Abstraction by Interchangeability in Resource Allocation

Berthe Y. Choueiry, Boi Ealtings and Rainer Weigel
Laboratoire d'Intelligence Artificielle
Swiss Federal Institute of Technology (EPFL)
IN-Ecublens, CH-1015 Lausanne, Switzerland
E-mail: [choueiry|faltings|weigel]@lia.di.epfl.ch

Abstract

Resource allocation is a difficult constraint satisfaction problem that has many practical applications. Fully automatic systems are often rejected by the ultimate users because, in many real-world environments, constraints cannot be formalized completely. On the other hand, humans are overwhelmed by the complexity of their task. We present a new way of solving the resource allocation, where a computer builds dynamic abstractions that simplify problem solving to the point that the user can intervene in the solution of the problem. These abstractions are based on the concept of interchangeability introduced by Freuder.
In this paper, we describe a heuristic for decomposing a resource allocation problem into abstractions that reflect interchangeable sets of tasks or resources. We assess the "quality" of the discovered neighborhood interchangeable sets by comparing them to the ones obtained by the exact algorithm described by Freuder, both for data taken from a real-world application and for randomly generated problems.

1 Introduction and motivation

The resource allocation (RA) problem is to assign resources to a set of tasks, scheduled at given time intervals, such that no resource is assigned to two different tasks at the same time. It arises in many real-world applications.

- In *manufacturing,* once the various jobs have been scheduled, it appears as the problem of distributing tools to the machining centers and of allocating routing vehicles to transfer the material.
- In the *operating units of a hospital,* the availability of personnel and the program of surgical operations to be executed are decided at various administrative levels, days or weeks in advance. On a daily basis, one or two human operators distribute the qualified personnel (nurses and technicians) to the surgical operations as required.
- In an *airline company^* the problem arises for allocating, to the various scheduled flights, aircraft, cabin crew, gates; as well as personnel, vehicles, and equipment for catering, baggage handling, cleaning, and fuel refilling.

Fig. 1 shows an example of a resource allocation problem. The problem can be represented as interval orders, as shown on the left of Fig. 1. In this paper, however,

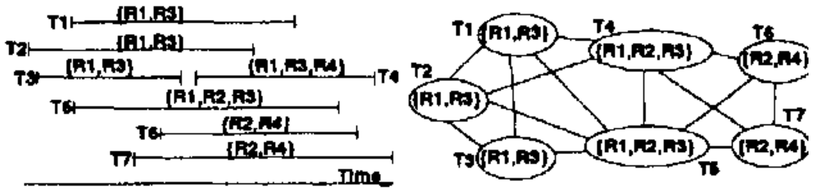

Figure 1: *Left:* Interval orders: a schedule of seven tasks whose start time and duration are fixed. For each task, a set of possible resources is shown. *Right:* the corresponding constraint graph.

we model the resource allocation problem as a discrete Constraint Satisfaction Problem (CSP) [7] in which the constraints among variables are binary and denote *mutual exclusion* with respect to the values, as shown in Fig. 1 right. The nodes of the *constraint graph* represent tasks to be executed, and their labels are sets of resources that can carry out, the tasks. Arcs link nodes that intersect in time and have at least one resource in common. Note that we consider only resources that are reusable and non-sharable, see [6].

Complexity. Arkin and Silverberg [l] showed that the resource allocation problem is NP-complete (i.e., the existence of a polynomial solution method is unlikely). In [3], we showed that an optimization version of the resource allocation problem in which certain tasks (compulsory tasks) must absolutely be allocated a resource and the number of the remaining ones (optional tasks) must be maximized is at least as difficult as MAXSNP-complete, for which the existence of a polynomial-time approximation scheme is unlikely.

Requirement: interactivity vs. automation. Often, constraints cannot be exhaustively enumerated. For example, the constraint that two persons do not work well together is often not formalized. Moreover, constraints may vary over time. For example, a person may be sick and unable to carry out certain tasks. Fully automatic methods are, therefore, often rejected by the ultimate users, who have to revise solutions with the full complexity of the original problem. Thus, it is important to present them with compact representations that simplify the problem.

Abstractions as a new perspective. In this paper, we investigate the use of computers to dynamically build abstractions that structure the problem space and allow the users to solve the problem interactively. In [4], we introduced a heuristic called the Value-Assignment Delay heuristic (VAD) for decomposing a resource allocation problem into sub-problems. The VAD tries to solve sub-problems independently and isolates conflicts between them. The tradeoffs of using this decomposition scheme and a comparison to related work in the scheduling and CSP literature are reported in [3; 4].

By decomposing a problem, the VAD implicitly generates *equivalence classes* of resources and tasks. We argue that these new objects structure the solution space of a resource allocation problem in a compact manner and identify equivalent choices. Thus, they serve as a basis for abstraction. In this paper, we identify these various abstraction classes and characterize them in terms of interchangeability sets [5].

The paper is organized as follows. In Section 2, we recall the main definitions of interchangeability that are of interest to us. In Section 3, we describe and formalize the interchangeable sets discovered by the VAD; then we show how they can be used as abstractions to simplify problem solving. In Section 4, we evaluate the discovered neighborhood interchangeable sets with respect to the ones found by the exact algorithm. Section 5 presents our conclusions.

2 Interchangeability: definitions

In [5], Freuder introduces the concept of interchangeability and defines various kinds of value interchangeability. In this section, we quickly recall the definitions of interest to us. The phrase "(i,j) satisfies *C*" means that values *i* and *j* for two variables linked by a constraint *C* are consistent with respect to this constraint.

Definition 1 *full interchangeability: A* value 6 for a CSP variable *V* is *fully interchangeable* with a value c for *V* if and only if every solution to the CSP that assigns *b* to *V* remains a solution when *c* is substituted for 6 in *V* and vice versa.

This means that values 6 and c can be switched for variable *V* in a given solution without affecting at all the rest of the problem and regardless of the constraints that apply to the variables. Freuder notices that computing fully interchangeable sets may require, in general, computing all solutions, which can be a quite costly operation.

Definition 2 *Neighborhood interchangeability: A* value *b* for a CSP variable *V* is neighborhood interchangeable with a value c for *V* if and only if for every constraint *C* on *V:*

$$\{i \mid (b,i)\ satisfies\ C\} = \{i \mid (c,i)\ satisfies\ C\}$$

Neighborhood interchangeability is a stronger condition than full interchangeability (i.e., not all fully interchangeable sets are neighborhood interchangeable), but is easier to compute. Freuder describes a polynomial-time procedure for computing all neighborhood interchangeable sets[1].

Definition 3 *Substitutability:* Given two values 6 and *c* for a CSP variable V, 6 is substitutable for c if and only if substituting 6 in any solution involving c yields another solution.

Substitutability is 'one-way' full interchangeability.

Definition 4 *Neighborhood substitutability:* For two values 6 and c, for a CSP variable *V, b* is neighborhood substitutable For *c* if and only if for every constraint *C* on *V:*

$$\{i \mid (b,i)\ satisfies\ C\} \supseteq \{i \mid (c,i)\ satisfies\ C\}$$

Neighborhood substitutability is 'one-way' neighborhood interchangeability.

Definition 5 *Partial Interchangeability:* Two values are partially interchangeable with respect to a subset *S* of variables if and only if any solution involving one implies a solution involving the other, with possibly different values for variables in *S.*

Note that full interchangeability is partial interchangeability with *S* — 0. Indeed, Freuder outlines that: "Partial interchangeability captures the idea that values for variables may differ among themselves, but be fully interchangeable with respect to the world".

Using partial interchangeability, we may isolate a sub-problem in which the partial solution can be affected by a change of the value of a variable in the isolated sub-problem, without having to update the rest of the solution. This idea of localizing the effect of a modification is very important in scheduling applications, where one tries to keep the stability of a global solution while adjusting a partial solution locally to accommodate unforeseen events. Thus, partial interchangeability sets can serve as a basis for *reactive* scheduling strategies. In [5], Freuder argues that interchangeable values are redundant and their removal simplifies the problem space.

3 Interchangeability in RA

The algorithm described in [5] is applicable to all types of constraints, however, it only finds neighborhood interchangeabilities (NI)[2]. Below, we introduce a decomposition heuristic, called the VAD heuristic, which is only applicable to constraints of mutual exclusion; but, in addition to the NI sets, it also determines other types of interchangeability for which no other algorithm is known so far.

[1]The worst-case complexity of this procedure is $0(n^2a^2)$ where n is the number of nodes and *a* is the domain size.

[2]In [5], Freuder proposes also a generalized version for finding A:-interchangeability.

3.1 The Value-Assignment Delay heuristic

The idea of the VAD heuristic, as stated in [4], is as follows: "Delay the assignment of the most solicited values in a list coloring problem and try to solve the simplified problem without them. Distribute the delayed values to those variables that really cannot do without them."

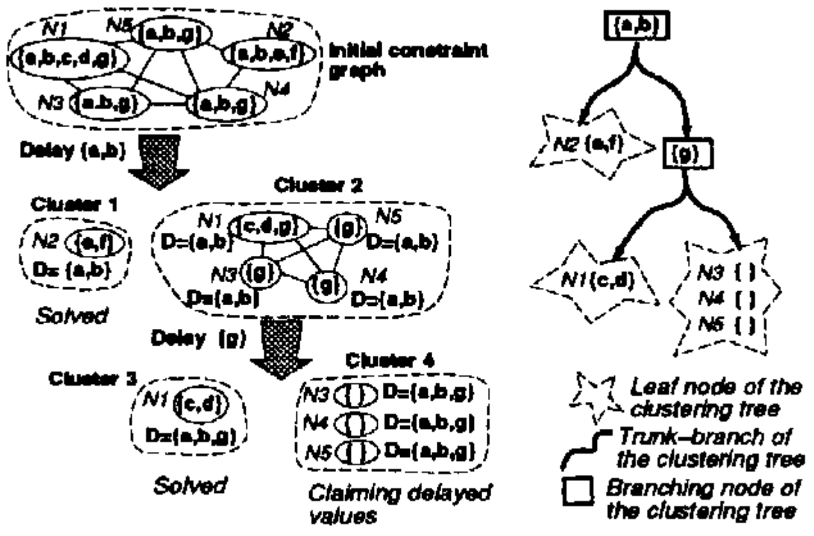

Figure 2: *Left:* Applying the VAD heuristic to a simple resource allocation problem. *Right:* Clustering tree.

The application of this heuristic to a simple example is shown in Fig. 2. In the first step, the assignment of values o and 6 is delayed. The set {a, 6} is called the set of *delayed values.* This has the effect of breaking up the initial graph into two components, because *N2* now has no value in common with, and thus no link to, the rest of the graph. This process is repeated iteratively, and yields a *clustering tree* in which the *leaves* represent simplified sub-problems, the *nodes* are sets of delayed values and the *branches* show how these components relate (Fig. 2 right). Some of the sub-problems can be easily solved in isolation without any effect on the rest of the problem. For example, nodes in *Cluster* and *Cluster₃* can be assigned any element in {e, /} and {c, *d)* respectively. Such commitments are safe and never need to be undone. All the other parts need to *claim* de-

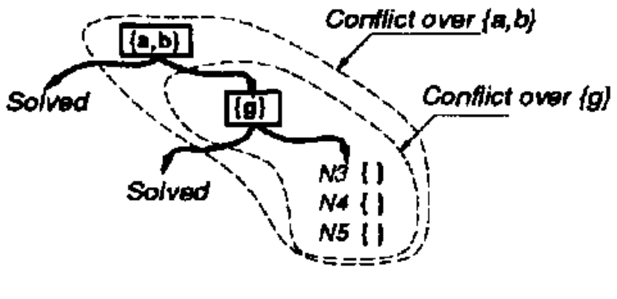

Figure 3: *Isolating conflicts*

layed resources for their solution. A *conflict* is defined around each value set claimed by several nodes in the leaves, see Fig 3. Conflicts may be resolved according to some heuristic and domain dependent preferences or by *interactive* users' intervention. Users may decide to *relax* constraints and "borrow" any of the values that remain unassigned.

By isolating easy sub-problems, identifying conflicts, and localizing interactions among the various components of a resource allocation problem, the heuristic defines a tree structure useful for interactive decision making. The algorithm and termination criteria are described in detail in [4; 3], where we show that the worst-case complexity of the method is $0(n^3 + (n+a)an \log a)$, n being the number of tasks and *a* the number of resources.

3.2 Interchangeability found by the VAD

The VAD heuristic is by 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 similar tasks and resources: it makes explicit the similarities and differences among these entities as far as value contention is concerned. For instance, for the example shown in Fig. 2, the VAD discovered that c and d are equivalent with respect to N_1 and that a and b are equivalent with respect to $N_1, N_2, \dots, N_5$.

Thus, the VAD heuristic implicitly builds pools of resources and aggregates of tasks, which are equivalence classes of objects or objects that are interchangeable with respect to the constraints. These equivalence classes can be considered as problem-specific abstractions useful for solving this particular problem instance. We now show how the components discovered by the VAD identify interchangeable sets.

We use a simple example[3] taken from real-life case study to illustrate the various kinds of interchangeability discovered during decomposition by the VAD heuristic. The problem domain is the allocation of operating rooms and technicians to surgical operations in a hospital.

In Fig. 4 right, we show the tree structure generated by the application of the VAD heuristic and the resources delayed at the branching nodes. For each leaf cluster, we display the tasks (surgical operations) but not the links between the tasks as in a constraint graph. The proposition "$\star$ Op$_x$ missing" indicates that Op$_x$ needs to be allocated a resource of type garçon. The set of resources following the proposition is the set of values that have not been delayed and are still directly available. In this example, the set {Henri, Erika} was delayed and the initial problem broke down into eight leaves.

Fig. 5 shows the interchangeability relations discovered in one of the leaf clusters of Fig. 4. For a variable V_i in a leaf cluster, $\{v_i\}$ denotes the set of values still available for V_i. For instance, for variable Op$_{10}$ in cluster C_2 in Fig. 4, $\{v_i\}$ = {Mobwete, Fernando, Giorgio, Alberto, Rolf, Martine}. We claim that:

Lemma 1 For every variable V_i in a leaf cluster C, all values in the set $\bigcap_{V_j \in C} \{v_j\}$ are partially interchangeable with respect to $S = \{V_j \in C\}$.

[3]This is the example "test 1-a" of Table 1 in Section 4, slightly modified to illustrate all the types of interchangeability addressed in this section.

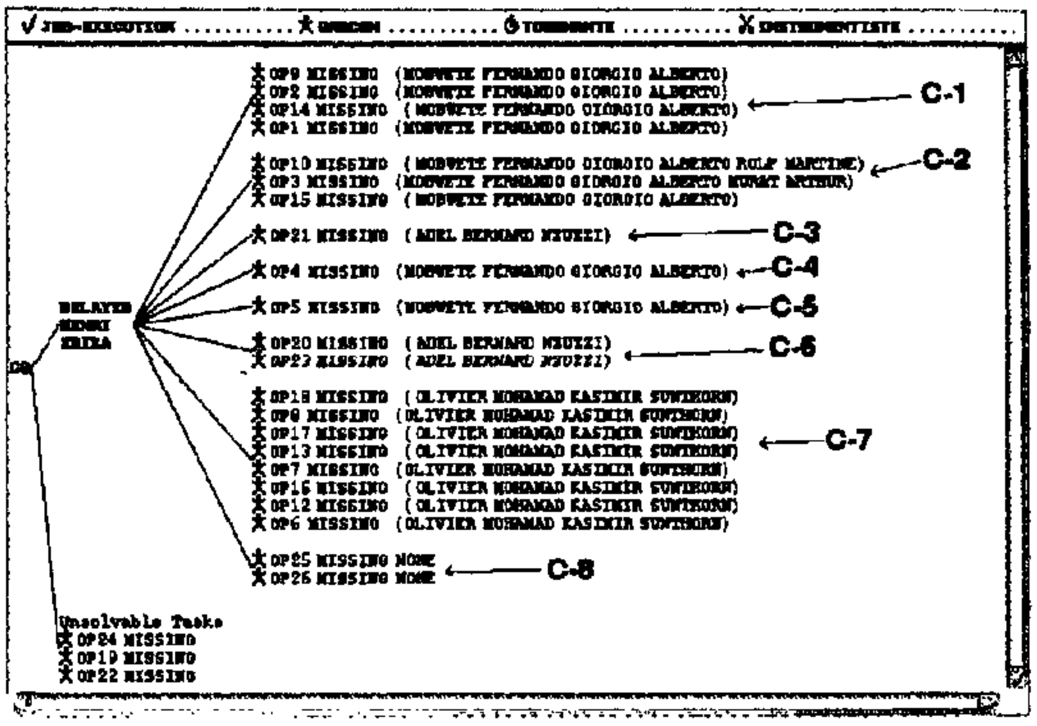
OP9 MISSING (MOBWETE FERNANDO GIORGIO ALBERTO)
OP2 MISSING (MOBWETE FERNANDO GIORGIO ALBERTO)
OP14 MISSING (MOBWETE FERNANDO GIORGIO ALBERTO)
OP1 MISSING (MOBWETE FERNANDO GIORGIO ALBERTO)
C-1
OP10 MISSING (MOBWETE FERNANDO GIORGIO ALBERTO ROLF MARTINE)
OP3 MISSING (MOBWETE FERNANDO GIORGIO ALBERTO MURAT ARTHUR)
OP15 MISSING (MOBWETE FERNANDO GIORGIO ALBERTO)
C-2
OP21 MISSING (ADEL BERNARD NZUZZI)
C-3
OP4 MISSING (MOBWETE FERNANDO GIORGIO ALBERTO)
C-4
OP5 MISSING (MOBWETE FERNANDO GIORGIO ALBERTO)
C-5
OP20 MISSING (ADEL BERNARD NZUZZI)
OP23 MISSING (ADEL BERNARD NZUZZI)
C-6
OP18 MISSING (OLIVIER MOHAMAD KASIMIR SUNTHORN)
OP8 MISSING (OLIVIER MOHAMAD KASIMIR SUNTHORN)
OP17 MISSING (OLIVIER MOHAMAD KASIMIR SUNTHORN)
OP13 MISSING (OLIVIER MOHAMAD KASIMIR SUNTHORN)
OP7 MISSING (OLIVIER MOHAMAD KASIMIR SUNTHORN)
OP16 MISSING (OLIVIER MOHAMAD KASIMIR SUNTHORN)
OP12 MISSING (OLIVIER MOHAMAD KASIMIR SUNTHORN)
OP6 MISSING (OLIVIER MOHAMAD KASIMIR SUNTHORN)
C-7
OP25 MISSING NONE
OP26 MISSING NONE
C-8
DELAYED
HENRI
ERIKA
Unsolvable Tasks
OP24 MISSING
OP19 MISSING
OP22 MISSING

Figure 4: *Decomposition by the VAD heuristic.*

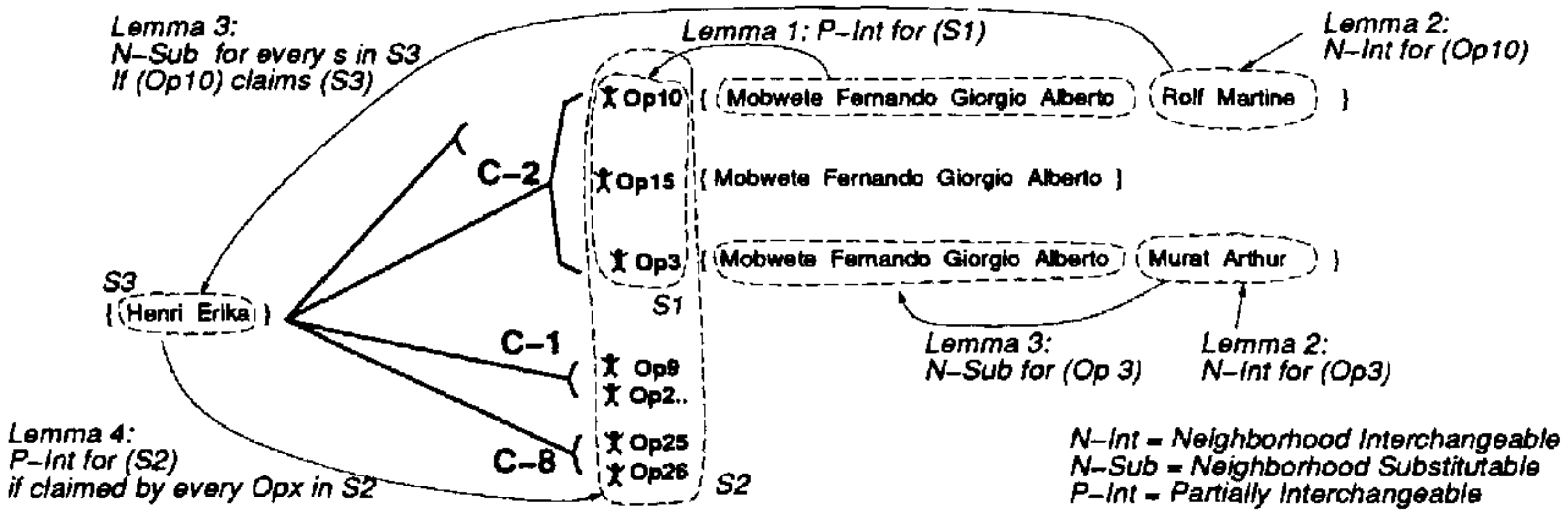
Lemma 3:
N-Sub for every s in S3
if (Op10) claims (S3)
Lemma 1: P-Int for (S1)
Lemma 2:
N-Int for (Op10)
Op10 { Mobwete Fernando Giorgio Alberto Rolf Martine }
Op15 { Mobwete Fernando Giorgio Alberto }
Op3 { Mobwete Fernando Giorgio Alberto Murat Arthur }
C-2
S1
S3
{ Henri Erika }
C-1
Op9
Op2..
Lemma 3:
N-Sub for (Op 3)
Lemma 2:
N-Int for (Op3)
Lemma 4:
P-Int for (S2)
if claimed by every Opx in S2
C-8
Op25
Op26
S2
N-Int = Neighborhood Interchangeable
N-Sub = Neighborhood Substitutable
P-Int = Partially Interchangeable

Figure 5: *Illustration of various types of interchangeabilities discovered by the VAD heuristic.*

Proof: Let $T = \bigcap_{V_j \in C} \{v_j\}$. Since the values in T have not been delayed, they are needed by no other variable in the problem within the time window of the cluster C, except the ones in C. Thus, no conflict with variables outside C is possible and any decision about the assignment of elements of T can only affect variables in C. Since any value in T is acceptable for any variable in the cluster, two values for a variable can be always swapped in any solution by a permutation of the values in T over the variables in C, for instance. Thus, there cannot be solutions involving one value in T without there being solutions involving the other values in T. □

Example from Fig. 5: In C-2, {Mobwete, Fernando, Giorgio, Alberto} are partially interchangeable for {Op$_{10}$, Op$_{15}$, Op$_3$ }.

For lemmas 2 and 3, let U_i denote the set of values uniquely available to variable V_i in cluster C. U_i is given by the formula $U_i = \{v_i\} - \bigcup_{V_j \in C, j \neq i} \{v_j\}$.

Lemma 2 For every variable V_i in a leaf cluster C, all values in the set U_i are neighborhood interchangeable.

Proof: U_i may not create any conflict outside C for the same reason as in the proof of Lemma 1. Within C, all values in U_i are possible only for V_i, by construction. Thus no conflict is possible with any other variable $\{V_j \in C, j \neq i\}$. □

Example from Fig. 5: In C-2, {Rolf, Martine} are neighborhood interchangeable for Op$_{10}$, and {Murat, Arthur} are neighborhood interchangeable for Op$_3$.

Lemma 3 **For every variable V_i in a leaf cluster C, each value in U_i is neighborhood substitutable for every other value in the label of V_i. In particular, each value in U_i is neighborhood substitutable for every value in $\bigcap_{V_j \in C} \{v_j\}$ and for every value in every delayed set that V_i may claim.**

Proof: Since no value in *Ui* was delayed, this means that *Ui* is needed only for *Vi* during *Vi* and assigning any element of *Ui* to Vi does not rule out any choice for the rest of the problem. Therefore, the set of values for nodes adjacent to V*, for the case where Vi is assigned an element of C_i, will be a superset of that for the case where it is assigned any other value in its label. □

Example from Fig. 5: In C-2, each of {Rolf, Martine} are neighborhood substitutable for each of {Henri, Erika, Mobwete, Fernando, Giorgio, Alberto} for Opio-

Lemma 4 All values in a given set of delayed values *D* are partially interchangeable with respect to the set *S* of variables located downstream that claim *D.*

Proof: The set of delayed values is chosen at each step as the set of values simultaneously claimed by a set of variables, see [4]. Thus, if a variable claims any element of the set of delayed values, it also claims all the others, by construction. D

Example from Figures 4 and 5: {Henri, Erika} are partially interchangeable for all the nodes in the leaf clusters $C_1, \ldots, C_s$ since they are admissible for all these nodes (although this information is not explicitly shown).

In [5], Freuder suggests that interchangeability can also be computed dynamically during problem resolution, (i.e., *dynamic interchangeability).* An example of this is shown in Fig. 6. Suppose that a conflict resolution procedure (or the user), decides to assign b to Task K. The delayed value set {a} is now fully interchangeable with {e, f} for variable Task L.

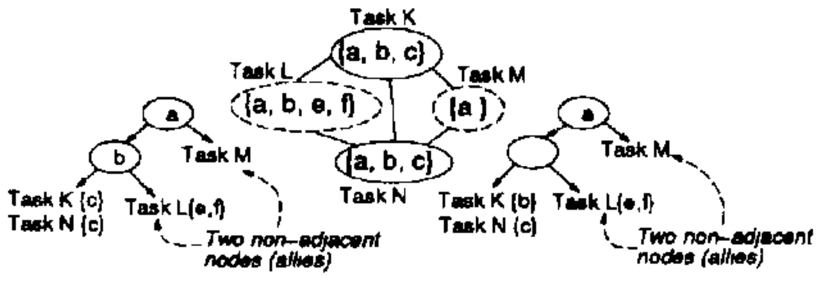

Figure 6: *Full interchangeability discovered dynamically.*

Lemma 5 **Any delayed value claimed only by a set N of non-adjacent nodes[4] is fully interchangeable with all $\{v_i\}$ for each $V_i \in N$.**

Proof: Since the values in the delayed set are claimed only by non-adjacent nodes, whether they are assigned to two or more nodes simultaneously cannot affect the problem in any way. In particular, each of them is neighborhood substitutable for all *{v_i* in each *V_i.* Since all *{v^* can be proven neighborhood substitutable for each of these values using Lemma 3, they are fully interchangeable. D

As these lemmas show, in addition to the NI sets, the VAD also determines other types of interchangeability for which no other algorithm is known so far. In the case of partial interchangeability, the VAD also determines the sets of nodes with respect to which the values are interchangeabl e.

3.3 Interchangeable sets as abstractions

We claim that the VAD decomposition scheme structures the solution space into *compact families* of partial solutions: qualitatively equivalent solutions can be generated by locally modifying the partial solutions in the isolated interchangeable sets. In Fig. 4, it is easy to see that many solutions can be generated simply by switching partially interchangeable values for tasks in the same leaf cluster. This switching operation affects only the nodes in the leaf, while keeping the rest of the 'world' unchanged. Thus, by only viewing the clustering tree generated by the VAD, the user can easily assess the variety of possible solutions. Classical enumerative methods fail to organize the solution space in such a compact manner: they present solutions to the users in a jumble without showing similarities and differences between alternative solutions. In particular, they fail to identify the boundaries within which the effect of a change remains local.

In *interactive problem solving,* interchangeable sets help the human decision maker to view alternative choices in a concise way. Full or neighborhood interchangeable values can be replaced by one 'meta-value'. Partial interchangeability identifies the boundaries in which changes are permitted. When a value *a* is neighborhood substitutable for a value 6 for a given variable the user is guaranteed to be able to replace 6 by a anytime a solution involving 6 for the variable is found, if *b* is not acceptable for some unquantifiable or subjective reason.

In *search,* the main advantages of discovering interchangeability sets are: (1) compacting the solution space representation by grouping *families* of solutions that are equivalent, thus allowing the search process to remain as local as possible, and (2) enhancing the performance of backtracking and consistency checking by removing redundant values.

In *dynamic concept formation,* see [3], interchangeability identifies groups of objects (sets of variables and sets of values) to become the basic components for a generalization process aimed at providing explanation.

4 Evaluation of NI sets

The VAD heuristic is not guaranteed to discover all interchangeable sets. Its performance depends on the struc-

[4]Two non-adjacent variables are called *allies* since, when they claim the same delayed set, they may be assigned values from this set simultaneously, see [4].

ture of the problem at hand- The interchangeable sets approximated by the VAD may differ from the exact ones in two ways: (1) the sets discovered by the VAD may be only subsets of the largest possible ones and (2) some possible sets may be missed.

The conditions under which the VAD discovers all exact sets of interchangeable values are so far unknown. We have evaluated the performance both on data from a case study in a hospital and on a variety of randomly generated resource allocation problems. The characteristics of the hospital case-study in terms of CSP measures are reported in Table 1. The random problems are generated following on the evolutionary model for generating random interval orders by Scheinerman [8] and a resource model where the number and selection of resources follow a uniform distribution.

We report the results only for problem sizes *n* — 20 and *n* — 50, while varying the adjacency probability[5] of the interval graph in {0.1, 0.2, ..., 0.9, 0.99}, and the maximum domain size *a* = {n/10, 2n/10,..., 9n/10,n} For each case, ten random problems were generated and each point in the experiments reported below is obtained by taking the average over the ten cases.

Each time, we have compared the NI sets obtained by the VAD decomposition with those obtained by the exact algorithm by Freuder according to the two criteria introduced above (i.e., maximality and existence).

Let *G* — *(V, E)* be a constraint graph of *V* vertices (or variables) and *E* edges (or constraints), and

- $s_e(v)$ be the set of NI values for variable v found by the exact algorithm by Freuder [5]
- $S_{total} = \{s_e(v); \forall v \in V \text{ and } s_e(v) \neq \emptyset\}$ be the set of all NI sets for the variables
- $s_v(v)$ be the set of NI values for variable v identified by our decomposition method, $s_v(v) \subseteq s_e(v)$
- $V_{missed} = \{v;\ s_v(v) \subset s_e(v)\}$, this is the set of variables with non empty NI sets and for which the NI sets discovered by the VAD are different from the exact ones
- $V_I = \{v \in V;\ s_e(v) \neq \emptyset\}$, this is the set of variables that have a non empty NI set.

We now consider the results of the evaluation regarding three different criteria: occurrence p1 (1), coverage p2 (2), and accuracy p_3 (3)[6] as discussed below.

Occurrence: existence of NI sets. *p* measures the "occurrence" of neighborhood interchangeability in a problem and is computed using the exact algorithm. A small value for pi indicates that few NI sets exist in *G.*

$$p_1 = \frac{|V_I|}{|V|} \tag{1}$$

In the real-world examples reported in Table 1, we notice that the occurrence of NI is rather rare *(p* is small). In fact, the other types of interchangeability characterized in Section 3.2 seem to occur more often (for example, see Fig. 5). The evaluation of the latter has not yet been carried out.

For the randomly generated data, Fig. 7 shows the following behavior: (1) As Benson and Freuder experienced [2j, only problems of small adjacency probability present interchangeability features: the number of NI sets becomes nearly zero for p > 0.4. (2) For problems with a "small" value for p, the number of interchangeable sets increases with the number of resources. (3) The number of interchangeable sets decreases with the size of the problem[7] even when the ratio of the number of values to the number of variables *(a/n)* is held constant. This is foreseeable because the resources are not structured (chaos increases).

Coverage: how many NI sets are approximated. p2 measures the number of sets that are affected by the behavior of the VAD. A small value for p_2 indicates that-few of the NI sets discovered by the VAD are different from the exact ones. A large value for *p2* means that many sets are truncated.

$$p_2 = \frac{|V_{missed}|}{|S_{total}|} \tag{2}$$

For the set of real-world data described in Table 1, *no set of neighborhood interchangeable values was missed* (p2 — 0 in all cases). For the randomly generated problems, the measured values of p_3 are reported in Fig. 8 and Table 2. We can draw the following observations from Fig. 8. For problems with small values for p, very few NI sets are truncated. The VAD nearly finds all exact sets. This is the most important region since it contains most NI sets (p1 is large). When p increases, the number of NI sets that are not complete also increases. However, one should bear in mind that, for these problems, *p* is small, which means that few NI sets exist. This explains why P2 deteriorates significantly even though the proportion of NI sets found is still fairly high.

Accuracy: the extent of NI sets approximation. The parameter p3 measures the accuracy to which the VAD computes the NI sets[8]

$$p_3 = \frac{\sum_{\{v \in V_{missed}\}} \frac{|s_v(v)|}{|s_e(v)|}}{|V_{missed}|}$$

A small value for p3 indicates those sets affected by the approximation are heavily distorted (truncated). A value p3 = 0 means that all NI sets were completely missed by the VAD. Large values for p_3 are desirable since they indicate that the NI sets discovered by the VAD are hardly altered.

For the set of real-world data described in Table 1, all the discovered NI sets were exact (p_3 is undefined in all cases).

[5]The adjacency probability, also called constraint probability, is $p = \frac{e}{e_{max}}$, where e is the number of edges.

[6]Existence of NI sets is measured by p_I and maximality by p_2 and p_3.

7The number of NI was so small (too small for any evaluation) for the cases n — 50 and a < 50 that we decided to generate also, for this problem size, the cases *a* = 60, 70, and 80.

[8]Note that p_3 is defined only when $V_{missed} \neq \emptyset$.

	CSP parameters				Evaluation of NI sets		
test-id	n	a	d	t	p_1	p_2	p_3
test1-a	22	5	0.376	0.094	0.136	0	-
test2	25	6	0.529	0.108	0.04	0	-
test3	25	5	0.417	0.137	0.04	0	-
test4	35	7	0.405	0.093	0	-	-
test5	36	8	0.455	0.087	0	-	-
test1-b	44	24	0.420	0.043	0	-	-
test6-a	47	25	0.570	0.043	0	-	-
test7-a	49	26	0.454	0.041	0	-	-
test1	66	24	0.433	0.037	0.046	0	-
test8	68	28	0.437	0.038	0.015	0	-
test9	72	29	0.444	0.037	0	-	-

Table 1: *Characteristics of the real-world examples.* 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 $e_{max} = \frac{n(n-1)}{2}$.

The numerical results for the random problems are reported in Table 3 and should be understood in light of the previous ones, (i.e., p_1 and p_2): *p_1 percent of the variables have non empty NI sets; p_2 percent of these sets are distorted by the VAD; and the distorted sets are p_3 percent accurate.* From comparing the results, one may conclude that:

- Problems with small adjacency probability (p is small) present many NI sets (p_1 is large). Many of these NI sets are faithfully rendered by the VAD (p_2 is small). Those that are approximated are not seriously altered (p_3 is large). For example, for $p = 0.1$, $n = 20$, $a = 14$, we have $p_1 = 26.0\%$, $p_2 = 7.4\%$, $p_3 = 56.0\%$.
- Conversely, problems with high adjacency probability (p is big) present few NI sets (p_1 is small) many of which are distorted by the VAD (p_2 is large) and those that are approximated are seriously affected (p_3 is small).

5 Conclusion

Because a changing and complex world cannot be completely formalized, resource allocation problems should not be addressed using fully automatic methods. In this paper, we propose the paradigm of dynamic abstractions as a new way of applying computers for such tasks. Here, a computer is used essentially to structure a problem space into relevant abstractions. This allows a user to participate in solving the actual problem while taking into account the unformalized constraints which hold at that moment.

We claim that interchangeability is a useful notion to characterize adequate abstractions, then we present and analyze a heuristic for dynamically generating abstractions for resource allocation. Although one still has to study the theoretical conditions under which the discovered NI sets are maximally defined, the approximation of NI sets by the VAD is acceptable given that:

- NI sets are more frequent in problems with small p and the effect of the "distortion" due to the VAD decreases with p.
- The VAD can be computationally less expensive than the exact algorithm[9].
- The computation of NI sets is not the purpose of the VAD, whose goal is to localize interactions and conflicts among sub-problems; these results are only a fortunate "side effect".

As a general comment, we recall again that neighborhood interchangeability is *not* the only type of interchangeability discovered by the VAD and that there exists, so far, no other algorithm for computing other types of interchangeability except for the obvious process of computing all possible solutions. Note that the possibility of discovering interchangeable sets increases in structured domains with some semantics. Thus, various real-world problems, such as scheduling, configuration, and design, may benefit greatly from finding interchangeability sets. Note also that the three proposed parameters, *p*, p2 and p3, can be exploited to measure quality of interchangeable sets in general.

In future work, we hope to extend our approach in two directions: exploiting interchangeability sets in rescheduling and building a generalized theory of abstraction in CSPs using interchangeability concepts.

Acknowledgments

The authors are indebted to Dean Allemang, Guevara Noubir, Bradley Richards and the anonymous reviewers for their comments.

References

[1] Esther Arkin and Ellen Silverberg. Scheduling Jobs with Fixed Start and End Times. *Discrete Applied Mathematics,* 18:1-8, 1987.

[2] Brent W. Benson and Eugene C. FVeuder. Interchangeability Preprocessing Can Improve Forward Checking Search. In *Proc. of the 10 th ECAI*, pages 28-30, Vienna, Austria, 1992.

[3] Berthe Y. Choueiry. *Abstraction Methods for Resource Allocation.* PhD thesis, Swiss Federal Institute of Technolody in Lausanne (EPFL), Switzerland, October 1994. Thesis No 1292.

[4] 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.

[5] Eugene C. Freuder. Eliminating Interchangeable Values in Constraint Satisfaction Problems. In *Proc. of AAAI-91,* pages 227-233, Anaheim, CA, 1991.

[6] Nancy B. Lehrer, *Knowledge Representation Specification Language (KRSL).* DARPA/Rome Laboratory Planning and Scheduling Initiative, June 1992. DRAFT.

[7] Alan K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence,* 8:99-118, 1977.

[8] Edward R. Scheinerman. An Evolution of Interval Graphs. *Discrete Mathematics,* 82:287-302, 1990.

[9] This holds when the number of resources is comparable to the number of tasks.

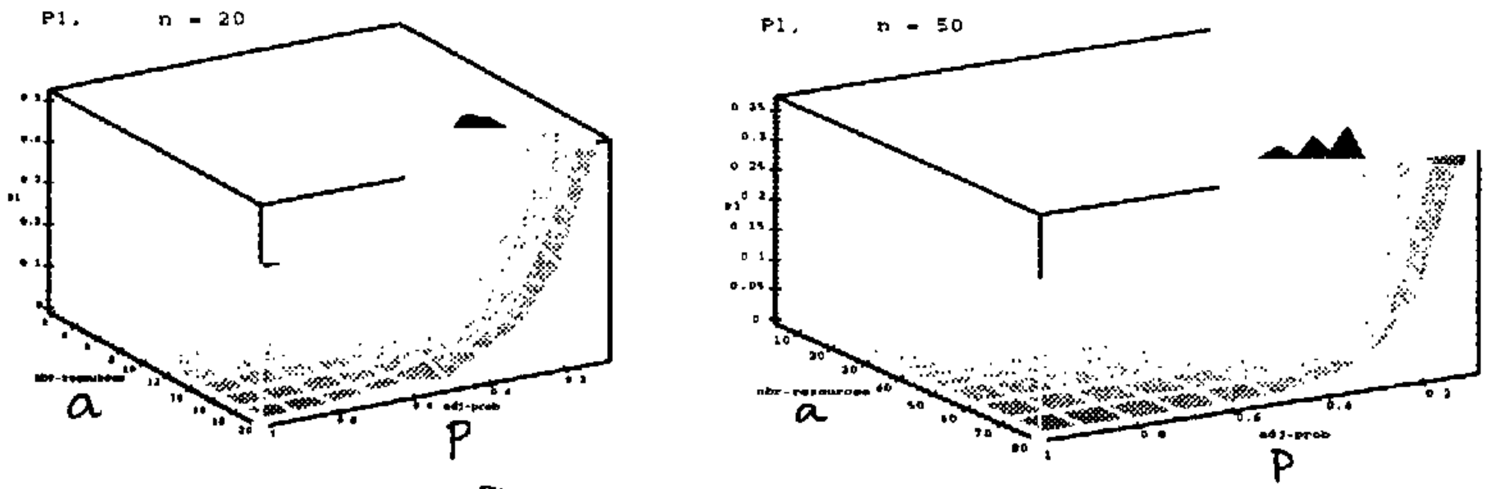

Figure 7: *p_1 for various problem sizes.*

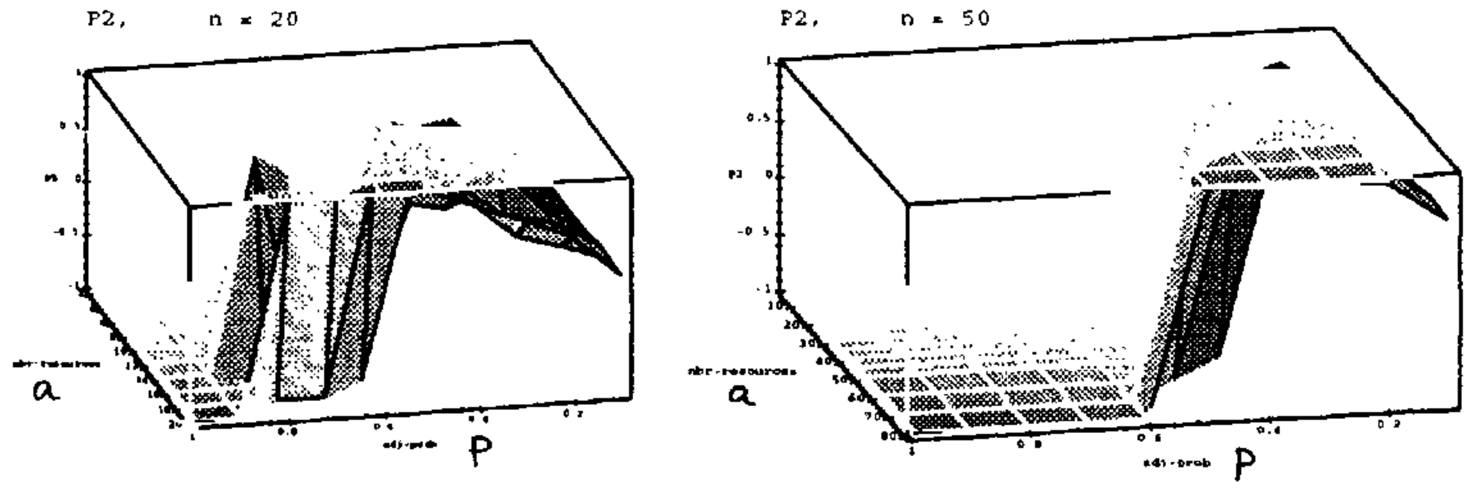

Figure 8: *p_2 for various problem sizes.* Problems that do not exhibit any interchangeability ($p > 0.4$) are neglected, this i indicated by imposing $p_2 = -1$.

n = 20, p2%										
p \ a	2	4	6	8	10	12	14	16	18	20
0.1	0.0	2.2	3.9	5.6	4.4	6.3	7.4	12.7	11.6	10.2
0.2	0.0	18.6	23.8	32.2	18.3	27.1	37.5	35.6	26.2	48.8
0.3	0.0	45.2	66.7	54.8	66.7	42.6	77.1	45.0	38.9	66.7
0.4	-	50.0	0.0	75 0	60.0	87.5	40.0	75.0	75.0	82.1
0.5	-	-	100	-	100	100	50.0	75.0	100	100
0.6	-	0.0	100	50.0	-	0.0	-	-	100	100
0.7	-	-	-	-	-	66.7	-	100	-	100
0.8	-	-	-	-	0.0	-	100	-	-	100

n = 50, p2%													
p \ a	5	10	15	20	25	30	35	40	45	50	60	70	80
0.1	17.5	33.9	49.6	58.5	50.0	58.7	56.6	54.3	67.4	61.7	59.6	64.6	61.5
0.2	50.0	75.0	95.2	83.3	95.8	100	89.6	92.1	96.3	88.0	100	100	95.5
0.3	-	100	80.0	-	100	100	100	100	100	100	100	100	100

Table 2: *Values of p_2 for problems where NI is likely to occur.*

n = 20, p3%										
p \ a	2	4	6	8	10	12	14	16	18	20
0.1	-	0	25.0	44.1	4.8	40.4	56.0	45.6	44.8	17.8
0.2	-	50.0	69.4	51.6	40.0	54.7	41.8	34.5	35.2	45.3
0.3	-	16.7	33.3	30.0	12.5	34.7	25.2	27.9	14.6	28.2
0.4	-	33.3	-	11.1	38.0	28.9	27.8	15.6	38.7	15.2
0.5	-	-	0	-	33.3	39.5	0	19.4	23.8	33.3
0.6	-	-	16.6	40	-	-	-	-	12.5	0
0.7	-	-	-	-	-	16.6	-	50.0	-	29.0
0.8	-	-	-	-	-	-	0.0	-	-	0.0

n = 50, p3%													
p \ a	5	10	15	20	25	30	35	40	45	50	60	70	80
0.1	37.5	49.1	38.7	38.2	34.8	35.5	34.5	37.0	34.3	27.4	24.5	22.9.	33.7
0.2	0.0	20.3	19.2	11.5	14.6	9.4	8.8	16.4	13.2	7.8	8.8	7.2	4.0
0.3	-	2.0	25.7	-	11.1	15.0	16.9	4.1	4.1	11.4	8.3	3.3	2.0
0.4	-	0.0	-	0.0	-	-	25.0	16.6	-	-	8.3	0.0	6.5
0.5	-	-	-	-	-	-	-	-	-	-	-	33.3	0.0

Table 3: *Values of p_3 for problems where NI is likely to occur.*