ELSEVIER

Artificial Intelligence 123 (2000) 133–156

Artificial Intelligence

www.elsevier.com/locate/artint

Decomposable constraints [☆]

Ian Gent [a], Kostas Stergiou [b,*], Toby Walsh [c]

[a] APES Research Group, School of Computer Science, University of St Andrews, Scotland, UK
[b] APES Research Group, Department of Computer Science, University of Strathclyde, Glasgow, Scotland G1 1HX, UK
[c] APES Research Group, Department of Computer Science, University of York, York, England, UK

Received 1 December 1999

Abstract

Many constraint satisfaction problems can be naturally and efficiently modelled using non-binary constraints like the "all-different" and "global cardinality" constraints. Certain classes of these non-binary constraints are "network decomposable" as they can be represented by binary constraints on the same set of variables. We compare theoretically the levels of consistency which are achieved on non-binary constraints to those achieved on their binary decomposition. We present many results about the level of consistency achieved by the forward checking algorithm and its various generalizations to non-binary constraints. We also compare the level of consistency achieved by arc-consistency and its generalization to non-binary constraints, and identify special cases of non-binary decomposable constraints where weaker or stronger conditions, than in the general case, hold. We also analyze the cost, in consistency checks, required to achieve certain levels of consistency, and we present experimental results on benchmark domains that demonstrate the practical usefulness of our theoretical analysis. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Constraint satisfaction; Search; Decomposable constraints; Generalized arc consistency; Maintaining arc consistency

1. Introduction

Constraint satisfaction problems occur in many real-life applications such as resource allocation, time tabling, vehicle routing, frequency allocation, etc. Many constraint satis-

---

[☆] Supported by EPSRC award GR/L/24014. The authors wish to thank other members of the APES research group.

[*] Corresponding author.

E-mail addresses: ipg@dcs.st-andrews.ac.uk (I. Gent), ks@cs.strath.ac.uk (K. Stergiou), tw@cs.york.ac.uk (T. Walsh).

PII: S0004-3702(00)00051-5

faction problems can be naturally and efficiently modelled using non-binary constraints like the "all-different" and "global cardinality" constraints [21,22,24]. Certain classes of these non-binary constraints are "network decomposable" [6,19] as they can be represented by binary constraints on the same set of variables. Throughout this paper, we will abbreviate this to *decomposable*. For example, an all-different constraint is decomposable into a clique of binary not-equals constraints. As a second example, a monotonicity constraint is decomposable into a sequence of ordering constraints on pairs of variables. Although many non-binary constraints fail to decompose into binary constraints on the same set of variables (for example, the parity constraint even $(x_1 + x_2 + x_3)$ cannot be represented as a binary constraint satisfaction problem without the introduction of additional variables), decomposable constraints like the all-different constraint are very common and important in a large number of problems. For example, all-different constraints occur frequently in scheduling problems like exam and sports timetabling, in frequency assignment problems, and in many combinatorial problems like Golomb rulers, all interval series and Langford's number problems. CSPLib at www.csplib.org provides many examples of such problems.

In this paper, we compare theoretically the levels of consistency which are achieved on non-binary constraints to those achieved on their binary decomposition. This paper includes material first appearing in [24]. However it both extends this material and includes other material that covers several new topics. To be precise, we present many new results about the level of consistency achieved by the forward checking algorithm and its various generalizations to non-binary constraints. We also compare the level of consistency achieved by arc-consistency and its generalization to non-binary constraints, and identify special cases of non-binary decomposable constraints where weaker or stronger conditions, than in the general case, hold. We correct an error in [24] that suggested that neighborhood inverse consistency on the binary decomposition is an upper bound on the level of consistency achieved by generalized arc-consistency on decomposable non-binary constraints. We also analyze the cost, in terms of consistency checks, required to achieve certain levels of consistency. In addition, we present experimental results on benchmark domains that demonstrate the practical usefulness of our theoretical analysis.

The remainder of this paper is organized as follows. In Section 2, we give formal background on constraint satisfaction problems and define various levels of consistency. In Section 3, we compare the level of consistency achieved by the forward checking algorithm and its generalizations on decomposable non-binary constraints. In Section 4, we repeat this analysis for arc-consistency and its generalization. In Section 5, we make an analysis of the number of consistency checks required to achieve certain levels of consistency. In Section 6, we present experimental results that demonstrate the practical relevance of the theoretical results. In Section 7, we discuss related work, and finally, in Section 8, we conclude and discuss future work.

2. Formal background

A constraint satisfaction problem (CSP) is a triple (X, D, C). X is a set of variables. For each $x_i \in X$, D_i is the domain of the variable. In this paper, a value $a \in D_i$ is noted (i, a). Each k-ary constraint $C_{i\ldots k} \in C$ is defined over a set of variables $(x_i, \ldots, x_k)$ by the subset of the cartesian product $D_i \times \cdots \times D_k$ which are consistent values. For a binary constraint C_{ij}, $C_{ij}(a,b)$ denotes that $((i,a),(j,b)) \in C_{ij}$, or in words, that (j,b) is a *support* for (i,a) on C_{ij} and vice verca. A CSP can be represented by a constraint graph where the nodes correspond to variables and the (hyper)edges correspond to constraints. In the remainder of this paper when we refer to the constraint graph of a decomposable CSP we will mean the constraint graph that corresponds to the binary decomposition, unless otherwise stated. Thus, by edges (or arcs) we will mean binary constraints. A *projection* of a k-ary constraint C over a subset $\{x_1,\ldots,x_i\}$ of its variables, where $i < k$, is a new constraint over those i variables. This constraint is defined by the subtuples of the consistent tuples in the original k-ary constraint that only contain values for the i projected variables. A solution to a CSP is an assignment of values to variables that is consistent with all constraints.

Many lesser levels of consistency have been defined for binary constraint satisfaction problems (see [5] for full references). A problem is *node-consistent* iff for every variable x_i and every value (i,a) in the domain of x_i, value (i,a) satisfies all unary constraints on x_i. A problem is (i,j)*-consistent* iff it has non-empty domains and any consistent instantiation of i variables can be extended to a consistent instantiation involving any j additional variables [9]. A problem is *arc-consistent* (AC) iff it is $(1,1)$-consistent. A problem is *path-consistent* (PC) iff it is $(2,1)$-consistent. A problem is *strong path-consistent* iff it is node-consistent, arc-consistent, and path-consistent. A problem is *path inverse consistent* (PIC) iff it is $(1,2)$-consistent. A problem is *neighbourhood inverse consistent* (NIC) iff any value for a variable can be extended to a consistent instantiation for its immediate neighbourhood [10]. A problem is *restricted path-consistent* (RPC) iff it is arc-consistent and if a value assigned to a variable is consistent with just a single value for an adjoining variable then for any other variable there exists a value compatible with these instantiations. A problem is *singleton arc-consistent* (SAC) iff it has non-empty domains and for any instantiation of a variable, the problem can be made arc-consistent. Many of these definitions can be extended to non-binary constraints. For example, a (non-binary) constraint satisfaction problem is *generalized arc-consistent* (GAC) iff for any variable in a constraint and value that it is assigned, there exist compatible values for all the other variables in the constraint [18]. Fig. 1 gives formal definitions of the above properties.

Following [5], we call a consistency property A *stronger* than B ($A \geqslant B$) iff in any problem in which A holds then B holds, and *strictly stronger* ($A > B$) iff it is stronger and there is at least one problem in which B holds but A does not. We call a local consistency property A *incomparable* with B ($A \sim B$) iff A is not stronger than B nor vice versa. Finally, we call a local consistency property A *equivalent* to B iff $A \geqslant B$ and $B \geqslant A$. The following identities summarize results from [5] and [10]: strong PC > SAC > PIC > RPC > AC, NIC > PIC, NIC ~ SAC, and NIC ~ strong PC.

Many algorithms enforce a certain level of consistency at every node in a search tree. For example, the *forward checking* algorithm (FC) maintains a restricted form of AC that ensures that the most recently instantiated variable and those that are uninstantiated are arc-consistent. If all remaining values for a variable are removed, a *domain wipe-out* occurs and the algorithm backtracks. Forward checking can be generalized to an algorithm for non-binary constraints (called nFC0 in [2]) which makes every k-ary constraint with $k-1$ variables instantiated arc-consistent. No pruning is performed on k-ary constraints with

- A binary CSP is (i, j)-*consistent* iff $\forall i \in X$, $D_i \neq \emptyset$ and any consistent instantiation of i variables can be extended to a consistent instantiation involving any j additional variables.
- A binary CSP is *arc-consistent* iff $\forall D_i \in D$, $D_i \neq \emptyset$ and D_i is arc-consistent. A domain D_i is arc-consistent iff $\forall a \in D_i$, $\forall j \in X$, with $C_{ij} \in C$, there exists $b \in D_j$ so that $C_{ij}(a, b)$.
- A binary CSP is *path-consistent* iff $\forall i, j \in X$, (i, j) is path-consistent. A pair of variables (i, j) is *path-consistent* iff $\forall (a, b) \in C_{ij}, \forall k \in X$, there exists $c \in D_k$ so that $C_{ik}(a, c)$ and $C_{jk}(b, c)$.
- A binary CSP is *strong path-consistent* iff it is $(j, 1)$-consistent for $j \leqslant 2$.
- A binary CSP is *path inverse consistent* iff $\forall (i, a) \in D, \forall j, k \in X$ so that $j \neq i \neq k \neq j$, $\exists (j, b) \in D$ and $(k, c) \in D$ so that $C_{ij}(a, b)$ and $C_{ik}(a, c)$ and $C_{jk}(b, c)$.
- A binary CSP is *neighbourhood inverse consistent* iff $\forall (i, a) \in D$, (i, a) can be extended to a consistent instantiation in the immediate neighbourhood of i.
- A binary CSP is *restricted path-consistent* iff $\forall i \in X$, $D_i \neq \emptyset$ and D_i is arc-consistent and, $\forall (i, a) \in D, \forall j \in X$ so that (i, a) has a unique support b in D_j, $\forall k \in X$ so that $C_{ik}, C_{jk} \in C, \exists c \in D_k$ so that $C_{ik}(a, c)$ and $C_{jk}(b, c)$.
- A binary CSP is singleton arc-consistent iff $\forall i \in X$, $D_i \neq \emptyset$ and $\forall (i, a) \in D$, $P|_{D_i=\{a\}}$ has an arc-consistent sub-domain. By $P|_{D_i=\{a\}}$ we denote the CSP obtained by restricting D_i to $\{a\}$ in a CSP P, where $i \in X$.
- A non-binary CSP is *generalized arc-consistent* iff $\forall D_i \in D, D_i \neq \emptyset$ and D_i is generalized arc-consistent. A domain D_i is generalized arc-consistent iff $\forall a \in D_i, \forall j, \ldots, k \in X$, with $C_{j,\ldots,i,\ldots,k} \in C$, there exists tuple $t = \{b, \ldots, a, \ldots, c\}$ allowed by $C_{j,\ldots,i,\ldots,k}$ so that t is a support for (i, a) on $C_{j,\ldots,i,\ldots,k}$.

Fig. 1. Formal definitions of various consistencies.

less than $k - 1$ variables instantiated. As required, this reduces to the forward checking algorithm FC, when applied to purely binary constraints.

Alternative and stronger generalizations of forward checking to non-binary constraints are studied in [2]. nFC1 applies (one pass of) AC on each constraint or constraint projection involving the current variable and exactly one future variable (by comparison, nFC0 does not use the constraint projections). nFC2 applies (one pass of) GAC on each constraint involving the current variable and at least one future variable. nFC3 makes the set of constraints involving the current variable and at least one future variable GAC. nFC4 applies (one pass of) GAC on each constraint involving at least one past variable and at least one future variable. nFC5 makes the set of constraints involving at least one past variable and at least one future variable GAC. As required, all these generalizations reduce to FC when applied to binary constraints.

Even higher levels of consistency can be maintained at each node in the search tree. For example, the *maintaining arc-consistency* algorithm (MAC) enforces AC at each node in the search tree [12]. If enforcing AC removes all remaining values for a variable, a *domain wipe-out* occurs and the algorithm backtracks. For non-binary constraints, the algorithm that *maintains generalized arc-consistency* (MGAC) on a (non-binary) constraint satisfaction problem enforces GAC at each node in the search tree. When comparing the amount of search performed by different backtracking algorithms, we assume that we are looking for all solutions and there is a static variable ordering. Many of these results will extend to a number of different dynamic variable and value ordering heuristics, again provided we compute all solutions. However, the proofs are significantly more complex and add very little to our understanding of the impact of decomposition on search and constraint propagation. We say that algorithm A *dominates* algorithm B ($A \geqslant B$) if when A visits a node then B also visits the equivalent node in its search tree, and *strictly dominates* ($A > B$) if it dominates and there is one problem on which it visits strictly fewer nodes. Algorithm A and B are *incomparable* if neither A dominates B or vice versa ($A \sim B$). The following identities summarize results from [2]: nFC2 $>$ nFC1 $>$ nFC0, nFC5 $>$ nFC3 $>$ nFC2, nFC5 $>$ nFC4 $>$ nFC2, nFC3 $\sim$ nFC4.

3. Forward checking on decomposable constraints

We will compare the level of consistency achieved by FC (and its generalizations) on decomposable non-binary constraints. We repeat this analysis for AC in the next section. We first identify a lower bound on the performance of FC applied to the binary decomposition.

Theorem 1. *For a decomposable non-binary constraint satisfaction problem, the forward checking algorithm* FC *on the binary decomposition strictly dominates the generalized forward checking algorithm* nFC0.

Proof. Consider a node in the search tree explored by the nFC0 algorithm. Assume that forward checking removes the value a for some variable x. Then x occurs in an k-ary constraint in which all $k - 1$ other variables have been assigned values. In the binary decomposition, not all arcs between x and these $k - 1$ variables can support assigning x the value of a otherwise this would be a consistent extension in the non-binary representation. Hence forward checking on at least one of these binary arcs will remove the value a.

To show strictness, consider a ternary constraint $x < y < z$ with x, y and z all having the domains $\{1, 2\}$. Assume a lexicographic variable ordering and a numerical value ordering (similar results are obtained with other variable and value orderings). FC first assigns 1 to x. Forward checking then reduces the domain of y to 2. After assigning this unit, forward checking discovers a domain wipeout for z. We therefore backtrack to the root of the search tree and assign 2 to x. Forward checking then discovers a domain wipeout for y. The problem is therefore insoluble, and FC shows this in 2 branches. The nFC0 algorithm takes longer to solve this problem as it must assign 2 values before the ternary constraint is checked. It therefore takes 4 branches to show insolubility. □

We can generalize the example in the last proof to show that nFC0 applied to non-binary constraints can explore exponentially more branches than FC on the binary decomposition. This proof holds for a wide variety of variable orderings. A variable ordering which instantiates variables with unit domains before those without is called an *unit preference* ordering.

Lemma 1. *There exists a decomposable non-binary constraint satisfaction problem in n variables on which the forward checking algorithm* FC *applied to the binary decomposition explores* 2 *branches, but the generalized forward checking algorithm* nFC0 *explores* 2^{n-1} *branches using any value ordering and any unit preference variable ordering.*

Proof. Consider the n-ary constraint $x_1 < x_2 < \cdots < x_n$ with each variable x_i having the domain $\{1, 2\}$. The variable and value ordering heuristics in the forward checking algorithm FC first assign a value, 1 or 2 to some variable x_i. The proof divides into four cases. If $1 < i \leqslant n$ and the value assigned to i is 1 then forward checking discovers a domain wipeout for x_{i-1}. If $1 \leqslant i < n$ and the value assigned to i is 2 then forward checking discovers a domain wipeout for x_{i+1}. If $i = 1$ and the value assigned to i is 1 then forward checking reduces x_2 to an unit domain, the unit preference variable ordering assigns this variable next and discovers a domain wipeout for x_3. In the final case, $i = n$ and the value assigned to i is 2. Forward checking then reduces x_{n-1} to an unit domain, the unit preference variable ordering assigns this variable next and discovers a domain wipeout for x_{n-2}. After each case, we backtrack, assign the alternative value to i and discover a domain wipeout. FC thus shows that the problem is insoluble in 2 branches. On the other hand, the nFC0 algorithm takes longer to solve this problem as it must assign $n - 1$ values before the n-ary constraint is checked. It therefore takes 2^{n-1} branches to show insolubility whatever the variable and value ordering. □

We can also give a simple upper bound on the performance of FC on the binary decomposition.

Theorem 2. *For a decomposable non-binary constraint satisfaction problem,* nFC1 *strictly dominates the forward checking algorithm* FC *on the binary decomposition.*

Proof. Since the problem is decomposable, the constraint projections that involve the current and one future variable are a superset of the arcs that FC applied to the binary decomposition makes arc-consistent. Hence, if FC on the binary decomposition prunes a value, so will the nFC1 algorithm.

To show strictness, consider a problem with four variables $\{x_1, x_2, x_3, x_4\}$, with domains $\{1\}$, $\{1, 2, 3\}$, $\{3, 4, 5\}$ and $\{5\}$ respectively. There is a "not-equals" constraint between x_1 and x_2 and a ternary constraint on $\{x_2, x_3, x_4\}$ specifying that each pair of variables in the constraint has a difference of more than 1. The only allowed tuple for this constraint is $\{1, 3, 5\}$. Assuming a lexicographic variable ordering and a numerical value ordering, FC on the binary decomposition will forward check the assignment of x_1 and remove 1 from the domain of x_2. Then, FC will assign 2 to x_2, remove 3 from the domain of x_3, assign 4 to x_3 and discover a dead-end. FC will then assign 5 to x_3, and fail again. It will then backtrack, assign 3 to x_2 and 5 to x_3, and will finally discover that there is no solution after 3 branches have been explored. nFC1 will only explore 2 branches because after value 1 is removed from the domain of x_2 the projections involving x_2 and future variables become empty. Therefore, as soon as one of the two remaining values of x_2 is tried, nFC1 closes the branch. □

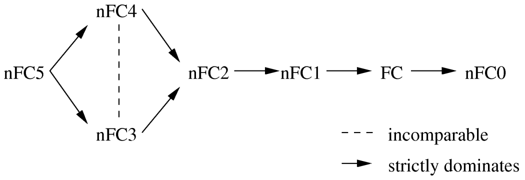

Fig. 2. The performance of the forward checking algorithm, FC on the binary decomposition of a set of decomposable non-binary constraints compared to the various generalizations of forward checking, nFC0 to nFC5 applied to the non-binary constraints.

We can also prove that FC on the binary decomposition and nFC0 may explore exponentially more branches than algorithms nFC1–nFC5. This proof holds for any variable and value ordering heuristics.

Lemma 2. *There exists a decomposable non-binary constraint satisfaction problem in n variables on which the* nFC1–nFC5 *algorithms explore just $n-1$ branches, but on which the forward checking algorithm* FC *applied to the binary decomposition takes $(n-1)!$ branches, and the* nFC0 *algorithm explores $(n-1)^{n-1}$ branches.*

Proof. Consider an n-ary all-different constraint on the variables $x_1, x_2, \ldots, x_n$, each with the domain $\{1, 2, \ldots, n-1\}$. FC explores $(n-1)!$ branches to show that the problem is insoluble. The nFC0 algorithm assigns $n-1$ values to the x_i $(1 \leqslant i \leqslant n)$ before the n-ary all-different constraint is checked. It therefore takes $(n-1)^{n-1}$ branches to prove that the problem is insoluble. By comparison, algorithms nFC1–nFC5 show that the problem is insoluble in $n-1$ branches since as soon as the first variable is instantiated with any one of its $n-1$ values, we enforce GAC (AC in the projections for nFC1) and discover that the current subproblem (the constraint projections) admit no satisfying tuples. □

These results, as well as those from [2], are summarized in Fig. 2.

4. Arc-consistency on decomposable constraints

Algorithms that enforce even higher levels of consistency than forward checking have been shown to be highly effective at solving binary and non-binary constraint satisfaction problems (see, for example, [3,24]). In this section, we characterize the level of consistency achieved by (generalized) AC on decomposable constraints. The following theorem puts a lower bound on the level of consistency achieved by GAC on decomposable constraints with respect to the binary decomposition.

Theorem 3. *Generalized arc-consistency on decomposable constraints is strictly stronger than arc-consistency on the binary decomposition.*

Proof. Consider any variable and value assignment. GAC ensures that each value in each variable of a k-ary constraint can be extended to a consistent k-tuple of values for the other variables in the constraint. Hence, each generalized arc-consistent value in each variable can be extended to a consistent 2-tuple. Hence, the binary decomposition is arc-consistent.

To prove strictness, consider an all-different constraint on 3 variables, each with domain $\{0, 1\}$. The binary decomposition of this constraint is arc-consistent but enforcing GAC shows that it is insoluble. □

As we show later on in this section, this lower bound is exact since we can exhibit a large class of problems on which GAC is equivalent to AC on the binary decomposition. We can also prove that an algorithm that maintains AC on the binary decomposition may explore exponentially more branches than an algorithm that maintains GAC.

Lemma 3. *There exists a decomposable non-binary constraint satisfaction problem in n variables ($n > 2$) that an algorithm that maintains generalized arc-consistency solves without search, but on which an algorithm that maintains arc-consistency on the binary decomposition explores $(n-1)!$ branches whatever the variable ordering.*

Proof. Consider an all-different constraint on n variables, each with the same $n-1$ values. Enforcing generalized arc-consistency shows that the problem is insoluble without search. Consider an algorithm that maintains arc-consistency on the binary decomposition. This is arc-consistent so we choose one of the $n-1$ values for one of the variables to instantiate. For $n > 3$, enforcing arc-consistency will prune this value from the other variables, leaving an arc-consistent subproblem in $n-1$ variables, each with the same $n-2$ values. Our argument therefore recurses. For $n = 3$, we are left with just 2 uninstantiated variables. Enforcing arc-consistency now causes a domain wipe-out and backtracking. The algorithm therefore explores $(n-1)!$ branches. □

The following theorem shows that NIC on the binary decomposition, as well as all the levels of consistency between strong PC and RPC, are incomparable to GAC. (See Theorem 6 for one condition under which NIC becomes strictly stronger than GAC.)

Theorem 4. *Generalized arc-consistency on decomposable constraints is incomparable to neighbourhood inverse consistency, to strong path-consistency, to singleton arc-consistency, to path inverse consistency, and to restricted path-consistency on the binary decomposition.*

Proof. Consider a problem with three all-different constraints on $\{x_1, x_2, x_3\}$, on $\{x_1, x_3, x_4\}$, and on $\{x_1, x_4, x_2\}$, in which x_1 has the unitary domain $\{1\}$ and every other variable has the domain $\{2, 3\}$. This problem is GAC, but enforcing RPC, and all properties stronger than RPC, shows that it is insoluble.

Consider the following 2-colouring problem. We have 5 variables, x_1 to x_4 which are arranged in a ring. Each variable has the same domain of size 2. Between each pair of neighbouring variables in the binary decomposition, there is a not-equals constraint. In the non-binary representation, we post a single constraint on all 5 variables. Note, that this is not an all-different constraint but a non-binary constraint that states that all neighbouring variables must have different values. This problem is NIC, but enforcing GAC on the non-binary representation shows that the problem is insoluble. Consider an all-different constraint on 4 variables, each with the same domain of size 3. The binary representation of the problem is strong PC, SAC, PIC and RPC, but enforcing GAC shows that it is insoluble. □

The upper bounds we can give for GAC tend to be rather weak. This is perhaps not surprising as we can post very large arity non-binary constraints. GAC may therefore achieve very high levels of consistency. The first upper bound we give is rather trivial. If the largest (non-binary) constraints involve k or fewer variables, then $(1, k-1)$-consistency is strictly stronger than GAC. This can be clearly understood if we consider that when trying to enforce GAC we take into account only explicit constraints, while $(1, k-1)$-consistency takes into account all implicit constraints between every set of k variables.

Theorem 5. *For decomposable non-binary constraints of arity k or less, $(1, k-1)$-consistency on the binary decomposition is strictly stronger than generalized arc-consistency on the non-binary constraints.*

Proof. Consider any variable and value assignment. $(1, k-1)$-consistency ensures that we can assign consistent values to the (at most) $k-1$ variables that appear with this variable in any given (non-binary) constraint. Hence, this constraint is generalized arc-consistent. Thus, $(1, k-1)$-consistency of the binary decomposition implies GAC of the original problem.

To prove strictness, consider a non-binary problem in 4 variables: x_1, x_2 and x_3 each with domains $\{1, 2\}$, and x_4 with domain $\{2, 3\}$. We post a ternary all-different constraint on x_2, x_3 and x_4, and not-equals constraints between x_1 and x_2, and x_1 and x_3. Now each of these constraints is generalized arc-consistent, so no values are removed. However enforcing $(1, 2)$-consistency shows that the problem is insoluble because of the constraints on x_1, x_2 and x_3. (In this example, x_4 is only there to guarantee that there is a ternary constraint in the problem.) □

A much stronger upper bound for GAC can be given if all the non-binary constraints in a problem decompose into cliques of binary constraints. For example, an all-different constraint decomposes into a clique of binary not-equals constraints. Under such a restriction, NIC on the binary decomposition is strictly stronger than GAC on the original (non-binary) problem.

Theorem 6. *If each non-binary constraint decomposes into a clique of binary constraints then neighbourhood inverse consistency on the binary decomposition is strictly stronger than generalized arc-consistency on decomposable constraints.*

Proof. Consider any variable and value assignment. NIC ensures that we can assign consistent values to the variable's neighbours. However, as the decomposition is into a clique, any (non-binary) constraint including this variable has all its variables in the neighbourhood. Hence, the (non-binary) constraint is generalized arc-consistent.

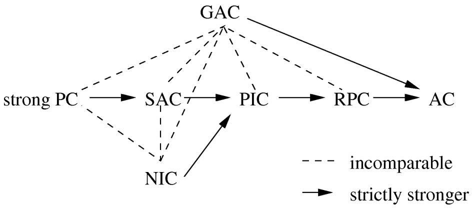

Fig. 3. The consistency of GAC on a set of decomposable non-binary constraints compared to various consistency techniques stronger than or equal to AC on the binary decomposition.

To prove strictness, consider again the problem with three all-different constraints from the proof of Theorem 4. This problem is generalized arc-consistent, but enforcing neighbourhood inverse consistency shows that it is insoluble. □

We had hoped to give weaker conditions under which NIC is strictly stronger than GAC. For example, we considered adding the binary constraints implied by path consistency to the binary decomposition. However, this is not enough to ensure that NIC implies GAC. In general, you may need *any* of the implied binary constraints. This may lead to prohibitively large neighbourhoods in the binary decomposition, with any variable that has a value removable by GAC connected to every other variable. On a minor note, this last upper bound is exact since we can exhibit a class of problems in which the non-binary constraints decompose into cliques of binary constraints and on which GAC is equivalent to NIC on the binary decomposition.

These results are summarized in Fig. 3.

Not surprisingly an algorithm that maintains GAC on decomposable constraints also strictly dominates the strongest generalized forward checking algorithm nFC5. Naturally, this means that it also dominates algorithms nFC0–nFC4 and also the forward checking algorithm FC applied to the binary decomposition.

Lemma 4. *For a decomposable non-binary constraint satisfaction problem, an algorithm that maintains generalized arc-consistency strictly dominates any of the generalized forward checking algorithms* nFC0 *to* nFC5*, as well as strictly dominating the forward checking algorithm* FC *on the binary decomposition.*

Proof. Algorithms nFC2–nFC5 enforce GAC in subsets of the problem and are therefore dominated by an algorithm that maintains GAC in the whole problem. From Theorems 1 and 3 it trivially follows that such an algorithm also dominates FC on the binary decomposition, nFC0 and nFC1.

To show strictness we only need to give an example where maintaining GAC explores less branches than nFC5. Consider an all-different constraint on the variables x, y and z, each with the domain $\{1, 2\}$. Assume a lexicographic variable ordering and a numerical value ordering (similar results are obtained with other variable and value orderings). nFC5 first assigns 1 to x, and then discovers there are no satisfying tuples for the all-different constraint. We therefore backtrack to the root of the search tree and assign 2 to x. This branch ends in failure by a similar argument. The problem is thus insoluble and FC5 shows this in 2 branches. All the other forward checking algorithms also explore 2 branches. By comparison, enforcing GAC immediately shows the problem is insoluble without any search. □

We can generalize the example in the last proof to show that FC on the binary decomposition can explore exponentially more branches than an algorithm that maintains GAC. This proof holds for any variable and value ordering heuristics.

Lemma 5. *There exists a decomposable non-binary constraint satisfaction problem in n variables on which the forward checking algorithm* FC *applied to the binary decomposition explores* $(n-1)!$ *branches, whilst* GAC *shows that it is insoluble without search.*

Proof. Consider an n-ary all-different constraint on the variables $x_1, x_2, \ldots, x_n$, each with the domain $\{1, 2, \ldots, n-1\}$. At each level in the search tree of the forward checking algorithm, one more value is removed from the domain of the remaining uninstantiated variables. The branching rate therefore decreases from $n-1$ to 1. When the $(n-1)$th variable is instantiated, the remaining variable suffers a domain wipeout and backtracking occurs. Forward checking therefore visits $(n-1)!$ branches before the problem is shown insoluble. By comparison, enforcing GAC immediately shows that the problem is insoluble. □

4.1. Tree decomposable constraints

We next identify a class of decomposable non-binary constraints on which GAC meets its lower bound (*viz.* AC on the binary decomposition). A special case of decomposable constraints are "tree decomposable" constraints in which the constraint graph of the binary decomposition forms a tree (or forest of independent trees). For example, the non-binary constraint that a list of variables is monotonically increasing is tree decomposable into a set of binary inequality constraints. Such monotonicity constraints are frequently used when we model real problems as they can be made to break unwanted symmetries. As the next two theorems demonstrate, tree decomposability topologically characterizes when GAC may be of benefit. If the constraint graph is a tree then GAC performs no more pruning than AC on the binary decomposition. On the other hand, if the constraint graph is not a tree, then GAC can be more pruningful. We first prove that GAC on tree decomposable constraints is no more effective than AC on the binary decomposition.

Theorem 7. *Generalized arc-consistency on tree decomposable constraints is equivalent to arc-consistency on the binary decomposition.*

Proof. (⇒) Consider a tree decomposable problem that is generalized arc-consistent. Consider two variables, x_i and x_j, and a value for x_i. The proof divides into two cases. Either x_i and x_j are directly connected to each other in some tree, or they are not. If they are connected, since the problem is generalized arc-consistent, there is a consistent value for x_j. If they are not connected then any value for x_j is consistent. Thus, the tree decomposition is arc-consistent.

($\Leftarrow$) Consider the tree decomposition of a problem that is arc-consistent. Consider a variable, x_i and a value from its arc-consistent domain. We now show how to find consistent values for all the other variables. We take the parent and each of the children of x_i. As the tree decomposition of the problem is arc-consistent, we can find consistent values for these variables. We repeat this process until we reach the root and the leaves. We now consider any uninstantiated children of the root. Again, as the tree decomposition of the problem is arc-consistent, we can find consistent values for these variables. We then consider the children of these variables and repeat until all variables are instantiated. Hence, there exists a consistent extension for the value assigned to x_i, and the problem is generalized arc-consistent. □

This result is perhaps rather unsurprising. Freuder has shown that when the constraint graph of a binary constraint satisfaction problem is a tree, we can solve problems by enforcing AC and then instantiating the variables in a suitable order [8]. Hence, as AC essentially determines global consistency, GAC is unable to achieve anything higher. In fact, even AC is too much since a restricted form of AC called "directional arc-consistency" is enough to ensure backtrack free solutions in constraint trees [7]. What is perhaps more surprising is that tree decomposition precisely characterizes when GAC can do more pruning than AC on the binary decomposition. To be more precise, as soon as the constraint graph of the binary decomposition is no longer a tree (or forest of trees) but contains one or more cycles, there are problems on which GAC performs more pruning than AC on the binary decomposition.

Lemma 6. *Given a binary constraint graph which has one or more cycles, then there exists a non-binary problem with this decomposition on which generalized arc-consistency is strictly stronger than arc-consistency on the binary decomposition.*

Proof. By Theorem 3, GAC is stronger than AC on the binary decomposition. To show strictness, given a binary constraint graph containing one or more cycles, we construct a non-binary problem with this decomposition on which GAC performs more pruning than AC on the binary decomposition. We first find a cycle in the binary decomposition. We then construct a non-binary constraint on the variables in this cycle. Each variable is given a domain with the same two values. If the cycle found is of odd length, then we construct a non-binary constraint that ensures that neighbouring variables in the chain take different values. If the cycle found is of even length, then we construct a non-binary constraint that ensures that neighbouring variables in the chain take different values except for one pair of variables which must take equal values. Enforcing GAC on this non-binary constraint will show that the problem is insoluble. By comparison, the binary decomposition is arc-consistent. □

In the next section, we characterize a large class of problems which are not tree decomposable and on which GAC is guaranteed to achieve levels of consistency much higher than AC on the binary decomposition.

4.2. Triangle preserving constraints

By imposing some slightly stronger conditions on the type of non-binary constraints, we can prove that generalized AC is significantly stronger than AC on the binary decomposition. One such condition (first studied in [24]) is when the non-binary constraints contain all length 3 cycles (triangles). The intuition is that the constraints then capture an inherent non-binary aspect of the problem. We say that a set of decomposable constraints is *triangle preserving* if for every triangle of variables in the constraint graph of the binary decomposition those 3 variables are involved together in a non-binary constraint.

For example, an all-different constraint is triangle preserving as it decomposes into a clique of binary not-equals constraints. Therefore, all variables occuring in a triangle of not-equals constraints in the binary decomposition are involved together in the all-different constraint. However, a set of all-different constraints may not be triangle preserving as the binary decomposition may contain 3-cliques that are not explicitly involved in an all-different constraint in the n-ary representation. For example, the first problem in the proof of Theorem 4 is not triangle preserving because there is a clique of not-equals constraints in the binary encoding between variables x_2, x_3, x_4 and these variables do not occur together in a constraint in the n-ary representation. Binary constraints can still occur in a triangle preserving set of non-binary constraints, but only if they do not form part of a triangle. A triangle preserving set of non-binary constraints is trivially not tree decomposable. Under the restriction to triangle preserving constraints, GAC is strictly stronger than path inverse consistency, which itself is strictly stronger than AC.

Theorem 8. *On a triangle preserving set of constraints, generalized arc-consistency is strictly stronger than path inverse consistency on the binary decomposition.*

Proof. Consider a triple of variables, x_i, x_j, x_k and any value for x_i from its generalized arc-consistent domain. The proof divides into four cases. In the first, x_i and x_j appear in one constraint, and x_i and x_k in another. As each of these constraints is arc-consistent, we can find a value for x_j consistent with x_i, and for x_k consistent with x_i. As the (non-binary) constraints are triangle preserving, there is no direct constraint between x_j and x_k so the values for x_j and x_k are consistent with each other. Hence, the binary representation of the problem is PIC. Up to symmetry, there are three other cases: x_i, x_j, and x_k can have no direct constraints between each other; x_j and x_k can have a constraint between then but neither has a direct constraint with x_i; and x_i, x_j, and x_k all have direct constraints between each other. Each case follows a similar argument. To show that GAC is strictly stronger, consider an all-different constraint on 4 variables each with domains of size 3. This problem is PIC but not GAC. □

We can also prove that for a triangle preserving set of constraints an algorithm that maintains path inverse consistency on the binary decomposition may explore exponentially more branches that an algorithm that maintains GAC.

Lemma 7. *There exists a triangle preserving set of constraints in n variables* ($n > 2$) *that an algorithm that maintains generalized arc-consistency solves without search, but on* *which an algorithm that maintains path inverse consistency on the binary decomposition explores* $(n-1)!$ *branches whatever the variable ordering.*

Proof. We can use the example in the proof of Theorem 3, observing that, on the binary decomposition, path inverse consistency does no more pruning than arc-consistency. □

A corollary of Theorem 8 is that GAC on a triangle preserving set of constraints is strictly stronger than restricted path-consistency or AC on the binary decomposition. Even when restricted to triangle preserving sets of constraints, GAC remains incomparable to strong path-consistency, singleton AC, and neighbourhood inverse consistency.

Theorem 9. *On a triangle preserving set of constraints, generalized arc-consistency is incomparable to strong path-consistency, to singleton arc-consistency and to neighbourhood inverse consistency on the binary decomposition.*

Proof. Consider an all-different constraint on 4 variables, each with the same domain of size 3. The binary representation of the problem is strong PC and SAC, but enforcing GAC shows that it is insoluble. Consider the second problem in the proof of Theorem 4 with the addition of an all-different constraint on variables x_1, x_5, x_6, with x_5 and x_6 having the domain $\{0, 1, 2\}$. This problem is NIC but enforcing GAC on the non-binary representation shows that the problem is insoluble. Note, that since there is no clique of 3 variables in the binary encoding of this problem, it is triangle preserving.

Consider a problem with five all-different constraints on $\{x_1, x_2, x_3\}$, on $\{x_1, x_3, x_4\}$, on $\{x_1, x_4, x_5\}$, on $\{x_1, x_5, x_6\}$, and on $\{x_1, x_6, x_2\}$. in which x_1 has the unitary domain $\{1\}$ and every other variable has the domain $\{2, 3\}$. Note, that this set of constraints is triangle preserving since all triangles of variables in the binary decomposition are involved in an all-different constraint in the n-ary representation. The problem is GAC, but enforcing NIC, or strong PC, or SAC shows that it is insoluble. □

These results are summarized in Fig. 4.

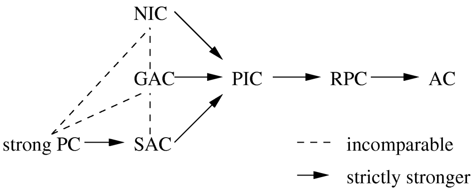

Fig. 4. The consistency of GAC on a triangle preserving set of non-binary constraints compared to various consistency techniques stronger than or equal to AC on the binary decomposition.

5. Consistency checks

In the previous two sections we compared the levels of consistency achieved by generalized algorithms on decomposable constraints to the levels achieved by FC and AC on the binary decomposition. We now analyze the relative numbers of consistency checks required to achieve these consistencies.

5.1. Forward checking

FC performs $O(d)$ consistency checks between the currently assigned variable and each future variable, where d is the maximum domain size. If $C_{c,f}$ is the number of constraints between the current variable and future variables then FC performs $O(C_{c,f}d)$ consistency checks at each node. Bessière et al. [2] give upper bounds in the number of consistency checks that algorithms nFC0–nFC5 perform at each node of the search tree. nFC0 forward checks an *n-ary* constraint when $n-2$ variables have been assigned and the $(n-1)$th variable is the current one. If $C_{c,1}$ is the number of constraints that involve the current variable and only one future variable then nFC0 performs at maximum $O(C_{c,1}d)$ consistency checks at each node. The complexities of algorithms nFC1–nFC5 depend on the levels of consistency that they enforce, and also on the complexity of the AC algorithm they use. We should note that nFC1 has the requirement that all the n-consistent tuples of the n-ary constraint have been precomputed. This, in general, increases the number of consistency checks by factor that is exponential in the constraint arity. There are cases, like the all-different constraint, where computing the allowed tuples can be done in polynomial time. However, since the number of tuples is exponential, there can be space restrictions if we want to store explicitly all the tuples of all-different constraints with high arity.

The main observation regarding the complexities of the forward checking algorithms is that there can only be a polynomial difference in the number of consistency checks performed by any two algorithms at any node. This means that the results of Sections 3 and 4 regarding exponential differences between algorithms in terms of visited nodes are also true in terms of consistency checks. However, the dominance results do not carry through to consistency checks. Examples 1 and 2 show that for a decomposable non-binary constraint satisfaction problem, FC on the binary decomposition is incomparable to algorithms nFC0 and nFC1 in terms of consistency checks.

As in [1], we count k primitive consistency checks to check if an k-tuple of a k-ary constraint is consistent. As [1] explains, this is true both in the case where the constraint is represented intensionally and the case where it is represented extensionally. If a k-ary constraint is represented intensionally as a predicate then checking if a tuple is consistent takes k operations since the predicate must consider all k values for the variables involved in the constraint. In case a k-ary constraint is represented extensionally as a boolean array then checking for the consistency of a tuple again requires k operations since we must index into an array of k dimensions. Since we count k checks for a k-ary constraint, for consistency, we count 2 checks for a binary constraint.

Example 1. Consider a ternary constraint $x < y < z$ with x having the domain $\{0, 1, 2\}$, y having the domain $\{1, 2, 3\}$ and z having the domain $\{2, 3, 4\}$. Assume a lexicographic variable ordering and a numerical value ordering. FC on the binary decomposition first assigns 0 to x and forward checks it against y. This takes 6 consistency checks (2 for each value of y). Then, 1 is assigned to y and the assignment is forward checked against z taking 6 more consistency checks. nFC0 assigns x to 0, y to 1, and then forward checks taking 9 consistency checks (3 for each value of z), which is 3 checks less than FC.

Now consider the same constraint with the variables having domains $\{0, 1\}$. FC will show insolubility in two branches, performing 12 consistency checks. nFC0 will explore 4 branches and perform 24 consistency checks in total.

Example 2. To prove that FC and nFC1 are incomparable in terms of consistency checks, consider a ternary constraint $x_1 < x_2 < x_3$, with x_1 having the domain $\{0, 1, 2\}$, x_2 having the domain $\{3, 4, 5\}$ and x_3 having the domain $\{6, 7, 8\}$. FC will take 12 consistency checks to solve the problem while nFC1 will take 18 consistency checks.

Now consider the example in the proof of Theorem 3. FC on the binary decomposition takes 28 consistency checks to prove insolubility while nFC1 takes only 18.

We can also show that algorithms nFC2–nFC5 are incomparable in consistency checks to FC on the binary decomposition and also incomparable with one another using more complicated examples.

5.2. Arc-consistency

For any non-binary constraint, specified by a predicate, GAC can be established by the best known algorithm, GAC-*schema* [4], with $O(d^k)$ worst-case complexity, where d is the maximum domain size of the variables and k is the arity of the constraint. AC can be enforced on the binary decomposition of a decomposable constraint with $O(ed^2)$ optimal worst-case complexity, where e is the number of binary constraints that the initial constraint decomposes into. We can identify a cross-over point in the size, e, of the binary decomposition. That is, if $e > d^{k-2}$ then GAC is asymptotically cheaper than AC on the binary decomposition. In practice, e is $O(k^2)$, and therefore, GAC is cheaper than AC, in the worst case, only when the arity of the constraints and the domain size of the variables are small. However, for certain types of non-binary constraints, like the all-different constraint, there exist algorithms that achieve GAC with much lower cost than $O(d^k)$.

An all-different constraint on k variables can be decomposed into a clique of $O(k^2)$ binary constraints. AC can be achieved on the decomposition of an all-different constraint in $O(k^2d^2)$ checks, when a generic AC algorithm is used. However, since we are dealing with “not equals” constraints, AC can be achieved with $O(k^2)$ worst-case complexity. This is a correction on the bound for such constraints given in [25] and it is based on the following observations: First, for a network of “not equals” constraints, an AC-3 like algorithm will revise each edge at most once. And second, for a “not equals” constraint, AC may remove a value from the domain of one of the variables if and only if the other variable has a unary domain. As a result, AC has $O(e)$ worst-case complexity, which is $O(k^2)$ for the decomposition of all-different constraints. This is better than Régin’s specialized filtering algorithm which achieves GAC on the non-binary representation with

$O(k^2d^2)$ worst-case complexity. However, as we demonstrated in Section 4, GAC on the non-binary representation is stronger than AC on the decomposition. Also, experimental results presented in the following section strongly suggest that Régin's algorithm is much more efficient than an AC algorithm. Finally, If we use GAC-schema to achieve GAC then the complexity depends on the number of allowed tuples which is $O(d!/(d-k)!)$ for one constraint. If we compare that with the complexity of Régin's algorithm it is obvious that GAC-schema is inferior. Even for ternary constraints the difference is substantial as GAC-schema would perform $O(d^3)$ consistency checks on one constraint, compared to the $O(d^2)$ checks of Régin's algorithm.

6. Experimental results

To demonstrate the practical relevance of these theoretical results, we ran experiments in two benchmark domains. We compare the impact of enforcing GAC on the non-binary representation with enforcing AC on the binary decomposition.

6.1. Quasigroup completion

Gomes and Selman have proposed random quasigroup completion problems as a benchmark that combines some of the best features of random and structured problems [13]. A quasigroup is an ordered pair $(Q, \cdot)$, where Q is a set and $(\cdot)$ is a binary operation on Q such that the equations $a \cdot x = b$ and $y \cdot a = b$ are uniquely solvable for every pair of elements a, b in Q. The constraints on a quasigroup are such that its multiplication table forms a Latin square. That is, each element occurs exactly once in every row or column of its n by n multiplication table. The order n of the quasigroup is the cardinality of the set Q.

Quasigroup completion problem is the NP-complete problem of determining whether the remaining entries of a partially filled n by n table can be filled in such a way that a full quasigroup multiplication table is obtained. A quasigroup completion problem can be represented as a CSP with n^2 variables, each with a domain of size n. The constraints can be represented by $2n$ all-different n-ary constraints (one for each row and column). Alternatively, we can use binary "not equal to" constraints, giving a constraint graph with $2n$ cliques of size n. For quasigroup completion problems, there is a phase transition from a region where almost all problems are soluble to a region where almost all problems are insoluble as we vary the percentage of variables preassigned. The solution cost peaks around the transition, with approximately 42% of variables preassigned [13].

The constraints in a quasigroup completion problem are triangle preserving and decompose into cliques. This means that GAC can achieve very high levels of consistency. In fact, as shown in [24], GAC on quasigroup problems is equivalent to NIC on their binary decomposition. The following example demonstrates that enforcing GAC on the all-different constraints of a quasigroup completion problem is strictly stronger than enforcing AC on the not-equals constraints of the binary decomposition. We have a 3×3 quasigroup where each variable/square has the domain $\{r, g, b\}$ and there are two squares preassigned to r.

$\{r\}$	$\{r, g, b\}$	$\{r, g, b\}$
$\{r, g, b\}$	$\{r\}$	$\{r, g, b\}$
$\{r, g, b\}$	$\{r, g, b\}$	$\{r, g, b\}$

Enforcing AC on the binary decomposition gives,

$\{r\}$	$\{g, b\}$	$\{g, b\}$
$\{g, b\}$	$\{r\}$	$\{g, b\}$
$\{g, b\}$	$\{g, b\}$	$\{r, g, b\}$

However, enforcing GAC on the all-different constraints of the n-ary representation filters out two more values in the bottom right square,

$\{r\}$	$\{g, b\}$	$\{g, b\}$
$\{g, b\}$	$\{r\}$	$\{g, b\}$
$\{g, b\}$	$\{g, b\}$	$\{r\}$

Table 1
Percentiles in branches searched to complete a quasigroup of order 10 using algorithms that maintain either arc-consistency on the binary decomposition (MAC) or generalized arc-consistency on the non-binary representation (MGAC). $*$ means that the instance was abandoned after 10,000 branches. 100 problems were solved at each data point

p	MAC		MGAC	
	100th	90th	100th	90th
10	163	1	1	1
20	*	1	1	1
30	*	15	2	1
35	*	124	2	1
40	*	1726	2	1
42	*	*	2	1
45	*	*	2	1
48	*	2771	2	1
50	5692	1263	2	1
55	324	71	2	1
60	47	7	1	1
70	2	2	1	1
80	2	2	1	1
90	2	2	1	1

For our experiments we encoded the problem in ILOG Solver, a C++ constraint toolkit which includes Régin's algorithm for maintaining GAC on all-different constraints [20]. We used the Brelaz heuristic for variable selection (an in [13]) and Geelen's promise heuristic for value ordering (as in [17]). Although this value ordering is expensive to compute in general, the structure of quasigroup problems reduces the cost significantly. Gomes et al. observed that search costs to solve random quasigroup completion problems can be modeled by a "heavy-tailed" distribution [14]. We therefore focus on the higher percentiles. Table 1 gives branches explored to complete an order 10 quasigroup with $p\%$ of entries preassigned, maintaining either AC on the binary representation or GAC on the all-different constraints. We see a very significant advantage in enforcing GAC. With a random value ordering, the worst case for MGAC was also 2 branches. This advantage was also reflected on the CPU times used to process the instances. For the harder instances, MGAC took a couple of seconds to find a solution or prove insolubility while MAC was cut off after more than twenty times that time.

Table 2 shows that, as we increase problem size, almost all the problems remain trivial to solve. The only exception was a single order 25 problem with 42% of its variables preassigned. Search was abandoned at the cuttoff limit of 10,000 branches. Apart from this, all instances were solved in less than 5 branches. This is a significant improvement over the results of [14] where, despite the use of random restarts to enhance performance, problems of order 25 were too expensive to solve, especially at the phase transition. We should note that as the size of the problems increased there was an exponential growth in the difference between the CPU times used by MGAC and MAC. For many instances of more than order 10 size, MAC took hours to find a solution or prove that none exists, when no cutoff limit was imposed.

Table 2
Percentiles in branches explored to complete quasigroups of order 10, 15, 20 and 25 using MGAC

p	Order 10		Order 15		Order 20		Order 25	
	100th	90th	100th	90th	100th	90th	100th	90th
10	1	1	1	1	1	1	1	1
20	1	1	1	1	1	1	1	1
30	2	1	1	1	2	1	2	1
40	2	1	2	1	2	1	2	1
42	2	1	2	1	2	1	*	1
45	2	1	3	1	3	1	3	1
48	2	1	2	1	2	1	2	1
50	2	1	2	1	2	1	3	1
60	1	1	1	1	4	1	1	1
70	1	1	1	1	1	1	1	1
80	1	1	1	1	1	1	1	1
90	1	1	1	1	1	1	1	1

6.2. Quasigroup existence

A variety of automated reasoning programs have been used to answer open questions in finite mathematics about the existence of quasigroups with particular properties [11]. Is GAC useful on these problems? We follow [11] and look at the so-called QG3, QG4, QG5, QG6 and QG7 class of problems. For example, the QG5 problems concern the existence of idempotent quasigroups (those in which $a \cdot a = a$ for each element a) in which $(ba \cdot b)b = a$. For the definition of the other problems, see [11]. In these problems, the structure of the constraint graph is disturbed by additional non-binary constraints. These reduce the level of consistency achieved compared to quasigroup completion problems. Nevertheless, GAC significantly prunes the search space and reduces run times.

To solve these problems, we again use the Solver toolkit, maintaining either GAC on the all-different constraints, or AC on their binary representation, and the fail-first heuristic for variable ordering. The additional non-binary constraints were handled using ILOG Solver's

Table 3
Branches explored to solve a variety of quasigroup existence problems using either an algorithm that maintains AC on the binary representation (MAC) or an algorithm that maintains GAC on the all-different constraints (MGAC)

Order	QG3		QG4		QG5	
	MAC	MGAC	MAC	MGAC	MAC	MGAC
6	7	4	6	4	0	0
7	64	48	59	42	5	3
8	1,511	821	1,227	707	15	10
9	65,001	31,274	88,460	40,582	30	19
10	–	–	–	–	268	74
11	–	–	–	–	1,107	292
12	–	–	–	–	6,832	910
13	–	–	–	–	>1,000,000	27,265

Order	QG6		QG7	
	MAC	MGAC	MAC	MGAC
6	0	0	6	4
7	5	2	67	39
8	9	3	415	314
9	36	26	4,837	4,211
10	199	167	94,433	80,677
11	2,221	1,876	–	–
12	42,248	34,741	–	–
13	–	4,730,320	–	–

Table 4
Number of models found and branches explored on QG5 problems by a variety of different programs

Order	Models	Branches					
		MGTP	FINDER	MACE	SATO	SEM	MGAC
7	3	9	3	4	5	6	3
8	1	34	13	8	8	11	10
9	0	239	46	14	11	29	19
10	0	7,026	341	37	21	250	74
11	5	51,904	1,728	112	43	1,231	292
12	0	2,749,676	11,047	369	277	8,636	1,156

standard bound consistency propagation. To eliminate some of the symmetric models, as in [11], we added the constraint that $a \cdot n \geqslant a - 1$ for every element a. Table 3 demonstrates the benefits of an algorithm that maintains GAC over one that maintains AC. In QG3 and QG4, an algorithm that maintains AC explores twice as many branches as an algorithm that maintains GAC, in QG5 the difference is orders of magnitude, whilst there is only a slight difference in QG6 and QG7. An algorithm that maintains GAC dominates in terms of CPU time as well as in terms of explored branches, although the difference in CPU times is not as large since enforcing GAC is a more expensive operation than enforcing AC. It would be interesting to identify the features of QG5 that gives an algorithm that maintains GAC such an advantage over MAC, and those of QG6 and QG7 that lessen this advantage.

We now compare our results with those of FINDER [23], MACE [16], MGTP [11], SATO [26], and SEM [27]. Table 4 shows that our simple Solver code, by maintaining GAC, outperforms MGTP and FINDER by orders of magnitude, and explores less branches than SEM. SEM, MACE, and SATO have very sophisticated branching heuristics and complex rules for the elimination of symmetric models that are far more powerful than the symmetry breaking constraint we use [27]. It is therefore impressive that our simple Solver program is competitive with well-developed systems like SEM and SATO.

To conclude, despite the addition of non-binary constraints that disturb the structure of the constraint graph, maintaining GAC significantly reduces search and run times on quasigroup existence problems. We conjecture that the performance of SEM and SATO could be improved by the addition of a specialized procedure to maintain GAC on the all-different constraints.

7. Related work

Montanari looked at the approximation of non-binary constraints by binary constraints on the same set of variables [19]. He constructs a “minimal network” of binary constraints by projecting each non-binary constraint onto the pairs of variables it contains. The minimal network has a set of solutions that is a superset of the set of solutions of the original non-binary constraints. It is the best upper bound to the set of solutions of the non-binary constraints as no other binary approximation has fewer solutions. The minimal network of a decomposable non-binary constraint is simply the binary decomposition.

Dechter has studied the representation of non-binary constraints by binary constraints with additional (hidden) variables [6]. She identifies a trade-off between the number of additional variables required and the size of their domains. In particular, any non-binary constraint can be expressed by binary constraints with the addition of hidden variables with three or more values. By comparison, with domains of size 2, additional variables do not improve the expressive power. Bacchus and van Beek compared the forward checking algorithm, nFC0 on non-binary constraints with the forward checking algorithm FC applied to binary encodings that introduce extra (hidden) variables [1]. They showed that FC and nFC0 are incomparable on such constraints both in visited nodes and consistency checks.

8. Conclusions

We have performed a detailed theoretical comparison of the effects of binary and non-binary constraint propagation on decomposable non-binary constraints. We proved that the number of nodes visited by the forward checking algorithm, FC applied to the binary decomposition lies between the number visited by the generalized forward checking algorithms, nFC1 and nFC0 when applied to the non-binary constraints (assuming equivalent variable and value ordering). We also proved that generalized arc-consistency on decomposable constraints is strictly stronger than arc-consistency on the binary decomposition. Indeed, under a simple restriction, it is strictly stronger than path inverse consistency on the binary decomposition. By generalizing the arguments of [15], these results show that a search algorithm that maintains generalized arc-consistency on decomposable constraints strictly dominates a search algorithm that maintains arc-consistency on the binary decomposition, which itself strictly dominates the forward checking algorithm, FC and any of its generalizations, nFC0 to nFC5.

We corrected a result of [24] that claims that neighbourhood inverse consistency on the binary decomposition is strictly stronger than generalized arc-consistency. In general, neighbourhood inverse consistency on the binary decomposition is incomparable to generalized arc-consistency. However, we identify a simple condition under which neighbourhood inverse consistency on the binary decomposition is guaranteed to be strictly stronger than generalized arc-consistency. We also defined a class of decomposable non-binary constraints on which generalized arc-consistency collapses down onto AC on the binary decomposition. We also made an asymptotic analysis on the number of consistency checks required to achieve certain levels of consistency. The high levels of consistency achieved by GAC on decomposable constraints compared to AC on the binary decomposition result in less search for an algorithm that maintains GAC compared to an algorithm that maintains AC. We demonstrated the practical importance of this result by running experiments on benchmark domains. We showed that the dominance of GAC over AC has a significant effect on problems with decomposable constraints such as quasigroup completion and quasigroup existence.

What general lessons can be learnt from this study? First, the representation of problems can have a very large impact on the efficiency of search. Our results show the comparison of different representations is very complex, even when restricted to a limited set of consistency properties and algorithms. The study of different representations thus deserves further work, both theoretical and practical. Second, a non-binary representation can offer considerable advantages over a binary representation. Decomposing non-binary constraints into binary constraints can significantly reduce the level of consistency achieved by our constraint propagation techniques.

References

[1] F. Bacchus, P. van Beek, On the conversion between non-binary and binary constraint satisfaction problems, in: Proc. AAAI-98, Madison, WI, 1998.

[2] C. Bessière, P. Meseguer, E. Freuder, J. Larrosa, On forward checking for non-binary constraint satisfaction, in: Proc. 5th International Conference on Principles and Practice of Constraint Programming (CP-99), Alexandria, VA, 1999, pp. 88–102.

[3] C. Bessière, J.C. Régin, Mac and combined heuristics: Two reasons to forsake fc (and cbj?), in: Proc. CP-96, Cambridge, MA, 1996, pp. 61–75.

[4] C. Bessière, J.C. Régin, Arc consistency for general constraint networks: Preliminary results, in: Proc. IJCAI-97, Nagoya, Japan, 1997, pp. 398–404.

[5] R. Debruyne, C. Bessière, Some practicable filtering techniques for the constraint satisfaction problem, in: Proc. IJCAI-97, Nagoya, Japan, 1997, pp. 412–417.

[6] R. Dechter, On the expressiveness of networks with hidden variables, in: Proc. AAAI-90, Boston, MA, 1990, pp. 555–562.

[7] R. Dechter, J. Pearl, Tree clustering for constraint networks, Artificial Intelligence 38 (1989) 353–366.

[8] E. Freuder, A sufficient condition for backtrack-free search, J. ACM 29 (1982) 24–32.

[9] E. Freuder, A sufficient condition for backtrack-bounded search, J. ACM 32 (4) (1985) 755–761.

[10] E. Freuder, C.D. Elfe, Neighborhood inverse consistency preprocessing, in: Proc. AAAI-96, Portland, OR, 1996, pp. 202–208.

[11] M. Fujita, J. Slaney, F. Bennett, Automatic generation of some results in finite algebra, in: Proc. IJCAI-93, Chambéry, France, 1993, pp. 52–57.

[12] J. Gaschnig, Performance measurement and analysis of certain search algorithms, Technical Report CMU-CS-79-124, Carnegie-Mellon University, Pittsburgh, PA, 1979.

[13] C.P. Gomes, B. Selman, Problem structure in the presence of perturbations, in: Proc. AAAI-97, Providence, RI, 1997, pp. 221–226.

[14] C.P. Gomes, B. Selman, N. Crato, Heavy-tailed probability distributions in combinatorial search, in: Proc. CP-97, Vienna, Austria, 1997, pp. 121–135.

[15] G. Kondrak, P. van Beek, A theoretical evaluation of selected backtracking algorithms, Artificial Intelligence 89 (1997) 365–387.

[16] W. McCune, A Davis–Putnam program and its application to finite first-order model search: Quasigroup existence problems, Technical Report ANL/MCS-TM-194, Argonne National Laboratory, 1994. Available from http://www-unix.mcs.anl.gov/AR/mace/.

[17] P. Meseguer, T. Walsh, Interleaved and discrepancy based search, in: Proc. ECAI-98, Brighton, UK, 1998, pp. 239–243.

[18] R. Mohr, G. Masini, Good old discrete relaxation, in: Proc. ECAI-88, Munich, Germany, 1988, pp. 651–656.

[19] U. Montanari, Networks of constraints: Fundamental properties and applications to picture processing, Inform. Sci. 7 (1974) 95–132.

[20] J.F. Puget, A C++ implementation of CLP, Technical Report 94-01, ILOG S.A., Gentilly, France, 1994.

[21] J.C. Régin, A filtering algorithm for constraints of difference in csps, in: Proc. AAAI-94, Seattle, WA, 1994, pp. 362–367.

[22] J.C. Régin, Generalized arc consistency for global cardinality constraint, in: Proc. AAAI-96, Portland, OR, 1996, pp. 209–215.

[23] J. Slaney, FINDER, Finite domain enumerator: Notes and guide, Technical Report TR-ARP-1/92, Automated Reasoning Program, Australian National University, 1992.

[24] K. Stergiou, T. Walsh, The difference all-difference makes, in: Proc. IJCAI-99, Stockholm, Sweden, 1999, pp. 414–419.

[25] P. Van Hentenryck, Y. Deville, C. Teng, A generic arc consistency algorithm and its specializations, Artificial Intelligence 57 (1992) 291–321.

[26] H. Zhang, M. Stickel, Implementing the Davis–Putnam algorithm by tries, Technical Report, University of Iowa, 1994.

[27] J. Zhang, H. Zhang, SEM: A system for enumerating models, in: Proc. IJCAI-95, Montreal, Quebec, Vol. 1, 1995, pp. 298–303.