Decomposable Constraints

Ian Gent[1], Kostas Stergiou[2], and Toby Walsh[3]

[1] University of St Andrews, St Andrews, Scotland. ipg@dcs.st-and.ac.uk
[2] University of Strathclyde, Glasgow, Scotland. ks@cs.strath.ac.uk
[3] University of York, York, England. tw@cs.york.ac.uk

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 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 also analyze the cost, in consistency checks, required to achieve certain levels of 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 satisfaction problems can be naturally and efficiently modelled using non-binary constraints like the "all-different" and "global cardinality" constraints [15,16,17]. Certain classes of these non-binary constraints are "network decomposable" [14,6] 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. Not all non-binary constraints are decomposable into binary constraints on the same set of variables. For example, the parity constraint $\text{even}(x_1 + x_2 + x_3)$ cannot be represented as a binary constraint satisfaction problem without the introduction of additional variables.

In this paper, we compare theoretically the levels of consistency which are achieved on non-binary constraints to those achieved on their binary decomposition. We extend the results of [17] and include 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

K.R. Apt et al. (Eds.): New Trends in Constraints, LNAI 1865, pp. 134–149, 2000.

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 [17] 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. A few of the results presented here have appeared in [17] but are repeated to make this paper a more complete survey.

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 discuss related work, and finally, in Section 7, 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. Each k-ary constraint $c \in C$ is defined over a set of variables $(x_1, \ldots x_k)$ by the subset of the cartesian product $D_1 \times \ldots D_k$ which are consistent values. A solution 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 $(i, j)-$*consistent* iff it has non-empty domains and any consistent instantiation of i variables can be extended to a consistent instantiation involving 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 $(j, 1)$-consistent for $j \leq 2$. 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 variable assigned to a value 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 [13].

Following [5], we call a consistency property A *stronger* than B ($A \geq B$) 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 implies B and vice versa. The following identities summarize results from [5] and elsewhere: strong PC $>$ SAC $>$ PIC $>$ RPC $>$ AC, NIC $>$ PIC, NIC $\sim$ SAC, and NIC $\sim$ strong PC.

Backtracking based algorithms perform depth-first search in a tree of variable assignments. Each node of the tree corresponds to a different set of assignments. Leaf nodes in a search tree are also called branches. 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 less than $k-1$ variables instantiated. As required, this reduces to 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 nodes in the search tree. For example, the *maintaining arc-consistency* algorithm (MAC) enforces AC at each node in the search tree [11]. 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. We say that algorithm A *dominates* algorithm B ($A \geq B$) 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.

Theorem 2. *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 $x_1 < x_2 < \ldots < x_n$ 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 $1 < i \leq n$ and the value assigned to i is 1 then forward checking discovers a domain wipeout for x_{i-1}. If $1 \leq i < n$ 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 3. *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 between 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 an all-different constraint on four variables each with the same domain of three elements. FC shows that the binary decomposition is insoluble in 6 branches whatever the variable and value ordering. By comparison, nFC1 take just 3 branches. □

We can generalize the example in the last proof to show 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.

Theorem 4. *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 $(1 \leq i \leq n)$ 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 Figure 1.

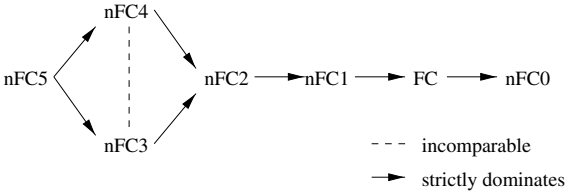

Fig. 1. The performance of the forward checking algorithm, FC on the binary decomposition of a set of decomposable non-binary constraints compared to the various generalizations of forward checking, nFC0 to nFC5 applied to the non-binary constraints.

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, [3,17]). In this section, we characterize the level of consistency achieved by (generalized) AC on decomposable constraints. The following theorem (from [17]) puts a lower bound on the level of consistency achieved by GAC on decomposable constraints with respect to the binary decomposition.

Theorem 5. *Generalized arc-consistency on decomposable constraints is strictly stronger than arc-consistency on the binary decomposition.*

Proof. See [17]. □

As we show later on in this section, this lower bound is strict since we can exhibit a large class of problems on which GAC is equivalent to AC on the binary decomposition. In [17], we claimed that NIC (on the binary decomposition) was an upper bound on the level of consistency achieved by GAC on decomposable non-binary constraints. This was wrong as the following theorem shows (see Theorem 8 for one condition under which NIC becomes strictly stronger than GAC).

Theorem 6. *Generalized arc-consistency on decomposable constraints is incomparable to neighbourhood inverse 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 generalized arc-consistent, but enforcing neighbourhood inverse consistency shows that it is insoluble.

Consider the following 2-colouring problem. We have 5 variables, x_1 to x_5 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. This problem is neighbourhood inverse consistent but enforcing GAC on the non-binary representation shows that the problem 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.

Theorem 7. *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) $k-1$ variable's 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 second upper bound can be given when the non-binary constraints 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 8. *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.

To prove strictness, consider again the problem with three all-different constraints from the proof of Theorem 6. 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 strict since we can exhibit a class of problems in which the non-binary constraints decompose into cliques of binary constraint and on which GAC is equivalent to NIC on the binary decomposition.

GAC on decomposable constraints is incomparable to all levels of consistency between strong path-consistency and restricted path-consistency on the binary decomposition [17].

Theorem 9. *Generalized arc-consistency on decomposable constraints is incomparable to strong path-consistency, to singleton arc-consistency, to path inverse consistency, and to restricted path-consistency on the binary decomposition.*

Proof. See [17]. □

These results are summarized in Figure 2.

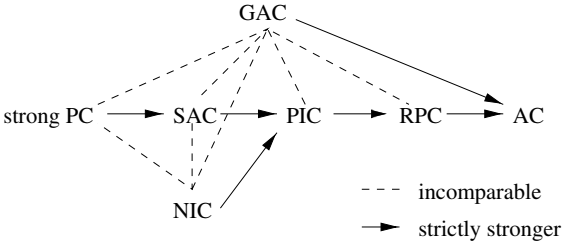

Fig. 2. The consistency of GAC on a set of decomposable non-binary constraints compared to various consistency techniques stronger than or equal to AC on the binary decomposition.

Not surprisingly an algorithm that maintains GAC on decomposable constraints also strictly dominates the forward checking algorithm, FC applied to the binary decomposition (as well strictly dominating any of the generalizations of FC to non-binary constraints).

Theorem 10. *For a decomposable non-binary constraint satisfaction problem, an algorithm that maintains generalized arc-consistency strictly dominates the forward checking algorithm, FC on the binary decomposition, as well as strictly dominating any of the generalized algorithms, nFC0 to nFC5.*

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.

Theorem 11. *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

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

Proof. ($\Rightarrow$) 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.

Theorem 13. *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 5, 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 [17]) 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 all triangles of variables in the constraint graph of the binary decomposition occur together in non-binary constraints. For example, an all-different constraint is triangle preserving as it decomposes into a clique of binary not-equals constraints. 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 14. *On a triangle preserving set of constraints, generalized arc-consistency is strictly stronger than path inverse consistency on the binary decomposition.*

Proof. See [17]. □

A corollary of this result 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 15. *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. See [17]. □

These results are summarized in Figure 3.

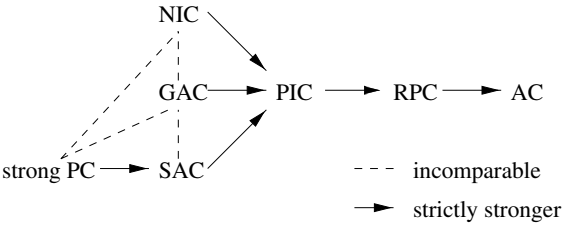

Fig. 3. The consistency of GAC on a triangle preserving set of non-binary constraints compared to various consistency techniques stronger than or equal to AC on the binary decomposition.

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. [2] gives 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, adds an exponential, to the arity of the constraint, number of consistency checks when the constraint is not defined by the allowed tuples. 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 explicitly store 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. The following examples 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 n primitive consistency checks to check if an n-tuple of an $n-$ary constraint is consistent, which means that 2 checks are counted 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 C, 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 asympotically 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 [18] 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 Regin'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 Regin'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 $O(\frac{d!}{(d-k)!})$ for one constraint. If we compare that with the complexity of Regin'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 Regin's algorithm.

6 Related Work

Montanari looked at the approximation of non-binary constraints by binary constraints on the same set of variables [14]. 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].

7 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 [12], 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 [17] 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.

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 that 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 constraint into binary constraints can significantly reduce the level of consistency achieved by our constraint propagation techniques.

Acknowledgements. The third author is supported by EPSRC award GR/L/24014. The authors are members of the APES research group (http://www.cs.strath.ac.uk/~apes) and wish to thank the other members for their comments and feedback.

References

1. F. Bacchus and P. van Beek. On the conversion between non-binary and binary constraint satisfaction problems. In *Proceedings of 15th National Conference on Artificial Intelligence*, pages 311–318. AAAI Press/The MIT Press, 1998.

2. C. Bessiere, P. Meseguer, E.C. Freuder, and J. Larrosa. On forward checking for non-binary constraint satisfaction. In *Proceedings of IJCAI-99 Workshop on Non-binary constraints*. International Joint Conference on Artificial Intelligence, 1999.
3. C. Bessière and J-C. Régin. MAC and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problems. In E.C. Freuder, editor, *Proceedings of Second International Conference on Principles and Practice of Constraint Programming (CP96)*, pages 61–75. Springer, 1996.
4. C. Bessière and J.C. Régin. Arc consistency for general constraint networks: Preliminary results. In *Proceedings IJCAI-97*, pages 398–404, 1997.
5. R. Debruyne and C. Bessière. Some practicable filtering techniques for the constraint satisfaction problem. In *Proceedings of the 15th IJCAI*, pages 412–417. International Joint Conference on Artificial Intelligence, 1997.
6. R. Dechter. On the expressiveness of networks with hidden variables. In *Proceedings of the 8th National Conference on AI*, pages 555–562. American Association for Artificial Intelligence, 1990.
7. R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34:1–38, 1988.
8. E. Freuder. A sufficient condition for backtrack-free search. *Journal of the Association for Computing Machinery*, 29(1):24–32, 1982.
9. E. Freuder. A sufficient condition for backtrack-bounded search. *Journal of the Association for Computing Machinery*, 32(4):755–761, 1985.
10. E. Freuder and C.D. Elfe. Neighborhood inverse consistency preprocessing. In *Proceedings of the 12th National Conference on AI*, pages 202–208. American Association for Artificial Intelligence, 1996.
11. J. Gaschnig. Performance measurement and analysis of certain search algorithms. Technical report CMU-CS-79-124, Carnegie-Mellon University, 1979. PhD thesis.
12. G. Kondrak and P. van Beek. A Theoretical Evaluation of Selected Backtracking Algorithms. *Artificial Intelligence*, 89:365–387, 1997.
13. R. Mohr and G. Masini. Good old discrete relaxation. In *Proceedings of the European Conference on Artificial Intelligence (ECAI-88)*, pages 651–656, 1988.
14. U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Science*, 7:95–132, 1974.
15. J-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the 12th National Conference on AI*, pages 362–367. American Association for Artificial Intelligence, 1994.
16. J-C. Régin. Generalized arc consistency for global cardinality constraints. In *Proceedings of the 13th National Conference on AI*, pages 209–215. American Association for Artificial Intelligence, 1996.
17. K. Stergiou and T. Walsh. The difference all-difference makes. In *Proceedings of 16th IJCAI*. International Joint Conference on Artificial Intelligence, 1999.
18. P. Van Hentenryck, Y. Deville, and C. Teng. A Generic Arc Consistency Algorithm and its Specializations. *Artificial Intelligence*, 57:291–321, 1992.