Techniques for Bundling the Solution Space of Finite Constraint Satisfaction Problems

Berthe Y. Choueiry and Amy Beckwith
Constraint Systems Laboratory
Department of Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln NE 68588-0115
Email: {choueiry | abeckwit}@cse.unl.edu

Technical Report CSL-01-02

Abstract

We study the backtrack-search procedure with forward checking (FC-BT) for finding all solutions to a finite Constraint Satisfaction Problem (CSP). We describe how to use *dynamic* interchangeability to enhance the performance of search and represent the solution space in a compact manner. We evaluate this strategy (FC-DNPI) in terms of the numbers of nodes visited, constraints checked, and solution bundles generated by comparing it, theoretically and empirically, to other search strategies. We show that FC-DNPI is equivalent to search with the Cross Product Representation (FC-CPR) of [Hubbe and Freuder 1992] in terms of the numbers of solution bundles and constraint checks, while it reduces the number of nodes visited. We establish that both strategies are always superior to FC-BT in terms of all three criteria and dynamic bundling is always beneficial. Further, we compare FC-DNPI to the search procedure of [Haselböck 1993], which exploits static, pre-computed interchangeability relations. We show that the former never generates more solution bundles nor expands more nodes than the latter, and often reduces the number of constraint checks. We also propose, without evaluating them, amendments to the strategy of [Haselböck 1993] to improve its performance and reduce the number of constraint checks.

Contents

List of Tables

List of Figures

1 Motivation

Many problems in engineering, computer science, and management can be naturally represented as a Constraint Satisfaction Problem (CSP), which, in its general form, is likely to be intractable. Backtrack search remains the ultimate mechanism for solving such problems. Various strategies for enhancing this basic search procedure have been proposed. These strategies are mainly based on mechanisms for intelligent backtracking, algorithms for consistency checking, and heuristics for variable and/or value ordering.

Another opportunity to improve the performance of search is the identification and exploitation of symmetries in the problem or a particular instance of it. Glaisher [Glaisher 1874], Brown et al. citeBrown88, Fillmore and Williamson [1974], and Ellman [1993] proposed to exploit *declared* symmetries among values in the problem to improve the performance of search. While the first three papers considered *exact* symmetries only, the latter proposed to include also *necessary* and *sufficient approximations* of symmetry relations. Freuder [1991] introduced a classification of various types of symmetry, which he called interchangeability. He proposed an efficient algorithm, based on building a discrimination tree (DT), that *discovers* an exact but local form of interchangeability, neighborhood interchangeability (NI). Haselböck [1993] simplified NI to a weaker form that we call *neighborhood interchangeability according to one constraint* (NIC). He showed how to exploit NIC advantageously in backtrack search (BT), with and without forward-checking (FC), for finding all the solutions of a CSP. In [Choueiry and Noubir 1998] we proposed *neighborhood partial interchangeability* (NPI) that can be controlled to compute interchangeability anywhere between, and including, NI and NIC, as shown in Fig. 1. We generalized the DT of [Freuder 1991] into the

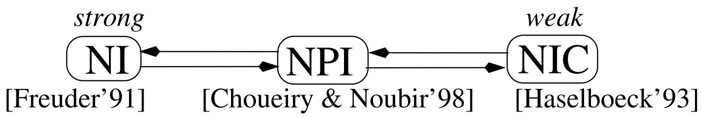

Figure 1: *Three types of neighborhood interchangeability.*

joint discrimination tree (JDT) to allow the computation of this new type of interchangeability. In Proposition 3.2 below, we relate the computation of NPI to that of NIC.

In parallel to the work on interchangeability, Hubbe and Freuder [1992] introduced the Cross Product Representation (CPR) to represent in a compact manner the partial solutions of a CSP during search. They proposed two search procedures based on CPR, with and without FC, that find all solutions to a CSP and reduce significantly the number of constraint checks.

In this paper, we concentrate on mechanisms to uncover and exploit symmetries in search while finding all solutions to a CSP, like the ones reported in [Haselböck 1993] and [Hubbe and Freuder 1992]. We propose one such new method based on the repeated computation of the JDT during search to find dynamic NPI sets (DNPI). While NIC or NPI can be computed in a pre-processing step prior to search, thus providing *static bundling*, CPR and DNPI are computed during search and provide *dynamic bundling*, which yields a better compaction of the solution space. We show how to exploit the structure of the JDT to *replace* the forward-checking mechanism. We show that our mechanism is equivalent to FC-CPR while it further reduces the number of nodes visited. Finally, we elucidate previously unstated relationships among the above listed algorithms with respect to their performance. We prove these relationships theoretically and verify them empirically.

2 Motivation and contributions

In addition to reducing the size of the search space, symmetry in general, and interchangeability in particular, can be used to represent the *solution space*, partially or entirely, in a compact manner by identifying families of qualitatively equivalent solutions useful in practical applications [Choueiry *et al.* 1995]. Our long-term objective is to organize the solution space of a CSP in order to draw, first, a landscape of this space then, to characterize regions of this landscape (e.g., as regions where solutions are numerous or rare, stable or brittle, easy or time-consuming to modify, etc.) Such a landscape is useful in practical applications as it allows a human user to: (1) Rank regions with respect to some optimization function or a qualitative property; (2) Use constraints that are hard to model, such as personal preferences, to discriminate among the individual solutions in a given region; and (3) In time-critical applications, quickly retrieve an alternative to a solution that is made inconsistent by an unforeseen event. Further, in a distributed environment where a problem is run independently through a number of specialized solvers (automated or human), global solutions can be obtained by intersecting compact solution sets of the individual solvers. The alternative strategy of having the distributed solvers collaborate on an individual solution, amending it iteratively or in parallel, is likely to cause loops and undermine the convergence of the problem-solving process.

There are industrial applications that require, or may benefit from, computing all the solutions to a CSP. For example, a qualitative simulation of a dynamic physical system requires the computation of all behavior trajectories so that the behavior of the system can be verified and validated for safety, stability, etc. In a product design or configuration task, a landscape of solutions will be a terrific support to the designers, especially in a collaborative environment.

With this perspective on practical applications in mind and as a step towards (our dream of) building a landscape of the solution space, we study in this paper the effects of static and dynamic bundling of the solution space of a CSP. The contributions of this paper are as follows:

1. We show how to exploit the JDT to compute interchangeabilities dynamically (DNPI) during backtrack search, yielding a better bundling of the solution space than NIC [Haselböck 1993].

2. We show that the JDT *readily* provides the domains of all future variables as they would result from applying forward checking, and for the same number of constraint checks. While this fact can be exploited to improve the performance of the search process proposed in [Haselböck 1993], its benefits remain superior in our approach.

3. We uncover the relationship between the concept of interchangeability [Freuder 1991] and the CPR mechanism [Hubbe and Freuder 1992] and show that result of search with DNPI is equivalent to that with CPR.

4. We establish theoretically and demonstrate empirically order relations between the FC-BT and the (static/dynamic) bundling algorithms with respect to three criteria that assess the search effort and the 'compaction' of the solution space. Note that the empirical evaluations are only meant to *verify* and *support* our theoretical results, and are not used to infer results.

This paper is organized as follows. Section 3 recalls the definition of a CSP, how to solve it, and the definitions of the interchangeability relations used in this paper. The three search strategies we examine are introduced in Section 4, then compared and evaluated in Section 5. Section 6 concludes the paper with directions for future research.

3 Definitions

A finite CSP is defined as $\mathcal{P}$=($\mathcal{V}$, $\mathcal{D}$, $\mathcal{C}$); where $\mathcal{V}$=$\{V_1, V_2, \ldots, V_n\}$ is the set of variables, $\mathcal{D}$=$\{D_{V_1}, D_{V_2}, \ldots, D_{V_n}\}$ is a set of their corresponding domains (the domain of a variable is a set of possible values), and $\mathcal{C}$ a set of constraints that specifies the acceptable combinations of values for variables. A solution to the CSP is to assign to each variable a value from its domain such as all constraints are satisfied. The question is to find one or all solutions. When all constraints are satisfied, the solution is said to be consistent (otherwise, it is inconsistent); when all variables are instantiated, the solution is said to be global (otherwise, it is partial). A CSP is often represented as a constraint (hyper-)graph in which the variables are represented by nodes, the domains by node labels, and the constraints between variables by (hyper-)edges linking the nodes in the scope of the corresponding constraint. We study CSPs with finite domains and binary constraints (i.e., they apply to two variables).

3.1 FC-BT

Backtrack search systematically instantiates one variable at a time constructively expanding a partial solution to a global one. In this paper we consider backtrack search with forward checking (FC-BT) [Haralick and Elliott 1980]. FC-BT ensures that each time a variable is assigned a value (current variable, V_c) the domain of each uninstantiated variable (future variable, V_f) connected to the current variable is revised to exclude values inconsistent with the assignment of the current variable. Consequently, FC-BT expands only partial solutions that are consistent. Further, the domains of all future variables are always consistent with that of every instantiated variable (past variable, V_p) given the binary constraints; thus, eliminating the need for back-checking (i.e., consistency checking against past variables). In this paper we study FC-BT with the various extensions that aim at compacting the search space. In Section 4, we justify why we choose forward checking rather than other look-ahead strategies.

3.2 Interchangeability

The general idea of interchangeability is that of a mapping between values of variables in a CSP such that any assignment remains a global consistent solution under the mapping (see functional interchangeability in [Freuder 1991]). In this paper, we consider only local forms of interchangeability, which are tractable. These are necessary approximations of the global forms, which are likely to be intractable. We consider here neighborhood interchangeability (NI) [Freuder 1991] and its weaker forms; namely neighborhood interchangeability according to one constraint (NIC) [Haselböck 1993], and neighborhood partial interchangeability (NPI) [Choueiry and Noubir 1998].

Definition 3.1. Neighborhood interchangeability (NI): *A value b for a CSP variable V is neighborhood interchangeable (NI) with a value c for V if and only if for every constraint C on V:* $\{x \mid (b,x)\text{ satisfies } C\} = \{x \mid (c,x)\text{ satisfies } C\}$.

Freuder provided an algorithm, the discrimination tree (DT), for computing the NI-sets of a given CSP variable V, which we summarize below and illustrate in

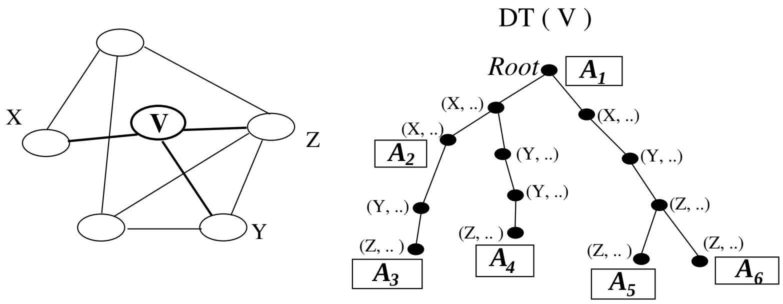

Figure 2: *The discrimination tree of* V. The nodes of the tree, except *Root*, are variable-value pairs from the neighborhood of V. The rectangles denote the partition of the domain of V.

Fig. 2. The idea is based on a simple consistency-checking mechanism between a variable and its neighbors and results in a partitioning of the domain of the variable into equivalence classes of values. DT operates as follows. It iterates over the values of V and checks their consistency, with respect to the constraints incident to V, with the values of all variables adjacent to V, i.e. $\{X, Y, Z\}$ in Fig. 2. It generates a tree structure. Each node in this tree represents a variable-value pair of the neighborhood of V (e.g., (X, x) where X is adjacent to V and $x \in D_X$). Further, the DT attaches to some nodes in the tree annotations, A_i's, that consist of values of V. These annotations determine a partition of the domain of V into its NI sets provided the variable-value pairs (X, x) are examined in a fixed order, such as a lexicographical one. Importantly, the path between a given annotation A_i and the root of the tree gives the values for the variables adjacent to V that are consistent with the values of V in A_i. These are *exactly* the new domains of the variables adjacent to V should forward-checking revise their domains after assigning the values in A_i to V. Thus, these variable-value pairs along each path can be used to update the domains of the future variables and eliminate the need for an explicit forward-checking procedure. *As a consequence, a (joint) discrimination tree provides not only the equivalence sets of values in the domain of a variable, but also, and at no extra cost, the new domains of the neighboring variables for each assignment of the variable to one of its domain partitions.* This information has not yet been exploited. It is the base of the improvement we propose for the strategy of [Haselböck 1993], and the mechanism that guarantees that our strategy never requires more constraint checks than FC-BT[1]. The time complexity of DT for computing the NI sets for one variable is $O(na^2)$ and its space complexity

[1]To this end, the implementation of the JDT has to stop expanding a path once it is clear that the domain of a neighboring variable is annihilated.

is $O(na)$, where n is the number of variables in the CSP and a is the maximum domain size.

Haselböck [1993] considered the partition of the domain according to *one* given constraint C_x among the ones incident to V and neglected the effects of all others constraints. This type of neighborhood interchangeability for one constraint C_x is denoted here 'NIC according to C_x' (NIC). The complexity of computing the NIC sets for a variable V according to a constraint C_x is $O(a^2)$ in time and $O(a^2)$ in space.

In [Choueiry and Noubir 1998] we showed how to extend the discrimination tree into the joint discrimination tree (JDT) to partition the domain of a variable V into sets of values that are neighborhood partially interchangeable (NPI). This weakened version of NI allows us to ignore the influence on V of variables specified within a boundary of change $\mathcal{S}$. We showed that when the JDT is restricted to computing the NPI sets of only one variable in $\mathcal{S}$ [2], it has a time complexity of $O((n-s)a^2)$ and a space complexity of $O((n-s)a)$, where s is the size of $\mathcal{S}$. Thus, it is cheaper than NI although more expensive than NIC, as we summarize in Table 1 (Bundle generation).

	Bundle generation			**Overhead to search**	
	Task	Time	Space	Time	Space
NI	One domain partition	$O(na^2)$	$O(na^2)$	$O(n^2a^2)$	$O(na)$
NIC	One domain partition	$O(a^2)$	$O(a^2)$	$O(n^2a^2)$	$O(n^2a)$
CPR	One Cross product	$O(na^2)$	$O(na^2)$	$O(n^2a^3)$	$O(n^2a)$
DNPI	One domain partition	$O((n-s)a^2)$	$O((n-s)a^2)$	$O(n^2a^3)$	$O(n(n-s)a)$
a: maximum domain size; n: number of variables. s: number of variables in $\mathcal{S}$.					

Table 1: *Cost of interchangeability.*

At the left of Fig. 3 we show again the relations between these three types of interchangeability.

In [Choueiry and Noubir 1998] we showed that the NPI partitions corresponding to a boundary of change $\mathcal{S}$ starting with $\mathcal{S} = \{V\}$ and gradually including the variables in the neighborhood of V can be organized in a hierarchy in which: (1) The finest partition, at the bottom of the hierarchy, corresponds to the NPI sets of V with $\mathcal{S} = \{V\}$ (i.e., NI sets). (2) The coarsest partition, at the highest level in the hierarchy, is the domain of V and corresponds to NPI partition of V with $\mathcal{S}'$ = $\{V$ and at least all the variables in its neighborhood$\}$. And, (3) two consequent

[2] The same JDT can be used to compute the NPI sets of all the variables in $\mathcal{S}$.

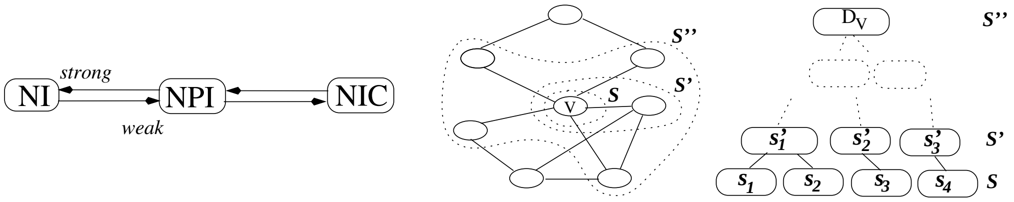

Figure 3: *Some types of interchangeability:* their relations and the corresponding hierarchy of domain partitions induced.

levels in the hierarchy correspond to boundaries of change that differ in at least one variable in the neighborhood of V; moreover, sets in a partition at a given level are either the same or the union of the ones at any lower level. This is illustrated in Fig. 3. Hence, we can infer the following relation between the NIC and NPI partitions of the domain of V:

Proposition 3.2. The NPI partition of the domain of V for $\mathcal{S} = \{V, V_1, V_2, \ldots, V_k\}$ is equal to the intersection of the NIC partitions of the domain of V according to V_1, V_2, ..., and V_k.

The proof of this proposition is straightforward given that the sets in a partition determine a relation of equivalence among the elements of the set. The intersection of k partitions can be done in $O(ka^2)$.

Finally, Freuder [1991] noted that interchangeability sets can be recomputed after instantiations are made during backtrack search, thus yielding *dynamic* interchangeability. We propose here a new type of dynamic interchangeability based on NPI.

Definition 3.3. Dynamic NPI (DNPI): *Given a variable ordering in a backtrack search integrating any kind of look-ahead scheme, the partition of the domain of the current variable, V_c, obtained by the JDT of V_c with $\mathcal{S}$ = $\{V_{p \leq c}\}$, defines a new type of NPI, which we call DNPI.*

For the *same* boundary of change, the DNPI partition of the domain of V_c is never finer than its NPI partition. Indeed, the fact that past variables are instantiated and the domains of the future variables are filtered accordingly, new opportunities for interchangeability of the values of V_c are likely, which cannot increase the number of sets in a partition.

4 Search with interchangeability

In this section, we discuss the use of interchangeability in search with forward-checking with partial look-ahead (FC-BT), see Section 3.1. The strategies we consider find all the solutions and assume the same ordering for variable and values across strategies. While we are aware that a promising alternative to FC is a full-lookahead scheme, we restrict our investigations to FC in order to maintain consistency of the search conditions with both [Haselböck 1993] and [Hubbe and Freuder 1992]. Further, a full-lookahead scheme can *only enhance* the quality of the dynamic bundling and thus the performance of the strategy we advocate. Finally, our current code implements only forward-checking and relies on the pseudo-code described in [Prosser 1993]. However, we intend to experiment with the techniques for full-lookahead described in [Sabin and Freuder 1997], which will certainly improve bundling and likely the constraint-checks count.

4.1 FC-NIC

FC-NIC operates as follows. For each variable V_c, the NIC partitions according to each of the constraints that apply to V_c are first generated and stored. Since each variable has at most $(n-1)$ such sets, this pre-processing requires $O(n^2a^2)$ time and $O(n^2a)$ space, reserved *throughout* the search process. We can conceptually separate these partitions into two sets: (1) `NIC-with-past`, computed according to constraints between V_c and a past variable $V_{p<c}$; and (2) `NIC-with-future`, computed according to constraint between V_c and a future variable $V_{f>c}$. Partitions in `NIC-with-past` are used when the domain of V_c is *revised* by FC, and those in `NIC-with-future` are used when V_c is *instantiated*.

When a $V_{p<c}$ is instantiated, FC-NIC revises the domain of V_c using the corresponding partition for V_c in `NIC-with-past`. If a value in a given set of the partition of V_c is consistent with the assignment of V_p, the whole set is kept, otherwise the whole set is removed.

When V_c is instantiated, its partitions in `NIC-with-future` are used. V_c is assigned the sets of values (i.e., bundles) obtained by intersecting *all* its NIC partitions in the set `NIC-with-future`. According to Proposition 3.2, these bundles are exactly the sets of the (static) NPI partition of V_c with $\mathcal{S} = \{V_{p\leq c}\}$. Thus, if k is the number of future variables $\{V_f\}$, the computation of the partitions in `NIC-with-future` ($O(ka^2)$) and their intersection ($O(ka^2)$) can be replaced with the computation of the NPI partition of the domain of V_c with $\mathcal{S} = \{V_{p\leq c}\}$ ($O(ka^2)$), which saves the effort for computing the intersection. This offers a first opportunity to improve FC-NIC.

As argued in Section 3.2, the revised domains of the future variables V_f when

V_c is assigned an NI set can be directly obtained from the discrimination tree. Consequently, using the JDT of V_c with $\mathcal{S}$ would also save all the constraint checks otherwise spent by the revise step in FC-NIC. This constitutes a second opportunity to improve FC-NIC. We did not implement either improvement.

4.2 FC-CPR

During search, the Cross Product Representation (CPR) assigns to a current variable V_c a set of values instead of a single value. Hence, a partial solution represents as many partial solutions as there are elements in the cross product of the assignments of instantiated variables $V_{p\leq c}$. FC-CPR [Hubbe and Freuder 1992] operates as follows. All possible values for V_c are considered and, for each such possibility, the domains of future variables $V_{f>c}$ are revised by forward checking. The filtered domains respective to the same $V_{f>c}$ are then compared for equality. When equality holds for all $V_{f>c}$'s, the corresponding values for V_c are merged into a set, which constitutes the bundled assignment of V_c.

4.3 FC-DNPI

We introduce here FC-DNPI, a search procedure with forward checking using the *dynamic* computation of NPI sets. It operates as follows. For a current variable V_c, the JDT of V_c with $\mathcal{S} = \{V_{p\leq c}\}$ is computed, yielding a partition of the domain of V_c and, for each set in this partition, the new domains for all $V_{f>c}$. V_c is then assigned in turn each of these sets, and the domains of all future variables are updated, for *each* of these bundled assignments, as specified by the corresponding path in the JDT. Since the domain of $V_{f>c}$ may be different across these assignments, the JDT for V_{c+1} computed with $\mathcal{S}' = \{V_{p\leq c+1}\}$ must be computed once for each of the possible assignments for V_c.

Proposition 4.1. The bundles of values assigned to V_c by FC-CPR are the sets of the DNPI partition of V_c with $\mathcal{S} = \{V_{p\leq c}\}$, which is, by Definition 3.3, the NPI computed after instantiating all variables $V_{p<c}$ and propagating the effects of these instantiations.

This proof is simple. Note, again, that FC-DNPI, unlike FC-NIC and FC-CPR, does not use a revise procedure to update the domain of $V_{f>c}$: the new domain is directly available from the JDT of $\mathcal{S}$. Further, whereas FC-CPR generates as many nodes in the search tree as there are values for V_c *before* it bundles them (when possible), FC-DNPI generates one node per set of the DNPI partition, thus reducing the number of nodes visited, see Theorem 5.1 below.

5 Comparison of the FC strategies

All strategies discussed in Section 4 add to the cost of search the cost of computing the bundles. These costs are summarized in Table 1 (Overhead to search). On the one hand, these worst-case values can be neglected given the exponential cost of search; on the other, they fail to show the *positive* effect of the bundling on the cost search. For this reason, we choose to compare the above procedures, both theoretically and empirically, with respect to the following criteria: (1) number of nodes visited (`NV`), (2) number of constraints checked (`CC`), and (3) number of solution bundles found (`SB`). (In the worst case, a solution bundle contains exactly one solution.) The first two criteria are orthogonal standard measures [Kondrak and van Beek 1995] for assessing the performance of search *independently of the implementation details*. The third criterion measures the performance of the bundling, the lesser the number of generated bundles, the better. We also report the CPU time, although we believe it should *not* be considered too critically as we justify in detail in Section 5.2.

5.1 Theoretical comparisons

The results stated below and illustrated in Fig. 4 are easy to prove. These results hold for all static or dynamic variable orderings, provided the orderings are the same for all strategies and search computes all solutions.

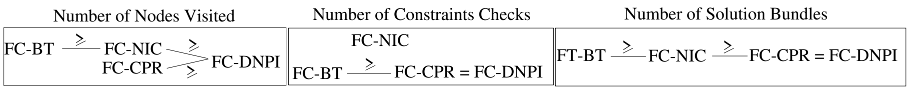

Figure 4: *Comparing bundling strategies.*

Theorem 5.1. *For the number of nodes visited (`NV`), the following orders hold:* `NV`*(FC-BT)* $\geq$ `NV`*(FC-NIC)* $\geq$ `NV`*(FC-DNPI) and* `NV`*(FC-CPR)* $\geq$ `NV`*(FC-DNPI).*

Theorem 5.2. *For the number of constraints checked (`CC`), the following orders hold:* `CC`*(FC-BT)* $\geq$ `CC`*(FC-CPR)=* `CC`*(FC-DNPI). However,* `CC`*(FC-NIC) is comparable to neither* `CC`*(FC-BT) nor* `CC`*(FC-DNPI).*

Theorem 5.3. *For the number of solution bundles generated (`SB`), the following total order holds:* `SB`*(FC-BT)* $\geq$ `SB`*(FC-NIC)* $\geq$ `SB`*(FC-CPR)* $=$ `SB`*(FC-DNPI)*

Informal justification: (1) Because FC-DNPI computes the domain partition of a current variable *during* search, it cannot generate more search nodes or more bundles than FC-NIC. (2) Because FC-CPR bundles after generating all future subproblems for a current variable, it may visit more nodes than FC-DNPI and thus cannot be compared with FC-NIC. (3) The domains of future variables in FC-DNPI are retrieved from the JDT, requiring the same number of constraint checks as FC-CPR requires to filter the domain of future variables. Further, because of the bundling, this number cannot exceed that required by FC-BT. (4) Finally, FC-NIC computes only once the NIC partitions for a variable V_c, whereas FC-DNPI requires the computation of a JDT for V_c for each of the bundled assignments for V_{c-1}. However, these additional constraint checks in FC-DNPI are often largely compensated for by the reduction of the solution space due to dynamic interchangeability.

5.2 Empirical evaluations

In order to verify and support the above claims, we implemented: elementary backtrack search (BT), with forward checking (FC-BT), static bundling strategies with NIC (BT-NIC and FC-NIC), and dynamic bundling strategies with CPR (FC-CPR) and DNPI (FC-DNPI). Note that FC-CPR and FC-DNPI have neither been implemented nor tested before. We used a static variable ordering according to the least domain (LD) heuristic. In order to reduce the duration of our experiments to a reasonable value, we chose to make *all* problems arc-consistent (AC-3) before search is started. Since this is done uniformly in all experiments and for all strategies, it does not affect the quality of our conclusions, although, for a given type of interchangeability relation, it may improve the bundling.

We conducted tests on puzzles and on randomly generated problems, and compared the strategies with respect to the three criteria cited above. We also included CPU time to show that the effort for bundling with CPR and DNPI is never prohibitive and almost always beneficial to search, even when no solution bundling is possible. Note however that: (1) the implementation of a JDT, dynamically built as a tree of structures, can be easily improved using arrays; (2) our Common Lisp code is experimental and our compiled code has not been optimized for run-time; (3) the CPU time includes time for saving intermediary numerical results on files; and (4) the resolution of the clock is of 10 ms, fractions are due to averaging effects. Thus, all reported CPU times should be considered, at best, for order of magnitude.

5.2.1 Negative examples:

It is well-known that the N-queens problem may not benefit from 'simple' interchangeability such as NIC [Freuder and Sabin 1997; Benhamou 2000], and thus, *a* *fortiori*, NI or NPI. We noticed this is also true for puzzles, such as Zebra. To handle such cases, we should investigate other types of symmetries, such as isomorphic interchangeability. Indeed, a careful examination of each of the constraints of the Zebra problem shows that FC-NIC necessarily yields bundles of individual solutions[3]. Further, the N-queen problem, while it presents some NIC partitions with non-singleton elements[4], 'resists bundling.' In both cases, the pre-processing step advised in [Haselböck 1993] adds to the number of constraint checks while drawing no benefits.

	Search	NV	CC-1	CC-2	CC	SB	**Time** [ms]
8-Queens	BT	26632	72462	-	72462	92	1340
	BT-NIC	3420	62752	7168	69920	92	1430
	FC-BT	2186	15508	-	15508	92	290
	FC-NIC	2186	15028	7168	22196	92	1020
	FC-CPR	2186	15508	-	15508	92	270
	FC-DNPI	2134	15508	-	15508	92	540
Zebra-1	BT	2213	5851	-	5851	1	150
	BT-NIC	384	3567	3112	6679	1	190
	FC-BT	209	972	-	972	1	30
	FC-NIC	209	1686	3112	4798	1	190
	FC-CPR	209	972	-	972	1	40
	FC-DNPI	175	972	-	972	1	40
Zebra-210	BT	23725666	62339123	-	62339123	210	1264660
	BT-NIC	4745344	62339123	5892	62345015	210	1330710
	FC-BT	285668	1803980	-	1803980	210	47050
	FC-NIC	285668	2012450	5892	2018342	210	111690
	FC-CPR	285668	1803980	-	1803980	210	54360
	FC-DNPI	268812	1803980	-	1803980	210	51980

CC-1: constraint checks during search
CC-2: constraint checks for computing interchangeability sets.
CC: sum of CC-1 and CC-2.
Time: compile code not optimized for run time, clock resolution 10 ms.

Table 2: *Results on puzzles.* **NUMERICAL VALUES NOT ACCURATE. HAVE BEEN IMPROVED.**

Table 2 reports the results of tests on the 8-queens, Zebra-1, and Zebra-210. Zebra-210 is obtained by removing the unary constraints from Zebra and has 210 solutions. The entries in the table support *each* of the theorems in Section 5.1.

[3]The same statement holds for the pigeon-hole problem.

[4]In the 8-queens problem where each variable is a queen in a row, the positions (1, 8) and (8, 1) for Q1 are NIC according to the constraint between Q1 and Q8.

Further, (1) Forward checking is always beneficial (compare $\langle$BT, BT-NIC$\rangle$ and $\langle$FC-BT, FC-NIC$\rangle$); (2) NIC degrades the number of constraint checks but not that of nodes visited and may sensibly degrade time performance by an order of magnitude; (3) Even when no *solution bundling* is possible, CPR and DNPI, which bundle dynamically, never do more constraint checks or expand more nodes than FC-BT. Moreover, FC-DNPI visits even less nodes, because it is bundling 'no-goods.' Time performance of dynamic bundling is of the same order of magnitude and the difference is not significant given the implementation.

In summary, dynamic interchangeability is shown to be worthwhile even for known counter-examples where NIC cannot possibly be effective.

5.2.2 Random problems:

Using the random-problem generator of [Bacchus and van Run 1995], we tested the above-listed procedures on random CSPs, with 10 variables ($n = 10$), a fixed domain size ($a = 5$), constraint density $d = \{.1, .5, .9\}$ and constraint tightness t going from 0.04 to 0.92 by steps of 0.08. We generated 20 random instances for each value of density and tightness, and averaged the values of `NV`, `CC`, `SB`, and time over the 20 instances. Numerical results for $t \leq 0.44$ are reported in Table 3. For $t > 0.44$, the results do not exhibit information worth mentioning and we do not show them here for lack of space.

Note first a case that seems to contradict our claims. For $\langle t = .44, d = 0.9\rangle$ where no solution exists, the values of `NV` do not respect Theorem 5.1, and the values of `CC` do not comply with Theorem 5.2 (except for FT-NIC where constraints checks for the computation of NIC before search are wasted). The difference is not significant and is caused by our ordering heuristics, which, by breaking ties randomly, did not produce the same results across the strategies.

Concerning the number of nodes visited (Theorem 5.1), we clearly see in Table 3 any bundling strategy always outperforms FC-BT (except for $\langle t = .44, d = .9\rangle$, see above justification). Fig. 5 (A) shows that:

1. `NV`(FC-DNPI)/`NV`(FC-NIC) is always less than 1 (light column),

2. `NV`(FC-CPR) and `NV`(FC-NIC) are not always comparable (dark column), and

3. FC-DNPI outperforms FC-CPR (light column never higher than dark column).

Concerning the number of constraint checks (Theorem 5.2), Table 3 shows `CC`(FC-CPR)= `CC`(FC-DNPI), except for $\langle t = .44, d = .9\rangle$, see above justification.

		Nodes Visited `NV`			**Constraint Checks** `CC`			**Solution Bundles** `SB`			**Time** `[ms]`		
$t \setminus d$		**0.1**	**0.5**	**0.9**	**0.1**	**0.5**	**0.9**	**0.1**	**0.5**	**0.9**	**0.1**	**0.5**	**0.9**
0.04	FC-BT	7356600	4325913	2573138	2708141	3978502	3364460	5710371	3256031	1859533	114577	122483	93760
	FC-NIC	2530	31813	142848	4320	55799	289700	624	9177	46403	280	4044	21760
	FC-CPR	6570	56276	187549	12043	161051	685384	481	5353	23178	385	3992	16490
	FC-DNPI	2038	20098	77460	12043	161051	685384	481	5353	23178	440	5378	23239
0.12	FC-BT	2638985	516245	106796	1310551	539847	214263	1890654	324581	53049	52198	17652	6389
	FC-NIC	30701	110709	61365	46913	219258	152561	9166	36268	18875	2842	13458	9533
	FC-CPR	50192	118384	53827	63797	249995	163366	5607	17773	9301	1881	6542	4250
	FC-DNPI	19835	61197	35587	63797	249995	163366	5607	17773	9301	2621	9929	6963
0.20	FC-BT	828805	56534	4593	561741	90370	17609	520957	25361	1138	21904	2957	580
	FC-NIC	47531	29802	4143	95700	65784	20156	14278	8240	752	4892	3796	1062
	FC-CPR	57899	24774	3720	77275	57899	16781	7895	3942	467	2337	1586	472
	FC-DNPI	28049	16971	3193	77275	57899	16781	7895	3942	467	3316	2456	766
0.28	FC-BT	230354	5926	372	156328	14278	3014	130375	1560	19	6388	459	108
	FC-NIC	23425	4558	369	42816	15668	6929	5848	788	16	2240	850	294
	FC-CPR	27663	3940	350	36797	12284	2975	3514	472	11	1140	404	109
	FC-DNPI	15078	3210	333	36797	12284	2975	3514	472	11	1625	560	148
0.36	FC-BT	73610	535	68	67515	2587	839	33493	50	0	2617	90	45
	FC-NIC	11637	488	72	22871	4998	4711	2879	30	0	1204	196	243
	FC-CPR	12172	475	68	18888	2504	838	1698	23	0	564	94	51
	FC-DNPI	7670	432	66	18888	2504	838	1698	23	0	802	111	64
0.44	FC-BT	17784	136	12	16665	918	167	6030	4	0	691	46	36
	FC-NIC	3283	132	13	7121	3256	1804	614	2	0	380	137	106
	FC-CPR	3347	126	13	5077	898	175	374	2	0	178	46	44
	FC-DNPI	2160	119	13	5077	898	178	374	2	0	245	50	42

Table 3: *Results on random problems.* **NUMERICAL VALUES NOT ACCURATE. HAVE BEEN IMPROVED.**

It also shows that it is not always a good idea to compute interchangeability as a preprocessing step: FC-NIC may wastefully increase to the number of constraint checks, see $\langle t = .20, d = .9\rangle$ and $\langle t = .28, .36, .44, d = .5, .9\rangle$. Fig. 5 (B) shows that for $t \leq .20$, dynamic bundling is always superior to static bundling (NIC) in spite of the repeated computation of the JDT.

The column on the *number of solution bundles* in Table 3 shows that all bundling strategies are pretty effective, as stated in Theorem 5.2. The number of solution bundles for FC-BT, which does no bundling, is in fact the number of solutions to the CSP. Fig. 5 (C) shows that dynamic bundling consistently outperforms static bundling.

Finally, *concerning CPU time*, by looking at the right-most column in Table 3, we see that bundling is always worthwhile, give or take experimental precision, except sometimes for FC-NIC (e.g., $\langle t = .12, d = 0.9\rangle$, $\langle t \leq .20, d = .5, .9\rangle$), however, the difference is not significant given the precision of the experiments. Fig. 5 (D) shows that for $t \leq .12$ dynamic bundling consistently outperforms static bundling (FC-NIC). The light column being always higher than the dark one indicates that FC-DNPI is always more costly than FC-CPR although it expands less

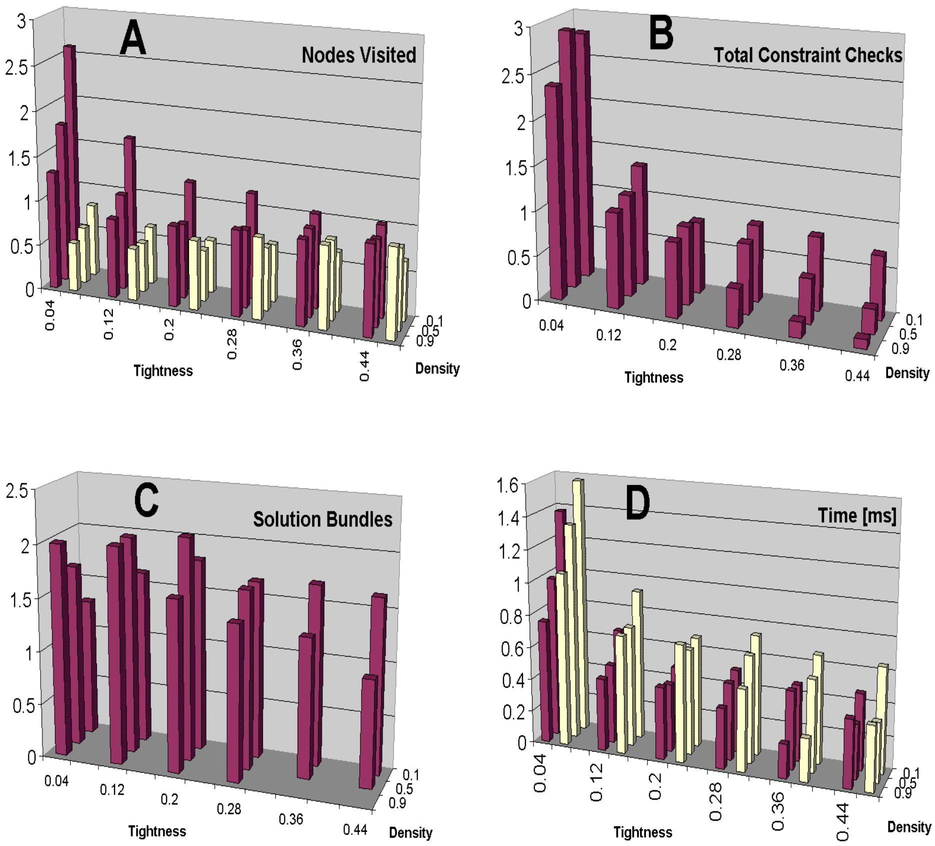

Figure 5: **A–Nodes visited:** Dark column: NV(FC-CPR)/NV(FC-NIC). Light column: NV(FC-DNPI)/NV(FC-NIC). **B–Constraint checks:** CC(FC-DNPI)/CC(FC-NIC). **C–Solution bundles:** SB(FC-DNPI)/SB(FC-NIC). **D–CPU time:** Dark column: Time(FC-CPR)/Time(FC-NIC). Light column: Time(FC-DNPI)/Time(FC-NIC).

nodes and does the same amount of constraint checks. This is obviously due to our expensive implementation of the JDT and can be fixed using arrays.

6 Conclusions and future research

In this paper we study the use of dynamic bundling in backtrack search for finding all the solutions of a CSP. We propose a new search procedure (FC-DNPI) based on the computation of dynamic neighborhood partial interchangeability. We discuss the relation between this new strategy and the known FC-NIC and FC-CPR procedures. We compare these strategies theoretically and empirically in terms of three criteria for assessing the performance of search and shown that our strategy can *never* cause any degradation *even when no bundling is possible*. We also propose improvements to FC-NIC to increase its performance and reduce the number of constraint checks. Our investigations can be continued in a number of directions:

- The effects of variable ordering on the bundling are investigated in a companion paper[Beckwith and Choueiry 2001]. Note, however, that no ordering heuristic can guarantee finding FI [Lesaint 1994].
- All the results reported here hold for finding all solutions. We are currently investigating to what extent they hold for finding a single, or a maximal bundle [Lesaint 1994], alternatively, a pre-specified number of bundles.
- Since these techniques, while harmless, cannot significantly improve the solving of puzzle-like problems, it is tempting to integrate DNPI with the symmetry-detection algorithm described in [Benhamou 2000], although computing those symmetries is known to be equivalent to graph isomorphism [Crawford 1992; Benhamou 2000].
- We are currently working on interchangeability for non-binary constraints and hope to report some results in the (near) future.
- We firmly believe that the use of interchangeability should be extended to CSPs with continuous domains, especially CSPs with monotonic constraints, functional constraints, and what we call pseudo-functional constraints, which are constraints that can be represented by block-diagonal binary matrices.

Acknowledgments

We are grateful to Chester Davis for advice on producing the charts in Excel, to Rob Glaubius for his hard work organizing the data, and to anonymous reviewers of a previous version of this paper for challenging us to defend our approach. Special thanks to Eugene C. Freuder for his invaluable feedback and stunning availability.

References

Bacchus, Fahiem and Run, P.van 1995. Dynamic Variable Ordering in CSPs. In *Principles and Practice of Constraint Programming, CP'95. Lecture Notes in Artificial Intelligence 976*. Springer Verlag. 258–275.

Beckwith, Amy M. and Choueiry, Berthe Y. 2001. Effects of Dynamic Ordering and Bundling on the Solution Space of Finite Constraint Satisfaction Problems. Technical Report CSL-01-03. `consystlab.unl.edu/CSL-01-03.ps`, University of Nebraska-Lincoln.

Benhamou, Belaid 2000. Symmetry and Dominance in Constraint Satisfaction Problems (Draft). Research Report TR-353, LIM, Centre de Mathématiques et Informatique de Marseille.

Choueiry, Berthe Y. and Noubir, Guevara 1998. On the Computation of Local Interchangeability in Discrete Constraint Satisfaction Problems. In *Proc. of AAAI-98*, Madison, Wisconsin. 326–333. Revised version KSL-98-24, `ksl-web.stanford.edu/KSL_Abstracts/KSL-98-24.html`.

Choueiry, Berthe Y.; Faltings, Boi; and Weigel, Rainer 1995. Abstraction by Interchangeability in Resource Allocation. In *Proc. of the 14 th IJCAI*, Montréal, Québec, Canada. 1694–1701.

Crawford, James M. 1992. A Theoretical Analysis of Reasoning by Symmetry in First-Order Logic (Extended Abstract). In *Working Notes of the Workshop on Tractable Reasoning, AAAI-92*, San Jose, CA.

Ellman, Thomas 1993. Abstraction via Approximate Symmetry. In *Proc. of the 13 th IJCAI*, Chambéry, France. 916–921.

Fillmore, Jay P. and Williamson, S.G. 1974. On Backtracking: A Combinatorial Description of the Algorithm. *SIAM Journal on Computing* 3 (1):41–55.

Freuder, Eugene C. and Sabin, Daniel 1997. Interchangeability Supports Abstraction and Reformulation for Multi-Dimensional Constraint Satisfaction. In *Proc. of AAAI-97*, Providence, Rhode Island. 191–196.

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

Glaisher, J.W.L. 1874. On the Problem of the Eight Queens. *Philosophical Magazine, series 4* 48:457–467.

Haralick, Robert M. and Elliott, Gordon L. 1980. Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Artificial Intelligence* 14:263–313.

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

Hubbe, Paul D. and Freuder, Eugene C. 1992. An Efficient Cross Product Representation of the Constraint Satisfaction Problem Search Space. In *Proc. of AAAI-92*, San Jose, CA. 421–427.

Kondrak, Grzegorz and Beek, Petervan 1995. A Theoretical Evaluation of Selected Backtracking Algorithms. In *Proc. of the 14 th IJCAI*, Montréal, Québec, Canada. 541–547.

Lesaint, D. 1994. Maximal Sets of Solutions for Constraint Satisfaction Problems. In *Proc. of the 11 th ECAI*, Amsterdam, The Netherlands. 110–114.

Prosser, Patrick 1993. Hybrid Algorithms for the Constraint Satisfaction Problem. *Computational Intelligence* 9 (3):268–299.

Sabin, Daniel and Freuder, Eugene C. 1997. Understanding and Improving the MAC Algorithm. In *Principles and Practice of Constraint Programming, CP'97. Lecture Notes in Artificial Intelligence 1330*. Springer Verlag. 167–181.