Improved Algorithm for Finding (a,b)-super Solutions

Student name: Emmanuel Hebrard
Supervisor name: Toby Walsh

NICTA and UNSW, Sydney, Australia
ehebrard@cse.unsw.edu.au

Abstract. Super solutions are a mechanism to provide robustness to constraint programs. We introduce a new algorithm that exploits the similarity between a super solution and its repairs in order to do inference during search. It improves on previous methods since it is more space efficient and also faster in practice.

1 Introduction

The super model framework [4], and its equivalent in constraint programming (super solutions [3]) allow to characterize the notion of fault tolerance. An (a, b)-super solution is a solution in which, if a small number of variables lose their values, we are guaranteed to be able to repair the solution with only a few changes. This concept is useful in dynamic and uncertain environments when robustness is a valuable property. We introduce a new algorithm for finding super solutions that improves upon the method introduced in [2] in several dimensions. This algorithm is more space efficient as it only requires to double the size of the original constraint satisfaction problem. We decompose the problem into a *master problem* and a number of *sub-problems* generated during search. This approach is simple and can be implemented using most of the constraint toolkits currently available. We then show how we can do inference while solving a subproblem to reduce the master problem.

2 Formal background and notations

A constraint satisfaction problem (CSP) P consists of a set of variables $\mathcal{X}$, a set of domains $\mathcal{D}$ such that $\mathcal{D}(X_i)$ is the finite set of values that can be taken by the variable X_i, and a set of constraints $\mathcal{C}$ that specify allowed combinations of values for subsets of variables. We use upper case for variables (X_i) and lower case for values (v). A full or partial instantiation $S = \{\langle X_1 : v_1\rangle, \dots \langle X_n : v_n\rangle\}$ is a set of assignments $\langle X_i : v_j\rangle$ such that $v_j \in X_i$. We will use $S[i]$ to denote the value assigned to X_i in S. A (partial) solution is an instantiation satisfying the constraints. Given a constraint C_V on a set of variables V (its scope), a *support* for $X_i = v_j$ on C is a partial solution involving the variables in V and containing

$X_i = v_j$. A variable X_i is *generalized arc consistent* (GAC) on C iff every value in $\mathcal{D}(X_i)$ has support on C. A constraint C is GAC iff each constrained variable is GAC on C, and a problem is GAC iff all constraints in $\mathcal{C}$ are GAC. Given a CSP P and a subset $A = \{X_{i_1}, \ldots X_{i_k}\}$ of $\mathcal{X}$, a solution S of the restriction of P to A (denoted $P|_A$) is a partial solution on A such that if we restrict $\mathcal{D}(X_i)$ to $\{S[i]\}$ for $i \in [i_1..i_k]$, then P can be made GAC without domain wipe-out.

We introduce some notations used later in the paper. the function $H(S, R)$ is defined to be the Hamming distance between two solutions R and S, i.e., the number of variables assigned to different values in S and R. We also define $H_A(S, R)$ to be the Hamming distance restricted to the variables in A: $H_A(S, R) = \sum_{X_i \in A}(S[i] \neq R[i])$. An *a-break* A on a solution S is a combination of a variables among the variables in S. A *b-repair* of S for A is a solution R such that $H_A(S, R) = |A|$ and $H(S, R) \leq (a + b)$. In other words, R is an alternative solution for S such that if the assignments of the variables in A are forbidden, the remaining "perturbation" is restricted to b variables.

Definition 1. *A solution S is an (a, b)-super solution iff for every $a' \leq a$, and for every a'-break A of S, there exists a b-repair of S for A.*

3 The basic algorithm

We first describe a very simple and basic version of the algorithm without any unnecessary features. Then we introduce some inference rules that make the algorithm more efficient and more effective.

The basic idea is to ensure that the current partial solution is also a partial super solution. In order to do so, we create as many sub-problems as possible breaks for this partial solution, that is $\sum_{i \leq a}\binom{k}{i}$ for a partial solutionover k variables. The solutions to these sub-problems are partial repair solutions. We therefore work on a copy of the original problem that we change and solve for each test of *repairability*. Note that the sub-problem is much easier to solve than the main problem. They are in fact polynomial to solve for fixed a and b. Indeed, since a repair solution must have less than $a + b$ discrepancies with the main solution, the number of possibilities is bounded by $n^{a+b}d^{a+b}$. Furthermore, we shall demonstrate later that we can infer inconsistent values in the master problem from the process of looking for a repair. Pruning the master problem is critical, as it reduces the search tree, and consequently the number of sub-problems to be solved.

Initialization: The input is a CSP, i.e., a triplet $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ and the output a (a, b)-super solution S. We first create a copy P' of P, where $X'_i \in \mathcal{X}'$ iff $X_i \in \mathcal{X}$ and $C' \in \mathcal{C}'$ iff $C \in \mathcal{C}$. This copy will be used to find b-repairs. At any point in the algorithm, $\mathcal{D}(X_i)$ (resp. $\mathcal{D}'(X'_i)$) is the *current* domain of X_i (resp. X'_i) . The set $Past \subseteq \mathcal{X}$ contains all variables that are already bound to a value and we denote $Past'$ the set containing the same variables, but "primed", $Past' = \{X'_i | X_i \in Past\}$.

Main Backtracker: Algorithm 1 searches and backtracks on the main problem P. It is in very similar to a classical backtracker that maintains GAC at

each node of the search tree, except that we also add a call to the procedure `repairability` at each node. Note that any solver or local/global consistency property can be used instead as long as the procedure `repairability` is called. A possible way of implementing `repairability` –in a standard constraint toolkit– can be as a global constraint containing internally the extra data structure P' and an associated specialised solver.

Algorithm 1: `backtrack`$(P, P', S, Past, a, b)$: Bool

```
if Past = 𝒳 then return True;
choose X_i ∈ 𝒳 \ Past;
Past ← Past ∪ {X_i};
foreach v ∈ 𝒟(X_i) do
    save 𝒟;
    𝒟(X_i) ← {v};
    S ← S ∪ {⟨X_i : v⟩};
    if AC-propagate(P) & repairability(P, P', S, Past, a, b) then
        if backtrack(P, S, Past, a, b) then return True;
    restore 𝒟;
    S ← S − {⟨X_i : v⟩};
Past = Past − {X_i};
return False;
```

Enforcing repairability: The procedure `repairability` (Algorithm 2) ensures that each a-break of the solution S has a b-repair. If $|S| = k$ then we check all combinations with maximally a variables in S, that is $\sum_{j \leq a} \binom{k}{j}$ breaks. This expression has no closed form, although it is bounded above by k^a. For each a-break, we model the problem of the existence of a b-repair using P'. Given the main solution S and a break A, we need to find a b-repair, that is, a solution R of $P'|_{Past'}$ such that $H_A(S, R) = |A|$ and $H(S, R) \leq |A| + b$. The domains of all variables are set to their original state. Then for any $X'_i \in A$, we remove the value $S[i]$ from $\mathcal{D}'(X'_i)$, thus making sure that $H_A(S, R) = |A|$. We also add an AtMost$k$Diff constraint that ensures $H(S, R) \leq k$, where $k = |A| + b$. Finally, we solve $P'|_{Past'}$, it is easy to see that any solution is a b-repair.

Algorithm 2: `repairability`$(P, S, Past, a, b)$:Bool

```
foreach A ⊆ Past' such that |A| ≤ a do
    foreach X_i ∈ A do
        𝒟'(X'_i) ← 𝒟(X_i) − {S[i]};
    k ← (|A| + b);
    S' ← solve(P'|_Past' + AtMostkDiff(𝒳', S));
    if S' = nil then return False;
return True;
```

Propagating the AtMost$k$Diff **constraint:**

Definition 2. AtMost$k$Diff$(X'_1, \ldots X'_n, S)$ *holds iff* $k \geq \sum_{i \in [1..n]} (X'_i \neq S[i])$

This constraint ensures that the solution we find is a valid partial b-repair by constraining the number of discrepancies to the main solution to be lower than $a + b$. To enforce GAC on such a constraint, we first compute the smallest expected number of discrepancies to S. Since S is a partial solution we consider

the *possible* extensions of S. Therefore, when applied to the auxiliary CSP P' this number is simply $d = |\{i|\mathcal{D}'(X'_i) \cap \mathcal{D}(X_i) = \emptyset\}|$

We have three cases:

1. If $d < k$ then the constraint is GAC as every variable can be assigned any value providing that all other variables X'_i take a value included in $\mathcal{D}(X_i)$, and we will still have $d \leq k$.
2. If $d > k$ then the constraint cannot be satisfied.
3. If $d = k$ then we can set the domain of any variable X'_i such that $\mathcal{D}(X'_i) \cap \mathcal{D}(X_i) \neq \emptyset$ to $S[i]$.

4 Improvements

The observation that we make in order to improve the search for a super solution is that there must be at least $n-(a+b)$ variables assigned equally in the master problem and any sub-problem. For instance consider the particular case of a $(1,0)$-super solution. Suppose that we are checking the break $A = \{X_i\}$, thus we have $X'_i \neq S[i]$. Moreover, since $n-1$ variables must be assigned as in P, and since X'_i cannot be equal to X_i, we can post $X'_j = X_j$, $\forall j \neq i$. Now consider a value v such that for some $j \neq i$, $X_j = v$ is consistent (for a given local consistency) whilst $X'_j = v$ is not. This is for instance possible if $X_i = S[i]$ was the only support of $X_j = v$. We can prune $X_j = v$ in both the master problem and the sub-problem. This pruning can in turn trigger some propagation, hence reducing the search space of the master problem.

However, as soon as one repair (or more) is allowed, that is $b > 0$, then identifying the variables, for a given sub-problem, that must be assigned the same as in the master problem is more difficult. Indeed, any variable can, at the outset, be assigned differently as in the master problem (this will be the repair). We now explore some ways to deduce equality constraints between a variable and its primed homologue.

The first idea is to enforce a given level of consistency as pre-processing on the sub-problem P'. We will from now on consider that the level of consistency is GAC, however it may be worthwhile to explore stronger consistencies (such as Singleton Arc Consistency [1]). Suppose that after enforcing GAC on P', the least number of discrepancies is exactly $a+b$, that is, $Diff = \{i|\mathcal{D}(X_i) \neq \mathcal{D}'(X'_i)\}$ & $|Diff| = a+b$. We can deduce $\forall j \notin Diff, X'_j = X_j$.

The second idea is that, intuitively, a repair must be *close* to the break in the constraint graph. For instance, in a $(1,1)$-super solution, any "repaired" variable must share a constraint with the "broken" variable (say X). Indeed if it was not the case, then it would mean that all the constraints involving X are satisfied by the solution S for both $X = S[X]$ and $X = v$ for a value $v \neq S[X]$. Moreover, we know that S satisfies all constraints, since it is a solution, therefore, the break $\{X\}$ need no repair at all, a valid alternative is $X = v$. We shall extend this reasoning to any a, b, but first, let us introduce some necessary notation:

A path linking two variables X and Y is a sequence of constraints $C_{V_1}, \ldots C_{V_k}$ such that $i = j+1 \Rightarrow V_i \cap V_j \neq \emptyset$ and $X \in V_1$ and $Y \in V_k$, k is the length of the

path. The distance between two variables $\delta(X, Y)$ is the length of the shortest path between these variables ($\delta(X, X) = 0$). $\Delta_d(X)$ denotes the neighborhood at a distance exactly d of X, i.e., $\Delta_d(X) = \{Y \mid \delta(X, Y) = d\}$. $\Gamma_d(X)$ denotes the neighborhood up to a distance d of X i.e., $\Gamma_d(X) = \{Y \mid \delta(X, Y) \leq d\}$. Similarly, we define the neighborhood $\Gamma_d(A)$ (resp. $\Delta_d(A)$) of a subset of variable A as simply $\bigcup_{X \in A} \Gamma_d(X)$ (resp. $\Delta_d(A)$).

Now we can state the following lemma. Informally, it shows that if there exists a b-repair for a particular a-break A, then all reassignments are within the neighborhood of A up to a distance b.

Lemma 1. *Given a solution S and a set A of a variables, the following equivalence (where R and R' both denote repair solutions for the break A, that is $H(S|_A, R|_A) = a$ and $H(S|_A, R'|_A) = a$) holds:*

$$\exists R \text{ s.t. } H(S, R) < d \Leftrightarrow \exists R' \text{ s.t. } H(S|_{\Gamma_{d-a}(A)}, R'|_{\Gamma_{d-a}(A)}) = H(S, R') < d$$

Proof. We prove this lemma constructively. We start from two solutions S and R that satisfy the premise of this implication and construct R' such that S, R' satisfy the conclusion, the converse is straightforward. We have $H(S, R) = k_1 < k$, therefore exactly k_1 variables are assigned differently between S and R. We also know that $H(S|_A, R|_A) = |A| = a$ therefore only $b = k_1 - a$ are assigned differently outside A. Now we change R into R' in the following way. Let d be the smallest integer such that $\forall X_i \in \Delta_d(A),\ R[i] = S[i]$. It is easy to see that $d \leq b$ as $\Delta_{d_1}(A)$ and $\Delta_{d_2}(A)$ are disjoint iff $d_1 \neq d_2$. We leave all variables in $\Gamma_d(A)$ unchanged, and for all other variables we set $R'[i]$ to $S[i]$. Now we show that R' satisfies all constraints. Without loss of generality, consider any constraint C_V on a set of variables V. By definition, the variables in V belongs to at most two sets $\Delta_{d_1}(A)$ and $\Delta_{d_2}(A)$ such that d_1 and d_2 are consecutive (or possibly $d_1 = d_2$). We have 3 cases:

1. $d_1 \leq d$ and $d_2 \leq d$: any $X \in V$ is assigned as in R, therefore C_V is satisfied.
2. $d_1 > d$ and $d_2 > d$: any $X \in V$ is assigned as in S, therefore C_V is satisfied.
3. $d_1 = d$ and $d_2 = d+1$: any $X \in \Delta_{d_2}(A)$ is assigned as in S, and by definition of R', any $X \in \Delta_{d_1=d}(A)$ is assigned as in S, therefore C_V is satisfied.

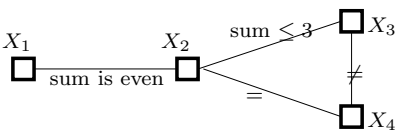
□

Computing this neighborhood can be done as a preprocessing step by a simple breadth first search on the constraint graph, i.e., the graph were any two variables are connected iff they are constrained by the same constraint. The neighborhood $\Gamma_d(A)$ of a break A is recomputed each time, however it just requires a simple union operation over the neighborhood of the elements in A.

Using this lemma we can infer the following equality constraints, given a sub-problem P' for a break A: $\forall i,\ X_i \notin \Gamma_b(A) \Rightarrow X'_i = X_i$.

We give an example of such pruning, consider the problem P:

$X_1 = \{1, 2, 4\}$
$X_2 = \{1, 2\}$
$X_3 = \{1, 2\}$
$X_4 = \{1, 2\}$



It is easy to see that P is arc consistent. Now suppose that we look for a $(1,1)$-super solution, and our first decision is to assign the value 1 to X_1. The domains are reduced so that P remains arc consistent:

$$X_1 = \{1\}, X_2 = \{1\}, X_3 = \{1,2\}, X_4 = \{1,2\}$$

Then we want to make sure that there exists a 1-repair for the break $\{X_1\}$. We then consider P' where $\mathcal{D}(X'_1)$ is set to $\mathcal{D}(X'_1) \setminus \{1\}$. Moreover the constraint AtMost2Diff is posted on $\Gamma_1(\{X_1\}) = (X_1, X_2)$:

$$X'_1 = \{2,4\}, X'_2 = \{1,2\}, X'_3 = \{1,2\}, X'_4 = \{1,2\}$$

Since $P'|_{\{X'_1\}}$ is satisfiable, (for instance, $\{\langle X'_1 : 2\rangle\}$ is a partial solution that does not produce a domain wipe out in any variable of P') we continue searching. However, if before solving P' in order to find a repair we first propagate arc consistency, then we obtain the following domains:

$$X'_1 = \{2,4\}, X'_2 = \{2\}, X'_3 = \{1\}, X'_4 = \{2\}$$

Observe that $2 \in \mathcal{D}(X_3)$ whilst $2 \notin \mathcal{D}(X'_3)$, this means that no repair for X_1 can assign the value 2 to X_3. However, by lemma 1 we can post the following equality constraints: $X'_3 = X_3$ & $X'_4 = X_4$, since $X'_3, X'_4 \notin \Gamma_1(X'_1)$. We can thus conclude prune the value 2 from X_3. In this toy example, this removal will make P arc inconsistent, and therefore we can conclude without searching that X_1 cannot be assigned to 1.

5 Conclusion and Future Work

A preliminary implementation of this algorithm showed considerable improvement upon earlier methods for finding super solutions to the job-shop scheduling problem. Although this method does not yet scale up to large instances, these improvements might prove very useful when applied to the optimization version [2]. In this case, we do not require all breaks to be repairable, but we try to maximize their number. With this approach, we can tackle much larger instances since we start from a regular solution, and then only, improve its *repairability*. We expect this optimization algorithm to profit from the techniques introduced in this paper.

References

1. Romuald Debruyne and Christian Bessière. Some practicable filtering techniques for the constraint satisfaction problem. In *IJCAI'97*, pages 412–417, 1997.
2. E. Hebrard, B. Hnich, and T. Walsh. Robust solutions for constraint satisfaction and optimization. In *Proceedings ECAI'04*, 2004.
3. E. Hebrard, B. Hnich, and T. Walsh. Super solutions in constraint programming. In *Proceedings CP-AI-OR'04*, 2004.
4. A. Parkes M. Ginsberg and A. Roy. Supermodels and robustness. In *Proceedings AAAI'98*, pages 334–339, 1998.