Algorithms for Quantified Constraint Satisfaction Problems

Nikos Mamoulis[1] and Kostas Stergiou[2]

[1] Department of Computer Science and Information Systems
University of Hong Kong
[2] Department of Information and Communication Systems Engineering
University of the Aegean

Abstract. Many propagation and search algorithms have been developed for constraint satisfaction problems (CSPs). In a standard CSP all variables are existentially quantified. The CSP formalism can be extended to allow universally quantified variables, in which case the complexity of the basic reasoning tasks rises from **NP**-complete to **PSPACE**-complete. Such problems have, so far, been studied mainly in the context of quantified Boolean formulae. Little work has been done on problems with discrete non-Boolean domains. We attempt to fill this gap by extending propagation and search algorithms from standard CSPs to the quantified case. We also show how the notion of value interchangeability can be exploited to break symmetries and speed up search by orders of magnitude. Finally, we test experimentally the algorithms and methods proposed.

1 Introduction

The basic decision task in a CSP is to determine whether there exist values for all variables such that all constraints in the problem are satisfied. It is well known that in standard CSPs, where all variables are existentially quantified, this task is **NP**-complete. A natural generalization of the standard CSP formalism is to consider the case where some of the variables may be universally quantified. A CSP that allows universal quantification of variables is called a Quantified Constraint Satisfaction Problem (QCSP) [3, 2]. The generalization of CSPs to QCSPs increases the expressiveness of the framework, but at the same time the complexity of the decision task rises from **NP**-complete to **PSPACE**-complete [2]. As an example consider the problem $\exists x_i \exists x_j \forall x_k \ (x_i > x_k \wedge x_j \neq x_k)$. This is a CSP where one of the variables is universally quantified, and it reads "there exist values for x_i and x_j such that for every value of x_k the constraints $x_i > x_k$ and $x_j \neq x_k$ are satisfied".

QCSPs can be used to model various **PSPACE**-complete problems from domains like game playing, planning, and belief revision. Recently, there is an increasing interest in algorithms for quantified constraint reasoning, especially in the cases of Quantified Propositional Satisfiability (QSAT or QBF) and quantified CSPs with continuous domains. Most of the proposed algorithms are extensions of already existing algorithms from the standard case (e.g. SAT) to the quantified one. However, very little has been done as far as algorithms for QCSPs with discrete non-boolean domains are concerned.

The only work we are aware of is [3], where ways to implement arc consistency for some classes of constraints in QCSPs are proposed.

In this paper we extend standard propagation and search algorithms for binary CSPs to the case of binary QCSPs. We describe an arc consistency (AC) algorithm that can deal with arbitrary binary constraints. We then extend the BT, FC, and MAC algorithms so that they can handle quantification. We then show how the notion of value interchangeability can be exploited in QCSPs to break symmetries and speed up search by orders of magnitude. Finally, we give some preliminary experimental results.

2 Preliminaries

A *Constraint Satisfaction Problem* (CSP) consists of a set of variables $X = \{x_1, \ldots, x_n\}$, a set of domains $D = \{D(x_1), \ldots, D(x_n)\}$, where $D(x_i)$ is the finite set of possible values for variable x_i, and a set C of constraints over subsets of the variables. A constraint c on variables $x_i, \ldots, x_j$ is a subset of the Cartesian product $D(x_i) \times \ldots \times D(x_j)$ that specifies the allowed combinations of values for variables $x_i, \ldots, x_j$. An assignment of a value a to variable x_i is denoted by (x_i, a). In standard CSPs all variables are considered to be existentially quantified. In what follows we will focus on binary constraints. A binary constraint on variables x_i, x_j will be denoted by c_{ij}. The goal in a QCSP can be either to determine satisfiability or to find a consistent instantiation of the existential variables for all instantiations of the universal ones.

A constraint $c_{ij} = (x_i, x_j)$ is *arc consistent* (AC) iff for each value a in $D(x_i)$ there exists a value b in $D(x_j)$ so that the assignments (x_i, a) and (x_j, b) satisfy c_{ij}. In this case we say that b is a *support* for a on constraint c_{ij}. The operation performed to determine whether a constraint is satisfied by a given tuple of assignments is called a *consistency check*. A binary CSP is arc consistent if all its constraints are arc consistent. We now give a formal definition of QCSPs.

Definition 1. ([3]) A *Quantified Constraint Satisfaction Problem* (QCSP) is a formula of the form $Q_1 x_1 \ldots Q_n x_n \ (c_1 \wedge \ldots \wedge c_m)$, where each Q_i denotes a quantifier ($\forall$ or $\exists$) and each c_i is a constraint relating some variables among $x_1, \ldots, x_n$.

3 Arc Consistency

[3] extends the definition of AC to QCSPs and gives rules to compute AC for constraints on 0-1 variables (e.g. $\neg x = y$) and numerical constraints (e.g. $x + y = z$). We complement the work of [3] by describing simple rules that can be used to devise a generic AC algorithm for QCSPs. When applying AC on a constraint c_{ij}, the filtering achieved depends on the type of quantification for variables x_i, x_j and on the order in which the variables appear in the quantification formula. For a binary constraint there are four possible orders. We can define AC for a constraint c_{ij} using the following general rules; one for each order of quantification.

$\exists\, x_i\ \exists\, x_j$: This is the case of standard CSPs. Constraint c_{ij} is AC iff each value $a \in D(x_i)$ is supported by at least one value $b \in D(x_j)$. If a value $a \in D(x_i)$ has no support in $D(x_j)$ then AC will remove a from $D(x_i)$. If $D(x_i)$ becomes empty then the problem is unsatisfiable.

$\forall\, x_i \,\forall\, x_j$: Constraint c_{ij} is AC iff each value $a \in D(x_i)$ is supported by all values in $D(x_j)$. If a value $a \in D(x_i)$ is not supported by all values in $D(x_j)$ then the problem is unsatisfiable.

$\forall\, x_i \,\exists\, x_j$: Constraint c_{ij} is AC iff each value $a \in D(x_i)$ is supported by at least one value in $D(x_j)$. If a value $a \in D(x_i)$ has no support in $D(x_j)$ then the problem is unsatisfiable.

$\exists\, x_i \,\forall\, x_j$: Constraint c_{ij} is AC iff each value $a \in D(x_i)$ is supported by all values in $D(x_j)$. If a value $a \in D(x_i)$ is not supported by all values in $D(x_j)$ then AC will remove a from $D(x_i)$. If $D(x_i)$ becomes empty then the problem is unsatisfiable.

Based on the above rules we can easily devise an AC algorithm for quantified binary CSPs. The algorithm takes a QCSP with a set X of existentially or universally quantified variables in a given order, and and a set C of binary constraints, and computes the arc consistent sub-domains in case the problem is arc consistent or returns FALSE in case the problem is not arc consistent. The algorithm uses the previously defined rules to check the consistency of a constraint according to the type and order of quantification of the variables involved in the constraint.

4 Search Algorithms

Algorithms BT and FC for binary QCSPs are easily devised by extending the corresponding CSP algorithms. I we apply AC before search, we do not have to consider constraints of the form $\exists\, x_i \forall\, x_j, c_{ij}$ or $\forall\, x_i \forall\, x_j, c_{ij}$ in the algorithms. All values of variable x_i, in such constraints, are definitely consistent with all values of variable x_j. If some value was not consistent then it would have been removed by AC.

BT is a straightforward extension of the corresponding algorithm for standard CSPs. It proceeds by checking assignments of values to variables until the truthness of the quantified problem is proved or disproved. The extension of standard FC to QCSPs, which we call FC0, operates in a way similar to BT with the difference that each variable assignment is forward checked against values of future existential variables constrained with the current one. We have also experimented with a modified version of FC, which we call FC1, that is better suited to QCSPs. FC1 has the exactly same behavior as FC0 when the current variable is existentially quantified. If the current variable x_i is universally quantified then we forward check each value of x_i against all future variables before assigning a specific value to x_i. If one of x_i's values causes a domain wipeout then we backtrack to the last existential variable. Otherwise, we proceed in the usual way. In this way we can discover dead-ends earlier and avoid fruitless exploration of search tree branches. It is easy to see that FC1 will always visit at most the same number of search tree nodes as FC0. The two algorithms are incomparable in the number of consistency checks they perform. That is, depending on the problem, FC0 may perform less checks than FC1 and vice versa.

Based on the above, we can easily adapt the MAC algorithm to QCSPs. MAC can also be modified in the same way as FC to yield MAC1, an algorithm analogous to FC1. That is, when the current variable x_i is universally quantified we can apply AC for each instantiation (x_i, a_j), $j \in \{1, \ldots, d\}$ before committing to a particular instantiation. If one of the instantiations causes a domain wipe-out then we backtrack. Otherwise, we commit to one of the values and proceed with the next variable.

5 Symmetry Breaking via Value Interchangeability

Many CSPs contain symmetries which means that for a given solution there are equivalent solutions. This can have a profound effect on the search cost when looking for one or (mainly) all solutions to a CSP. We propose the exploitation of value interchangeability as a dynamic symmetry breaking technique in QCSPs.

A value a of a variable x_i is *neighborhood interchangeable* (NI) with a value b of x_i, if a and b are supported by exactly the same values of all variables adjacent to x_i [4] A set of NI values can be replaced by a single representative of the set without in effect losing any solutions. In the context of QCSPs we can exploit interchangeability to break symmetries by "pruning" the domains of universal variables. That is, for each set (sometimes called bundle) of NI values we can keep one representative and remove the others, either permanently before search, or temporarily during search. If the algorithm proves that the representative value is consistent (i.e. satisfies the QCSP) then so are the rest of the NI values. Therefore, checking the consistency of such values is redundant. Consider the following example. We have the formula $\forall x_i \exists x_j, x_k \ (x_i \neq x_j, x_i \neq x_k)$, where the domains of the variables are $D(x_i) = \{0,1,2,3,4\}$, $D(x_j) = \{0,1\}$, $D(x_k) = \{0,2\}$. Values 3 and 4 of x_i are NI since they are supported by the same values in both x_j and x_k. Therefore, they can be replaced by a single value or to put it differently one of them can be pruned out of the domain.

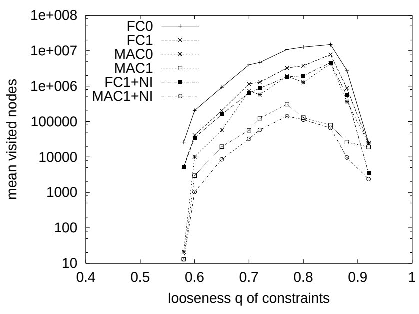
We can detect NI values as a preprocessing step in QCSPs to remove values from the domains of universal variables, and we can also detect them dynamically during search to avoid repeated exploration of similar subtrees.

6 Experimental Evaluation

To compare the performance of the algorithms presented is the previous sections, we ran experiments on randomly generated QCSPs. The random generator we used takes five parameters: $< n, n_\forall, d, p, q >$, where n is the total number of variables, $n_\forall$ is the number of universally quantified variables, d is the uniform domain size of the variables, p is the density of the constraint graph (i.e., the fraction of constrained variable pairs), and q is the uniform looseness of the constraints (i.e., the fraction of allowed tuples). To generate an instance, we first randomly assign universal quantification to $n_\forall$ out of the n variables, and then generate the constraint graph and the constraint matrices according to the widely used model B of standard CSPs. For each generated constraint c_{ij} the quantification of the variables is either $\exists x_i \ \exists x_j$ or $\forall x_i \ \exists x_j$. That is, we do not generate constraints that can be handled by preprocessing alone.

[5] showed that models for random generation of QCSPs can suffer from a local flaw that makes almost all of the generated instances false, even for small problem sizes. Since no satisfactory flawless model for QCSPs has been proposed, in the experiments reported below we disregard flawed instances.

Figures 1 and 2 present a comparison of algorithms FC0, FC1, MAC0, MAC1, FC1+NI, MAC1+NI on problems with $n = 20$, $d = 5$, $p = 0.15$, $n_\forall = 5$, and varying q. For each value of q shown in the figures, 100 problem instances were generated. We measure the number of node visits and consistency checks performed. As we can see, there is an easy-hard-easy pattern in the search cost as in standard CSPs and QSAT.



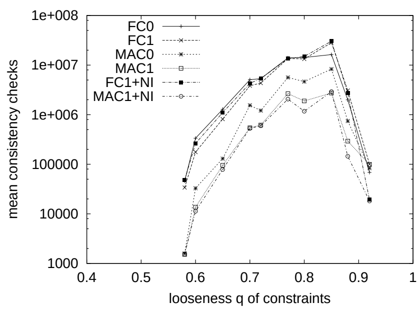


Fig. 1. Average visited nodes.

Fig. 2. Average consistency checks.

The MAC variants outperform the corresponding FC ones, as expected. FC1 and MAC1 are able to detect dead-ends early, which makes them considerably better than FC0 and MAC0 respectively. The best algorithm is MAC1+NI, followed by MAC1. Note that MAC1+NI is competitive with the QSAT solver operating on encodings of QCSPs into QSAT used in [5]. While FC1+NI was significantly outperformed by the QSAT solver (see [5]), preliminary experiments showed that MAC1+NI is five times faster (median cpu times) in $< 20, 5, 5, 0.15, 0.72 >$ problems. However, mean cpu times showed a similar advantage in favor of the QSAT solver.

7 Conclusion

In this paper we studied algorithms for QCSPs with discrete non-boolean domains. The QCSP framework can be used to model various **PSPACE**-complete problems from domains such as planning and game playing. We first described an AC algorithm that can deal with arbitrary binary constraints. We then extended the BT, FC, and MAC algorithms so that they can handle quantification. We also proposed modifications of FC and MAC that are better suited to QCSPs. We showed how value interchangeability can be exploited in QCSPs to break symmetries and speed up search by orders of magnitude. Finally, we tested experimentally the algorithms and methods proposed. There is a lot of future work to be done. We intend to follow two directions; first to further improve the presented algorithms through dynamic variable ordering heuristics and techniques like backjumping and learning, and second to apply the QCSP framework in real problems, such as planning under uncertainty, and compare it with existing approaches .

Acknowledgements

Kostas Stergiou is a member of the APES research group and would like to thank all other members. We especially thank I. Gent, P. Nightingale, and A. Rowley.

References

1. C. Bessière and J.C. Régin. Refining the basic constraint propagation algorithm. In *Proceedings of IJCAI-2001*, pages 309–315, 2001.

2. F. Boerner, A. Bulatov, P. Jeavons, and A. Krokhin. Quantified constraints: algorithms and complexity. In *Proceedings of CSL-2003*, pages 244–258, 2003.
3. L. Bordeaux and E. Monfroy. Beyond NP: Arc-consistency for quantified constraints. In *Proceedings of CP-2002*, 2002.
4. E. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings of AAAI-91*, pages 227–233, 1991.
5. I. Gent, P. Nightingale, and A. Rowley. Encoding quantified csps as quantified boolean formulae. Technical Report APES-79-2004, APES Research Group, February 2004. Available from http://www.dcs.st-and.ac.uk/˜apes/apesreports.html.