On the Computation of Local Interchangeability in Discrete Constraint Satisfaction Problems

Berthe Y. Choueiry
Knowledge Systems Laboratory
Stanford University
Stanford, CA, 94305-9020
choueiry@ksl.stanford.edu

Guevara Noubir
Data Communications Group
Centre Suisse d'Electronique et de
Microtechnique (CSEM), Rue Jaquet-Droz 1
CH-2007 Neuchâtel, Switzerland
guevara.noubir@csemne.ch

Abstract

In [4], Freuder defines several types of interchangeability to capture the equivalence among the values of a variable in a discrete constraint satisfaction problem (CSP), and provides a procedure for computing one type of local interchangeability. In this paper, we first extend this procedure for computing a weak form of local interchangeability. Second, we show that the modified procedure can be used to generate a conjunctive decomposition of the CSP by localizing, in the CSP, independent subproblems. Third, for the case of constraints of mutual exclusion, we show that locally interchangeable values can be computed in a straightforward manner, and that the only possible type of local interchangeability is the one that induces locally independent subproblems. Finally, we give hints on how to exploit these results in practice, establish a lattice that relates some types of interchangeability, and identify directions for future research.

1 Introduction

Interchangeability among the values of a variable in a *Constraint Satisfaction Problem* (CSP) captures the idea of 'equivalence' among these values and was first formalized by Freuder [4]. Choueiry and Faltings [2] show that interchangeability sets are abstractions of the CSP with the following advantages: (1) The reduction of the computational complexity of a problem, and the improvement of the performance of the search technique used to solve it. (2) The identification of elementary components for interaction with the users.

This paper studies the computation of local interchangeability, and is organized as follows. In Section 2, we first review the definitions of a CSP and interchangeability; we discuss the advantages drawn from computing interchangeable sets; then we restate the procedure introduced in [4] for computing a strong type of local interchangeability. In the rest of the paper we describe our contributions. In Section 3.1, we extend the above mentioned procedure; we show that this extension enables the computation of a weak form of interchangeability (Section 3.2) as well as the identification of locally independent subproblems (Section 3.3); then we describe how these interchangeable sets are organized in a hierarchy (Section 3.4). Further, we sketch how to use their properties in practice (Section 4), and show that, for the case of constraints of mutual exclusion, local interchangeability can be easily computed, and is equivalent to identifying locally independent subproblems (Section 5). Finally, we introduce a lattice that situates various contributions reported in the literature (Section 6), and draw directions for future research (Section 7).

We intentionally restrict ourselves here to presenting the concepts, hinting on their usefulness, and illustrating them on simple problems. Although we have already identified several properties useful for problem solving, we do not discuss them here for lack of space.

2 Definitions

A CSP is defined by $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$, where $\mathcal{V} = \{V_1, V_2, \ldots, V_n\}$ is the set of variables, $\mathcal{D} = \{D_{V_1}, D_{V_2}, \ldots, D_{V_n}\}$ the set of domains (*i.e.*, sets of values) associated with the variables, and $\mathcal{C}$ is the set of constraints that apply to the variables. A constraint C_{V_i,V_j}, applicable to two variables V_i and V_j, restricts the combination of values that can be assigned simultaneously to V_i and V_j, and thus defines a relation $R_{V_i,V_j} \subseteq D_{V_i} \times D_{V_j}$, which is the set of tuples allowed by C_{V_i,V_j}. When the relation is exactly the Cartesian product of the variable domains (*i.e.*, $R_{V_i,V_j} = D_{V_i} \times D_{V_j}$), the corresponding constraint is said to be *universal*. To solve a CSP is to assign one value to each variable such that all constraints are simultaneously satisfied. A CSP is commonly represented by a *constraint graph* in which the variables are represented by nodes, the domains by node labels, and the constraints by edges that link the relevant nodes. Universal constraints are omitted from the constraint graph. In this document, we restrict our study to discrete binary CSPs: each domain D_{V_i} is a finite set of discrete values, and each constraint applies to two variables. We define the *neighborhood* of a set of variables $\mathcal{S}$, denoted $\texttt{Neigh}(\mathcal{S})$, to be the set of variables

adjacent to $\mathcal{S}$ in the constraint graph.

2.1 Interchangeability

Freuder introduces several types of value interchangeability for a CSP variable. Below, we recall those relevant to our study, while illustrating each of them on the list coloring problem of Fig. 1.

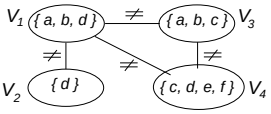

Figure 1: *An example of a list coloring problem.*

Definition 2.1 *Full interchangeability:* A value b for a CSP variable V_i is fully interchangeable (FI) with a value c for V_i if and only if every solution to the CSP that assigns b to V_i remains a solution when c is substituted for b in V_i and vice versa.

Two FI values b and c can be switched for variable V_i in any solution regardless of the constraints that apply to V_i. In Fig. 1, d, e, and f are fully interchangeable for V_4. Indeed, we inevitably have $V_2 = d$, which implies that V_1 cannot be assigned d in any consistent global solution. Consequently, the values d, e, and f can be freely permuted for V_4 in any global solution. No efficient general algorithm for computing FI has to date been reported: in fact, determining FI may require computing all solutions. Neighborhood interchangeability (NI) only considers local interactions, and can thus be efficiently computed:

Definition 2.2 *Neighborhood interchangeability:* A value b for a CSP variable V_i is neighborhood interchangeable (NI) with a value c for V_i if and only if for every constraint C on V_i:

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

In Fig. 1, e and f are NI for V_4. NI and FI are special cases of k-interchangeability, which introduces gradually levels of full interchangeability in all subproblems of size k, moving from NI for $k = 2$ (local), towards FI for $k = n$ (global). k-interchangeability (including FI and NI) is concerned with changing values of one variable, while keeping those of all other variables unchanged. Another type of interchangeability introduced in [4], partial interchangeability, allows a subset of the variables ($\mathcal{A} \subseteq \mathcal{V}$) to be affected when switching the values of V_i, while the rest of the 'world' ($\mathcal{V} - V_i - \mathcal{A}$) remains the same. Informally, partial interchangeability is about extending a boundary (which is the set of variables $\mathcal{S}$ affected by the switching operation) by *weakening* the requirement on what may be affected.

Definition 2.3 *Partial Interchangeability:* Two values are partially interchangeable (PI) with respect to a subset $\mathcal{A}$ of variables if and only if any solution involving one implies a solution involving the other, with possibly different values for variables in $\mathcal{A}$.

In Fig. 1, a and b are PI for V_1 with respect to the set $\mathcal{A} = \{V_3\}$. There is no known efficient algorithm for computing the PI-sets for a CSP variable. In addition to the difficulty of computing PI, one also needs to specify the set $\mathcal{A}$, which is not a straightforward task.

In this paper, we extend the polynomial algorithm for computing NI to efficiently compute a localized version of PI, which we call NPI and define in Section 3.2. Notice that FI corresponds to PI with $\mathcal{A} = \emptyset$; and similarly, NI corresponds to NPI with $\mathcal{A} = \emptyset$. Fig. 2 illustrates the relations between these types of interchangeability (from local to full, and from strong to weak). Informally, a variable V_i affects the problem

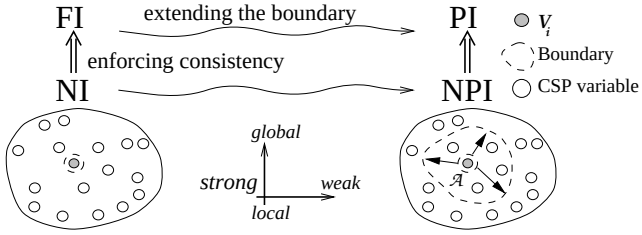

Figure 2: *Interchangeability.* Partial ($\rightsquigarrow$): from strong to weak. Full ($\Rightarrow$): from local to global.

'through' the constraints that link V_i to other variables in the problem, thus through V_i's neighborhood. Two values x and y that are NI for V_i 'affect' $\texttt{Neigh}(\{V_i\})$ in exactly the same fashion. They are bound to carry the same effect on the whole problem, and are inevitably FI for V_i. More formally, Freuder shows that NI is a sufficient but not a necessary condition for FI. (Indeed, in the example of Fig. 1, d and e for V_4 are FI but not NI.) It is easy to show that the same relation holds between NPI and PI.

We introduce the relation $\equiv_{\mathcal{C}}$ that links values x and y, variables V_i and V_j in $\mathcal{S} \subseteq \mathcal{V}$, and the constraints $\mathcal{C}$ of the CSP to indicate that the variable-value pairs (V_i, x) and (V_j, y) are compatible with exactly the same variable-value pairs in $\texttt{Neigh}(\mathcal{S})$.

$$(x, V_i, \mathcal{S}) \equiv_{\mathcal{C}} (y, V_j, \mathcal{S}) \quad (1)$$

If x and y are NI for V_i, we have $(x, V_i, \{V_i\}) \equiv_{\mathcal{C}} (y, V_i, \{V_i\})$; if they are PI, we have $(x, V_i, \{V_i\} \cup \mathcal{A}) \equiv_{\mathcal{C}} (y, V_i, \{V_i\} \cup \mathcal{A})$. This relation is symmetric and transitive, because it is in essence a relation of equivalence.

2.2 Advantages of interchangeable values

We identify three main ways to use interchangeable values in practice:

1. Strongly interchangeable values can be replaced by one 'meta-value'.
2. An asymmetric type of interchangeability, called substitutability (see Definition 5.4), can be used to accommodate unquantifiable constraints or subjective preferences.
3. Partial interchangeability localizes the effect of modifications to some variables and identifies *compact*

families of partial solutions: qualitatively equivalent solutions can be generated by modifying the values of the indicated variables only.

Classical enumerative methods fail to organize the solution space in such a compact manner: they present solutions 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 practice, these characteristics can be used as follows, beyond and including search:

In interactive problem solving. Interchangeable sets can be used to help the human decision-maker view alternative choices in a concise way [2]. More specifically, meta-values can be used to avoid displaying too much information to the user; substitutability allows the compliance to users' preferences; and finally, partial interchangeabilities delimit the extent of certain modifications, so that the users can modify a solution locally to cope with change in a dynamic environment.

In search. Interchangeability sets can be used (1) to monitor the search process to remain as local as possible, by compacting the solution space representation and by grouping solution families, and (2) to enhance the performance of both backtracking and consistency checking by removing redundant values, as shown in [1; 7].

In explanation. Interchangeability identifies groups of objects (sets of variables and sets of values) to become the basic components for a concept generation process aimed at providing explanation. In real-world applications, it is reasonable to suspect that the set of objects discovered to be interchangeable share common characteristics [4]. In [2], a concept generation procedure uses background knowledge, in the form of concept hierarchies, to generate dynamically concise descriptions of the interchangeable sets.

2.3 The discrimination tree

Below, we recall the procedure, introduced in [4], for computing the NI-sets for a variable by building its discrimination tree (DT). The complexity of this procedure is $O(nd^2)$, where n is the size of $\mathcal{V}$, and d the size of the largest domain. It is important, in this procedure, that variables and values be ordered in a canonical way.

Algorithm 1 DT for V_i (D_{V_i}, $\texttt{Neigh}(\{V_i\})$)

Create the root of the discrimination tree
Repeat for each value $v \in D_{V_i}$:
  Repeat for each variable $V_j \in \texttt{Neigh}(\{V_i\})$:
    Repeat for each $w \in D_{V_j}$ consistent with v for V_i:
      Move to if present, construct and move to if not,
      a child node in the tree corresponding to '$V_j = w$'.
  Add '$V_i, \{v\}$' to annotation of the node (or root),
  Go back to the root of the discrimination tree.

The collection of the annotations in the discrimination tree of a variable V_i yields the following set:

$$\texttt{DT}(V_i) = \{d_{1i}, d_{2i}, \ldots, d_{ki}\} \quad (2)$$

where $1 \leq k \leq |D_{V_i}|$ is the number of annotations in the tree, and $d_{1i}, d_{2i}, \ldots, d_{ki}$ determine a partition of D_{V_i}. The NI-sets of V_i are expressed as follows:

$$\texttt{NI}(V_i) = \{d_{ki} \in \texttt{DT}(V_i) \text{ such that } |d_{ki}| > 1\} \quad (3)$$

Although this was not explicitly stated in [4] or in other papers on this topic, there may be in general any number of NI-sets per variable. In Fig. 3, we show the graph and constraints of a simple CSP. The annotations in the discrimination tree for V_1 are: $\texttt{DT}(V_1) = \{\{1,2\},\{3,4\}\}$. In this case, $\texttt{NI}(V_1) = \texttt{DT}(V_1)$. As for V_2 and V_3, we have $\texttt{DT}(V_2) = \{\{3\},\{6,7\}\}$, $\texttt{NI}(V_2) = \{\{6,7\}\}$, $\texttt{DT}(V_3) = \{\{3\},\{4\},\{9\}\}$ and $\texttt{NI}(V_3) = \emptyset$.

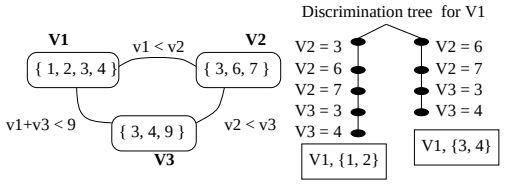

Figure 3: *Left:* CSP. *Right:* DT for V_1.

3 Weakening local interchangeability

First, we extend the DT associated with a variable to be a joint discrimination tree (JDT) associated with a set of variables. We identify the interchangeability sets determined by the JDT, and discuss a special case in which the discovered sets induce locally independent subproblems. Then we show that the annotations of the JDT determine partitions of the variable domains that are organized in a hierarchy.

3.1 Joint discrimination tree (JDT)

We extend Algorithm 1, and apply it to a set $\mathcal{S}$ of variables in order to identify how these variables, when considered together and regardless of the constraints that apply among them, interact through their neighborhood with the rest of the problem. The generalization to a set of variables is straightforward and is obtained by replacing the second argument of Algorithm 1, $\texttt{Neigh}(\{V_i\})$, by $\texttt{Neigh}(\mathcal{S})$ and repeating this algorithm for each of the variables in $\mathcal{S}$ while using the same tree structure. Given a set $\mathcal{S}$ of size s, the time complexity of the algorithm is $O(s(n-s)d^2)$ and the space complexity for storing the tree is $O((n-s)d)$, where n is the size of $\mathcal{V}$, and d the size of the largest domain. The annotations of the *joint discrimination tree* (JDT) of a set $\mathcal{S} = \{V_1, V_2, \ldots, V_k\} \subseteq \mathcal{V}$ yield[1]:

$$\begin{aligned}\texttt{JDT}(\mathcal{S}) = \{\ & \{(V_1, d_{11}), (V_2, d_{12}), \ldots, (V_k, d_{1k})\}, \\ & \{(V_1, d_{21}), (V_2, d_{22}), \ldots, (V_k, d_{2k})\}, \ldots, \\ & \{(V_1, d_{m1}), (V_2, d_{m2}), \ldots, (V_k, d_{mk})\}\ \}\end{aligned} \quad (4)$$

It is important to highlight the following:

[1] When $\mathcal{S} = \{V_i\}$, a comparison of Expressions (2) and (4) yields: $\texttt{JDT}(\{V_i\}) = \{\{(V_i, d_{ki})\} \,|\, d_{ki} \in \texttt{DT}(V_i)\}$.

- In order to comply with Expression (4), the annotations in this tree that do not contain a variable-domain pair for every variable in $\mathcal{S}$ are completed with all pairs of a missing variable and an empty domain for this variable.
- In Expression (4), m denotes the number of annotations in the tree ($1 \leq m \leq \sum_{i=1}^{i=k} |D_{V_i}|$).
- $\forall V_j \in \mathcal{S}$, the sets d_{ij} for $1 \leq i \leq m$ determine a partition of D_{V_j}.
- Since all variable-value pairs in any annotation in the JDT for $\mathcal{S}$ are compatible with exactly the same value-variables pairs in $\texttt{Neigh}(\mathcal{S})$, we have $\forall V_k \in \mathcal{S}$, $\forall 1 \leq i \leq m$, and $\forall x, y \in d_{ik}$:

$$(x, V_k, \mathcal{S}) \equiv_c (y, V_k, \mathcal{S}) \tag{5}$$

and, using the same notation, $\forall V_k, V_h \in \mathcal{S}$, $\forall x, y$:

$$\exists 1 \leq i \leq m, x \in d_{ik}, y \in d_{ih} \iff (x, V_k, \mathcal{S}) \equiv_c (y, V_h, \mathcal{S}) \tag{6}$$

Fig. 4 shows an example of a CSP along with the JDT of $\mathcal{S} = \{V_1, V_2\}$. We have:

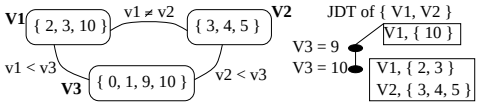

Figure 4: *Left:* CSP. *Right:* JDT for $\mathcal{S} = \{V_1, V_2\}$.

$$\texttt{JDT}(\{V_1, V_2\}) = \{\{(V_1, 10), (V_2, \emptyset)\}, \{(V_1, \{2, 3\}), (V_2, \{3, 4, 5\})\}\} \tag{7}$$

Fig. 5 shows, to the left, the JDT for $\mathcal{S} = \{V_1, V_3\}$ of the CSP of Fig. 1, and, to the right, the JDT for $\mathcal{S} = \{V_2, V_3\}$ of the CSP of Fig. 3. We have:

$$\texttt{JDT}(\{V_1, V_3\}) = \{\{(V_1, \{a, b\}), (V_3, \{a, b\})\}, \{(V_1, \{d\}), (V_3, \emptyset)\}, \{(V_1, \emptyset), (V_3, \{c\})\}\} \tag{8}$$

$$\texttt{JDT}(\{V_2, V_3\}) = \{\{(V_2, \{6, 7\}), (V_3, \{3, 4\})\}, \{(V_2, \{3\}), (V_3, \emptyset)\}, \{(V_2, \emptyset), (V_3, \{9\})\}\} \tag{9}$$

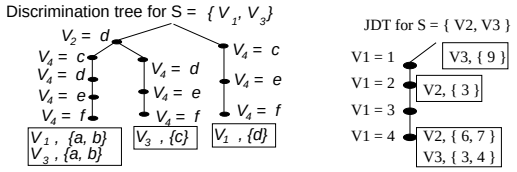

Figure 5: *Left:* JDT for $\mathcal{S} = \{V_1, V_3\}$ of CSP of Fig. 1. *Right:* JDT for $\mathcal{S} = \{V_2, V_3\}$ of CSP of Fig. 3.

3.2 Neighborhood partial interchang. (NPI)

For any element of the set $\texttt{JDT}(\mathcal{S})$ in Expression (4), such as $\{(V_1, d_{m1}), (V_2, d_{m2}), \ldots, (V_k, d_{mk})\}$, for any $V_i \in \{V_1, V_2, \ldots, V_k\}$, if d_{mi} for V_i is the empty set, there seems to be no obvious way profitably to use this element of the $\texttt{JDT}(\mathcal{S})$ in practice. The same is true when all d_{mi} are singletons (*i.e.*, $\forall 1 \leq i \leq k, |d_{mi}| = 1$).

In all other cases, any $x, y \in d_{mi}$ are partially interchangeable (PI) for V_i by construction of the JDT and, according to Definition 2.3, $\mathcal{A} = \mathcal{S} - \{V_i\}$ in this case. Since these d_{mi} sets are computed locally, *i.e.* considering only $\texttt{Neigh}(\mathcal{S})$, they are only subsets of the 'complete' PI-sets, and the elements of each d_{mi} are said to be *neighborhood partially interchangeable* (NPI) for V_i. Thus the definition[2]:

$$\texttt{NPI}(\mathcal{S}) = \{\{(V_1, d_{m1}), (V_2, d_{m2}), \ldots, (V_k, d_{mk})\} \in \texttt{JDT}(\mathcal{S}) \text{ such that } (\forall\, 1 \leq i \leq k, d_{mi} \neq \emptyset) \wedge (\exists\, 1 \leq i \leq k, |d_{mi}| > 1)\ \} \tag{10}$$

The extension of Algorithm 1 to a set of variables yields powerful results: For a given $\mathcal{S}$, it efficiently and simultaneously determines the NPI-sets for all the variables in $\mathcal{S}$. Obviously, if one is interested in the NPI-sets of a subset of $\mathcal{S}$, this procedure needs to iterate only over the variables of the subset.

Using Expression (8), we have, for the example of Fig. 4, $\texttt{NPI}(\{V_1, V_2\}) = \{\{(V_1, \{2, 3\}), (V_2, \{3, 4, 5\})\}\}$. For the following two expressions, we have respectively: $\texttt{NPI}(\{V_1, V_3\}) = \{\{(V_1, \{a, b\}), (V_3, \{a, b\})\}\}$ and $\texttt{NPI}(\{V_2, V_3\}) = \{\{(V_2, \{6, 7\}), (V_3, \{3, 4\})\}\}$.

3.3 Subproblem identification

Sometimes a discrimination tree exhibits a branch worth singling out from the other branches in the tree. In Fig. 6, we show in a constraint graph of a CSP, the set $\mathcal{S} = \{V_1, V_2, \ldots, V_k\}$, and $\texttt{Neigh}(\mathcal{S}) = \{V_{n1}, V_{n2}, \ldots, V_{nl}\}$; we also sketch the JDT for $\mathcal{S}$. Consider the path in this figure whose leaf node is

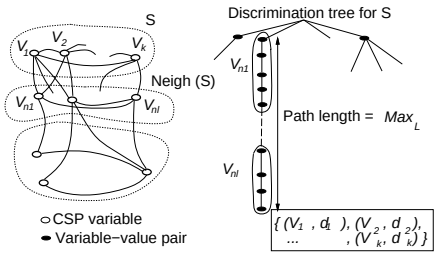

Figure 6: *Left:* Constraint graph. *Right:* Discrimination tree with a path of maximal length.

annotated with $\{(V_1, d_1), (V_2, d_2), \ldots, (V_k, d_k)\}$. When the length of this path (from the root to the leaf) is equal to the sum of the sizes of all the variable domains in $\texttt{Neigh}(\mathcal{S})$ (*i.e.*, $\sum_{i=n1}^{i=nl} |D_{V_i}|$, denoted Max_L), this indicates that $\forall V_i \in \mathcal{S}$, any value in d_i is consistent with *all* the values of the variables in $\texttt{Neigh}(\mathcal{S})$. As long as we deliberately assign to V_i values exclusively chosen from d_i, no consistency checks with the variables in $\texttt{Neigh}(\mathcal{S})$ need be carried out. This appears as if all the constraints between $\mathcal{S}$ and $\texttt{Neigh}(\mathcal{S})$ have been replaced by universal constraints. We can thus generate a disjunctive decomposition[3] of the CSP into

[2] When $\mathcal{S} = \{V_i\}$, a comparison of Expressions (2) and (10) yields: $\texttt{NPI}(\{V_i\}) = \{\{(V_i, d_{ki})\} \mid d_{ki} \in \texttt{NI}(V_i)\}$.

[3] Note that, according to the terminology introduced in [5], this decomposition is consistent, simplifying, complete, and non-redundant.

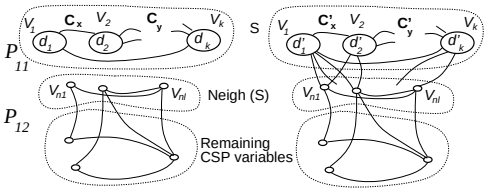

Figure 7: *Left:* $\mathcal{P}_1$. *Right:* $\mathcal{P}_2$.

the two CSPs $\mathcal{P}_1$ and $\mathcal{P}_2$ shown in Fig. 7. $\forall V_i \in \mathcal{S}$, D_{V_i} is replaced by d_i in $\mathcal{P}_1$, and by $d_i' = D_{V_i} - d_i$ in $\mathcal{P}_2$. Naturally, the constraints among the variables in $\mathcal{S}$, and also between $\mathcal{S}$ and $\texttt{Neigh}(\mathcal{S})$, need to be updated accordingly, which is a trivial task. Further, we can generate a conjunctive decomposition of $\mathcal{P}_1$ into two subproblems: $\mathcal{P}_{11}$ containing the variables in $\mathcal{S}$ (and their new domains and updated constraints) and $\mathcal{P}_{12}$ containing the variables in $(\mathcal{V} - \mathcal{S})$. The advantage of this conjunctive decomposition is obvious. A solution to $\mathcal{P}_1$ can be obtained by simply 'concatenating' any solution to $\mathcal{P}_{11}$ and any solution to $\mathcal{P}_{12}$, both of which are smaller than $\mathcal{P}_1$, and the complexity of solving $\mathcal{P}_1$ is reduced to that of solving the bigger subproblem. Hence, we considerably reduce the complexity of solving $\mathcal{P}_1$, and as a result $\mathcal{P}$. Because we use local analysis (*i.e.*, the JDT for $\mathcal{S}$) to determine that $\mathcal{P}_{11}$ is an independent subproblem of $\mathcal{P}_1$, we choose to call $\mathcal{P}_{11}$ a *neighborhood independent subproblem*(NIS). Moreover, at most one path of the JDT for $\mathcal{S}$ can be of maximal length.

Theorem 3.1 *The locally independent subproblem for a given set $\mathcal{S}$ is unique.*

Proof: Any path in the tree that is of maximal length must contain nodes for all variable-value pairs in the neighborhood of $\mathcal{S}$. If two such paths exist, they necessarily consist of exactly the same nodes, which is impossible according to the construction rule of Algorithm 1. □

The set $\texttt{NIS}(\mathcal{S})$ is defined to be the element of the set $\texttt{JDT}(\mathcal{S})$ that annotates a path in the tree of length Max_L. In the example of Fig. 4, $\texttt{NIS}(\{V_1, V_2\}) = \emptyset$. In the example of Fig. 3, $\texttt{NIS}(\{V_2, V_3\}) = \{(V_2, \{6, 7\}), (V_3, \{3, 4\})\}$, and $\texttt{NIS}(\{V_1, V_3\}) = \{(V_1, \{1, 2\}), (V_3, \{9\})\}$. Note how the JDT reveals nontrivial independent subproblems that would, otherwise, have remained unnoticed.

3.4 Hierarchical structure of domain partitions

As stated in Section 3.1 and Expression (4), $\forall V_k \in \mathcal{S}$, the sets d_{ik} for $1 \leq i \leq m$ determine a partition of D_{V_k}. If we were to extend the set $\mathcal{S}$ for which the discrimination tree is built to a set $\mathcal{S}'$ ($\mathcal{S} \subset \mathcal{S}'$), the sets d_{ik} for variable V_k can only increase in size. In fact, for $\mathcal{S} \subset \mathcal{S}' \subset \mathcal{V}$ and for $V_k \in \mathcal{S}$, any element of the partition of D_{V_k} determined by $\texttt{JDT}(\mathcal{S}')$ is either an element or a union of elements of the partition of D_{V_k} determined by $\texttt{JDT}(\mathcal{S})$. The sign $\oplus$ below denotes exclusive OR (*i.e.*, XOR). More formally:

Property 3.2 *$\forall V_k \in \mathcal{S}$ we have:*

$$\exists s_1, s_2 \in \texttt{JDT}(\mathcal{S}) \mid (V_k, d_{ik}) \in s_1; (V_k, d_{jk}) \in s_2 \iff \begin{cases} (\exists s_1', s_2' \in \texttt{JDT}(\mathcal{S}') \mid (V_k, d_{ik}) \in s_1', (V_k, d_{jk}) \in s_2') \\ \oplus \\ (\exists s' \in \texttt{JDT}(\mathcal{S}') \mid (V_k, d_{lk} \supseteq d_{ik} \cup d_{jk}) \in s') \end{cases}$$

Indeed, the set $\texttt{Neigh}(\mathcal{S}')$ contains no more nodes from the set $\texttt{Neigh}(\{V_k\})$ than does the set $\texttt{Neigh}(\mathcal{S})$, plus some nodes that are not connected to V_k. This means that $\texttt{JDT}(\mathcal{S}')$ cannot discriminate more among the values in D_{V_k} than does $\texttt{JDT}(\mathcal{S})$. This fact yields a hierarchical representation of the partitions of D_{V_k} of increasingly coarser granularity as $\texttt{JDT}(\mathcal{S}')$ is gradually expanded to encompass more variables including V_k. One such hierarchy is illustrated in Fig. 8.

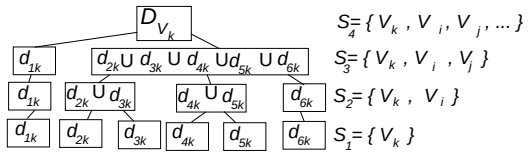

Figure 8: *A hierarchy of partitions of the domain of variable V_k, $D_{V_k} = d_{1k} \cup d_{2k} \cup \ldots \cup d_{6k}$.*

Moreover, when $\mathcal{S}$ is enlarged to $\mathcal{S}'$, the respective domain partitions of two or more variables are either maintained or reduced by union. Informally, this appears as if the lines in Expression (4) were either 'conserved' or 'unified,' as far as the 'old' variables are concerned. Consider two variables $V_k, V_h \in \mathcal{S}$. Consider two sets of the partition determined by $\texttt{JDT}(\mathcal{S})$ on D_{V_k}, respectively on D_{V_h}. Let these sets be d_{ik} and d_{jk}, respectively d_{ih} and d_{jh}, as in:

$$\begin{aligned} \texttt{JDT}(\mathcal{S}) = \{\ldots, \quad & \{\ldots, (V_k, d_{ik}), (V_h, d_{ih}), \ldots\}, \quad (11) \\ \ldots, \quad & \{\ldots, (V_k, d_{jk}), (V_h, d_{jh}), \ldots\}, \quad \ldots\} \end{aligned}$$

Property 3.3 *$\forall V_k, V_h \in \mathcal{S}$ we have:*

$$\exists s_1, s_2 \in \texttt{JDT}(\mathcal{S}) \mid (V_k, d_{ik}), (V_h, d_{ih}) \in s_1; (V_k, d_{jk}), (V_h, d_{jh}) \in s_2 \iff \begin{cases} (\exists s_1', s_2' \in \texttt{JDT}(\mathcal{S}') \mid (V_k, d_{ik}), (V_h, d_{ih}) \in s_1'; (V_k, d_{jk}), (V_h, d_{jh}) \in s_2') \\ \oplus \\ (\exists s' \in \texttt{JDT}(\mathcal{S}') \mid (V_k, d_{lk} \supseteq d_{ik} \cup d_{jk}), (V_k, d_{lh} \supseteq d_{ih} \cup d_{jh}) \in s') \end{cases}$$

This property can be extended to any number of variables in a straightforward manner.

4 Use of JDT-sets in practical applications

The practical benefit drawn from building the joint discrimination tree of a set of variables is two-fold: identification of independent subproblems and of partial interchangeability sets.

1. The advantage of isolating subproblems is obviously a reduction of the overall computational complexity of the problem, when these subproblems can be successfully solved.

   In case the independent subproblem is found to be insoluble, the combination of variables and variable domains can be remembered as a *compact no-good* set. The identified unsolvable subproblem can serve as a component in the 'factor out failure' decomposition scheme proposed in [6] or to improve the performance of backtracking during traditional search.

2. The benefit of computing partially interchangeable sets of values for a set of variables $\mathcal{S} \subset \mathcal{V}$ becomes apparent when one tries to explore alternative solutions to the CSP that may require updating the values of the variables in $\mathcal{S}$ while keeping the values of the variables outside $\mathcal{S}$ unchanged. The idea of localizing the effect of a modification is very important in several practical applications such as scheduling and resource allocation, where one tries to keep the stability of a global solution while locally adjusting a partial solution to accommodate unforeseen events. Thus, partial interchangeability sets can serve as a basis for *reactive* strategies, such as rescheduling.

Starting from the interchangeability sets of a variable V_i with $\mathcal{S} = \{V_i\}$, one can compute the JDT for an increasingly larger 'environment' that encompasses V_i. Alternatively, starting from a large environment, one can gradually refine it by restricting the computations to a subset of $\mathcal{S}$. We are investigating strategies for guiding both of these processes. We have already studied some properties of these sets to be exploited while building such strategies. These results are not reported here for lack of space. Below, we give hints on how such strategies may proceed:

Enlarging $\mathcal{S}$. We should be careful to enlarge $\mathcal{S}$ to include variables that share a common neighborhood this would allow us to increase the size of the NPI-set and to identify independent subproblems.

Refining $\mathcal{S}$. Properties 3.2 and 3.3 can be used as the basis of consistency checking techniques more efficient than the traditional ones. Such strategies would operate at coarse levels of detail by using the output of a JDT computed over a relatively large set $\mathcal{S}$ to remove 'coarse' inconsistent combinations of values between the variables in $\mathcal{S}$ and the rest of the problem, then gradually refining $\mathcal{S}$ to check and filter finer combinations of values among variables.

5 Constraints of mutual exclusion

In general, as in the example of Fig. 3, it is necessary for determining local interchangeability to compute a discrimination tree. In this section, we study the constraints of mutual exclusion, as in coloring problems, which are widely used in practical applications (*e.g.*, resource allocation [2]). These constraints are naturally highly disjunctive, and propagation algorithms perform poorly in this case unless values are assigned to variables (*i.e.*, search is carried out). Indeed, when the sizes of variable domains are strictly greater than 1, arc-consistency fails to rule out any value. In this section, we show that, when the constraints that link $\mathcal{S}$ to $\texttt{Neigh}(\mathcal{S})$ are constraints of mutual exclusion[4], local interchangeabilities can be easily determined and do not require the computation of a discrimination tree. Let $D_{\texttt{Neigh}} = \bigcup_{V_j \in \texttt{Neigh}(\mathcal{S})} D_{V_j}$.

Theorem 5.1 *When $\mathcal{S}$ and $\texttt{Neigh}(\mathcal{S})$ are linked by constraints of mutual exclusion, $\forall V_i \in \mathcal{S}$, $\forall x, y \in D_{V_i}$, and $x \neq y$, we have:*

$$(x, V_i, \mathcal{S}) \equiv_C (y, V_i, \mathcal{S}) \Longleftrightarrow x, y \notin D_{\texttt{Neigh}} \qquad (12)$$

Proof: Suppose that $y \in D_{\texttt{Neigh}}$, then $\exists V_j \in \texttt{Neigh}(\mathcal{S})$ such that $y \in D_{V_j}$. Since $x \neq y$, and C_{V_i,V_j} is a constraint of mutual exclusion, then $V_i = x$ is compatible with $V_j = y$. Since x, y are NI for V_i, this implies that $V_i = y$ is compatible with $V_j = y$, which is impossible because C_{V_i,V_j} is a constraint of mutual exclusion. Thus, $y \notin D_{\texttt{Neigh}}$. Similarly, $x \notin D_{\texttt{Neigh}}$.

Further, $\forall x, y \in D_{V_i}, x, y \notin D_{\texttt{Neigh}}$ means that both x and y for V_i are compatible with all the elements in $D_{\texttt{Neigh}}$; thus they are NI for V_i. □

The results listed below can be easily proven using this theorem; the proofs are omitted because of the space limit. The following theorem indicates how $\texttt{NIS}(\mathcal{S})$ is obtained without a discrimination tree.

Theorem 5.2 *When $\mathcal{S}$ and $\texttt{Neigh}(\mathcal{S})$ are linked by constraints of mutual exclusion, we have:*

$$\texttt{NIS}(\mathcal{S}) = \{(V_i, d_i), \forall V_i \in \mathcal{S}, d_i = D_{V_i} - D_{\texttt{Neigh}}\} \qquad (13)$$

For the example of Fig. 1, $\texttt{NIS}(\{V_4\}) = \{(V_4, \{e, f\})\}$ and $\texttt{NIS}(\{V_1, V_3\}) = \{(V_1, \{a, b\}), (V_3, \{a, b\})\}$. The following theorem states that the only kind of local interchangeability that can show up in this situation is the one that induces a locally independent subproblem.

Theorem 5.3 *When $\mathcal{S}$ and $\texttt{Neigh}(\mathcal{S})$ are linked by constraints of mutual exclusion, we have:*

$$\texttt{NPI}(\mathcal{S}) = \begin{cases} \{\texttt{NIS}(\mathcal{S})\} & \text{if } \exists\ (V_i, d_i) \in \texttt{NIS}(\mathcal{S}) \text{ such} \\ & \text{that } |d_i| > 1 \\ \emptyset & \text{otherwise} \end{cases} \qquad (14)$$

Freuder also introduced *neighborhood substitutability* (NSUB), which is 'one-way' neighborhood interchangeability[5] [4]:

Definition 5.4 *Neighborhood substitutability:* For two values b and c, for a CSP variable V, b is neighborhood substitutable (NSUB) for c if and only if for every constraint C on V:

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

[4]Note that the constraints in $\mathcal{C}$ need not all be constraints of mutual exclusion.

[5]Naturally, the concept local substitutability can be extended to apply to a set of variables, as we did for the concept of neighborhood interchangeability.

The following theorem states that, in the case of constraints of mutual exclusion, NSUB values necessarily induce a locally independent subproblem:

Theorem 5.5 *When V_i and* $\texttt{Neigh}(\{V_i\})$ *are linked by constraints of mutual exclusion, $\forall x, y \in D_{V_i}$, we have:*

$$x \text{ is NSUB for } y \iff x \in d_i \mid \texttt{NIS}(\{V_i\}) = \{(V_i, d_i)\} \quad (15)$$

6 Related work

In spite of its importance for practical applications and the challenge its poses for the development of theoretically new search mechanisms, interchangeability has received relatively little attention in contrast, for instance, to backtracking and consistency filtering. Below we review some contributions on this topic, draw the relations among them, and summarize these relations in the lattice of Fig. 9.

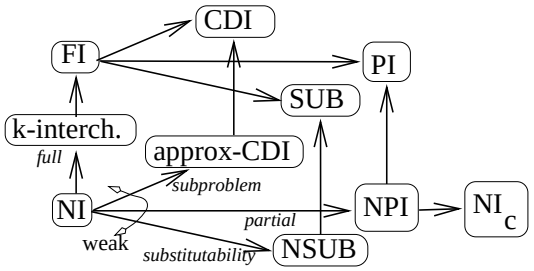

Figure 9: *A lattice of value interchangeability for a variable.* Weak interchangeability (*i.e.*, substitutability, partial, and subproblem [4]) and full interchangeability.

6.1 On the conceptual side

We use the example of Fig. 10 to compare three types of interchangeability with respect to 'partial interchangeability' (see Fig. 9). The NI-sets are computed by iter-

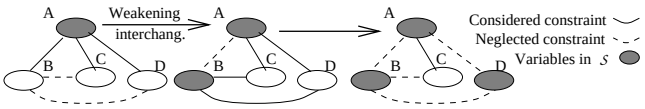

Figure 10: *Constraints for the computation of:* Left: NI for A. Center: NPI for $\{A, B\}$. Right: $\text{NI}_{C_{A,C}}$ for A.

ating over the neighborhood of a given variable, *e.g.* A in Fig. 10. This amounts to considering all the constraints that applies to A, namely: $C_{A,B}$, $C_{A,C}$, and $C_{A,D}$. NPI extends the boundaries within which change is permitted to comprise $\mathcal{S} = \{A, B\}$: it considers the constraints that apply to A or B, but not to A and B (*i.e.*, $C_{A,C}$, $C_{A,D}$, $C_{B,C}$, and $C_{B,D}$). NPI for A is weaker than NI for A: an NI-set for A is always a subset of some NPI-set for $A \in \mathcal{S}$. In [7], Haselböck introduced a form of local interchangeability that is obtained from NI by considering, for A, only one constraint that apply to A, *e.g.* $C_{A,C}$. We choose to call it 'NI for A according to $C_{A,C}$,' and denote it $\text{NI}_{C_{A,C}}(A)$. This is equivalent to extending the boundary to $\mathcal{S} = \texttt{Neigh}(\{\mathcal{A}\}) - \{C\}$, and to iterating, in the JDT, only over the values of A. 'NI according to $C_{A,X}$' is thus weaker than NPI computed for any set $\mathcal{S}$ such that $A \in \mathcal{S}$ and $X \notin \mathcal{S}$: an NPI-set for $\mathcal{S}$ is always a subset of an element of $\text{NI}_{C_{A,X}}(A)$. Thus, NPI sits between NI and NI_C from three standpoints: conceptually, with regard to the computational cost, and also with respect to the size of the sets it defines (see Fig. 8).

In [9], Weigel *et al.* looked for interchangeability in subproblems induced from the original CSP by reducing the domains of a selected set of variables, which they called *Context Dependent Interchangeability* (CDI), as shown at the left of Fig. 11. Further, they proposed to approximate CDI-sets by a local form of CDI, denoted approx-CDI in Fig. 9, obtained by computing the NI-sets in the subproblems. The success of this strategy for weakening local interchangeability depends crucially on the selection of the induced subproblems. These NI-sets cannot be compared with NPI-sets since they are derived for distinct problems. They are conceptually a localized form of another type of weak interchangeability called *subproblem interchangeability* [4].

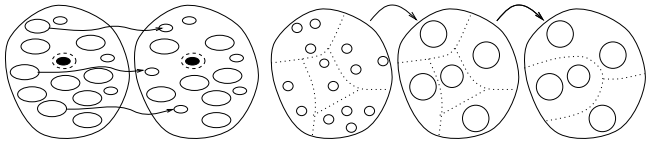

Figure 11: *Left:* Subproblem interchangeability by reduction of some variable domains. *Right:* Meta-interchangeability for variables.

In [8], Weigel and Faltings exploited NI-sets while constructing all solutions for a CSP. The iterative process is sketched at the right of Fig. 11. First, the variables are partitioned into subproblems, then each subproblem is replaced by a meta-variable that consists of all solutions to the subproblem (which are in fact partial solutions to the original problem). NI-sets are computed for each meta-variable, and the process iterates until a unique meta-variable is obtained, which represents all the solutions of the original CSP. This is a localized version of *meta-interchangeability* [4] for variables, not to be confused with k-interchangeability which requires full interchangeability in *all* subproblems of size k. As for approx-CDI, the choice of the subproblems is critical, and remains an open issue.

Most importantly, among all approaches explored for computing localized forms of weak interchangeability, our technique is the only one that gives full control of the coarseness of the interchangeability sets (through the selection, enlargement, and refinement of $\mathcal{S}$). This appears as if one can smoothly move along the axis labeled 'partial' in Fig. 9. In [4], Freuder gives an algorithm for computing, for any k, k-interchangeability for a variable (*i.e.*, moving along the vertical axis in Fig. 9), which naturally has a higher time complexity than our technique (*i.e.*, $O(n^{k-1}d^k)$ vs. $O(s(n-s)d^2)$).

6.2 Practical results

In [3], Choueiry *et al.* reported on various types of interchangeability discovered automatically by a decomposition heuristic, called VAD, for resource allocation. They showed how these sets are used for compacting the solution space and for supporting explanation and interaction with users. They also assessed the NI-sets discovered by this procedure with respect to the exact ones. Their evaluation of the interchangeabilities discovered by VAD can now, thanks to our results of Section 5, be extended to cover the case of NPI and NSUB-sets.

The practical usefulness of various kinds of local interchangeability has already been established on random problems for backtracking [1; 7], for arc-consistency [7], and for compacting solutions [8]. Freuder proved that "for *any* [k] there are cases in which preprocessing to remove redundant k-interchangeable values before backtracking, *k-interchangeability preprocessing*, will be cost effective" [4]. Since NPI is weaker than NI and than any level of k-interchangeability, it necessarily occurs more frequently, and the resulting benefits should be at least as good.

More generally, it is not clear whether one should search for 'symmetries' in random problems or in structured random problems, and whether this does not only reflect the power or shortcomings of the random generator itself. It seems to be more adequate to assess the usefulness of interchangeability in real-world problems, such as scheduling, configuration, and design. Since interchangeability is likely to be important in structured domains, it may greatly benefit practical applications. This has so far been shown for the case of resource allocation; experimentation with other application domains must be carried out.

7 Conclusions and Future directions

Interchangeability is known significantly to enhance search and backtracking, but has not yet been enough exploited for updating solutions. In this paper, we introduce NPI, which is weaker, and thus more frequent, than NI or FI. NPI is also an efficiently computable form of PI, which allows for local update of solutions in practical applications. We show how the procedure proposed by Freuder [4] is extended to compute the JDT-set, and thus identifying the NPI-sets and locally independent subproblems. We show how the JDT-sets are organized in a hierarchy, and give hints on how they can be exploited in practice. Further, we show that for the case of constraints of mutual exclusion, local interchangeability is easy to compute, and is equivalent to the existence of locally independent subproblems.

Our goal in this paper is to report the results summarized above and to draw the relations among the various types of interchangeability explored so far. Efforts now need to be invested in exploring strategies for selecting the sets of variables for which the JDT is built, and heuristics for monitoring the successive refinement or enlargement of these sets. We have already studied some properties of JDT-sets useful for this purpose, but do not report them here for lack of space. Another important issue is to combine the exploration of JDT-sets with the process of decomposing the CSP and solving it. Although we have not yet explored this, it should be possible to extend the techniques explored here to non-binary constraints by iterating, in the discrimination tree, over the constraints rather than over the domains of the neighboring variables. Further, and similarly to constraints of mutual exclusion, we believe that we must investigate the existence of constraint types for which interchangeability sets can be easily computed. We also believe that interchangeability may unveil novel strategies for domain splitting in a CSP involving continuous domains.

Acknowledgments

This work was started when the authors were at the Industrial Computing Lab. of the Swiss Federal Institute of Technology in Lausanne. B. Y. Choueiry is supported by a fellowship for advanced researchers from the Swiss-NSF.

References

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

[2] Berthe Y. Choueiry and Boi Faltings. Using Abstractions for Resource Allocation. In *IEEE 1995 International Conference on Robotics and Automation*, pages 1027–1033, Nagoya, Japan, 1995.

[3] Berthe Y. Choueiry, Boi Faltings, and Rainer Weigel. Abstraction by Interchangeability in Resource Allocation. In *Proc. of the 14 th IJCAI*, pages 1694–1701, Montreal, Canada, 1995.

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

[5] Eugene C. Freuder and Paul D. Hubbe. A Disjunctive Decomposition Control Schema for Constraint Satisfaction. In V. Saraswat and P. Van Hentenryck, editors, *Principles and Practice of Constraint Programming*, pages 319–335. MIT Press, Cambridge, MA, 1995.

[6] Eugene C. Freuder and Paul D. Hubbe. Extracting Constraint Satisfaction Subproblems. In *Proc. of the 14 th IJCAI*, pages 548–555, Montreal, Canada, 1995.

[7] Alois Haselböck. Exploiting Interchangeabilities in Constraint Satisfaction Problems. In *Proc. of the 13 th IJCAI*, pages 282–287, Chambéry, France, 1993.

[8] Rainer Weigel and Boi Faltings. Structuring Techniques for Constraint Satisfaction Problems. In *Proc. of the 15 th IJCAI*, pages –, Nagoya, Japan, 1997.

[9] Rainer Weigel, Boi Faltings, and Berthe Y. Choueiry. Context in Discrete Constraint Satisfaction Problems. In *12th European Conference on Artificial Intelligence, ECAI'96*, pages 205–209, Budapest, Hungary, 1996.