Approximating Partial Interchangeability in CSP Solutions

Nicoleta Neagu[1] and **Boi Faltings**[2]

[1]Whitestein Technologies AG
Pestalozzistrasse 24, 8024 Zurich
Switzerland
{nne}@whitestein.com
[2]Artificial Intelligence Laboratory (LIA)
Computer Science Department, EPFL
CH-1015, Ecublens, Switzerland
{boi.faltings}@epfl.ch

Abstract

The concept of *interchangeability* characterizes the possibilities for making local changes to CSP solutions. Often, interchangeability is only *partial* and also requires changing values assigned to other variables, called the *dependent set*. As partial interchangeability (PI) can only be computed by solving the whole problem, it needs to be approximated. We introduce the new concept of *neighborhood tuple interchangeability* (NTI) and show that it correctly approximates *partial interchangeability* (PI). We propose an algorithm for computing smallest dependent sets for NTI.

Keywords: constraint satisfaction, interchangeability.

Introduction and Background

Constraint Satisfaction Problems (CSP) prove to be a generic framework which can be applied for modeling and solving a wide range of combinatorial applications as planning, scheduling and resource sharing in many practical domains such as transportation, production, mass marketing, network management and human resources management. Many of this applications may require not only solving but also solution adaptation whereas an existing solution needs to be modified to satisfy additional criteria or accommodate changes in the problem. In this paper, we propose a method based on partial interchangeability for localizing changes in a CSP problem with the precise purpose of adapting CSP solutions. The algorithm we propose does not necessarily consider to bring yet another search algorithm for solving CSP problems, but a method for adapting solutions when this is required by the application domain. This can be exploited for example:

- in interactive configuration systems, where it is possible to show what parts of a solution might be affected by a local change ((Weigel & Faltings 1998)),
- in distributed problem solving, where it is possible to limit the set of agents that a change has to be coordinated with, and also to make local changes so that they do not spread through the entire problem, as shown by Petcu and Faltings ((Petcu & Faltings 2003)),
- in constraint-based control systems, where it is possible to choose control actions that have effects that are as local as possible,
- in problem abstraction, where a critical variable and the dependent set for making its domain interchangeable provide meaningful meta-variables, similar to the compilation technique in (Weigel & Faltings 1999).

Following its introduction by Freuder ((Freuder 1991)), interchangeability has been investigated by Choueiry and Faltings ((Choueiry, Faltings, & Rainer 1995)), for problem abstraction, by Neagu and Faltings ((Neagu & Faltings 2001)) for case adaptation and by Weigel and Faltings ((Weigel & Faltings 1998)), for configuration.

Interchangeability in binary constraint networks has been first proposed by Freuder ((Freuder 1991)) to capture equivalence among the values of a variable in a discrete constraint satisfaction problem. Value $x_i = a$ is *interchangeable* with $x_i = b$ if for any solution where $x_i = a$, there is an otherwise identical solution where $x_i = b$ and vice versa. *Full Interchangeability* considers all constraints in the problem and checks if values a and b for a certain variable x_i can be interchanged without affecting the global solution. The localized notion of *Neighborhood Interchangeability* considers only the constraints involving a certain variable x_i, as in the definition:

Definition 1 *([Freuder'91] Neighborhood Interchangeability - NI) Two values $x_i = a$ and $x_i = b$ are* neighborhood *interchangeable (NI) for variable x_i iff for every constraint C on x_i: $\{j|(a,j)satisfiesC\} = \{j|(b,j)satisfiesC\}$.*

NI values can be computed by the use of Discrimination Tree algorithm proposed by Freuder in ((Freuder 1991)). The construction of the DT proceeds in the following way: For each value of the variable x_i, we build a path containing in its nodes consistent values of variables in the neighborhood. Each time we start the process from the root node of the tree. For each neighboring variable value a node is constructed, but in the case it already exists in the next node of

the path the algorithm just makes the move to the node. The annotations of the leaves of the DT trees will be the equivalence classes of neighborhood interchangeable values.

For example, in the problem shown in Figure 1, values r and z of variable x_5 are neighborhood interchangeable (NI) [1]. Thus, interchanging r and z for variable x_5 in any solution does not require any changes in any other variable in order to remain a solution. NI is important because any values that are NI are also fully interchangeable, so NI can be used as an incomplete approximation of FI. Interchangeability as defined above is quite rare in practice. Usually, exchanging values also requires making changes elsewhere in the solution. Thus, in ((Freuder 1991)) Freuder also defined:

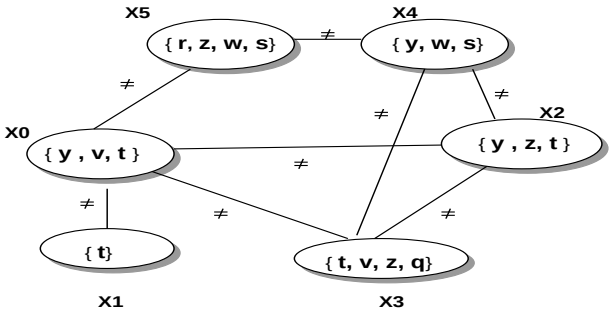

Figure 1: *Example of a CSP problem.*

Definition 2 *([Freuder'91] Partial Interchangeability - PI) Two values $x_i = a$ and $x_i = b$ are* partially *interchangeable (PI) with respect to a set of variables S iff for any solution involving one there also is an otherwise identical solution involving the other, except possibly different values for variables in S. We call x_i* the critical variable, *the set $\{a, b\}$* the interchangeable set *and the set S* the dependent set.

In the problem shown in Figure 1, values w and s for variable x_5 are partially interchangeable with respect to the dependent set of variables $S = \{x_4\}$. Thus, when interchanging values w and s for x_5 it may be necessary to change also the value of x_4 in order to maintain a consistent solution. There is no known algorithm for computing partial interchangeability without enumeration of all solutions. Choueiry and Noubir ((Choueiry & Noubir 1998)) proposed a localized algorithm for computing *neighborhood partial interchangeability* (NPI), using an algorithm based on a *Joint Discrimination Tree*. A formal definition of NPI concept is as follows:

Definition 3 *([Choueiry&Noubir'98] Neighborhood Partial Interchangeability - NPI) A value b for a CSP variable x_i is neighborhood partial interchangeable (NPI) with a value c for x_i given a boundary of change $\mathcal{S}$ (which include x_i) if and only if for every constraint C defined on the variables (x_i, x_k) where in $x_i \in \mathcal{S}$, $x_k \notin \mathcal{S}$, we have:* $\{j|(b,j)satisfiesC\} = \{j|(c,j)satisfiesC\}$.

[1] This example was inspired and further extended from one presented by Chouiery and Noubir in ((Choueiry & Noubir 1998)).

NPI values can be computed based on the JDT algorithm, see Algorithm 1. The JDT algorithm structures the possible values of the critical variable and the dependent set by considering the assignments of their neighborhood that they are consistent with. Each group of values that is compatible with the same set of assignments to the neighborhood becomes a leaf node and is indicated in the *annotation* of that node.

1: Create the root of the Joint Discrimination Tree.
2: **for** variable $X_i \in$ S **do**
3: **for** value $v_{il} \in D_{Xi}$ **do**
4: **for** variable variable $X_j \in$ Neigh(S) **do**
5: **for** value value $v_{jk} \in D_{Xj}$ consistent with v_{il} **do**
6: **if** there is a child node corresponding to '$X_j = v_{jk}$' **then**
7: Then move to it,
8: **else**
9: Construct such a node and move to it.
10: Add 'X_i, v_{il}' to annotation of the node (or root).
11: Go back to the root of the discrimination tree.

Algorithm 1: Algorithm for computing the Joint Discrimination Tree (JDT).

In the Figure 2, we show a graphical representation of the JDT algorithm for the set of variables $S = \{x_0, x_3\}$ of the problem in the Figure 1..

Not all partially interchangeable values can be detected by neighborhood-based algorithms. For example, the critical variable x_3 of the CSP in Figure 1 has a partially interchangeable set $\{t, z\}$ with respect to the dependent set $S = \{x_2\}$ because the variable x_0 will never take value t in a consistent solution as necessarily $x_1 = t$. Computing this fact requires computing global consistency and is not feasible with algorithms based on the neighborhood only.

Moreover, not all values detected by the NPI algorithm are partially interchangeable. For the problem shown in Figure 1, by applying the NPI algorithm for critical variable x_3 and dependent set $S = \{x_0\}$, we obtain that values v and q are NPI when $x_0 = \{v\}$. However, values v and q are not partially interchangeable for variable x_3 relatively to the dependent set $S = \{x_0\}$ because for example, the solution $x_0 = v, x_1 = t, x_2 = y, x_3 = q, x_4 = s, x_5 = w$ has no identical solution where $x_3 = v$ with other changes only in variable $S = \{x_0\}$. If x_3 takes value v the value of variable x_2 would also have to change. So, according to the Definition 2, values v and q for variable x_3 are not PI with respect to the set $S = \{x_0\}$ but only with respect to set $S = \{x_0, x_2\}$. So not all values detected by the NPI algorithm are PI.

Neighborhood Tuple Interchangeability

In order to guarantee consistencies among variables in the dependent set $\mathcal{S}$, we introduce the concept of *Neighborhood Tuple Interchangeability* (NTI) which is considering in the computation of PI interchangeable values not only the outer join semantics of the set $\mathcal{S} \cup \{x_i\}$ as NPI, but also its inner

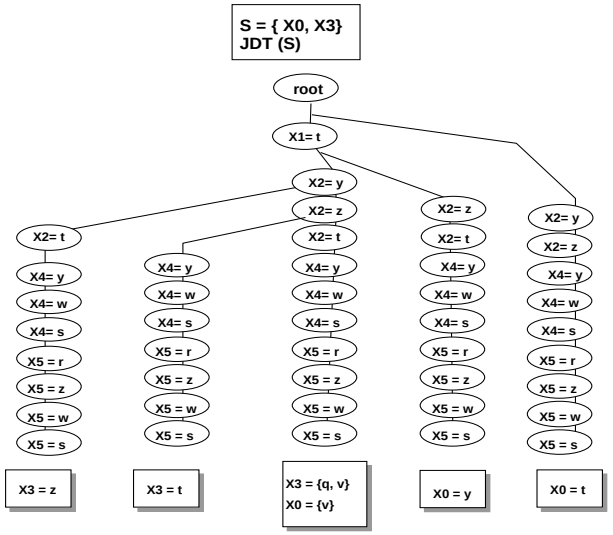

Figure 2: *Joint Discrimination Tree (JDT) for set* $S = \{x_0, x_3\}$.

join semantics. Thus, NTI is taking into account also consistency of values assigned to variables in the dependent set S and values to interchange of the critical variable x_i.

Definition 4 *(Neighborhood Tuple Interchangeability - NTI) Values* $x_i = a$ *and* $x_i = b$ *are* neighborhood tuple interchangeable *(NTI) with respect to a set of variables* S *if for every* consistent *tuple* t *of value assignments to* $S \cup \{x_i\}$ *where* $x_i = a$ *there is another* consistent *tuple* t' *where* $x_i = b$ *such that* t *and* t' *are consistent with the same value combinations for variables outside of* S*. Additionally, the same condition must hold with* a *and* b *exchanged.*

For example, by applying the NTI algorithm we propose in this paper for the critical variable x_3 in Figure 1 and the interchangeable sets $I_1 = \{t, z\}$ or $I_2 = \{q, v\}$, our NTI algorithm finds as dependent set the set of variables $S = \{x_0, x_2\}$ and its corresponding interchangeable tuples, see Table 1.

For NTI, we can show the following properties:

Theorem 1 *(Extensivity: NTI* $\Longrightarrow$ *NPI) Consider a critical variable* x_i*. If values* a *and* b *are NTI with dependent set* S*, then they are NPI with dependent set* S*. Consider two tuples* t *and* t' *of values assignments over a set of variables* S*. Then, if* t *and* t' *are neighborhood tuple interchangeable, this implies that they are neighborhood partial interchangeable.*

Proof: By Definition 4, if values a and b of variable x_i are NTI with respect to the dependent set S, for every consistent tuple t_a of value assignments to $S \cup \{x_i\}$ where $x_i = a$, there exists a consistent tuple t_b that admits $x_i = b$. t_a and t_b are consistent with the neighborhood of $S \cup \{x_i\}$. Then, if the set of variables S is removed from the problem, values a and b are both consistent with the neighborhood in the same way, thus NI with the problem. This makes them NPI according to the Definition 3.

Theorem 2 *(Extensivity: NTI* $\Longrightarrow$ *PI) Consider a critical variable* x_i*. If values a and b are NTI with dependent set* S*, then they are PI with dependent set* S*. Consider two tuples* t_a *and* t_b *of values assignments over a set of variables* S *and interchangeable set* $I = \{a, b\}$*. Then, if* t_a *and* t_b *are neighborhood tuple interchangeable, this implies that they are partial interchangeable.*

Proof: By Definition 4, if tuples t_a and t_b are NTI, they are compatible in the same way with the neighborhood of the dependent set S. Thus, by replacing t_a with t_b in a given solution, it will stay a solution. So, for a solution which contains a there is another solution which contains b with changes only in set S. By Definition 2, this means that values a and b in the interchangeable set I are PI.

Theorem 3 *Let* I *be a partially interchangeable set for critical variable* x_i *with dependent set* S*. Then* I *is also a neighborhood tuple interchangeable set for* x_i *with dependent set* $S' \supseteq S$*.*

Proof: If I is PI then there is a set of solutions to the entire problem that contain all values in I. These solutions make I NTI with respect to an S' which contains all variables of the problem.

In most cases, it will not be necessary to extend the dependent set to the entire problem to obtain NTI. For example, in the problem shown in Figure 1, the set $I = \{t, z\}$ is *partially interchangeable* for the variable x_3 with the dependent set $S = \{x_2\}$. This happens because variable x_0 would never take value t. Using NTI, we can find that I is a *neighborhood tuple interchangeable* set for variable x_3 with dependent set $S' = \{x_0, x_2\}$, where $S' \supseteq S$.

In this paper we propose an algorithm for computing tuple interchangeability. We first describe an algorithm that test for NTI using a novel structure called the joint tuple tree (JTT). We then present an algorithm that incrementally searches for a minimal dependent set containing NTI tuples. Based on Theorem 3, this algorithm also provides a way to test if a given interchangeable set can be PI at all. Finally, we give experimental results about the occurrence of tuple interchangeability and the size of dependent sets depending on the CSP structure.

Algorithm for Testing NTI

Firstly, we present an algorithm for testing whether a given interchangeable set is indeed neighborhood tuple interchangeable. This algorithm is based on the JDT algorithm described in previous section.

When searching for tuple interchangeability, it is sufficient to consider only a part of the JDT, namely the branch corresponding to the values of the interchangeable set. We thus define:

Definition 5 *(Reduced JDT) A* reduced JDT *is a JDT where we consider only the neighborhood assignments that are consistent with all values in the interchangeable set. We call these assignments the* Common Assignments.

In Figure 3, in the left side, one can see the reduced JDT for the set $S = \{S_1 \cup \{x_3\}\}$, where $S_1 = \{x_0\}$, relatively to critical variable x_3.

Thus, all branches of the reduced JDT involve only subsets of the common assignments. Each branch leads to a JDT

node that carries an annotation. We structure these nodes into a *Joint Tuple Tree (JTT)* that reflects the subset relations between the corresponding branches of the reduced JDT:

Definition 6 *(Joint Tuple Tree (JTT)) A* Joint Tuple Tree *is a tree which contains as nodes the leaves of the reduced JDT for a critical variable x_i and a dependent set S. A node n is a child of a node n' if the set of compatible assignments of n is a subset of that of n' and if there is no other n'' such that n'' would be child of n and n' child of n''. We annotate the arc between n and n' by the variables involved in assignments that are consistent with n' but not with n.* [2].

For example, if we consider that variable x_3 in Figure 1, an interchangeable set $I = \{q, v\}$ and a dependent set $S_1 = \{x_0\}$. Whereas the reduced JDT is shown in Figure 3 on the left side, the right side of Figure 3 shows the JTT obtained from the annotations of the reduced JDT. The root node contains the annotation with the critical variable x_3 and the interchangeable set $I = \{q, v\}$.

All the other nodes have assignments that are subsets of the root node assignment, and thus they become children of the root node. For example, as the annotation $x_0 = y$ has the assignments a subset of the annotation $x_3 = \{q, v\}$, it becomes its child.

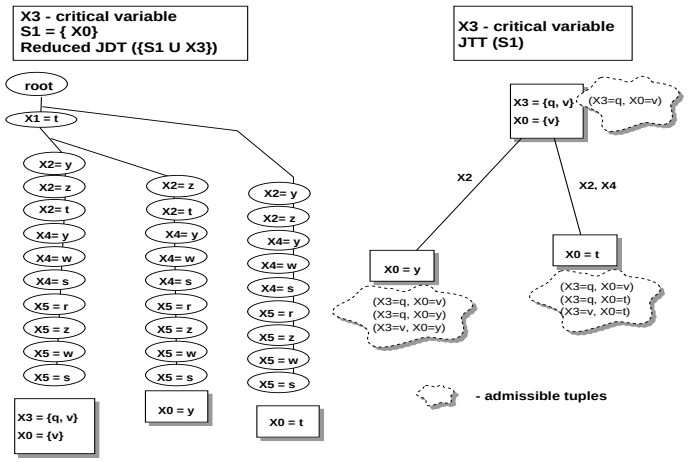

Figure 3: *Reduced Joint Discrimination Tree (JDT) for the critical variable x_3, interchangeable set $I = \{q, v\}$ and dependent set $S_1 = \{x_0\}$ (left side). Joint Tuple Tree (JTT) for the critical variable x_3, interchangeable set $I = \{q, v\}$ and dependent set $S_1 = \{x_0\}$ (right side).*

Algorithm 2 constructs the JTT from the reduced JDT. Note that as NTI implies NPI, there must be a leaf node of the reduced JDT that contains the interchangeable set in its annotations; otherwise, the values cannot be NTI. This leaf becomes the root node of the JTT. The other leaves become nodes of the JTT. Arcs between the nodes are constructed as in Definition 6 between nodes whose annotations are subsets of one another.

[2]The JTT arc between its nodes represent subset relations between the node assignments of the reduced JDT branches of the corresponding end nodes annotations.

We now consider the use of the JTT for computing tuple interchangeability. We associate the different consistent tuples of the dependent set with nodes of the JTT using the notion of admissibility:

Definition 7 *(Admissible Tuples) The admissible tuples at a node n of the Joint Tuple Tree(JTT) are all the combinations of the variable/value assignments to the critical variable and dependent set that are consistent with all assignments on the corresponding branch of the JDT.*

Lemma 1 *A consistent tuple t that contains a value of the interchangeable set and is consistent with the neighborhood will be admissible at some node of the JTT.*

Proof: The tuple must be consistent with at least one combination of assignments to the neighborhood, hence at least one branch of the JDT. As it contains a value of the interchangeable set, it is also in the reduced JDT, and hence in the JTT.

Theorem 4 *Admissible tuples of a JTT node are interchangeable.*

Proof: Let t and t' be two admissible tuples in a JTT node. Suppose that t is part of the solution tuple s of the CSP. That means that any value v_n of any variable x_n in the neighborhood is consistent with any value of the tuple t. By the construction of the JTT , any pairs of values in t and t' assigned to the same variable of the tuple are compatible with the same values of the variables in the neighborhood. Thus, one can interchange tuples t and t' in a solution and it will remain a solution, so they are interchangeable.

The JTT decomposes the different possible value assignments to variables in the neighborhood of the dependent set into environments represented as nodes. Each environment represents a combination of value assignments such that the

1: $T \leftarrow$ root node, r = leaf node with interchangeable set in reduced JDT.
2: $l \leftarrow$ remaining leaf nodes of the reduced JDT.
3: **repeat**
4: $n \leftarrow$ node in l such that no other node n' is compatible with all assignments on the path to n.
5: $p \leftarrow$ deepest node in T that is compatible with all the assignments on the path to n.
6: make n a child node of p; remove n from l.
7: annotate the arc between p and n with the variable involved in assignments consistent with the annotation of p but not consistent with the annotation of n.
8: **until** l is empty

Algorithm 2: Algorithm for computing the Joint Tuple Tree (JTT).

admissible tuples are the same. In order to have neighborhood tuple interchangeability, the admissible tuples for each environment must either contain a tuple for each interchangeable value, or no tuple with any of the interchangeable values. This is expressed by the notion of *compatibility*, defined as follows:

Definition 8 *(Compatibility) We call the subtree of the joint tuple tree rooted at node n* compatible *if either: (1) - for*

each value in the interchangeable set, the JTT admissible tuples at node n contain at least one tuple where the critical variable takes that value or (2) - the admissible tuples at node n contain no tuple where the critical variable takes a value in the interchangeable set, and all subtrees rooted at children of node n are compatible.

For example, in Figure 3 all the tree nodes of the JTT are compatible. The two children are compatible because their admissible tuples contain at least one tuple for the values to interchange $x_3 = q$ and $x_3 = v$, and the root node is also compatible since it has as children compatible nodes, even thou it does not have an admissible tuple for value $x_3 = v$.

To show that the values in the interchangeable set are indeed partially interchangeable, we need to show that every tuple containing one is an interchangeable tuple containing the other. Due to the structure of the JTT, we can test this using the following:

Lemma 2 *The values in the interchangeable set are NTI with respect to set S if and only if the JTT is compatible from the root node.*

Proof: Note that in the JTT, tuples admissible for any node on the path from a node n to the root node are compatible with all assignments as tuples admissible for n. Thus, they can be substituted for tuples in n. Consider a tuple t admissible at an arbitrary node n in the JTT. If the root of the JTT is compatible, there must be at least one node on the path from n to the root (including n) that contains a tuple for each value in the interchangeable set, and these tuples can be substituted for t.

Conversely, assume that any tuple t can be substituted with some other tuple t' where the critical variables takes a different value in the interchangeable set. Then t' must be in the admissible tuples of a node on the path from root to the node where t is admissible. Now consider t' instead of t and continue until we reach a t that is admissible at a node such that no node above it has any admissible tuples. Then it can only be substituted with other tuples admissible at the same node, and the subtree at that node must be compatible. Thus, the JTT is compatible from its root node.

In certain cases, the JTT also allows us to determine that values are not interchangeable for S or any superset:

Lemma 3 *If the root of the JTT is not compatible, and there is a value in the interchangeable set that does not occur in any of the admissible tuples of the nodes of the JTT, the values are not NTI for set S or any superset of set S.*

Proof: In this case, there are tuples admissible for some node in the JTT that cannot be substituted by a tuple where the critical variable would take on the missing value because of inconsistencies in the set $S \cup \{x_i\}$. Consequently, the values cannot be NTI, and this will not change when the set S is enlarged.

Complexity of JTT Algorithm: Consider the computation of the JTT for a dependent set S of size s and a maximum domain size d. There are at most $s \cdot d$ variable-value combinations. Each occurs in at most one leaf of the JDT so the number of leaves in the JDT and the number of nodes in the JTT is bounded to $s \cdot d$. In the worst case, the algorithm has to check for each leaf of the JDT whether the neighborhood values it is consistent with a subset of those for all other leaf nodes; there are $O((s \cdot d)^2)$ such tests. Each test requires at most $(n-s) \cdot d^2$ operations. The total complexity is $O(s^2 \cdot (n-s) \cdot d^4)$, thus polynomial in both s and d.

Algorithm for computing NTI

The NTI algorithm goal is to determine a minimal set to which the change in one CSP variable can be localized. For this purpose, we now consider the problem of determining a minimal dependent set S that would make a set I neighbourhood partial interchangeable.

The NTI algorithm 3 is based on the JTT (Algorithm 2), the discrimination algorithms proposed by Freuder in ((Freuder 1991)) and by Choueiry and Noubir in ((Choueiry & Noubir 1998)). It takes as input a critical variable x_i and an interchangeable set I. It determines a dependent set S of minimum size such that I is NTI with this dependent set. If no such set exists, it returns failure.

The algorithm first computes the discrimination tree (DT) for the critical variable x_i to check whether if its interchangeable set I is neighborhood interchangeable. If this is the case, it returns with $S = \phi$. If not, it uses the DT to determine which variables to include in the initial candidate dependent set S.

1: construct DT for x_i.
2: **if** interchangeable set values are NI **then**
3: return (success, $S = \phi$).
4: **else**
5: $S \leftarrow$ variables that are involved in DT assignments that are consistent with some values in the interchangeable set but not all.
6: $OPEN \leftarrow (\{S\})$
7: **repeat**
8: $S \leftarrow first(OPEN), OPEN \leftarrow rest(OPEN)$.
9: construct JDT for S.
10: construct JTT.
11: **if** root node compatible **then**
12: return (success, S).
13: **else**
14: c $\leftarrow$ sets of minimal combinations of nodes of the JTT such that the union of their admissible tuples contains each value of the interchangeable set at least once.
15: **for** c $\in$ C **do**
16: $NS \leftarrow S \cup \bigcup_{n \in C}$. annotations of the arcs which are on the path from root of JTT to node n.
17: **if** $NS \notin OPEN$ **then**
18: include NS in OPEN s. th. OPEN is ordered in increasing size.
19: **until** $OPEN = \phi$
20: return failure.

Algorithm 3: Algorithm for computing NTI. Input : critical variable x_i and interchangeable set I.

All assignments which place values in I in different

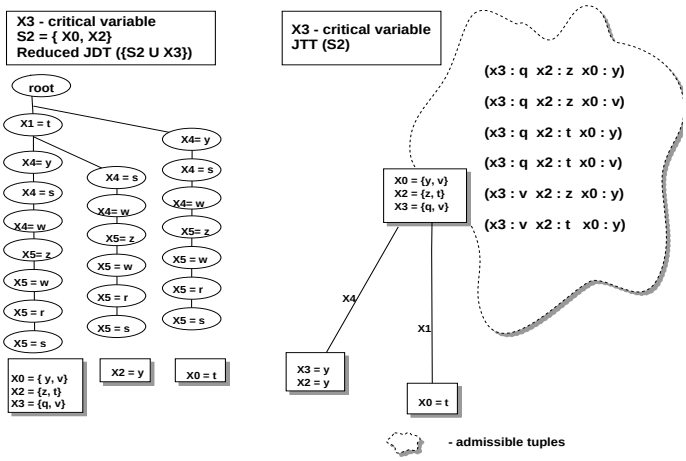

Figure 4: *Joint Discrimination Tree (JDT) for the critical variable x_3, interchangeable set $I = \{q, v\}$ and dependent set $S = \{x_0, x_2\}$ (left side). Joint Tuple Tree (JTT) for the critical variable x_3, interchangeable set $I = \{q, v\}$ and dependent set $S = \{x_0, x_2\}$ (right side).*

branches in the DT must be included in any dependent set S for the NTI.

It then enters a search for a minimal dependent set S, considering them in the order of increasing size to ensure that the smallest is found first. For each candidate set, it computes first the JDT and then the JTT using Algorithm 2. It then checks whether the JTT is compatible according to Definition 8. If it is, then S is a correct dependent set and the algorithm terminates.

If the JTT is not compatible, the algorithm generates all possible candidates for S that could provide a set of admissible tuples containing all values in the interchangeable set, and thus a compatible JTT. It adds these to the list of candidates, and continues with the next candidate.

Following the example given in the previous subsection for the computation of tuple interchangeability for variable x_3 and its interchangeable set $I = \{q, v\}$, we can see in Figure 3 that the root node of the JTT obtained in the right side of the figure is not compatible. Thus, the computation continues according to Algorithm 3 by including in the set S variable x_2 which makes the difference between the root node of the JTT and node $x_0 = y$, see Figure 3.

Note that we choose node $x_0 = y$ of the JTT since the other node would require to extend the set S more, with x_2 and x_4 as well. Further, the computation continues with the construction of the JDT for the new set S and its corresponding JTT as in Figure 4. The computation stops here since the root node of the JTT is compatible and returns the obtained set S and the corresponding interchangeable tuples.

The only consistent tuple we obtain from the root node annotation is: $(x_3 = q, x_0 = v)$. As there is no consistent tuple for the other value v of the interchangeable set we have to start searching through the JTT for a consistent child node.

The first child node $x_0 = x$, gives the following consistent tuples: $(x_3 = q, x_0 = v)$, $(x_3 = q, x_0 = x)$, $(x_3 = v, x_0 = x)$ as we consider also the annotation in its parent when we compute the tuples. As we obtain at least one tuple for each value from the interchangeable set $I = \{q, v\}$, node $x_0 = x$ is consistent. As variable x_2 makes a difference between assignments of the root node and the child node $x_0 = x$, we have to include it in the dependent set S which now becomes $S = \{x_0, x_2\}$ and reconstruct the JDT for the new S, see Figure 4. The new candidate sets are: $S = \{x_0, x_2\}$ and $S = \{x_0, x_2, x_4\}$ but we choose the smallest candidate first. This time the root node of the JTT is consistent as there is at least one consistent tuple for each value in the interchangeable set $I = \{q, v\}$.

The algorithm next examines the dependent set $S = \{x_0, x_2\}$. The interchangeable tuples for the root node of the JTT are displayed in Figure 4. As it now has tuples for all $x_3 = q$ and $x_3 = v$, the root node is compatible and the algorithm terminates.

x_5, $S = \{x_4\}$ $I_{x_5} = \{w, s\}$	x_3, $S = \{x_2, x_0\}$ $I_{x_3} = \{t, z\}$	x_3, $S = \{x_2, x_0\}$ $I_{x_3} = \{q, v\}$
(x_5, x_4)	(x_3, x_2, x_0)	(x_3, x_2, x_0)
(w, s)	(t, x, y)	(q, x, y)
(s, w)	(t, x, v)	(q, x, v)
(y, x)	(t, z, x)	(q, z, x)
(y, z)	(t, z, y)	(q, z, y)
	(t, z, v)	(q, z, v)
	(t, z, v)	(q, t, x)
	(t, q, x)	(q, t, y)
	(t, q, y)	(q, t, v)
	(z, x, y)	(t, q, y)
	(z, x, v)	(t, q, v)
	(z, t, x)	(q, x, y)
	(z, t, y)	(v, x, y)
	(z, t, v)	(v, z, x)
	(z, y, t)	(v, z, y)
	(z, y, v)	(v, t, x)
	(z, q, y)	(v, t, y)
	(z, q, t)	(q, z, x)
	(z, q, v)	(q, z, y)
		(q, z, t)
		(q, z, v)

Table 1: Interchangeable tuples for different critical variables and interchangeable sets for the CSP of Figure 1.

We display more results by applying the NTI algorithm for the variables of the problem shown in Figure 1 in the Table 1. We can see that for the critical variable x_5 and interchangeable set $I = \{w, s\}$, we obtained the dependent set $S = \{x_4\}$. For variable x_3 by interchanging values $\{t, z\}$ or $\{v, q\}$ the dependent set obtained is $S = \{x_2, x_0\}$ on different domain partitions and thus different values in tuples.

We can show:

Theorem 5 *Algorithm 3 is **sound**: if it returns a dependent set S, then the set I is Neighborhood Tuple Interchangeable for the critical variable x_i. Algorithm 3 is also **complete**: if the set I is NTI for x_i, then it will find a smallest dependent set S for this interchangeability.*

Proof: Soundness follows from the fact that the algorithm checks compatibility of the JTT by Lemma 2 this is a sufficient condition for NTI. Completeness follows from the fact that the algorithm checks all sets S that could satisfy the conditions of Lemma 2, and that the condition of Lemma 2 is also a necessary condition for NTI. Furthermore, when the set c becomes empty and there is no successor to a candidate S, then the algorithm has proven by Lemma 3 that there cannot be NTI with dependent set S or any superset of S, so it is not necessary to consider any possible indirect successors. Furthermore, the dependent sets are considered in order of increasing size so that the first set that is found is guaranteed to be smallest.

We note that the algorithm could be adapted to return all possible minimal dependent sets for NTI by returning them one at a time in step 12 and continuing the algorithm until the list $OPEN$ becomes empty.

Complexity of NTI Algorithm: The NTI algorithm complexity is generated by the computation of the JDT and JTT data structures it uses. The complexity of the DT when applied for only one variable of the CSP, the one for which we want to interchange the values, is $O(n \cdot d)$, where n represents the number of variables in the CSP and d the largest domain size. If the computation continues with the JDT algorithm for the set of variables to which the interchange propagates; its time complexity goes to $O(s \cdot (n-s) \cdot d^2)$, which bounds the former $O(n \cdot d)$. The complexity of the JTT algorithm as presented earlier in this paper is $O(s^2 \cdot (n-s) \cdot d^4)$ where n is the number of CSP variables, s is the size of the dependent set for which the JTT is constructed and d the largest domain size. In the worst case, the NTI algorithm constructs the JTT for all possible dependent sets in the neighborhood of the critical variable up to size s_{max} where an interchangeability is found or not. Thus, it can call the JTT algorithm at most $O(n^{s_{max}})$ times, and the worst case computation time is $O((n^{s_{max}})s_{max}(n-s_{max})d^4)$. Fortunately, we will see in the analysis on random problems that s_{max} is on average quite small.

Experimental results

In this section, we describe our results obtained during empirical study for NTI occurrence in random generated problems. During our experimental study we notice that NTI values occurrence depends mostly on the CSP tightness. We present these results in the following. In order to understand when NTI appears and how it depends on the problem structure, we concentrate again on four constraint problem parameters and tried to isolate them in terms of their effects on the tuple interchangeability. The problem characteristics on which we concentrate in our measurements were : the number of variables in the CSP, n, the maximum domain size of the variables, dom, the tightness of the constraints, t, and the constraint density, $dens$. Because lack of space we present here only the results of dependencies on the problem tightness parameter and problem size. In our experiments, we measured two parameters of the NTI occurrence:

- the average tuple size per variable, Av_s and
- the average number of interchangeable tuples per variable, Av_t.

Both measures are obtained by computing the sets of interchangeable tuples for each value pair of each variable normalized to the domain size of the variable.

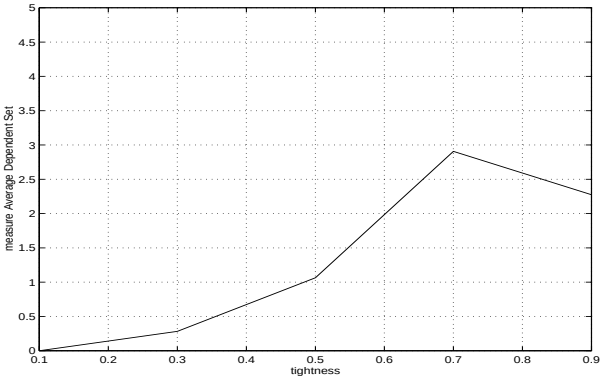

Figure 5: *The dependence of the interchangeable tuple size (Av_s) on the CSP tightness.*

In Figure 5 and Figure 6 we study how the interchangeable tuples measures defined above varies with the CSP tightness. For each data point, we generated 20 random problems with the following parameters : number variables $= 10$, domain size $= 10$, CSP density varies randomly in the interval [0.1, 0.9] and CSP tightness varies for each point in the set $\{0.1, 0.3, ..., 0.9\}$. We can observe that the average tuple size, Av_s, increases with the CSP tightness. The number of interchangeable tuples, Av_t, does not appear to depend on CSP tightness. Note that we count in our measurements also tuples of one variable, thus the NI values.

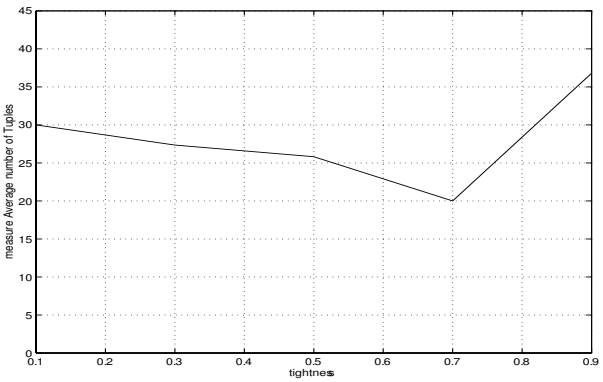

Figure 6: *The dependence of the number of interchangeable tuples (Av_t) on the CSP tightness.*

From our empirical experimentation we retrieve the following conclusions:

- the interchangeable tuple size and number do not depend on the CSP density;
- the interchangeable tuple size and number increases with the CSP tightness, where the number of tuples is more sensitive to the CSP tightness;
- the interchangeable tuples size does not depend on the problem size, while the number of interchangeable tuples

increases with the problem size. Both measures increases with the domain size.

Conclusions

Interchangeability is an interesting but not very deeply explored concept in constraint satisfaction. In this paper, we have developed the first algorithm that allows to find partial interchangeabilities without solving the entire problem. It is based on the concept of Neighborhood Tuple Interchangeability, which turns out to be more useful than Neighborhood Partial Interchangeability as defined earlier by Choueiry and Noubir ((Choueiry & Noubir 1998)).

We have given an algorithm that computes a smallest dependent set S for a desired interchangeability. If the algorithm finds such a set, it is guaranteed that the set is indeed partially interchangeable, but possibly with a smaller dependent set. An interesting result is provided by Theorem 3: if the set is found to be not NTI, then it can also be guaranteed to not be partially interchangeable at all. Thus, we actually have a complete method to compute all partial interchangeabilities; however, it does not necessarily find the smallest dependent sets. In experiments on random problems, we found that in general most values seem to become interchangeable with dependent sets of manageably small size. Thus, while the complexity of our methods are exponential in the size of the dependent sets, we do not expect this to be a great problem in practice.

In this paper, we have concentrated on computing NTI for a given critical variable and interchangeable set. In the future, it would be interesting to investigate if synergies can be obtained by computing NTI for all variables and domains in a single algorithm, in particular if it is possible to isolate rapidly those interchangeable sets for which the dependent sets are small.

References

Choueiry, B., and Noubir, G. 1998. On the Computation of Local Interchangeability in Discrete Constraint Satisfaction Problems. In *Proc. of AAAI-98*, 326–333.

Choueiry, B.; Faltings, B.; and Rainer, W. 1995. Abstraction by interchangeability in resource allocation. In *Proc. of the 14th IJCAI*, 1694–1701.

Freuder, E. C. 1991. Eliminating Interchangeable Values in Constraint Satisfaction Problems. In *In Proc. of AAAI-91*, 227–233.

Neagu, N., and Faltings, B. 2001. Exploiting Interchangeabilities for Case Adaptation. In *In Proc. of the 4th ICCBR-01*, 422–437.

Petcu, A., and Faltings, B. 2003. Applying Interchangeability Techniques to the Distributed Breakout Algorithm. In *In Proc. of the 18th IJCAI-03*.

Weigel, R., and Faltings, B. 1998. Interchangeability for Case Adaptation in Configuration Problems. *In Proc. of the AAAI98.*

Weigel, R., and Faltings, B. 1999. Compiling Constraint Satisfaction Problems. *Artificial Intelligence 115, pg. 257-287.*