they relate, isolates conflict sets, discovers abstractions of variable size, and serves as a basis for concept formation for assessing problem tightness and for helping in rescheduling and explanation.

[5] n is the number of variables, a the maximum domain size, $t = \frac{|forbidden\ tuples|}{|all\ tuples|}$ the constraint tightness, $d = \frac{e - e_{min}}{e_{max} - e_{min}}$ the constraint density, e the number of constraints, $e_{min} = (n-1)$, and $e_{max} = \frac{n(n-1)}{2}$.

REFERENCES

[1] E. Arkin and E. Silverberg, 'Scheduling Jobs with Fixed Start and End Times', *Discrete Applied Math.*, **18**, 1–8, (1987).

[2] P. M. Berry, *A Predictive Model for Satisfying Conflicting Objectives in Scheduling Problems*, Ph.D. dissertation, Dept. of Computer Science, University of Strathclyde, Glasgow, UK, 1991.

[3] M. E. Bertschi, *La Colorabilité Unique dans les Graphes Parfaits*, Ph.D. dissertation, Faculté des Sciences, University of Lausanne, Switzerland, 1988. In French.

[4] P. Cheeseman, B. Kanefsky, and W. M. Taylor, 'Where the Really Hard Problems Are', in *Proc. of the 12 th IJCAI*, pp. 331–337, Sidney, Australia, (1991).

[5] B. Y. Choueiry and B. Faltings, 'Temporal Abstractions and a Partitioning Heuristic for Interactive Resource Allocation', in *Working notes of the Workshop on Knowledge-Based Production Planning, Scheduling and Control, IJCAI'93*, pp. 59–72, Chambéry, France, (1993). Anonymous ftp liasun3.epfl.ch: /lia.

[6] B. Y. Choueiry and B. Faltings, 'Interactive Resource Allocation by Problem Decomposition and Temporal Abstractions', in *Current Trends in AI Planning.*, eds., C. Bäckström and E. Sandewall, 87–104, IOS Press, Amsterdam, (1994). Anonymous ftp liasun3.epfl.ch: /lia.

[7] B. Y. Choueiry, B. Faltings, and G. Noubir, 'Abstraction Methods for Resource Allocation', in *Workshop on Theory Reformulation and Abstraction*, Jackson Hole, Wyoming, (1994).

[8] R. Dechter, 'From Local to Global Consistency', *Artificial Intelligence*, **55**, 87–107, (1992).

[9] R. Dechter and J. Pearl, 'Tree Clustering for Constraint Networks', *Artificial Intelligence*, **38**, 353–366, (1989).

[10] E. C. Freuder, 'A Sufficient Condition for Backtrack-Bounded Search', *JACM*, **32 no. 4**, 755–761, (1985).

[11] E. C. Freuder, 'Eliminating interchangeable values in constraint satisfaction problems', in *Proc. of AAAI-91*, pp. 227–233, Anaheim, CA, (1991).

[12] P. A. Geelen, 'Dual viewpoint heuristics for binary constraint satisfaction problems', in *Proc. of the 10 th ECAI*, pp. 31–35, Vienna, Austria, (1992).

[13] F. Giunchiglia and T. Walsh, 'A Theory of Abstraction', *Artificial Intelligence*, **57**, 323–389, (1992).

[14] M. C. Golumbic, 'Algorithmic Aspects of Perfect Graphs', *Annals of Discrete Mathematics*, **21**, 301–323, (1984).

[15] U. J. Gupta, D. T. Lee, and J. Y.-T. Leung, 'An Optimal Solution for the Channel Assignment Problem', *IEEE Transactions on Computers*, **C-28 (11)**, 807–810, (1979).

[16] A. Haselböck, 'Exploiting Interchangeabilities in Constraint Satisfaction Problems', in *Proc. of the 13 th IJCAI*, pp. 282–287, Chambéry, France, (1993).

[17] A. B. Kempe, 'On the Geographical Problem of Four Colours', *American Jounal of Mathematics*, **2**, 193–204, (1879).

[18] C. G. Lekkerkerker and J. Ch. Boland, 'Representation of a Finite Graph by a Set of Intervals on the Real Line', *Fund. Math.*, **51**, 45–64, (1962).

[19] A. K. Mackworth, 'Consistency in Networks of Relations', *Artificial Intelligence*, **8**, 99–118, (1977).

[20] N. Sadeh, *Look-ahead Techniques for Micro-Opportunistic Job Shop Scheduling*, Ph.D. dissertation, Dept. of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1991.

[21] M. Shaohan and W. D. Wallis, 'Maximal-Clique Partitions of Interval Graphs', *Journal of Australian Math. Society*, **45**, 227–232, (1988).

[22] D. C. Wood, 'A Technique for Colouring a Graph Applicable to Large Scale Timetabling Problems', *Computer Journal*, **12**, 317–319, (1969).