Robust Solutions for Constraint Satisfaction and Optimization

Emmanuel Hebrard*
Cork Constraint Computation Centre
University College Cork, Ireland.
email: e.hebrard@4c.ucc.ie

Abstract

Super solutions are solutions 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. In this paper, we stress the need to extend the super solution framework along several dimensions to make it more useful practically. We demonstrate the usefulness of those extensions on an example from jobshop scheduling, an optimization problem solved through constraint satisfaction. In such a case there is indeed a trade-off between optimality and robustness, however robustness may be increased without sacrificing optimality.

Introduction

Many AI problems may be modeled 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. Supermodels were introduced in (Ginsberg, Parkes, & Roy 1998) as a framework to measure inherent degrees of solution stability. They are propositional satisfiability models such that we are guaranteed to have a close alternative solution in case of breakage. This notion extends to constraint satisfaction: an (a, b)-*super-solution* ((Hebrard, Hnich, & Walsh 2004)) is a solution to a constraint satisfaction problem (CSP) such that 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. We call this new solution (with at most $a + b$ variables reassigned) a *repair solution*. Consider the following CSP:

$$A < B < C \quad A, B, C \in [1..4]$$

$\langle A : 1, B : 2, C : 4\rangle$ is a $(1, 1)$-super-solution since for any loss of value of any single variable, another solution may be found by changing at most one other variable. For instance, a break on A is repaired by the following solution:

$$\langle A : 2, B : 3, C : 4\rangle$$

*supported by Science Foundation Ireland and an ILOG grant.

The problem of determining the existence of a super-solution has been proved to be NP-complete for a fixed a (Hebrard, Hnich, & Walsh 2004), as well as for supermodels of SAT theories (Ginsberg, Parkes, & Roy 1998).

Our aim is to be able to "constrain" a problem in a such way that a solution has the good properties of robustness we need. The approach taken in (Hebrard, Hnich, & Walsh 2004) was to explore in depth the simplest case, namely $(1, 0)$-super-solutions, i.e., solutions where the loss of a value taken by any variable may be repaired (the variable is reassigned) without changing the rest of the solution. An efficient algorithm based on the notion of super-consistency was introduced. The next step is to bring all (a, b)-super-solutions to the constraint satisfaction framework. Furthermore, in practice, we may know how the problem is likely to break, or how we may repair it. We explore here some extensions of the framework to take this knowledge into account. We classify those extentions as follows:

- **Break and repair set:** restrictions on the set of variables that may break, or may participate to a repair.
- **Alternative value:** restrictions on the alternative value given to a "broken" variable.
- **Value and variable:** restrictions on the reassignments required when "repairing" a break, either concerning the values or the variables.

Algorithm and Extensions

Constraint solving often involves search and inference. Local consistency is used to prune values in future variables' domains and and backtrack when a domain wipe-out occurs. We introduce an algorithm for finding (a, b)-super-solutions based the Maintain Arc-Consistency (MAC) algorithm (Sabin & Freuder 1994)(Bessiere & Regin 1996). On top of MAC's backtrack search, our algorithm fails also if the current partial assignment is not "repairable",i.e., the partial assignment is not a super-solution of the CSP restricted to the assigned variables, thus cutting branches which wouldn't have been cut by MAC. This algorithm returns both the super-solutions and the repair solutions.

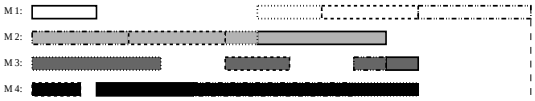
However, super-solutions alone are unable to model all the criteria of robustness one might want a solution to possess. For instance, consider the jobshop scheduling problem (jsp). A jsp involves a set of jobs and a set of machines. Each job is a sequence of activities, where each activity has

a duration for its execution on one of the machines. The objective is to schedule the activities such that their order is respected, no machine is required by two activities that overlap, and the makespan, i.e., the interval from the starting point of the first activity, and ending point of the last, is minimized.

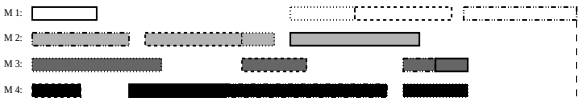
A jsp can be formulated as a CSP, with one variable for each activity, and a domain size equal to the makespan minus its duration. To find the minimal makespan, one starts with the makespan equal to a lower bound and increase it till a solution exists.

The following figure represents a jsp with 4 jobs and 4 machines. Bars are activities, organized in jobs. Their length are proportional to their duration. Each job's activities have the same border (respectively solid, dashed, dotted, dash and two dots), whilst colors correspond to machines.

The second figure shows an optimal solution.



Activities are now scheduled on each machine. One may argue that this solution is not robust. Indeed activities are tightly grouped and a "break" on a machine, and the subsequent delay, may trigger further delays in the schedule. Now, consider a $(1, 0)$-super-solution for this instance:



It's important to say that until now no assumption were made about the problem solved, and about what might be a robust solution to that problem. Nevertheless, with a small increase in the makesapn, we observe that a super-solution for a jsp contains more slacks between activities, making it more robust. However, this is not really satisfying. Indeed, the values are time points, the breakages are known only when they occur, thus we might want to change only events in the future. The next figure shows a super-solution with this (alternative value) restriction:

This solution is clearly more robust than the first one. However, the makespan increase is relatively important. One alternative, which ensures a smaller makespan is to optimize the *repairability*, i.e, the proportion of repairable variables. For instance, suppose that we don't want the makespan to increase by more than two time units. The next figure shows a solution with optimal repairability (i.e., more robust) whilst the makespan length is restricted:

There are a number of such improvements upon classical super-solution that we can gather from our knowledge of the problem. For instance, some machines may be reliable, one would then restrict the break-set accordingly. Moreover, the slacks should be linked to the duration for repairing a particular machine, this can be done by restricting the alternative values.

Another example is the instruction scheduling problem, where primitive instructions of a program are to be scheduled in order to minimize the time complexity of the program while keeping its semantic. When we model this problem, a variable can take the value "NOP", which stands for "no operation". Whilst a breakage may happen on any variable, one assigned to "NOP" cannot break. When some variables are set to such an assignment, no witness of repairability is required. Note that super-solutions offers no real difficulties to adapt to the optimization framework. However, it's not surprising that one has to sacrifice optimality in order to achieve robustness. An alternative is to optimize the *repairability* of a solution, that is the proportion of repairable breaks, without giving up optimality, or at least bounding the loss. More generally we have now a multi-criteria optimization problem.

Conclusion

We claim that super-solutions offer a useful framework to obtain more robust solution to constraint satisfaction or optimization problems. They are often a good starting point for characterizing robustness, without requiring any knowledge of the problem. However, we showed that integrating specific information about a problem can lead to much better results. In our future work we plan to define and implement a library of useful predicates and optimization criteria that can be used and combined in order to easily post robustness constraints. Our algorithm would simply use the appropriate predicates to check the validity of any repair. Then we would like to use this framework to obtain robust solutions of real problems subject to uncertainty.

References

Bessiere, C., and Regin, J.-C. 1996. MAC and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problems. In *Proceedings CP'96*, 61–75.

Ginsberg, M.; Parkes, A.; and Roy, A. 1998. Supermodels and robustness. In *Proceedings AAAI'98*, 334–339.

Hebrard, E.; Hnich, B.; and Walsh, T. 2004. Super solutions in constraint programming. In *Proceedings CPAIOR'04 (to appear)*.

Sabin, D., and Freuder, E. 1994. Contradicting Conventional Wisdom in Constraint Satisfaction. In Borning, A., ed., *Proceedings CP'94*, volume 874, 10–20.