Constraint Relaxation Techniques to Aid the Reuse of Knowledge Bases and Problem Solvers

Tomas Nordlander[1], Ken Brown[2], and Derek Sleeman[1]
[1]Department of Computing Science, University of Aberdeen, Scotland, UK
{tnordlan, sleeman}@csd.abdn.ac.uk
[2]Cork Constraint Computation Centre, Department of Computer Science, University College Cork, Ireland
k.brown@cs.ucc.ie

Abstract: Effective re-use of knowledge bases requires the identification of plausible combinations of both problem solvers and knowledge bases, which can be an expensive task. Can we identify impossible combinations quickly? The capabilities of combinations can be represented using constraints, and we propose using constraint relaxation to help eliminate impossible combinations. If a relaxed constraint representation of a combination is inconsistent then we know that the original combination is inconsistent as well. We examine different relaxation strategies based on constraint graph properties, and we show that removing constraints of low tightness is an efficient strategy which is also simple to implement.

1. Introduction

The MUSKRAT (Multistrategy Knowledge Refinement and Acquisition Toolbox) framework aims to unify problem solving, knowledge acquisition and knowledge-base refinement in a single computational framework [1]. Given a set of Knowledge Bases (KBs) and Problem Solvers (PSs), the MUSKRAT-Advisor [2] investigates whether the available KBs will fulfil the requirements of the selected PS for a given problem. The Advisor informs the user if the available KBs are sufficient. Our research addresses the problem of checking whether combinations of existing KBs could be reused with the selected PS. We propose to represent the KBs and PSs as Constraint Satisfaction Problems (CSPs), which can be combined to produce composite CSPs. If a combined CSP is solvable, then the original combination of KBs with the selected PS could be used to solve the given problem; if the resultant CSP is inconsistent, then the combination cannot be used. Identifying a suitable combination thus requires examining a series of CSPs, and rejecting insolvable ones until we find one with a solution. Proving CSPs insolvable can be a lengthy process; we would like to find a way to do this quickly. The method we propose here is to relax a CSP, and if we can prove that the relaxed version is insolvable then we know that the original CSP does not have a solution either. However, if the relaxed CSP has a solution, then the original CSP represents a *plausible* combination. To test this proposal, we investigate different relaxation strategies for binary CSPs and test them on randomly generated problems. We suggest that removing constraints with low tightness is an effective method for identifying insolvable combinations. Thus this paper reports a contribution to the challenging problem of Knowledge Reuse as it presents an aid based on Constraint Programming to enable a quick identification of inconsistent combinations.

2. Background

This work is supported by the Advanced Knowledge Technologies (AKT) Interdisciplinary Research Collaboration, which focuses on six challenges to ease substantial bottlenecks in the engineering and management of knowledge; reuse of knowledge is one of those challenges [3]. Current work in reuse has resulted in systems where a number of components have been reused, including ontologies, problem-solving methods (PSMs), and knowledge bases (KBs) [3]. The use of cases in Case Based Reasoning is a related activity [4].

2.1 Reusing Knowledge Based Systems

One of the main goals of the Knowledge Based System (KBS) community is to build new systems out of existing Problem Solvers (say Problem Solving Methods) and existing Knowledge Bases. At an early stage the Knowledge Engineering sub-area identified a range of Problem Solving Methods, which they argued covered the whole range of problem solving and included methods for Classification and Diagnosis through to Planning (so-called synthesis tasks) [5]. An early but powerful example of reuse of a PSM was the use of the EMYCIN shell with a variety of domain-specific knowledge bases [6]. More recently, systems like PROTÉGÉ have provided an option to write KBs in a standardised format like OKBC [7]. This then takes the goal of building new Knowledge-Based Systems (KBSs) one step further. A possible approach is to take the required KB and PSM and to produce manually, initially, a series of mappings, which will make the 2 components "comparable" [7], and thus to develop a new KBS from pre-existing components.

We have chosen to work with the domain of scheduling, mainly because the constraint community has been successful in addressing real world problems in this area. Also, we argue that the nature of scheduling problems are very close to CSPs, hence it would be relatively easy to transform PSs and KBs in this domain. We will now consider an example of a mobile phone manufacturer to understand our notion of KB and PS reuse. The manufacturer has two factories, three suppliers and two delivery companies to transport phones to wholesalers around the world (Figure 1).

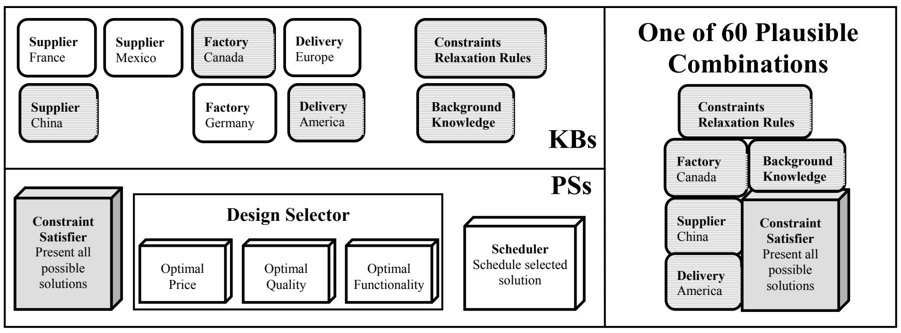

Figure 1. Combining KBs with selected PS

Along with the domain-specific KBs, the system also consists of background knowledge (e.g., ISO 9001, Safety Standards), and constraint relaxation rules to identify those constraints, which the system is allowed to relax (e.g. accept a screen with a maximum of 64 colours instead of 4096). This toy example has 5 PSs and 7 KBs (for each of the factories, suppliers, and delivery companies) to combine as well as the obligatory KBs about background knowledge and constraint relaxation rules (See Figure 1). To solve a task one needs to have: 1 Problem Solvers (out of 5), 1 supplier (out of 3), 1 factory (out of 2), 1 delivery company (out of 2), background knowledge and constraint relaxation. Since the number of possible combination is 60 (5*3*2*2*1*1=60), we would like to eliminate some of the insolvable combinations quickly, leaving a couple of plausible combinations, which need to be evaluated thoroughly. Let us assume that the manufacturer would like, if possible, to reuse knowledge to answer questions such as: Can we manufacture, within the guidelines of ISO 9001 and European safety standards (CE), a mobile phone with a 4096 colour screen, not heavier then 100g and have it delivered to the American market within 6 months?

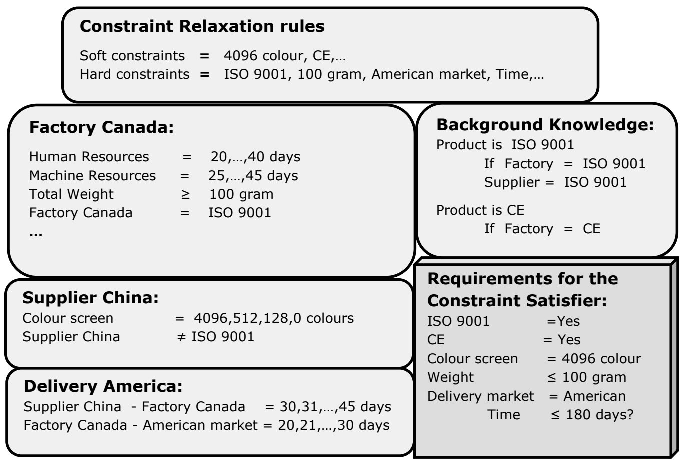
Constraint Relaxation rules
Soft constraints = 4096 colour, CE,…
Hard constraints = ISO 9001, 100 gram, American market, Time,…
Factory Canada:
Human Resources = 20,…,40 days
Machine Resources = 25,…,45 days
Total Weight ≥ 100 gram
Factory Canada = ISO 9001
…
Background Knowledge:
Product is ISO 9001
If Factory = ISO 9001
Supplier = ISO 9001
Product is CE
If Factory = CE
Supplier China:
Colour screen = 4096,512,128,0 colours
Supplier China ≠ ISO 9001
Requirements for the Constraint Satisfier:
ISO 9001 =Yes
CE = Yes
Colour screen = 4096 colour
Weight ≤ 100 gram
Delivery market = American
Time ≤ 180 days?
Delivery America:
Supplier China - Factory Canada = 30,31,…,45 days
Factory Canada - American market = 20,21,…,30 days

Figure 2. One of 60 Plausible Combinations

Figure 2 illustrates one candidate combination, as well as the requirements for the selected problem solver. With the knowledge and problem solving requirements in Figure 2 we can demonstrate that the combination is inconsistent, as one of the requirements given to the PS, viz., ISO 9001 certified, our background knowledge states that to make a phone certified, both the manufacture and the part supplier need to be ISO 9001 certified. Figure 2 shows that only the factory is certified hence the combination is inconsistent.

White & Sleeman [1] discussed the creation of Meta Problem Solvers, which check if combinations of KBs with a PS, represented as a CSP, are plausible. However, this approach was unable to verify if any discarded combination contained a solution. As noted earlier, we propose to use constraint programming as our PSM and to represent combination of KBs and PSs as CSPs. We will show that some of our strategies can not only identify insolvable combinations quickly, but also verify that their related CSPs do not have any solutions.

2.2 CSP

Constraint Satisfaction techniques attempt to find solutions to constrained combinatorial decision problems [8, 9] , and there are a number of efficient toolkits and languages available (e.g. [10, 11]). The definition of a constraint satisfaction problem (CSP) is:

- a set of variables $X=\{ X_1,..., X_n\}$,
- for each variable X_i, a finite set D_i of possible values (its domain), and
- a set of constraints $C_{<j>} \subseteq D_{j1} \times D_{j2} \times \ldots \times D_{jt}$, restricting the values that subsets of the variables can take simultaneously.

A solution to a CSP is an assignment of a value from its domain to every variable, in such a way that all constraints are satisfied. The main CSP solution technique interleaves consistency enforcement [12], in which infeasible values are removed from the problem, with various enhancements of backtracking search. The same approach also serves to identify insolvable problems.

The approach of relaxing CSPs has received considerable attention [13-15], but has focused on changing the CSP to introduce solutions. Relaxing problems is also a common technique in mathematical programming, e.g. to obtain better bounds in optimisation [16]. There has been extensive research on randomly generated CSPs [13, 17], which show a phase transition between solvable and insolvable problems, with the hardest problems being at the transition point. Problem classes in these CSPs can be described by a tuple <n,m,c,t>, where n is the number of variables and m is the number of values in each domain, c is the number of constraints, and t is the number of forbidden tuples in each constraint (a measure of problem tightness). Much of this research concentrates on binary CSPs, in which each constraint is defined over a pair of variables. Any CSP with finite domains can be modelled by a CSP with only binary constraints [8, 18].

3. Empirical Studies

The aim of relaxing CSPs is to produce new CSPs, which are easier to prove inconsistent. It is not obvious that relaxing an insoluble CSP will produce an easier problem. In fact, phase transition research (e.g. [19]) seems to indicate the opposite when the original CSP is inconsistent – as constraints become more loose, or the connectivity of the problems become more sparse, the time to prove inconsistency for random problems increases. If our approach is to work, we need to identify suitable relaxation strategies, which avoid the increase in solution time. In this section, we describe a series of experiments designed to test whether our approach is practical and to identify suitable relaxation strategies. For this paper, we limit the experiments to randomly generated binary CSPs, with a view to extending the results to real world problems. First we will describe the software, and then present the experiments. For the experiments, we generate a set of random problems, prove them inconsistent by search, and then apply various relaxation strategies and measure the reduction in search effort. The experiments are divided into three groups according to the distribution of the random problems.

3.1 CSP-Suite

The CSP-Suite [20] used in these experiments is written in SICStus Prolog [11] and consists of Generating, Relaxing, and Solving modules (Figure 3).

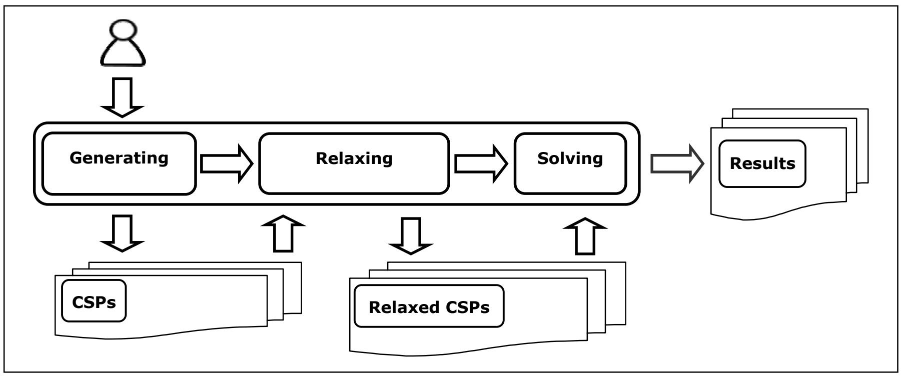

Figure 3. The CSP-Suite

The *Generating* module creates random binary CSPs. Our random CSPs are slightly different from most reported in the literature. We want to find local properties of a problem which indicate how to relax it, so rather than have every constraint in a problem with the same tightness, we instead allow the tightness to vary. This also produces random CSPs which are closer to real world problems. The tightness of individual constraints in our problems will be uniformly, normally, or exponentially distributed. Thus, for different tightness distributions we use different ways to describe the tightness tuple. The uniform problem classes have 5-tuples $<n,m,c,[t_\mu,r]>$, where t_μ is the average number of forbidden tuples, with the number for each constraint selected uniformly at random from the range $[t_\mu\text{-}r,\ t_\mu\text{+}r]$. The normal problem classes introduce standard deviation (sd) in our tightness tuple; $<n,m,c,[t_\mu,sd,r]>$. The standard deviation parameter controls the width of the bell curve. The exponential distribution requires a somewhat different notation: $<n,m,c,[t_m,stp,r]>$, where t_m (tightness middle) shows the middle of the distribution range, which is *not* the same as average distribution. The parameter *stp* has an effect on the steepness of the exponential probability curve. Even though only a positive exponential distribution has been tested in this paper (many low tightness constraints that decay exponentially to few high tightness constraints), it is also possible to generate a negative exponential tightness distribution. First we create a skeleton graph by successively adding nodes, then we randomly add constraints until we reach “c”, then take each constraint in turn, decide how many tuples it should forbid, and then randomly remove that number of tuples.

The *Relaxing* module generates relaxed CSPs from original CSPs by removing a specified number of constraints according to nine different strategies. Random Removal simply chooses the constraints randomly. Greedy Search considers each constraint in turn, removing it, solving the relaxed CSP, and replacing the constraint. It then selects the constraint whose removal gave the best performance improvement, removes it, and repeats the whole process on the resulting CSP. Greedy Ordering uses the first iteration of Greedy Search to generate an ordering list for all constraints then removes constraints in the order suggested. Node Degree selects constraints in ascending or descending order of the degree of their associated variables in the constraint graph. Isolate Node selects high or low degree nodes and removes a series of constraints incident on those nodes (i.e. it tries to remove variables from the problem). Tightness removes constraints in ascending or descending order of their tightness. Note that the two Greedy strategies would not be applicable in our eventual framework, since they must solve many CSPs each time. They are useful here as reference points showing what might be achievable. We also used the results of the Greedy search to analyse the properties of the most profitable removed constraints, and from that developed some of the other strategies.

The *Solving* module simply solves the CSPs using the finite domain library [21], and records search effort each time. Since the library does not report constraint checks, we use the resumption statistic instead. We have confirmed that resumptions correlate well with CPU time [22].

3.2 CSPs with Uniformly Distributed Tightness

First, we consider problems with a uniform distribution of tightness. In Figure 4, for the problem class <20,10,133, [65, 15]>, we show the resumption profit achieved (i.e. how much easier it is to solve the relaxed CSP compared to the original) by each strategy, removing up to 60 constraints each time. The problem class is in the over-constrained region, and in all cases we considered, remained over-constrained even after removing the constraints. The graph shows that removing low-tightness constraints is the most profitable of the applicable strategies, for this problem class. We assume that although such constraints are easy to satisfy, they are likely to be redundant when showing there is no solution, since they rule out so few combinations, and thus they introduce many extra branches into the search tree for no benefit.

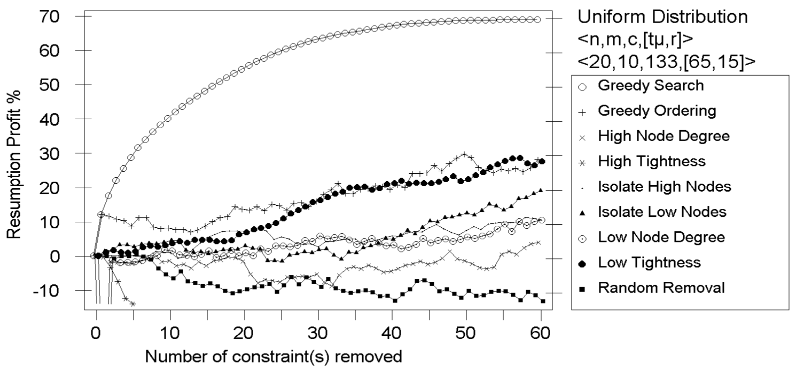

Figure 4. Relaxation Strategies when removing up to 60 constraints

We then concentrate on the Low Tightness strategy, and Figure 5 illustrates its effect on 4 different problem classes. Whilst the graph shows a negative result for the <20,10,133,[45,15]> curve, the others are positive. In the best case we can create relaxed CSPs that are up to 45% easier to solve than the original CSPs, still without introducing a solution. The graphs suggest that the Low Tightness Removal strategy works better on CSPs with high tightness and wide distribution.

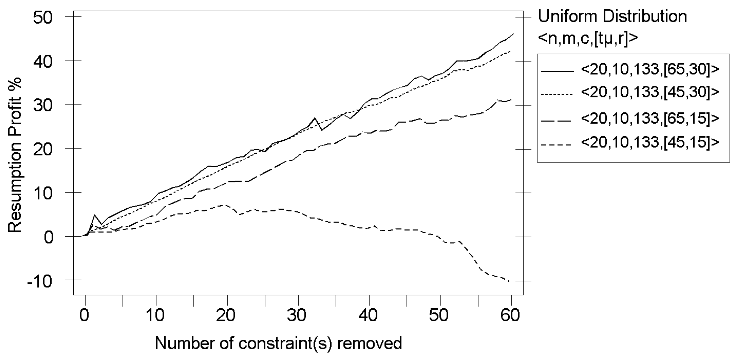

Figure 5. Low Tightness Removal Strategy when removing 60 constraints

As discussed at the start of section 3, strategy (1), randomly removing constraints from a randomly generated inconsistent CSP, is likely to make it harder to prove inconsistency. Does the same effect happen for Low Tightness Removal? I.e., is there a point at which removing constraints is no longer profitable? In Figure 6 we plot the search effort against the number of constraints in problems obtained by relaxing an original problem using Low Tightness Removal, and we include Random Removal for comparison. We start with a particular problem class on the right-hand side of the curve, remove constraints according to each strategy, and solve the new problem after each removal. In three cases, we avoid any significant hardness peak with the Low Tightness strategy, and for the fourth a peak appears only after removing approximately 40 constraints. That peak is further to the left than for Random Removal, and the peak normally coincides with the solubility transition. Figure 7 shows the transition curves, and we can see that in four cases, the transition point for Low Tightness Removal appears later (further left) than for Random Removal. This gives us some confidence that we can use our relaxation strategy reliably without introducing new solutions.

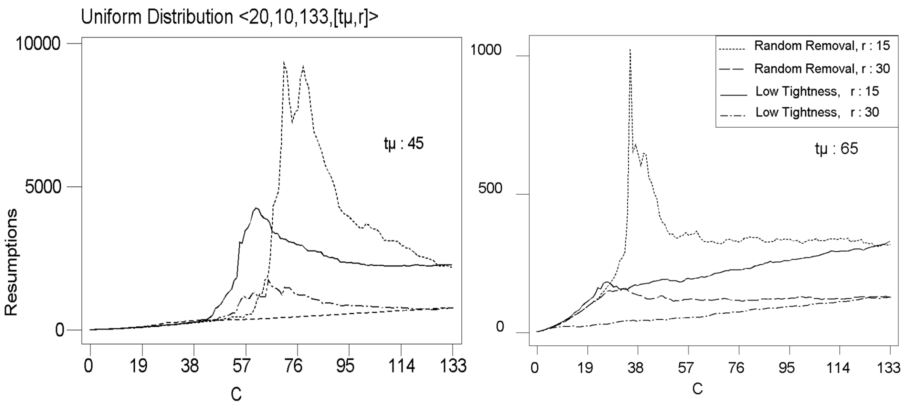

Figure 6. Search effort for Random Removal vs. Low Tightness Removal

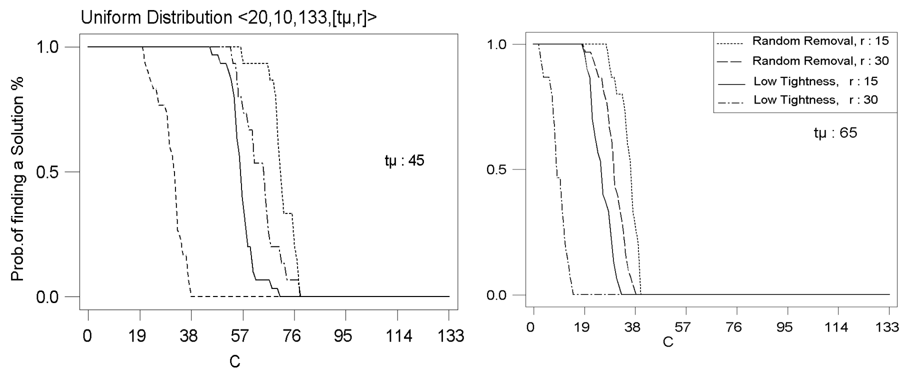

Figure 7. Transition phase for Random Removal vs. Low Tightness Removal

3.3 CSPs with Normally and Exponentially Distributed Tightness

We now repeat the experiments on problems with normally distributed tightness (Figures 8 to 11) and with exponentially distributed tightness (Figures 12 to 15). Although the graphs suggest that the results of the low tightness strategy are slightly better for CSPs with a uniformly distributed tightness that have a high average and a wide distribution range, it is evident from figures 8 &12 that low tightness is still the best strategy for all distributions.

There are only two differences in the results for these distributions when compared to Uniform. Firstly, as seen in figure 10, there are only two cases instead of three where we can avoid any significant hardness peak when relaxing using the Low Tightness strategy. Secondly, when we compare the Random curves (Figure 10 & 14) we notice that the hardness peak is not only a bit wider but also slightly higher than the uniform distribution graphs in section 3.2. Still, for all cases, the transition point for low tightness is later than for random (see Figures 11 & 15), which leaves us with some confidence that we can use our relaxation strategy reliably without introducing new solutions even when the tightness is exponentially distributed.

Note that the reason that we use a different standard deviation (*sd*) for problem classes with a different range is to obtain an equal bell-shaped distribution form for all our problem classes (Figure 9-11). Additionally, in order to achieve a similar exponential decay for the problem classes with different range values we use a different steepness (*stp*) (Figure 13-15).

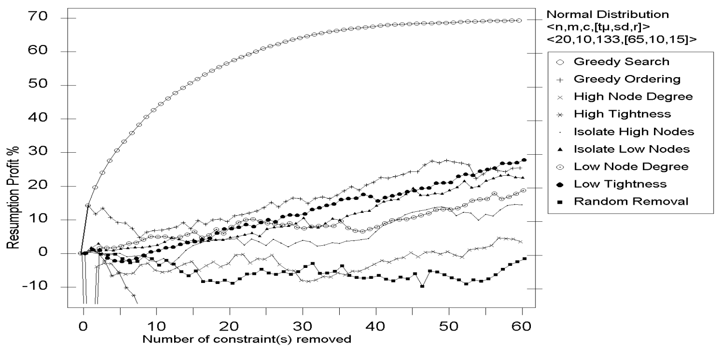

Figure 8. Relaxation Strategies when removing up to 60 constraints

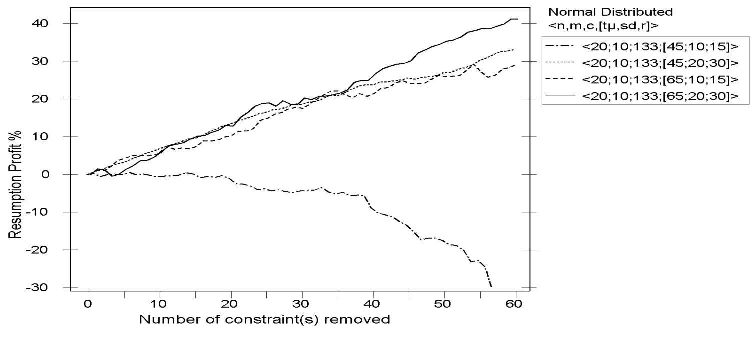

Figure 9. Low Tightness Removal Strategy when removing 60 constraints

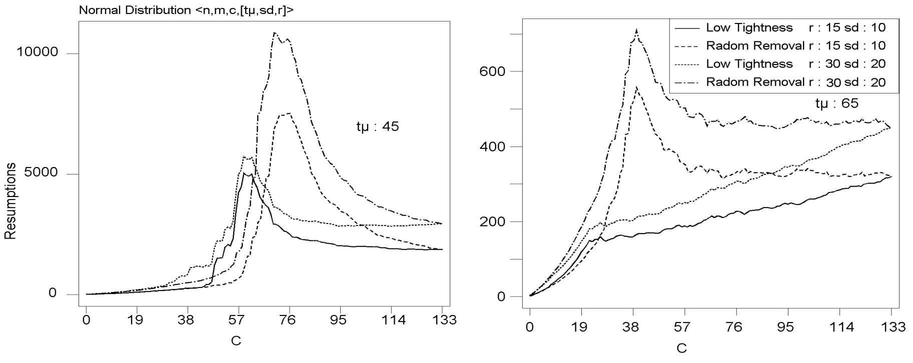

Figure 10. Search effort for Random Removal vs. Low Tightness Removal

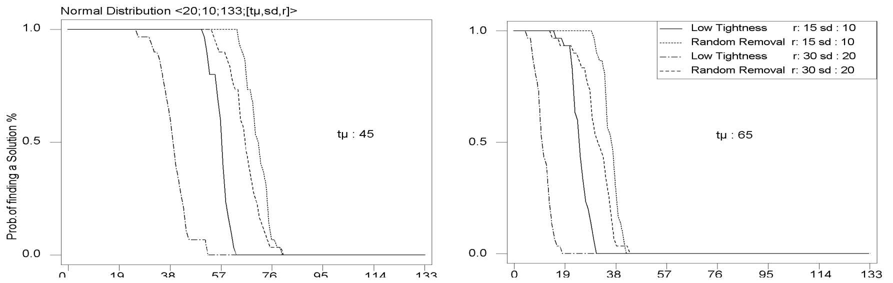

Figure 11. Transition phase for Random Removal vs. Low Tightness Removal

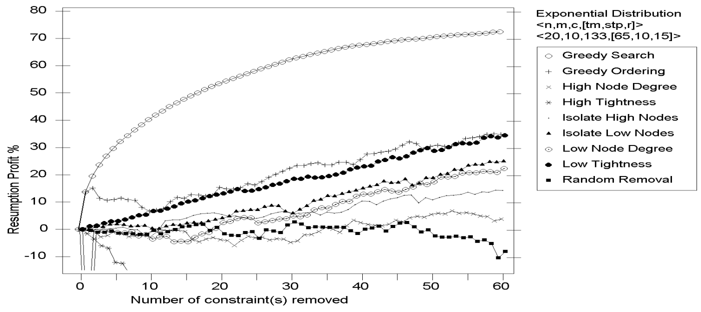

Figure 12. Relaxation Strategies when removing up to 60 constraints

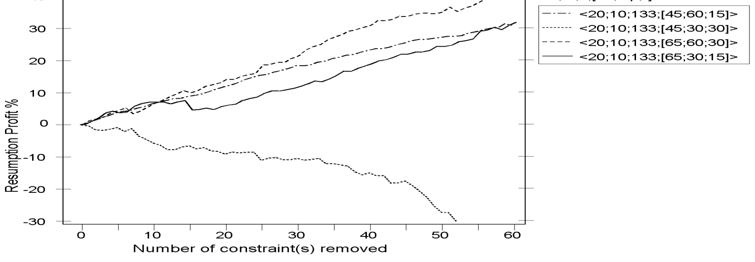

Figure 13. Low Tightness Removal Strategy when removing 60 constraints

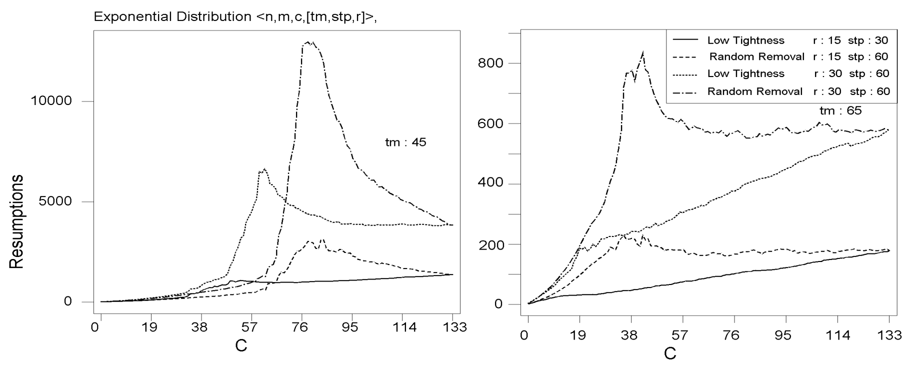

Figure 14. Search effort for Random Removal vs. Low Tightness Removal

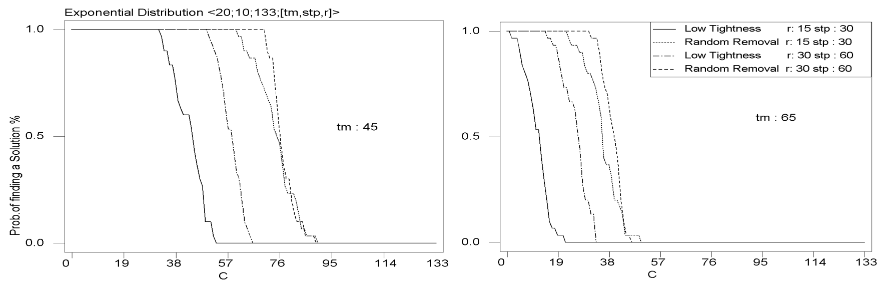

Figure 15. Transition phase for Random Removal vs. Low Tightness Removal

4. Future Work

In this paper, we have tested the effectiveness of constraint relaxation for quickly identifying inconsistent random CSPs with uniform, normal and exponential tightness distributions. Our original aim was to apply this technique to the problem of knowledge base reuse. Therefore, the main focus of future work will be on extending this approach to more realistic CSPs. We will focus on scheduling problems. These are likely to involve non-binary and global constraints, and constraint graphs with particular properties (e.g. [23]). We will also carry out investigations into theoretical CSP concepts, including higher consistency levels and problem hardness. One of our primary objectives was to detect combination of problem solvers (PSs) and knowledge bases (KBs), which are inconsistent. So we plan to take combination of PS and KBs and systematically explore the effect of relaxing one or more constraints. Thus when a new task, is encountered we will search the case-base for the nearest case(s) and perform the actions which these cases(s) suggest. We hope this approach will quickly and efficient lead us to detecting consistent and inconsistent PS-KBs combinations. As a related activity, we plan to also investigate scheduling ontologies [24, 25], to help characterise problems in terms of specific domain entities and relationships. We hope these studies will help us to further understand the problem of KB reuse, this being one of our original aims.

5. Summary

This paper has briefly discussed reuse of KBs and PSs, and described our method to alleviate part of this challenging issue. We propose to represent each KB and each PS as a Constraint Satisfaction Problem (CSP), which can be combined to obtain larger CSPs. In parallel to this theoretical discussion, we have conducted experiments on creating and relaxing binary CSPs with 9 different relaxation strategies. We have shown that our idea of relaxing CSPs can be useful for detecting inconsistency. In particular, we have shown that the simple strategy of removing constraints of low tightness is effective, and reduces the time to detect inconsistency. We will apply these results to determine whether combinations of KBs and a PS are plausible when reusing knowledge-based components for real world problems.

6. Acknowledgements

This work is supported under the EPSRC's grant number GR/N15764 and the Advanced Knowledge Technologies Interdisciplinary Research Collaboration, which comprises the Universities of Aberdeen, Edinburgh, Sheffield, Southampton and the Open University. We thank Mats Carlsson at SICTStus for helpful discussions about resumption, constraint checks and search effort.

7. References

1. White, S. & Sleeman, D., A Constraint-Based Approach to the Description & Detection of Fitness-for-Purpose, *ETAI*, vol. 4, pp. 155-183, 2000.
2. White, S. & Sleeman, D., Providing Advice on the Acquisition and Reuse of Knowledge Bases in Problem Solving, 11th Banff Knowledge Acquisition Workshop. SRDG Publications, Calgary Canada, 1998, pp. 21.
3. AKT, Reuse Knowledge, [WWW], Available from: http://www.aktors.org/publications/reuse/, [Accessed 1 June 2003].
4. Kolodner, J., *Case-Based Reasoning*. San Mateo, CA: Morgan Kaufmann, 1993.
5. Hayes-Roth, F., Waterman, D. & Lenat, D., *Building Expert Systems*. London: Addison-Wesley, 1983.
6. Bennett, J. & Engelmore, R., Experience using EMYCIN. In Rule-Based Expert Systems, in *Experience using EMYCIN. In Rule-Based Expert Systems*, B. Buchanan and E. Shortliffe, Eds. London: Addison-Wesley, 1983, pp. 314-328.
7. Gennari, J. H., Musen, M. A., Fergerson, R., et al., The Evolution of Protégé: An Environment for Knowledge-Based Systems Development., *International Journal of Human-Computer Studies*, vol. 58, pp. 89-123, 2003.
8. Barták, R., Online Guide to Constraint Programming, [WWW], Available from: http://kti.ms.mff.cuni.cz/~bartak/constraints/binary.html, [Accessed March 2003 1998].
9. Tsang, E., Foundations of Constraint Satisfaction. London & San Diego: Academic Press, 1993.
10. ILOG Solver, 5.3 ed. Paris: ILOG Inc., http://www.ilog.com/, 2003

11. SICStus Prolog, 3.9.1 ed. Kista: Swedish Institute of Computer Science, http://www.sics.se/sicstus/, 2001
12. Freuder, E., Synthesizing constraint expressions, CACM, 1978, pp. 958-966.
13. Freuder, E. & Wallace, R., Partial Constraint Satisfaction, *Artificial Intelligence*, vol. 58, pp. 2170, 1992.
14. Bistarelli, S., Montanari, U. & Rossi, F., Constraint Solving over Semi-rings, IJCAI'95, 1995, pp. 624--630.
15. Schiex, T., Fargier, H. & Verfaillie, G., Valued Constraint Satisfaction Problems: hard and easy problems, IJCAI'95, 1995, pp. 631-637.
16. Hooker, J., Logic-based methods for optimization : combining optimization and constraint satisfaction. New York, 2000, pp. 209.
17. MacIntyre, E., Prosser, P., Smith, B., et al., Random Constraint Satisfaction: Theory meets Practice, CP-98, 1998, pp. 325-339.
18. Bacchus, F., Chen, X., Beek, P. v., et al., Binary vs. non-binary constraints, *Artificial Intelligence*, vol. 140, pp. 1-37, 2002.
19. Prosser, P., Binary constraint satisfaction problems: Some are harder than others, Proceedings ECAI-94 (11th European Conference on Artificial Intelligence), 1994, pp. 95-99.
20. CSP-Suite, 1.90 ed. Aberdeen: University of Aberdeen, http://www.csd.abdn.ac.uk/~tnordlan/Proglog%20programs, 2002
21. SICStus, Constraint Logic Programming over Finite Domains, in *SICStus Prolog User's Manual*, vol. Release 3.9.1, R. 3.9.1, Ed. Kista: Intelligent Systems Laboratory Swedish Institute of Computer Science, 2002, pp. 345-381.
22. Nordlander, T., First Year Report, Aberdeen University, 2002.
23. Walsh, T., Search on high degree graphs, IJCAI-2001, 2001, pp. 266-274.
24. Rajpathak, D., Motta, E. & Roy, R., The Generic Task Ontology For Scheduling Applications, International Conference on Artificial Intelligence, Las Vegas, 2001.
25. Smith, S. F. & Becker, M. A., An Ontology for Constructing Scheduling Systems, *AAAI Symposium on Ontological Engineering*, Mars 1997.