Super Solutions in Constraint Programming

Emmanuel Hebrard*

Cork Constraint Computation Centre
University College Cork
e.hebrard@4c.ucc.ie

1 Introduction

Where changes to a solution introduce additional expenses or reorganization, solution robustness is a valuable property. A robust solution is less sensitive to changes. For example, a robust schedule will not collapse immediately when one job takes slightly longer to execute than planned. The schedule should change locally and in small proportions, and the overall makespan should change little if at all.

To improve solution robustness, we introduce the concept of super-solutions to constraint programming (CP). An (a, b)-super-solution is one in which if the values assigned to a variables are no longer available, the solution can be repaired by assigning these variables with a new values and at most b other variables. An (a, b)-super-solution is a generalization of both fault tolerant solutions in CP [14] and supermodels in propositional satisfiability (SAT) [9]. We show that finding (a, b)-super-solutions for any fixed a is NP-Complete in general. Super solutions are computed offline and do not require knowledge about the likely changes. A super-solution guarantees the existence of a small set of repairs when the future changes in a small way.

In this paper, we focus on the algorithmic aspects of finding $(1, 0)$-super-solutions, which are the same as fault tolerant solutions [14]. A $(1, 0)$-super-solution is a solution where if one variable loses its value, we can find another solution by re-assigning this variable with a new value, and no other changes are required for the other variables. We explore methods based both on reformulation and on search to find $(1, 0)$-super-solutions. Our reformulation methods transform the constraint satisfaction problem so that the only solutions are super-solutions. We review two reformulation techniques presented in [14], and introduce a new one, which we call the cross-domain reformulation. Our search methods are based on notions of super-consistency. We propose two new search algorithms that extend the maintaining arc consistency algorithm (MAC [7, 6]). We empirically compare the different methods and observe that one of them, super-MAC shows considerable promise.

* supported by the Science Foundation Ireland and an ILOG grant.

solutions	$(1,1)$-super-solutions	$(1,0)$-super-solutions
$\langle 1,1,1\rangle$, $\langle 1,1,2\rangle$	$\langle 1,1,2\rangle$, $\langle 1,1,3\rangle$	$\langle 1,2,3\rangle$
$\langle 1,1,3\rangle$, $\langle 1,2,2\rangle$	$\langle 1,2,2\rangle$, $\langle 1,2,3\rangle$	$\langle 1,2,2\rangle$
$\langle 1,2,3\rangle$, $\langle 1,3,3\rangle$	$\langle 1,3,3\rangle$, $\langle 2,2,2\rangle$	$\langle 2,2,3\rangle$
$\langle 2,2,2\rangle$, $\langle 2,2,3\rangle$	$\langle 2,2,3\rangle$, $\langle 2,3,3\rangle$	
$\langle 2,3,3\rangle$, $\langle 3,3,3\rangle$		

Fig. 1. solutions, $(1,1)$-super-solutions, and $(1,0)$-super-solutions for the CSP: $X \leq Y \leq Z$.

2 Super Solutions

Supermodels were introduced in [9] as a way to measure solution robustness. An (a,b)-supermodel of a SAT problem is a model (a satisfying assignment) with the additional property that if we modify the values taken by the variables in a set of size at most a (breakage variables), another model can be obtained by flipping the values of the variables in a disjoint set of size at most b (repair variables). We adapt this definition to handle non Boolean domains.

Definition 1. *A solution to a CSP is (a,b)-super-solution iff the loss of the values of at most a variables can be repaired by assigning other values to these variables, and modifying the assignment of at most b other variables.*

Example 1. Let us consider the following CSP: $X, Y, Z \in \{1,2,3\}, \quad X \leq Y \leq Z$. The solutions to this CSP are shown in Figure 1, along with the subsets of the solutions that are $(1,1)$-super-solutions and $(1,0)$-super-solutions. For the rest of the paper, we focus on $(1,0)$-super-solutions and refer to them as super-solutions when it is convenient.

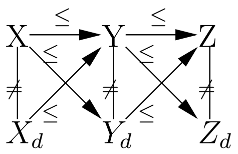
The solution $\langle 1,1,1\rangle$ is not a super-solution. If X loses the value 1, we cannot find a repair value for X that is consistent with Y and Z since neither $\langle 2,1,1\rangle$ nor $\langle 3,1,1\rangle$ are solutions. On the other hand, $\langle 1,2,3\rangle$ is a super-solution since when X breaks we have the repair solution $\langle 2,2,3\rangle$, when Y breaks we have the repair solution $\langle 1,1,3\rangle$, and when Z breaks we have the repair solution $\langle 1,2,2\rangle$. We therefore have a theoretical basis to prefer the solution $\langle 1,2,3\rangle$ to $\langle 1,1,1\rangle$, as the former is more robust.

Deciding if a SAT problem has an (a,b)-supermodel is NP-complete [9]. It is not difficult to show that deciding if a CSP has an (a,b)-super-solution is also NP-complete, even when restricted to binary constraints.

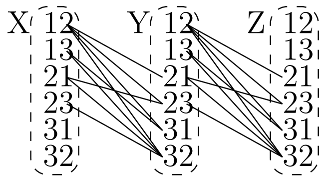
Theorem 1. *Deciding if a CSP has an (a,b)-super-solution is NP-complete for any fixed a.*

Proof. See [2].

A necessary but not sufficient condition to find supermodels in SAT or super-solutions in CSPs is the absence of backbone variables. A *backbone variable* is a variable that takes the same value in all solutions. As a backbone variable has



(a) $P + P$ (macrostructure)



(b) $P \times P$ (microstructure)

Fig. 2. Reformulations of example 1 such that any solution is a super-solution of the original

no alternative, a SAT or CSP problem with a backbone variable cannot have any (a, b)-supermodels or (a, b)-super-solutions.

Another important factor that influences the existence of super-solutions is the way the problem is modeled. For instance, the direct encoding into SAT (i.e., one Boolean variable for each pair variable-value in the CSP [12]) of the problem in Example 1 has no $(1, 0)$-supermodels, even though the original CSP had a $(1, 0)$-super-solution. Moreover, the meaning of a super-solution depends on the model. For example, if a variable is a job and a value is a machine, the loss of a value may mean that the machine has now broken. On the other hand, if a variable is a machine and the value is a job, the loss of a value may mean that the job is now not ready to start. The CP framework gives us more freedom than SAT to choose what variables and values stand for, and therefore to the meaning of a super-solution.

3 Finding $(1, 0)$-Super Solutions

Reformulation: To the best of our knowledge the only known method for finding super-solutions is a reformulation, which we call $P + P$, described in [14]. The fault tolerant solutions discussed in that paper are equivalent to (1.0)-super-solutions. Informally, $P + P$ duplicates the constraint network and ensures that any value assigned to a duplicated variable is a valid alternative.

We introduce a new reformulation approach which does not introduce new variables nor change the constraint graph. However, the domains become their own cross product, and the constraints are changed such that an element $\langle v_1, v_2 \rangle$ has a support $\langle w_1, w_2 \rangle$ iff, locally to those variables, v_2 (resp. w_2) is a valid alternative for v_1 (resp. w_1). We refer to this reformulation as $P \times P$.

Search Algorithms: We also introduce two *super-consistencies*, built on the model of local consistency for CSP. They lead naturally to search algorithms. The first one is a naive approach that augments the traditional arc-consistency (AC) by a further condition: after propagation, all domains must contain at least two values, since any variable need an alternative. We refer to this super-consistency as AC+, and to the correponding search algorithm as MAC+.

The second approach derives from the $P \times P$ reformulation. To avoid the quadratic reformulation we approximate it by dividing the values into a *super-domain* ($SD(X)$) and a *repair-domain*($RD(X)$) such that $SD(X) \subseteq RD(X) \subseteq D(X)$. A value v is in the super-domain of X iff for any other variable Y, there exists v' in super-domain of Y and r in repair-domain of Y such that $\langle v, v' \rangle$ and $\langle v, r \rangle$ are allowed and $v' \neq r$. A value v is in repair-domain of X iff for any other variable Y, there exists v' in super-domain of Y such that $\langle v, v' \rangle$ is allowed. We refer to this super-consistency as super-AC. For instance, in example 1, $Y = 1$ is not super-AC although it has one AC support on X and three on Z. Indeed it would require at least two supports on every variable, one of them being itself doubly supported.

Theoretical Properties: We can compare those consistencies with arc-consistency on the reformulations. We denote x(P) the proposition "P satisfies the consistency x". We have AC($P \times P$) $\Rightarrow$ super-AC(P) $\Rightarrow$ AC($P + P$) $\Leftrightarrow$ AC+(P).

3.1 Experimental Results

The experiments, on random CSPs and jobshop scheduling problems, confirm that MAC on $P \times P$ backtracks the least. However super-MAC, a search algorithm using super-AC, performs better than any other method both in terms of constraint checks and runtime. We give as example the results of the outlined approaches on two classes of 100 random CSP instances at the phase transition for super-Solubility.

	MAC+	MAC on $P{+}P$	MAC on $P{\times}P$	super-MAC
$\langle 50, 15, 0.08, 0.5 \rangle$				
CPU time (s)	788	43	53	**1.8**
backtracks	152601000	111836	**192**	2047
$\langle 100, 6, 0.05, 0.27 \rangle$				
CPU time (s)	2257	430	3.5	**1.2**
backtracks	173134000	3786860	**619**	6487

4 Related Work

Supermodels [9] and fault tolerant solutions [14] have been discussed earlier.

The notions of neighborhood interchangeability [5] and substitutability are closely related to our work, but whereas, for a given problem, interchangeability is a property of the values and works for all solutions, repairability is a property of the values in a given solution.

Uncertainty and robustness have been incorporated into constraint solving in many different ways. Some have considered robustness as a property of the algorithm, whilst others as a property of the solution (see, for example, dynamic CSPs [1, 8, 11], partial CSPs [4], dynamic and partial CSPs [10], stochastic CSPs [13], and branching CSPs [3]). In dynamic CSPs, for instance, we can reuse previous work in finding solutions, though there is nothing special or necessarily robust about the solutions returned. In branching and stochastic CSPs, on the other hand, we find solutions which are robust to the possible changes. However, both these frameworks require information, such as a probability distribution, about uncertainty.

5 Conclusion

To improve solution robustness in CP, we introduced the notion of super-solutions. We explored reformulation and search methods for finding super-solutions. We introduced the notion of super-consistency, and developed a search algorithm, super-MAC, that outperforms the other known methods.

The problem of seeking super-solutions becomes harder when multiples repairs are allowed, i.e, for $(1, b)$-super-solutions. We aim to generalize the idea of super-consistency to $(1, b)$-super-solutions. In a similar direction, we would like to explore tractable classes of $(1, b)$-super-CSPs. Furthermore, we plan to restrict or enlarge the class of super-solutions provided with respect to the problem being solved. For instance, in scheduling problems, repairs have to be later in time than breaks, which can be seen as a restriction on super-solutions.

References

1. A. Dechter and R. Dechter. Belief maintenance in dynamic constraint networks. In *Proceedings AAAI-88*, pages 37–42, 1988.
2. B. Hnich E. Hebrard and T. Walsh. 'super solutions in constraint programming', manuscript submited to CPAIOR-2003, 2003.
3. D. W. Fowler and K. N. Brown. Branching constraint satisfaction problems for solutions robust under likely changes. In *Proceedings CP-00*, pages 500–504, 2000.
4. E. C. Freuder. Partial Constraint Satisfaction. In *Proceedings IJCAI-89*, pages 278–283, 1989.
5. E. C. Freuder. Eliminating Interchangeable Values in Constraint Satisfaction Problems. In *Proceedings AAAI-91*, pages 227–233, 1991.
6. J. Gaschnig. A constraint satisfaction method for inference making. In *Proceedings of the 12th Annual Allerton Conference on Circuit and System Theory*. University of Illinois, Urbana-Champaign, USA, 1974.
7. J. Gaschnig. Performance measurement and analysis of certain search algorithms. Technical report CMU-CS-79-124, Carnegie-Mellon University, 1979. PhD thesis.
8. N. Jussien, R. Debruyne, and P. Boizumault. Maintaining arc-consistency within dynamic backtracking. In *Proceedings CP-00*, pages 249–261, 2000.
9. A. Parkes M. Ginsberg and A. Roy. Supermodels and robustness. In *Proceedings AAAI-98*, pages 334–339, 1998.
10. I. Miguel. *Dynamic Flexible Constraint Satisfaction and Its Application to AI Planning*. PhD thesis, University of Edinburgh, 2001.
11. T. Schiex and G. Verfaillie. Nogood recording for static and dynamic constraint satisfaction problem. *IJAIT*, 3(2):187–207, 1994.
12. T. Walsh. SAT v CSP. In *Proceedings CP-00*, 2000.
13. T. Walsh. Stochastic constraint programming. In *Proceedings ECAI-02*, 2002.
14. R. Weigel and C. Bliek. On reformulation of constraint satisfaction problems. In *Proceedings ECAI-98*, pages 254–258, 1998.