On Algorithms for Decomposable Constraints

Kostas Stergiou

Glasgow University, Glasgow, Scotland. kostas@dcs.strath.ac.uk

Abstract. Non-binary constraints are present in many real-world constraint satisfaction problems. Certain classes of these constraints, like the all-different constraint, are "decomposable". That is, they can be represented by binary constraints on the same set of variables. For example, a non-binary all-different constraint can be decomposed into a clique of binary not-equals constraints. In this paper we make a theoretical analysis of local consistency and search algorithms for decomposable constraints. First, we prove a new lower bound for the worst-case time complexity of arc consistency on binary not-equals constraints. We show that the complexity is O(e), where e is the number of constraints, instead of O(ed), with d being the domain size, as previously known. Then, we compare theoretically local consistency and search algorithms that operate on the non-binary representation of decomposable constraints to their counterparts for the binary decomposition. We also extend previous results on arc consistency algorithms to the case of singleton arc consistency.

1 Introduction

Many problems in the real world can be efficiently modelled as constraint satisfaction problems and solved using constraint programming techniques. Some examples are scheduling, planning, machine vision, temporal reasoning, car sequencing, vehicle routing, belief maintainance, and frequency allocation. Most of these problems can be naturally modelled using n-ary (or non-binary) constraints like the "all-different" and "global cardinality" constraints. Certain classes of these non-binary constraints are *decomposable* [6] as they can be represented by binary constraints on the same set of variables. For example, an all-different constraint can be decomposed into a clique of binary not-equals constraints. As a second example, a monotonicity constraint can be decomposed 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 constraint $(x_1 + x_2 < x_3)$ cannot be represented by binary constraints without the introduction of additional variables.

In this paper we make a theoretical analysis of some local consistency and search algorithms for decomposable constraints. In Section 2 we introduce the neccessary definitions from constraint satisfaction. In Section 3 we prove a new lower bound for the worst-case time complexity of arc consistency on binary not-equals constraints. We show that the complexity is O(e), where e is the number of constraints, instead of O(ed), with d being the domain size, as previously

I.P. Vlahavas and C.D. Spyropoulos (Eds.): SETN 2002, LNAI 2308, pp. 65–71, 2002.

known. This new complexity bound is lower than the corresponding complexity bound for the non-binary representation of not-equal constraints (i.e. the all-different constraint). However, as we discuss in Section 4, this does not mean that the binary decomposition is more efficient than the non-binary representation. In Section 4 we compare theoretically local consistency and search algorithms that operate on the non-binary representation of decomposable constraints to their counterparts for the binary decomposition. We show that the non-binary representation is more powerful than the binary one, and this makes up for the worse complexity bound. Finally, we extend previous results on arc consistency algorithms to the case of singleton arc consistency.

2 Formal Background

A *constraint satisfaction problem* (CSP) $\mathcal{P}$ is defined by a triple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$. $\mathcal{X}$ is a set of n variables. Each variable $x_i \in \mathcal{X}$ takes values from a domain $D_i \in \mathcal{D}$. $\mathcal{C}$ is a set of e constraints. Each k-ary constraint is defined over an ordered set of variables $\{x_1, \ldots, x_k\}$ by a subset of the Cartesian product $D_1 \times \ldots \times D_k$ that specifies the set of allowed value combinations (tuples). A constraint can be either defined extensionally by the set of allowed tuples or intensionally by a predicate or arithmetic function.

A value a in the domain D of variable x is *consistent* with a constraint c if x is not included in the variables of the constraint, or if it is included and there exists a valid tuple τ in c where $x = a$. In the latter case we say that τ is a *support* for a in c. Checking whether a tuple is a support for a variable value pair (x, a) is called a *consistency check*. A solution to a CSP is an assignment of values to variables that is consistent with all constraints. Many lesser levels of consistency (usually called local consistencies) have been defined for binary constraint satisfaction problems. 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. 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. 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. Some of the above local consistencies have been extended to the case of non-binary CSPs. The generalizations of AC and SAC to non-binary CSPs are called generalized AC (GAC) and singleton generalized AC (SGAC) respectively. For example, a (non-binary) CSP is generalized arc consistent 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.

Many search 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 which ensures that all values of the uninstantiated variables are consistent with the most recent variable instantiation. Various generalizations of FC for non-binary constraints have been proposed. These algorithms, starting from nFC0 up to nFC5 enforce increasingly higher levels of consistency (see [1]). 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. 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.

Following [2], we call a local consistency property A *stronger* than B iff for any problem enforcing A deletes at least the same values as B, and *strictly stronger* iff it is stronger and there is at least one problem where A deletes more values than B. We call A *equivalent* to B iff they delete the same values for all problems. Similarly, we call a search algorithm A stronger than a search algorithm B iff for every problem A visits at most the same search tree nodes as B, and strictly stronger iff it is stronger and there is at least one problem where A visits less nodes than B. A is equivalent to B iff they visit the same nodes for all problems.

3 Arc Consistency on Binary Not-equals Constraints

In this section we correct a result given in [10] regarding the complexity of achieving AC in a network of binary not-equals ($\neq$) constraints. In [10] it is claimed that AC can be optimally achieved with O(ed) worst-case time complexity, where e is the number of constraints and d the domain size of the variables. We will describe an algorithm that achieves AC in networks of binary not-equals constraints with O(e) worst-case complexity. In [10] it is claimed that $O(ed)$ is the optimal worst-case complexity of AC for any subclass of constraints, since, as they say, "it is reasonable to assume that we need to check each value in each domain at least once". We show that this is not the case for not-equals constraints, and as a result, the worst-case complexity is actually $O(e)$.

First, we start from the observation that for a not-equals constraint between variables x_i and x_j AC may remove a value from the domain of variable x_i or x_j only if the other variable has a unary domain. This is also mentioned in [10]. In general, a not-equals constraint between two variables x_i and x_j with domain sizes of more than one is always AC, since every value in the domain of x_i will have a support in the domain of x_j, and vice versa. Whenever a variable x_i is instantiated to a value a, AC will remove a from the domains of the variables adjacent to x_i and will only continue the propagation if some other variable has only one value in its domain. In other words, an optimal implementation of an AC algorithm will never process edges between variables that both have non-unary domains.

We now describe the steps of the AC algorithm with $O(e)$ worst-case complexity.

- For each edge (x_i, x_j), such that x_i has a unary domain mark the edge and put it in a queue.
- Extract the first edge (x_i, x_j) from the queue. Assuming that a is the unique value in the domain of x_i, if a is present in the domain of x_j, remove it.
- If the domain of x_j becomes empty, stop. The network is inconsistent.
- If x_j is left with a unary domain mark all the unmarked edges connected to x_j and put them in the queue. Checking whether a variable has only one value in its domain can be done in constant time through careful implementation. For example, using a flag that is set to 1 when the domain becomes singleton.
- Take the next edge out of the queue and continue in the same way. The algorithm will stop when a domain wipe-out occurs or the queue becomes empty.

Having described the algorithm, we can now prove the following proposition.

Proposition 1. *Arc consistency can be achieved with O(e) worst-case time complexity on a network of e binary not-equals constraints.*

Proof. We need to show that each of the e edges will be processed at most once, and that the processing can be done in constant time. Consider a not-equals constraint between variable x_i with the singleton domain $\{a\}$ and x_j with the domain $\{a, \ldots, z\}$. When this edge is extracted from the queue, AC will remove a from the domain of x_j. If at some point later x_j is left with a singleton domain, the algorithm we described will insert all edges that involve x_j into the queue. However, there is no point in including edge (x_i, x_j), since AC cannot remove a value from either variable. This was done earlier when (x_i, x_j) was first processed. Thus, an edge that has been processed once needs not to be processed again. This means that each of the e edges is made AC at most once. As mentioned, making an edge AC is equivalent to removing a value from a domain (if present) and checking whether the resulting domain has size of 0, 1 or more, both of which can be done in constant time with careful implementation. Therefore, AC can be achieved with $O(e)$ worst-case complexity.

As a result, if we have an all-different constraint on k variables the decomposition of this constraint into binary not-equals constraints can be made arc consistent with $O(k^2)$ worst-case complexity. This is a significant gain over the $O(k^2d^2)$ complexity of a generic optimal AC algorithm like AC-7. Also, the $O(k^2)$ complexity of AC is significantly lower than the $O(k^2d^2)$ complexity of Regin's algorithm for all-different constraints. However, as we discuss in the next section, this does not mean that we should decompose all-different constraints into binary.

4 Local Consistency and Search Algorithms

In this section we review some results from [9,3] where local consistency and search algorithms for non-binary decomposable constraints are compared to the

corresponding algorithms for the binary decomposition. [3] first compared the level of consistency achieved by FC on the decomposition to the levels of consistency achieved by the various generalizations of FC on the constraints of the n-ary representation. A lower bound on the performance of FC applied to the binary decomposition was first identified. It was proved that for a decomposable non-binary constraint satisfaction problem, the forward checking algorithm FC on the binary decomposition is strictly stronger than the generalized algorithm nFC0. [3] also gives a simple upper bound on the performance of FC on the binary decomposition. For a decomposable non-binary constraint satisfaction problem the generalized algorithm nFC1 is strictly stronger than FC on the binary decomposition.

[3] also investigated and compared the pruning efficiency of AC on the binary decomposition and GAC on the n-ary representation of decomposable constraints. They first gave a lower bound on the level of consistency achieved by GAC on decomposable constraints with respect to the binary decomposition. It was proved that GAC on decomposable constraints is strictly stronger than AC on the binary decomposition. As a result, an algorithm that maintains GAC on the non-binary representation of a set of decomposable constraints is strictly stronger than an algorithm that maintains AC on the binary decomposition. So, although we showed that AC on the decomposition can be achieved faster that GAC on the non-binary representation, it does not pay off because it is a weaker level of consistency. This is also demonstrated by the empirical results presented in [9,3].

Having established that GAC is stronger than AC, [3] compared GAC to stronger levels of consistency than AC in the binary decomposition. They showed that, in the general case, NIC on the binary decomposition, as well as all the levels of consistency between strong PC and RPC, are incomparable to generalized arc-consistency.

Another result from [3] is that an algorithm that maintains GAC on decomposable constraints strictly is stronger than the strongest generalized forward checking algorithm nFC5. Naturally, this means that it is also stronger than algorithms nFC0-nFC4 and also FC applied to the binary decomposition.

5 Singleton Arc Consistency

We now extend the analysis of [3] to the case of SAC and its generalization SGAC. As shown in [7], these very high levels of consistency can be very effective in certain classes of CSPs. First we prove that SGAC on on the non-binary representation is strictly stronger than SAC on the binary decomposition.

Theorem 1. *Singleton generalized arc consistency on decomposable constraints is strictly stronger than singleton arc consistency on the binary decomposition.*

Proof. SGAC ensures that every variable in the problem can be instantiated to any of the values in its domain and the resulting problem will be GAC. Since GAC on decomposable constraints is strictly stronger than AC on the binary

decomposition, for any instantiation of a variable, the binary decomposition of the resulting problem will be AC. Hence, the binary decomposition of the original problem is SAC.

To prove strictness, consider a problem with three all-different constraints on $\{x_1, x_2, x_3\}$, on $\{x_1, x_2, x_4\}$, and on $\{x_1, x_3, x_4\}$, in which all variables have the domain $\{1, 2, 3\}$. The binary decomposition of this problem is SAC, but enforcing SGAC on the original problem shows that it is insoluble. For example, if we assign 1 to x_2 then GAC on the all-different constraint $\{x_1, x_3, x_4\}$ detects inconsistency and, therefore, the resulting problem is not GAC. So 1 is removed from the domain of x_2. With similar arguments, values 2 and 3 are also removed from the domain of x_2 resulting in a domain wipe-out.

A corollary of this theorem is that SGAC is strictly stronger than PIC and RPC on the binary decomposition.

Corollary 1. *Singleton generalized arc consistency on decomposable constraints is strictly stronger than path inverse consistency and restricted path consistency on the binary decomposition.*

Proof. It trivially follows from Theorem 1 and the results of [2] where it is proved that SAC is strictly stronger than PIC and RPC.

The following theorem shows that NIC and strong PC on the binary decomposition are incomparable to SGAC on the n-ary representation of decomposable constraints.

Theorem 2. *Singleton generalized arc consistency on decomposable constraints is incomparable to neighbourhood inverse consistency and to strong path consistency, on the binary decomposition.*

Proof. For an example where NIC is stronger than SGAC, consider a problem with five variables $\{x_1, x_2, x_3, x_4, x_5\}$ and six 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 $\{x_1, x_2, x_5\}$, on $\{x_2, x_3, x_4\}$, and on $\{x_3, x_4, x_5\}$. All variables have the domain $\{1, 2, 3, 4\}$. This problem is SGAC because any instantiation of every variable results in a problem that is GAC. Enforcing NIC, however, shows that the problem is insoluble. Now, for an example where strong PC is stronger than SGAC, consider a problem with three variables $\{x_1, x_2, x_3\}$ and three not-equals constraints, $x_1 \neq x_2$, $x_1 \neq x_3$, $x_2 \neq x_3$. The domain of x_1 is $\{1, 2\}$ and the domains of x_2 and x_3 is $\{1, 2, 3\}$. This problem is SGAC but enforcing strong PC adds the constraint that either x_2 or x_3 must be 3.

For an example where SGAC is stronger than NIC, 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 NIC, but enforcing SGAC on the non-binary representation shows that the problem is insoluble. Finally, for an example where SGAC is stronger

than strong PC, 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 but enforcing SGAC shows that it is insoluble.

6 Conclusions

We made a theoretical analysis of local consistency and search algorithms for decomposable constraints. We proved a new lower bound for the worst-case time complexity of arc consistency on binary not-equals constraints. We showed that the complexity is $O(e)$ instead of $O(ed)$, as previously known. We compared theoretically local consistency and search algorithms that operate on the non-binary representation of decomposable constraints to their counterparts for the binary decomposition. We also extended previous results on AC and GAC algorithms to the case of SAC and SGAC. In general we showed that the representation of problems can have a very large impact on the efficiency of search. Also, a non-binary representation can offer considerable advantages over a binary representation in certain classes of constraints, such as decomposable constraints.

Acknowledgements. The author is a member of the APES research group and would like to thank Ian Gent, Patrick Prosser, and Toby Walsh.

References

1. C. Bessière, P. Meseguer, E. Freuder, and J. Larrosa. On forward checking for non-binary constraint satisfaction. In *Proceedings CP-99*, pages 88–102.
2. R. Debruyne and C. Bessière. Some practicable filtering techniques for the constraint satisfaction problem. In *Proceedings of IJCAI-97*, pages 412–417.
3. I. Gent, K. Stergiou, and T. Walsh. Decomposable Constraints. *Artificial Intelligence*, 123:133–156, 2000.
4. C. P. Gomes and B. Selman. Problem structure in the presence of perturbations. In *Proceedings of AAAI-97*, pages 221–226.
5. C. P. Gomes, B. Selman, and N. Crato. Heavy-tailed probability distributions in combinatorial search. In *Proceedings of CP-97*, pages 121–135.
6. U. Montanari. Networks of Constraints: Fundamental Properties and Applications to Picture Processing. *Information Science*, 7:95–132, 1974.
7. P. Prosser, K. Stergiou, and T. Walsh. Singleton consistencies. In *Proceedings of CP-2000*.
8. J. C. Régin. A filtering algorithm for constraints of difference in csps. In *Proceedings of AAAI-94*, pages 362–367.
9. K. Stergiou, and T. Walsh. The difference all-difference makes. In *Proceedings of IJCAI-99*.
10. P. Van Hentenryck, Y. Deville, and C. Teng. A Generic Arc Consistency Algorithm and its Specializations. *Artificial Intelligence*, 57:291–321, 1992.