Super CSPs

Emmanuel Hebrard, Brahim Hnich, and Toby Walsh*

Cork Constraint Computation Centre
University College Cork
{e.hebrard, brahim, tw}@4c.ucc.ie

Abstract. Fault tolerant solutions [12] and supermodels [8] are solutions with strong properties of stability. In this paper, we study super solutions, the generalization of supermodels to the constraint satisfaction and optimization framework. We explore two different approaches to find super solutions. In the first, we reformulate a constraint problem so that the only solutions are super solutions. In the second, we introduce notions of super consistency and enforce them during search. We also propose a branch and bound algorithm for finding the most robust solution, in case no super solutions exist. Finally, we run extensive experiments to compare the different approaches and study the difficulty of finding super solutions. We show that super MAC, a new search algorithm for finding super solutions outperforms the other techniques.

1 Introduction

Many AI problems may be modelled as constraint satisfaction and optimization problems. However, the real world is subject to change: machines may break, drivers may get sick, stock prices increase or decrease, etc. In such cases, our solutions to the problems may "break". In this context, one may want a solution to be robust, that is able to remain valid despite changes.

Uncertainty and robustness can be 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] [7] [10], partial CSPs [5], dynamic and partial CSPs [9], stochastic CSPs [11], 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 assume significant information about the likely changes (e.g. the stochastic CSP framework assumes we have independent probabilities for the values taken by the stochastic variables). In this paper we generalize a definition of solution robustness introduced in SAT [8] to constraint programming. This definition allows us to estimate the robustness of solutions without any additional knowledge.

* All authors are supported by the Science Foundation Ireland.

Solution stability[1] is the ability of a solution to share as many values as possible with a new solution if a change occurs. For example, a stable solution in a trip planning problem would not require cancelling a flight because of a train drivers' strike: the plan should change locally and in small proportion. Where large changes to a solution introduce additional expenses or reorganization, stability is valuable. Moreover, stability can help us find a new solution. Stability can then be seen as a particular form of robustness.

Fault tolerant solutions [12] and supermodels [8] are examples of solutions that exhibit strong properties of stability: a solution is fault tolerant if any of its values can be replaced by another one. For each part of our trip we need at least two different ways to get from one point to another (we can replace the train by a bus). Supermodels are models of SAT formula that can be repaired once a small number of variables have changed by changing only a few other variables. Supermodels are a powerful way to capture robustness and stability of solutions. Supermodels are computed offline, in advance of any changes. A supermodel guarantees the existence of a reasonably small repair in case of a small change in the future. Supermodels do not require any particular knowledge about future changes. However, supermodels have only been studied for SAT problems. In this paper, we generalize the concept of supermodels to constraint satisfaction problems (CSPs). We conjecture that constraint programming is in many ways a better framework for supermodels: they will be more likely, and they will more likely be useful. The definition of supermodels given in [8] has to be modified to deal with CSPs. From now on, we will refer to supermodels for SAT problems, whereas *super solutions* will denote stable solutions to CSPs. Note that our definition of super solutions (section 2) reduces to the definition of supermodels if the SAT variables are considered as CSP variables with binary domains.

2 Super Solutions

Supermodels were introduced in [8] as a framework to measure inherent degrees of solution stability. 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). A necessary but not sufficient condition that need to be satisfied in order to find a supermodel is the absence of backbone variables. A *backbone variable* is a variable that takes the same value in all solutions. The presence of a backbone variable in a SAT problem makes it impossible to find any (a, b)-supermodels as that particular variable has no alternative.

There are a number of ways we could generalize the definition of supermodels from SAT to constraint satisfaction as variables now can have more than two values. A break could be either "losing" the current assignment for a variable and then freely choosing an alternative value, or replacing the current assignment

[1] sometimes also referred to as "similarity" in the literature

with some other value. Since the latter is stronger and therefore less useful, we propose the following definition.

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

A number of properties follow immediately, for example, a (c,d)-super solution is a (a,b)-super solution if ($a \leq c$ or $d \leq b$) and $c+d \leq a+b$,

We will focus mostly on $(1,0)$-super solutions in the rest of the paper. They are called *fault tolerant solutions* and described in [12]. Deciding if a SAT problem has an (a,b)-supermodel is NP-complete [8]. 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. To see it is in NP, we need a polynomial witness that can be checked in polynomial time. This is simply an assignment which satisfies the constraints, and, for each of the $O(n^a)$ (which is polynomial for fixed a) possible breaks, the $a+b$ repair values.

To show completeness, we show how to map a binary CSP onto a new binary problem in which the original has a solution iff the new problem has an (a,b)-supersolution. We duplicate the domains of each of the variables, and extend the constraints so that the behave equivalently on the new values. For example, suppose we have a constraint $C(X,Y)$ which is only satisfied by $C(m,n)$. Then we extend the constraint so that is satisfied by just $C(m,n)$, $C(m',n)$, $C(m,n')$ and $C(m',n')$ where m' and n' are the duplicated values for m and n. Clearly, this binary CSP has a solution iff the original problem also has. In addition, any break of a variables can be repaired by replacing the a corresponding values with their primed values (or unpriming them if they are already primed) as well as any b other values.

3 Motivational Example

The approach taken in this paper, whilst it concerns repairs, is a proactive approach. A super solution is a solution to the deterministic, regular, CSP which we expect may change before we come to apply the solution. The changes occur *after* we have found a solution and must then be tackled. We aim to ensure that any break (loss of one value) will be repairable if it eventually occurs.

Let us consider the following CSP: $\quad X,Y,Z \in \{1,2,3\} \quad X \leq Y \wedge Y \leq Z$
The solutions to this CSP are shown in Figure 1, as well as the subsets of solutions that are $(1,1)$-super solutions and $(1,0)$-super solutions for this problem.

The solution $\langle 1,1,1 \rangle$ is not a $(1,0)$-super solution. The reason is that if X loses its value 1, we cannot find a repair value for X that is consistent with Y

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 problem $X \leq Y \leq Z$.

and Z because neither $\langle 2,1,1\rangle$ nor $\langle 3,1,1\rangle$ are solutions to the problem. Also, solution $\langle 1,1,1\rangle$ is not a $(1,1)$-super solution because when X loses its value 1, we cannot repair it by changing the value assignment of at most another variable, i.e., there exists no repair solution when X breaks because none of $\langle 2,1,1\rangle$, $\langle 3,1,1\rangle$, $\langle 2,2,1\rangle$, $\langle 2,3,1\rangle$, $\langle 2,1,2\rangle$, and $\langle 2,1,3\rangle$ is a solution to our problem. On the other hand, $\langle 1,2,3\rangle$ is a $(1,0)$-super solution because 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 $\langle 1,1,1\rangle$, the former being more "robust" or "stable". Note that all algorithms introduced in this paper provide offline (that is, in advance) the repairs as well as the solution. Hence finding and applying the repairs online takes constant time.

The way a given problem is modelled influences the super solutions. For instance, consider the encoding in SAT of this problem. One way to encode this is to add a Booloean variable x_i for every value i that X can take, $x_i = True$ means that $X = i$. In our case, $\{x_1, x_2, x_3\}$, $\{y_1, y_2, y_3\}$ and $\{z_1, z_2, z_3\}$. However, such an encoding has no $(1,0)$-supermodel. Any variable y_i standing for an assignment of y is in conflict with at least one other assignment on x or z. Moreover, one y_i must be set to $True$, since any solution gives a value to y. Therefore the variable in conflict with y_i must be set to $False$. If the assignment of this variable is modified, i.e, flipped to $True$, then at least y_i must be reassigned to $False$. Intuitively, in any encoding, the likelihood of finding an (a,b)-super solution will decrease with the number of variables and increase with the domain size. Moreover, the meaning of a super solution depends also on the model. For example, if a variable is a country and a value is a colour, the loss of a given value is equivalent to the loss of the given colour. On the other hand if every possible colouring of that country is encoded by a Boolean variable, the loss of a value means either that the colour is now forbidden or that this colour *must* be used. The CSP framework gives more freedom to choose what variables and values stand for, and therefore what being a super solution means.

4 Reformulation Approach to Finding Super Solutions

One possible approach to finding super solutions is to add further variables and constraints to the CSP that would eliminate those solutions that are not super solutions. In [12] a definition of *fault tolerant solutions* is given which matches exactly the definition of $(1,0)$-super solutions: two reformulations of CSP are given in [12] such that any solution of the reformulation corresponds to a fault tolerant solution of the original CSP. In the following subsections, we review those reformulation approaches and propose a new one, which we call the *cross-domains representation*.

4.1 Boolean Reformulation

The first approach in [12], associates a Boolean variable x_v to each value v of each variable X in a given CSP. Assigning this variable to 1 corresponds to the assignment $X = v$ in the CSP. Every disallowed tuple $\neg(X = v, Y = w)$ translates into the conflict clause which forbids the assignment (1,1) for the two corresponding variables. Finally, whereas in the original CSP any variable must implicitly be given exactly one value, here exactly *two* variables must be satisfied for every CSP variable. This model allows only fault tolerant solution, but not *all* of them and it is shown through the following CSP:

$$X = [1,2], Y = [1,2], X + Y < 4$$

This CSP translates into the following Boolean CSP (or SAT problem):

$$X_1 = X_2 = X_3 \in [0,1] \ (1)$$
$$(x_1, x_2) \in \{(1,1)\}, (y_1, y_2) \in \{(1,1)\} \ (2)$$
$$(x_2, y_2) \in \{(0,0), (0,1), (1,0)\} \ (3)$$

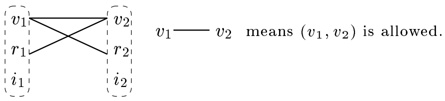
In (1) the Boolean variabless associated to the different assignments are given. In (2) exactly two of the Booleans variables must be true for each corresponding CSP variable, while in (3), the only disallowed tuple $\langle X = 2, Y = 2\rangle$ is encoded. The solution $\langle 1, 1\rangle$ is a fault tolerant solution of the original CSP, since for X or Y, the value 2 can replace the value 1. However, this Boolean CSP has no solution. This reformulation does not therefore give all the fault tolerant solutions.

4.2 Adding Extra Variables and Constraints

The second approach proposed in [12] simply duplicates the variables. The additional variables have the same domain as the original variables, and are linked with the same constraints to the same neighbourhood. A not_equal constraint is also posted between each original variable and its duplicate. The assignment to the original part of the CSP gives then the solution, while the duplicated part gives the repair for each variable. We refer to the reformulation of a CSP P with this encoding as $P{+}P$.

4.3 Cross-Domains Reformulation

We now present our last reformulation approach. Let $S = \langle v_1, v_2 \rangle$ be a (1,0)-super solution on *two* variables X_1 and X_2. If v_1 is lost, then there must be a value in $r_1 \in \mathcal{D}(X_1)$ that can repair v_1, that is $\langle r_1, v_2 \rangle$ is a compatible pair. Symmetrically, there must exists r_2 such that $\langle v_1, r_2 \rangle$ is allowed. Now consider the following subproblem involving two variables:

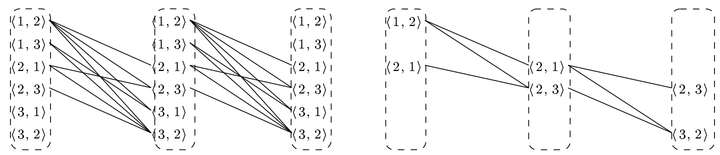


Since it satisfies the criteria above, $S = \langle v_1, v_2 \rangle$ is a super solution whilst any other tuple is not. One may suspect that values r_1, r_2, i_1, i_2 do not participate in any super solution and hence can be pruned. However r_1 and r_2 are *essential* for *providing* support to v_1 and v_2. So, we cannot simply reason about extending partial instantiations of values, unless we keep the information about the values that can be used as repair. So, let us instead think of the domain of the variables as pairs of values $\langle v, r \rangle$, the first element corresponding to the *super value* (which is part of a super solution), the second corresponding to the *repair value* (which can repair the former). We call $P \times P$ the reformulation of a CSP $P = \{\mathcal{X}, \mathcal{D}, \mathcal{C}\}$ such that any domain becomes its own cross-product (minus the doublons), D_i becomes $D_i \times D_i - \{\langle v, v \rangle | v \in D_i\}$. The constraints are built as follows, two pairs $\langle v_1, r_1 \rangle$ and $\langle v_2, r_2 \rangle$ are compatible iff

- v_1 and v_2 are compatible (the solution must be consistent at the first place),
- v_1 and r_2 are compatible (in case of break involving v_2, r_2 can be a repair),
- v_2 and r_1 are compatible.

The new domain $\mathcal{CD}(\mathcal{X})$ and $\mathcal{CD}(\mathcal{Y})$ of variable X and Y are
$\{\langle v_1, r_1 \rangle, \langle v_1, i_1 \rangle, \langle r_1, v_1 \rangle, \langle r_1, i_1 \rangle, \langle i_1, v_1 \rangle, \langle i_1, r_1 \rangle\}$
$\{\langle v_2, r_2 \rangle, \langle v_2, i_2 \rangle, \langle r_2, v_2 \rangle, \langle r_2, i_2 \rangle, \langle i_2, v_2 \rangle, \langle i_2, r_2 \rangle\}$
and the only one allowed tuple is $S = \langle \langle v_1, r_1 \rangle, \langle v_2, r_2 \rangle \rangle$. The example below shows the cross-domain representation of the CSP given in section 3 ($X \leq Y \leq Z, D(X) = D(Y) = D(Z) = [1, 2, 3]$). On this augmented CSP, arc consistency will prune *all* the pairs that are inconsistent. The process of reformulating with the cross-domain representation and enforcing a local consistency can therefore be seen as enforcing a local super consistency. A super solution of the original CSP can be extracted by keeping only the first element of every pair.

Since the constraint graph of this CSP is a tree, successively enforcing arc-consistency and assigning a variable leads to a solution without backtracking. The possible solutions to this augmented CSP are: $\langle \langle 1,2 \rangle, \langle 2,1 \rangle, \langle 2,3 \rangle \rangle$, $\langle \langle 1,2 \rangle, \langle 2,1 \rangle, \langle 3,2 \rangle \rangle$, $\langle \langle 1,2 \rangle, \langle 2,3 \rangle, \langle 3,2 \rangle \rangle$ and $\langle \langle 2,1 \rangle, \langle 2,3 \rangle, \langle 3,2 \rangle \rangle$. The first corresponds to $\langle 1,2,2 \rangle$, the second and the third to $\langle 1,2,3 \rangle$ and the fourth to $\langle 2,2,3 \rangle$



5 Finding Super Solutions Via Search

5.1 Super Consistency

Local consistency allows backtrack-based search algorithms such as MAC to detect unsatisfiable subproblems earlier. Local consistency can also be used to develop efficient algorithms for finding super solutions. We shall introduce three ways of incorporating local consistency into a search algorithm for seeking super solutions. The first (AC+), a naive approach, augments the traditionnal arc-consistency by a further condition, achieving a very low level of filtering. The second (arc-consistency on $P{\times}P$) allows us to infer all possible local information, just as in arc-consistency in a regular CSP [4]. However this comes at a high computational cost, though polynomial. The third (super AC) approach gives less inference, but is a good tradeoff between the amount of pruning and complexity. Informally, a consistent closure of a CSP contains only partial solutions for a given level of locallity. However, the situation with super solutions is a little bit more tricky because values that do not get used in any local super solution can still be essential as a *repair* and thus cannot be simply pruned. Throughout the rest of this paper we will refer only to $(1,0)$-super solutions, and thus to $(1,0)$-super arc-consistency.

5.2 Arc-Consistency+

If S is a super solution, then for every variable, at least *two* values are consistent with all the others values of S. Consequently, being arc consistent *and having non-singleton domains* is a necessary condition of satisfiability. AC+ can then be defined as follows: for a CSP $P = \{\mathcal{X}, \mathcal{D}, \mathcal{C}\}$: AC+$(P) \Leftrightarrow AC(P) \wedge \forall D \in \mathcal{D}, |D| > 1$. Whilst AC+ is usually too weak to give good results, it is the basis for an algorithm for the associated optimization problem (section 6).

5.3 Arc-Consistency on $P{\times}P$

$P{\times}P$ has the interesting property of having exactly the same topology as the original problem P. Moreover, each pair $\langle value, repair\rangle$ is explicitly represented, therefore, arc consistency on $P{\times}P$ makes all the possible inference, regarding arcs. The proof that AC on $P{\times}P$ is the tightest filtering introduced in this paper follows in section 5.5. As a corollary, tree and treewidth bounded tractable classes of CSPs are also tractable for finding $(1,0)$-super solutions, through cross-domain representation, since any tree structure is conserved by the transformation. In a similar way, if P is binary and Boolean, then $P{+}P$ is binary and Boolean, and hence tractable.

5.4 A Notion of Super Consistency

Arc Consistency on $P{\times}P$ allows us to infer all that can be inferred locally. In other words, we will prune any value in a cross-domain that is not locally consistent. However, this comes at high cost, maintaining arc-consistency will be $O(d^4)$ where d is the initial domain size. We therefore propose an alternative that does less inference, but at a much lower (for example, quadratic) cost. The main reason for the high cost is the size of the cross-domains. A cross-domain is quadratic in the size of the original domain since it explicitly represents the repair value for each super value. Here we will simulate (most of) the inference performed by super consistency, but will only look at one value at a time, and not pairs. We will divide the domain of the variable into two separate sets of domains: the "super domain" (SD) where only super values are represented and a "repair domain" (RD) where repair values are stored.

- 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 (v, v') and (v, r) 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 (v, v') is allowed.

The definition of super arc-consistency translates in a straightforward way into a filtering algorithm. The values are marked as either *super* or *repair*, and when looking for support of a super value, an additional and different support marked either as *super* or *repair* is required. The complexity of checking the consistency of an arc increases only by a factor of 2 and thus remains in $O(d^2)$. An algorithm that maintains such a consistency would branch only on the values in super domains and would fail if either the super domains becomes empty or the repair domains becomes singleton. The low cost of achieving super arc-consistency comes at the price of achieving a lower level of consistency compared to maintaining arc-consistency on $P{\times}P$, as shown in Figure 4. Moreover, the consistency must be maintained also on the domains of the variables already assigned. The super domain of an assigned variable is reduced to the chosen value and cannot change, but the repair domain can wipe out because of an assignment in the future. Figure 2 depicts an algorithm enforcing super AC on the CSP $X \leq Y \leq Z$. The first Figure shows the microstructure of the CSP. In the following Figure, super consitency is established: $Y = 1$ and $Y = 3$ have only one support, respectively on X and Z, thus they cannot be in super domain. Furthermore $X = 3$ and $Z = 1$ have each only one support on Y, respectively $Y = 1$ and $Y = 3$, which are not in the super domain. Hence $X = 3$ and $Z = 1$ are pruned. $Y = 2$ is the only value remaining in $SD(Y)$ and is thus assigned. In the last Figure, X is assigned to 2. As a result, $Y = 1$ is no longer supported and is pruned. Since it was the second support for $Z = 2$ on Y, $Z = 2$ is no longer in the super domain anymore. $Z = 3$ is the only value remaining in $SD(Z)$ and is assigned.

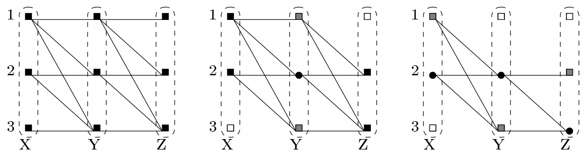


Fig. 2. Left: The microstructure of the CSP, Middle: its super consistent closure, Right: a super solution is reached after assigning $X = 2$.

- ● v is an assigned value ($SD = \{v\}$)
- ■ v is in super domain ($v \in SD$) and in repair domain($v \in RD$)
- ▣ v is in repair domain ($v \in RD$)
- □ v is pruned

5.5 Theoretical Properties

For the theorem and the proof below, we use the notation (x)(P) to denote that the problem P is "consistent" for the filtering (x). We compare AC on $P{\times}P$, AC on $P{+}P$, AC+ and super AC.

Theorem 2 (level of filtering). *For any subproblem P, $AC(P{\times}P) \Rightarrow$ super $AC(P) \Rightarrow AC(P{+}P) \Leftrightarrow AC{+}(P)$.*

Proof. **(1) AC($P{+}P$) $\Rightarrow$ AC+(P):** Suppose that P is not AC+, then in the arc consistent closure of P, there exists at least one domain D_i such that $|D_i| \leq 1$. $P{+}P$ contains P and then in its arc consistant closure, we have $|D_i| \leq 1$ as well. X_i is linked to a duplicate of itself which domain D'_i is then equal to D_i and therefore singleton (with the same value) or empty. However, recall that we force $X_i \neq X'_i$, thus $P{+}P$ is not AC.

(2) AC+(P) $\Rightarrow$ AC($P{+}P$): Suppose we have AC(P) and any domain D in P is such that $|D| > 1$, now consider $P{+}P$. The original constraints are AC since P is AC. The duplicated constraints are AC since they are identiqual to the original ones. The not_equal constraints between original and duplicated variables are AC since any variable has at least 2 values.

(3) super AC(P) $\Rightarrow$ AC+(P): Suppose that P is not AC+, then there exists two variables X, Y such that any value of X has at most one support on Y, therefore the corresponding super domain is wiped-out, and P is not super AC.

(4) super AC(P) $\not\Leftarrow$ AC+(P): See counterexample in Figure 3.

(5) AC($P{\times}P$) $\Rightarrow$ super AC(P): Suppose that AC($P{\times}P$), then for any two variables X, Y there exist two pairs $\langle v1, r1\rangle \in D(X) \times D(X), \langle v2, r2\rangle \in D(Y) \times D(Y)$, such that $\langle v1, r2\rangle, \langle r1, v2\rangle$ and $\langle v1, v2\rangle$ are allowed tuples. Therefore $v1$ belongs to the super domain of X and $v1$ and $r1$ belong to the repair domain of X. Thus, the super domain of X is not empty and the repair domain of X is not singleton. Therefore, P is super AC.

(6) AC($P\times P$) $\not\Leftarrow$ super AC(P): See counterexample in Figure 4. □

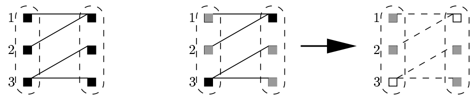


Fig. 3. The first graph shows the microstructure of a simple CSP, two variables and 3 values each, *allowed* combinations are linked. P is AC+ since the network is arc-consistent and every domain contains 3 values. However, P is not super AC since the greyed values (in the second graph) are not in super domains, they have only one support. In the second step, the whitened variables (in the third graph) are also removed from both repair and super domains since they don't have a support in a super domain.

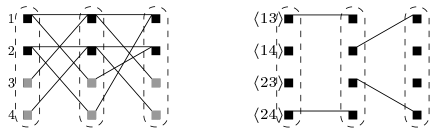


Fig. 4. The first graph shows the microstructure of a simple CSP, three variables and four values each, *allowed* combinations are linked. P is super AC since the super domains are of size 2 (black values), and the repair domains are of size 4 (black and grey values). The second graph shows $P\times P$, which is not arc-consistent.

5.6 Super Search Algorithms

MAC+. This algorithm establishes AC+ at each node. That is, establishes AC and backtracks if a domain wipes out *or* becomes singleton. In the MAC algorithm, we only prune future variables, since the values assigned to past variables are guaranteed to have a support in each future variable. Here, this also holds, but the condition on the size of the domains may be violated for an assigned variable because of an assignment in the future. Therefore, firstly arc-consistency must be established on the whole network, and not only on the future variables. Secondly variables are not assigned in a regular way (which is usually equivalent to reducing its domain to the chosen value) but one value is marked as super value, that is added to the current partial solution, and unassigned values are kept in the domain: they are possible repairs so far. The algorithm can be informally described as follows:

- Choose a variable X
- Assign a value $v \in D(X)$ to X, but keep the unassigned values in $D(X)$
- For all $Y \neq X$, backtrack if Y has not at least two supports for v
- Revise the constraints as the MAC algorithm, and backtrack if the size of any domain falls bellow 2.

Super MAC. We give the pseudo code of `super MAC` in Figure 5. This algorithm is very similar to MAC algorithm, the super domains ($\mathcal{SD}$) and repair domains ($\mathcal{RD}$), are both equal to the original domains for the first call. Most of the differences are grouped in the procedure `revise-Dom`. The values are pruned according to the rules described in section 5.4 (loop 1), and the algorithm backtracks if a super domain wipes out or a repair domain becomes singleton (line 2). Note that, as for MAC+, the consistency is also established on the domains of the assigned variables, (`super AC` loop 1).

We have established an ordering relation on the different filterings. However, for the two algorithms above, the process of assigning a value to a variable in the current solution doesn't lead to the same subproblem as in a regular algorithm. Whereas for a regular backtrack algorithm, the domains of the assigned variables are reduced to the chosen value, some unassigned values are still in their domain (but marked only as "repair") for the algorithms above. We have proved that a problem P is AC+ iff $P{+}P$ is AC. However, consider the subproblem P' induced by the assignment of X by MAC+. $P'{+}P'$ may have more than one variable in X, whereas the corresponding assignment in $P{+}P$ leaves only one value in the domain of X (see Figure 6). Therefore the ordering on the consistencies doesn't hold for the number of backtracks of the algorithms themselves. However, MAC($P{\times}P$) never backtracks when any other algorithm doesn't, and MAC+ always backtracks when any other algorithm does. Therefore any solution found by MAC($P{\times}P$) will eventually be found by other algorithms, and MAC+ will find any solution found by another algorithm. We prove that MAC+ is correct and MAC($P{\times}P$) is complete. Hence all four algorithms are correct and complete.

Theorem 3. *For any given CSP P, the sets of solutions of MAC+(P), of super MAC(P), of MAC($P{\times}P$), and of MAC($P{+}P$) are the same and is the set of all super solutions to P.*

Proof. MAC+ is correct: suppose that S is not a super solution, then there exists a variable X assigned to v in S, such that $\forall w \in D(X), v \neq w$, w cannot replace v in S. Therefore when all the variables are assigned, and thus, remain in the domains only the values that are arc-consistent with the current solution, $D(X) = \{v\}$ and then S is not returned by MAC+.
MAC($P{\times}P$) is complete: let S be a super solution, for any variables X, Y, let $v1$ be the value assigned to X in S, and $r1$ one of its possible repairs. Similarly $v2$ is the value assigned to Y and $r2$ its repair. It's easy to see that the pairs $\langle v1, r1\rangle$ and $\langle v2, r2\rangle$ are super arc-consistent, i.e, $\langle v1, v2\rangle$, $\langle v1, r2\rangle$ and $\langle r1, v2\rangle$ are allowed tuples. □

6 Finding the Most Robust Solutions

Finding super solutions may be difficult because (1) from a theoretical perspective, the existence of a backbone variable guarantees the non-existence of super solutions, and (2) from an experimental perspective (see next section), it is quite rare, even if we have no backbone variables, to have super solutions

Algorithm 1: `super MAC`

Data : CSP: $P = \{\mathcal{X}, \mathcal{SD}, \mathcal{RD}, \mathcal{C}\}$, solution: $S = \emptyset$, variables: $\mathcal{V} = \mathcal{X}$
Result : Boolean // $\exists S$ a $(1,0)$-super solution
if $\mathcal{V} = \emptyset$ **then** return True;
choose $X_i \in \mathcal{V}$;
foreach $v_i \in SD_i$ **do**
  save $\mathcal{SD}$ and $\mathcal{RD}$;
  $SD_i \leftarrow \{v_i\}$;
  if `super AC`$(P, \{X_i\})$ **then**
    if `super MAC`$(P, S \cup \{v_i\}, \mathcal{V} - \{X_i\})$ **then** return True;
  end
  restore $\mathcal{SD}$ and $\mathcal{RD}$;
end
return False;

Algorithm 2: `super AC`

Data : CSP: $P = \{\mathcal{X}, \mathcal{SD}, \mathcal{RD}, \mathcal{C}\}$, Stack: $\{X_i\}$
Result : Boolean // P is super arc consistent
while *Stack is not empty* **do**
  pop X_i from Stack;
1 **foreach** $C_{ij} \in \mathcal{C}$ **do**
    switch `revise-Dom`(SD_j, RD_j, SD_i, RD_i) **do**
      case *not-cons*
        return False;
      case *pruned*
        push X_j on Stack;
    endsw
  end
end
return True;

Procedure `revise-Dom`(SD_j, RD_j, SD_i, RD_i) : {pruned,not-cons,nop}

1 **foreach** $v_j \in SD_j$ **do**
  if $\nexists v_i \in SD_i, v_i' \in RD_i$ *such that* $\langle v_i, v_j\rangle \in C_{ij} \wedge \langle v_i', v_j\rangle \in C_{ij} \wedge v_i \neq v_i'$ **then**
    $SD_j \leftarrow SD_j - \{v_j\}$;
  end
end
foreach $v_j \in RD_j$ **do**
  if $\nexists v_i \in SD_i$ *such that* $\langle v_i, v_j\rangle \in C_{ij}$ **then**
    $RD_j \leftarrow RD_j - \{v_j\}$;
  end
end
if *at least one value has been pruned* **then** return pruned;
2 **if** $|SD_j| = 0 \vee |RD_j| < 2$ **then** return not-cons;
return nop;

Fig. 5. super MAC algorithm

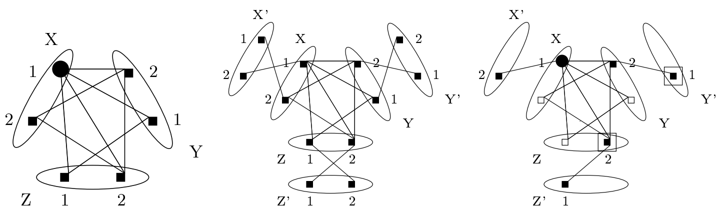


Fig. 6. Left: A CSP P, P is still AC+ after assigning X to 1. Middle: P+P, each variable has a duplicate which must be different, the constraints linking those variables are not represented here, the constraints on X' are exactly the same as the ones on X. Right: When the same assignment, $X = 1$ is done in P+P, we have the following propagation $X' \neq 1 \rightarrow Y \neq 1 \wedge Z \neq 1 \rightarrow Y' \neq 2 \wedge Z' \neq 2$. Now consider $(Y' : 1)$ and $(Z : 2)$ they are not allowed, the network is no longer arc-consistent.

where all variables can be repaired. To cure both problems, we propose finding the "most robust" solution that is as close as possible to a super solution. An optimal solution is defined as a solution that is as close as possible to a super solution.

For a given solution S, a variable is said to be repairable iff there exist at least a value in its domain different from the one assigned in S, and compatible with all other values in S.

The optimal solution is a solution where the number of repairable variables is maximal. Such an optimal robust solution is guaranteed to exist. If the problem is satisfiable, we will have a solution where, in the worst case, none of the variables are repairable. We hope, of course, to find some of the variables are repairable. For example, our experiments show that satisfiable instances at the phase transition and beyond have a core of roughly $n/5$ repairable variables. To find such solutions, we propose a branch and bound algorithm that finds an optimal robust solution. The algorithm implemented is very similar to MAC+ (see 5.6), where arc-consistency is established on the non-assigned as well as on the assigned variables. The current lower bound computed by the algorithm is the number of singleton domains, the initial upper bound is n. Indeed, each singleton domain corresponds to an *unrepairable* variable, since no other value is consistent with the rest of the solution. The rest is a typical branch and bound algorithm. The first solution (or the proof of unsatisfiability) needs exactly the same time as the underlying MAC algorithm. Afterwards, it will continue branching and discovering better solutions. It can therefore be considered as an *incremental anytime algorithm*. We refer to it as super B&B.

For optimization problems, the optimal solution may not be a super solution. We can look for either the most repairable optimal solution or the super solution with the best value for the objective function. More generally, an optimization problem then becomes a *multi-criterion* optimization problem, where we are optimizing the number of repairable variables and the objective function.

7 Experimental Results

Due to the lack of space, this section gives only some observations from our experiments, the interested reader is pointed to the technical report [2]. Our aims were firstly to study the difficulty of finding super solutions, rather than solutions. As expected, for a given density, the phase transition for MAC begins at a tightness much larger than the end of the phase transition for super MAC. That means that a hard problem is very likely not to have a super solution. Moreover, the highest point of the phase transition peak is orders of magnitude greater for finding super solutions than for finding solutions. However, we found that, on the bandwidth problem, for instance, by slightly relaxing the problem (in that case the optimality criterion) super solutions can be found, or alternatively, optimal non-super solutions with a relatively high number of repairable variables.

We present here the comparison between MAC on $P \times P$, super MAC, MAC+ and MAC on $P+P$, on two classes of random problem instances at the phase transition: (50 variables, 15 values, 100 constraints, 114 disallowed tuples per constraint) and (100 variables, 6 values, 250 constraints, 10 disallowed tuples per constraint). Figure 7 gives the cpu time, and the number of backtracks of these algorithms. We observe that (1) MAC on $P \times P$ effectively prunes more than all other methods, but is not practical when the domain size is too big, and (2) super MAC outperforms all other algorithms, in terms of runtime, as soon as the size of the problem increases.

	MAC+	MAC on $P+P$	MAC on $P \times P$	super MAC
$\langle n = 50\ d = 15\ p_1 = 0.08\ p_2 = 0.5 \rangle$				
CPU time (s)	788	43	53	**1.8**
backtracks	152601000	111836	**192**	2047
time out (3000 s)	12%	**0%**	**0%**	**0%**
$\langle n = 100\ d = 6\ p_1 = 0.05\ p_2 = 0.27 \rangle *$				
CPU time (s)	2257	430	3.5	**1.2**
backtracks	173134000	3786860	**619**	6487
time out (3000 s)	66%	7%	**0%**	**0%**

Fig. 7. Results at the phase transition. $*$ only 50 instances of this class were given to MAC+

8 Conclusion

Summary. We have studied the properties of supermodels within a CSP framework. We introduced the notion of super consistency, and based upon it, a search algorithm, super MAC is developed to solve the problem, which outperforms the other methods studied here. We also propose super B&B as an optimzation algorithm which finds the most robust solution that is as close as possible to a super solution.

Related work. Supermodels [8] and fault tolerant solutions [12] have been discussed earlier. Neighborhood interchangeability [6] 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, reparability is a property of the values according to a certain solution (it can be seen either as a property of a variable, or of the value given to this variable into a solution). Therefore those properties are incomparable. For instance, by definition if no solutions exists, then any two values are interchangeable whilst there is no repairable variable. On the other hand consider a CSP with two variables X, Y and the following solutions: $\langle 1,1\rangle, \langle 1,2\rangle, \langle 2,1\rangle, \langle 2,3\rangle$. This problem has two super solutions, $\langle 1,1\rangle$ and $\langle 2,1\rangle$. The value $X = 1$ is repairable in both super solutions by $X = 2$, and vice versa, but neither $X = 2$ is substitutable to $X = 1$, since $\langle 1,2\rangle$ is solution and not $\langle 2,2\rangle$, nor $X = 1$ is substitutable to $X = 2$, since $\langle 2,3\rangle$ is solution and $\langle 1,3\rangle$ is not.

Future directions. 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, as with dynamic CSPs, we wish to consider the loss of n-ary no-goods and not just unary no-goods.

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 csps. Technical Report APES-66-2003, APES Research Group, 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. A sufficient condition for backtrack-bounded search. *Journal of the ACM*, 32:755–761, 1985.
5. E. C. Freuder. Partial Constraint Satisfaction. In *Proceedings IJCAI-89*, pages 278–283, 1989.
6. E. C. Freuder. Eliminating Interchangeable Values in Constraint Satisfaction Problems. In *Proceedings AAAI-91*, pages 227–233, 1991.
7. N. Jussien, R. Debruyne, and P. Boizumault. Maintaining arc-consistency within dynamic backtracking. In *Proceedings CP-00*, pages 249–261, 2000.
8. A. Parkes M. Ginsberg and A. Roy. Supermodels and robustness. In *Proceedings AAAI-98*, pages 334–339, 1998.
9. I. Miguel. *Dynamic Flexible Constraint Satisfaction and Its Application to AI Planning*. PhD thesis, University of Edinburgh, 2001.
10. T. Schiex and G. Verfaillie. Nogood recording for static and dynamic constraint satisfaction problem. *IJAIT*, 3(2):187–207, 1994.
11. T. Walsh. Stochastic constraint programming. In *Proceedings ECAI-02*, 2002.
12. R. Weigel and C. Bliek. On reformulation of constraint satisfaction problems. In *Proceedings ECAI-98*, pages 254–258, 1998.