

Knowledge Base Reuse Through Constraint Relaxation

Tomas Eric Nordlander*

Cork Constraint Computation
Centre, Department of Com-
puter Science, University
College Cork, Ireland
tnordlan@4c.ucc.ie

Derek Sleeman

Department of Computing
Science, University of
Aberdeen, Scotland, UK
sleeman@csd.abdn.ac.uk

Ken N Brown

Cork Constraint Computation
Centre, Department of Com-
puter Science, University
College Cork, Ireland
k.brown@cs.ucc.ie

ABSTRACT

Effective reuse of Knowledge Bases (KBs) often entails the expensive task of identifying plausible KB-PS (Problem Solver) combinations. We propose a novel technique based on Constraint Satisfaction to enable more rapid identification of incompatible KBs, leaving fewer combinations on which to conduct a thorough investigation. In this paper, we describe our investigation process, its tools, and the latest empirical results applied to non-binary problems that demonstrate our relaxation approach is an effective method for plausibility testing.

Categories and Subject Descriptors

I.2.1 Applications and Expert Systems—Industrial automa-
tion. I.2.8 Problem Solving, Control Methods, and
Search—Graph and tree search strategies.

Keywords

Knowledge Base Reuse, Constraint Satisfaction Problem,
Relaxation Techniques, Scheduling.

INTRODUCTION

Knowledge Engineering is often a time-consuming and expensive process, particularly if it involves acquiring new knowledge and constructing new problem solving systems from scratch [6, 8, 25]. Knowledge Reuse addresses this issue by building new systems partly from existing components. The difficulty then becomes one of identifying which components can be reused to address the new task; this is still a demanding problem. One approach to dealing with the problem is to create toolboxes and advisory systems such as the MUSKRAT (Multistrategy Knowledge Refinement and Acquisition Toolbox) framework which aims to unify problem solving, knowledge acquisition and machine learning in a single computational framework [35]. Its main component is the Advisor, which if given a set of Knowledge Bases (KBs) and Problem Solvers (PSs), investigates whether combinations of the available KBs fulfil the requirements of the selected PS for a given task.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

K-CAP '05, October 2-5, 2005, Banff, Canada.

Copyright 2005 ACM 1-58113-000-0/00/0000...\$5.00

White proposed a Meta-PS [35], that conducts a tractable plausibility test which identifies and removes incompatible KB-PS combinations. However, its major shortcoming is that it can not guarantee that successful KB-PS combinations are not falsely discarded. The work described in this paper develops White's proposal, by considering how to generate a plausibility test using constraint satisfaction techniques which guarantees that no successful KB-PS combinations are falsely discarded. We have used a constraint solver as the PS and have represented existing scheduling KBs as Constraint Satisfaction Problems (CSPs) which can be combined to create a composite CSP. If the composite CSP is unsolvable, then that KBs-PS combination could not be reused to solve the given problem. Identifying plausible combinations thus requires examining a series of CSPs, and rejecting unsolvable ones. Proving a CSP inconsistent can be a lengthy process, so the method we propose to speed up this inconsistency detection is to relax the CSP by removing constraints, and if the relaxed version is demonstrated inconsistent then the original CSP will not have a solution either. Naively, we might assume that relaxing an unsolvable CSP will produce an easier problem. In fact, removing constraints randomly typically creates problems that are several times harder—as constraints become looser, or the connectivity of the problems becomes sparser, the time to demonstrate inconsistency for random problems increases [21]. To test our relaxation approach, we investigate different relaxation strategies on a variety of problem with scheduling characteristics.

BACKGROUND

Knowledge Base Systems (KBSs) have been developed for a variety of reasons, including: the archiving of rare skills, preserving the knowledge of retiring personnel, support in decision making, and to aggregate all of the available knowledge in a specific domain from several experts and/or machines. KBS is one of the Artificial Intelligence paradigms that have been easiest for companies to embrace and consequently there are numerous examples of successful KBSs in business [15, 18]. KBS appear most frequent in production, marketing, and customer service [16] and some of the most successful ones are in the scheduling domain [23, 26, 28].

* Work conducted when based at University of Aberdeen

Most KBSs are developed from scratch and the required knowledge acquisition is time consuming and therefore expensive [6, 8, 25]. If new systems could be built by reusing existing components, it might be done more quickly and hence money could be saved. Consequently one of the main goals of the KBS community has been to reuse KBS components [1], and several different components have been suggested for reuse. At an early stage researchers in the Knowledge Engineering sub-area identified a range of Problem Solving Methods (PSMs), which they argued covered the whole range of problem solving, and included methods for Classification and Diagnosis through to Planning (so-called Synthesis tasks) [10]. An early but powerful example of reuse of a PSM was the EMYCIN (Empty/Essential-MYCIN) [3] shell with a variety of domain-specific KBs in infectious diseases, analysis of building structures etc. Current work in reuse has resulted in systems where a number of components have been reused, including PS/PSM [12], ontologies [5, 12, 32] and KBs [1, 22, 24, 25, 28]. The use of cases in Case Based Reasoning is also a related activity [13].

Reusing KBs

There are several processes to assist in the hard task of reusing non-standardised KBs, such as searching, translation, comprehension, comparing, slicing, reformulation and merging [6]. Ontologies play an important role here [5, 22] by facilitating the search for structural and lexical similarities between the PS's knowledge requirement and existing knowledge in KBs which in turn increases the chances for mapping, merging and ultimate reuse of KBs. PROTÉGÉ now provides an option to write KBs in a standardised format like OKBC (Open Knowledge Base Connectivity) and OWL (Web Ontology Language) [9], which facilitates the necessary merging/mapping of KBs. Our research makes the strong assumption that the KBs to be investigated are standardised: written in the same language and use a common ontology. This might not always be the case for real-world problem.

Constraint Programming

Constraint programming has been successfully applied to many real-world problems because these problems can easily be modelled in terms of constraints, such as scheduling, planning, configuration, layout, resource allocation, and decision support [33]. Constraint Satisfaction techniques attempt to find solutions to CSPs. There are a number of efficient toolkits available (e.g. [11, 29]), especially designed to handle these problems. A CSP is defined by:

- a set of variables $X = \{X_1, \dots, X_n\}$,
- for each variable X_i , a finite set D_i of possible values (its domain), and
- a set of constraints $C_{\langle i \rangle} \subseteq D_{i_1} \times D_{i_2} \times \dots \times D_{i_t}$, restricting the values that subsets of the variables can take simultaneously.

A solution to a CSP is the assignment of values to every variable, in such a way that all constraints are satisfied simultaneously.

The arity of a constraint is the number of variables that the constraint is connected to; for example, a 'binary constraint' constrains two variables. Even though all constraints of an arity greater than two can be reformulated and represented with binary constraints [2], we have reservations about the practice of using solely binary constraints in CSPs and [21] argues against this practice. Problem classes of random binary CSPs are normally described by a 4-tuple $\langle n, m, c, t \rangle$ [14], 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 (the tightness).

A standard measure of effort for a CSP search algorithm is the number of constraint checks performed but other properties such as backtracking and resumption are also used. The search effort is dependent on the structure of the problem (how the constraints interact to rule out assignments); the individual constraints (some constraints are cheap to test/propagate, while others are expensive); and the number of solutions produced in a best-solution search or in an all-solution search. The main CSP search technique interleaves various forms of backtracking search with consistency enforcement, in which unfeasible values are removed from the problem through reasoning about the constraints. The use of higher order consistency algorithms also serves to identify unsolvable problems, but in [21] several reasons why this approach is unsuitable for identifying inconsistent KB-PS combinations are discussed. Finally, the concept of relaxing CSPs has received considerable attention [27], but in contrast to our relaxation approach this earlier research has focused on changing the CSP to introduce solutions.

OUR RELAXATION APPROACH

Our relaxation approach assists the MUSKRAT-Advisor by quickly identifying impossible KB-PS combinations. These impossible combinations can then be discarded, leaving a smaller number for the Advisor to evaluate. We have used a constraint solver as the PS and represented the existing scheduling KBs as CSPs which can be combined to create a composite CSP. If the composite CSP is found to be inconsistent, the KB-PS combination will not fulfil the PS requirements and can be discarded (note that if the relaxed CSP has a solution, then the original CSP represents a plausible combination and should be retained for further investigation). Proving a CSP inconsistent can be a lengthy process, so we propose a constraint relaxation approach to quickly identify inconsistent CSPs. Our approach relaxes the CSP by removing constraints and if the relaxed version is unsolvable then the original CSP will not have a solution either. This approach can only be profitable if the relaxed CSP is easier to solve than the original. It is reasonable to assume that if there is a problem and some of the constraints are removed from it, the new problem should be

easier to solve than the original. However, it is not certain that a relaxed CSP will pose an easier problem. In fact, phase transition research (e.g. [7]) seems to indicate the opposite when the original CSP is inconsistent; as constraints become looser, or the connectivity of the problems becomes sparser, the time required to demonstrate inconsistency for random problems increases. Moreover, part of our previous research [19] has shown that when random constraints are removed from an inconsistent binary CSP, the new relaxed CSP can be up to 10 times harder to solve. To statistically verify our relaxation idea we have developed the CSP-Suite, a Prolog test suite to assist in creating and evaluating relaxation strategies on a wide range of non-binary problem based on real-world scheduling problems.

Scheduling KBS

We created a prototype [21] of a Scheduling KBS in the mobile-phone manufacturing domain, and our reasons for exemplifying our relaxation approach on scheduling problems were threefold. *Firstly*, industry is currently using Scheduling KBSs, and has shown an increasing interest in the use of AI techniques to assist production scheduling problems [23, 25, 26]. *Secondly*, scheduling KBS in industry are already reusing standardised knowledge components successfully [26, 28]. *Thirdly*, scheduling problems are not only rich in detail and features, but are relatively easy to specify, commonly using a style that is very close to CSPs, hence it would be relatively easy to transform the necessary KBs. The decision to use the mobile phone manufacturing domain was made to highlight the benefit of reusing KBs from different stages in a global production chain; namely, factories, suppliers, shipping companies etc. Further, we believe the production chain in manufacturing mobile phones effectively illustrates the complexity of production scheduling, on a task that can be understood by non-domain experts.

In the prototype example, the manufacturer has a variety of existing standardised KBs at his/her disposal: four KBs concerned with factories, four with suppliers, and two with shipping companies used to transport phones to wholesalers around the world. Along with these slices of domain-specific KBs, the system also consists of background knowledge (e.g., ISO 9001, Safety Standards), and constraint relaxation rules that identify constraints which the system is or is not allowed to remove.

If the manufacturer would like to combine and reuse existing knowledge to answer a question such as ‘Can we manufacture, within the guidelines of the ISO 9001 and European safety standards (CE), a mobile phone with a 4096 colour screen, not heavier than 100g and have it delivered to the American market within 6 months?’ The manufacturer has 5 PSs (constraint satisfier, schedule etc.) and 12 KBs to combine and to determine if they can be used to answer the questions. The system then systematically investigates all possible KB-PS combinations. However there are some constraints; namely a KB-PS combina-

tion can only consist of 1 PS (out of 5), 1 supplier (out of 4), 1 factory (out of 4), 1 shipping company (out of 2), 1 background knowledge (out of 1) and 1 constraint relaxation rules (out of 1). This gives the manufacturer 160 (5×4×4×2×1×1) possible combinations to investigate; Figure 1 illustrates one possible combination.

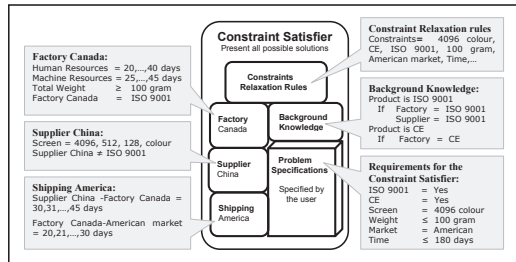


Figure 1. One of 160 possible KB-Combinations.

The Reuse Investigation Process

The process of inspecting KB-PS combinations for plausibility is done in three steps (Figure 2). *Firstly*, the KBs-PS combinations described in the previous section, are transformed by the constraint solver into composite CSPs. This transformation can only be done automatically if standardised KBs are used. *Secondly*, the CSP-Suite’s Relaxing module then relaxes these CSPs by carefully removing specified constraints according to its relaxation strategies to create a relaxed version of the problem. *Thirdly*, the Solving module with the task specification (find one, all, or the best solution) will then be used to solve the relaxed CSPs; if some of these relaxed CSPs are inconsistent then one can demonstrate that the original KBs-PS combinations are incompatible and can be discarded.

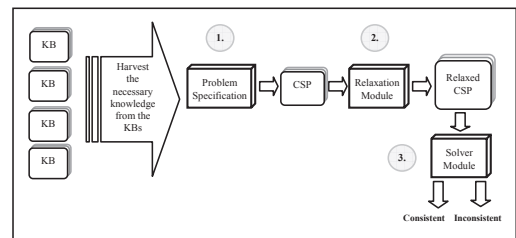


Figure 2. Our Process of Analysing Standardised KB-Combinations for reuse

The CSP-Suite

The CSP-Suite was designed to generate test-beds of CSPs. The suite is written in SICStus Prolog [29] and consists of modules to generate, relax and solve CSPs. In the test process, the user first specifies a problem class and relaxation strategies. The *Generating module* then randomly generates a set of CSPs, based on real world scheduling problems. The *Relaxing module* then creates relaxed versions of this

original CSP using pre-selected strategies to remove a specific number of constraints. Finally, the *Solving module* solves the original CSPs and their relaxed counterparts and records primarily the search effort.

Figure 3 shows the successful relaxation experiment for the Greedy Search strategy applied on problem class $30<20,10,133,65>$ along with the hardness curve for this problem class $30<20,10,Stepped,65>$, represented by the Random Removal strategy. The visual comparison between the two curves shows that while Random Removal fails to generate relaxed inconsistent CSPs that are easier to search, the greedy strategy manages not only to avoid the hardness peak (Random Removal hardness peak) but also creates relaxed CSPs that are $\sim 70\%$ easier to search compared to the original problem, after a small number of constraints are removed.

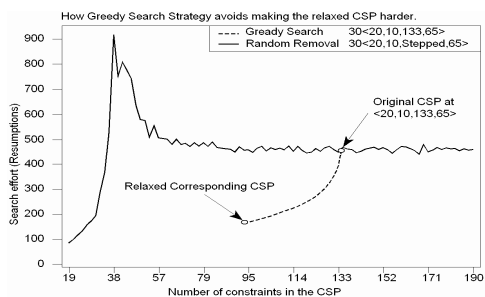


Figure 3. Relaxation Results for Greedy Search

The Generating Step

The Generating module produces a large number of CSPs with real-world scheduling properties. To help characterise real-world problems in terms of the entities and their relationships we have investigated the characteristics and ontologies [30] of scheduling problems as well as their constraint graphs' properties (e.g. [34]). This investigation was vital in our effort to close the gap between the properties of real-world scheduling problems and those our Generating module creates. There are several practical reasons for choosing to implement non-binary constraints into the CSP-Suite. *Firstly*, using non-binary, and particularly global constraints (i.e. constraints which can act on an arbitrary number of variables), in SICStus using its built-in propagation and search facilities, can give a more natural representation of real-world problems. Some of these available non-binary and global scheduling constraints are specially created to help formulate scheduling problems, so using these directly without having to perform a non-binary to binary transformation will reduce the formulation work done by the knowledge engineer. *Secondly*, translating from non-binary to binary can cause the problem size to explode. *Thirdly*, research has shown [31] that non-binary and global constraints can have much faster propagation than their binary equivalents, and therefore could speed up

the search process. *Fourthly*, non-binary constraints and global constraints may have different behaviour in constraint-solving toolkits so the characteristics of problems might change drastically in the transformation from binary to non-binary. *Lastly*, it is likely that standardised KBs will contain non-binary constraints, and to minimise the transformations required, the relaxation strategies should be applied directly to those constraints.

Because the Generating module creates a CSP with real-world properties; such as different statistical distributions on tightness (uniform, normal, and exponential distribution), as well as the mix of constraint type and arity, we use a somewhat different notation than that used earlier in the literature. We describe our problem class of CSPs as a tuple $\delta<n,m,c(c_{NB},A_{max}),D>$, where δ denotes the number of CSPs from the class². c is the number of constraints, c_{NB} denotes the number of non-binary constraints, and A_{max} is their maximum arity (i.e. the maximum number of variables constrained by a single constraint). The arity of the constraints is uniformly distributed in the range $[3, A_{max}]$. D denotes the distribution of the constraint tightness. For uniform distributions, D is of the form $U[t_{\mu},r]$, where the number of forbidden tuples in a constraint is uniformly distributed in the range $[t_{\mu}-r, t_{\mu}+r]$, with the average t_{μ} . For normal distributions, D is of the form $N[t_{\mu},sd,r]$, where sd is the standard deviation of the bell curve over the range $[t_{\mu}-r, t_{\mu}+r]$. For exponential distributions, D is of the form $E[t_m,stp,r]$, where the range is $[t_m-r, t_m+r]$, and stp measures the steepness of the probability curve³.

The module generates the problems in the following steps. First, a skeleton CSP is created using only binary constraints, to ensure that the underlying graph is connected. Then the module adds random binary constraints until the specified number ($c-c_{NB}$) is reached. Finally, the number of non-binary constraints is added (c_{NB}), selected randomly from the range (A_{max}), and the constraint is selected uniformly, normally, or exponentially from a predefined constraint table, ensuring that the tightness distribution is obeyed.

The Relaxing Step

The Relaxing module generates relaxed CSPs from the original CSPs by removing a specified number of constraints according to different strategies. For the relaxation approach to be successful, the relaxation strategies should not only be easy to implement but should also create a relaxed CSP that is easier to solve without introducing any early solutions. At this moment, eleven different strategies are implemented in the relaxation module: *Random Removal* simply chooses the constraints randomly. *Greedy Search* considers each constraint in turn, removing it, solving the relaxed CSP, and restoring the constraint. It then

² δ denotes the number of CSPs and are actually not part of the notation of the problem class.

³ Note that t_m is not the average of the exponential distribution

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. *Arity* removes constraints in ascending or descending order of their arity. Note that the two Greedy strategies would not be applicable in our eventual framework, since they must solve many CSPs to create the relaxed CSP. They are useful here as reference points showing what might be achievable. See [21] for a more detailed description of the eleven strategies as well as the rationale for their creation.

The Solving Step

The Solving module solves the original CSP and its relaxed counterpart; when one solution is found the search is stopped and different search statistics are recorded, using the SICStus finite domain constraint library [4]. The library does not report constraint checks, so instead we use the resumption statistic to measure the search effort. We have confirmed that resumptions correlate well with cpu time required to solve a task [21].

EMPIRICAL TESTS

There are a vast number of problem classes which can be investigated. Our previous work [20, 19] has shown that relaxing a binary CSP can speed up the detection of inconsistency, and in particular that removing constraints of low-tightness is an effective strategy. However, that work considered only binary CSP, while this paper shows the results when our strategies are applied on more realistic problems, based on scheduling properties and including non-binary constraints.

Figure 4, shows the strategies' relaxation findings when up to 60 constraints are removed from the non-binary problem class $30 < 20, 10, 133(66, 6), U[65, 15] >$. The Y-axis represents search effort spent on the relaxed CSP expressed as a percentage of that used on the original CSP; a positive value shows that the relaxed CSP is easier to solve than the original CSP, while a negative value indicates that the relaxed CSP is harder than the original CSP. The X-axis shows the number of constraints in the CSPs. These results illustrate that many of the relaxing strategies that were successful on the binary problem classes also perform well on non-binary CSPs. In addition, High Node Degree is a strategy that did not perform well on binary problem classes but produces good results along with High Arity on non-binary problem classes. Overall, Low Tightness is clearly the most profitable of the applicable strategies; it can identify the original

CSPs as inconsistent using ~30% less search effort than the original needs.

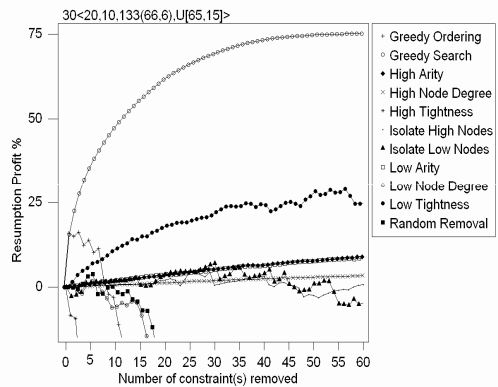


Figure 4. Resumptions Profit in % when Relaxation Strategies when Removing up to 60 Constraints

Figure 5 presents the most successful strategies on two new problem classes $30 < 20, 10, 133(66, 6), U[65, 30] >$ and $30 < 20, 10, 133(66, 6), U[45, 15] >$. The graphs illustrate the effect the strategies have on the non-binary problem classes, which differ in average tightness and the width or the tightness range. The lower graph, presents a very strong positive trend for Low Tightness, which achieves a resumption profit value of ~43%. Thereafter comes the Isolate Low Nodes strategy with ~12%, High Nodes, High Node Degree, and High Arity strategies produce a zero trend. The straight zero resumption trend for High Arity and High Node Degree strategies can be explained by the CSP representation; the solver checks the constraints from top to bottom, and the binary are formulated first while the random chosen non-binary are last. This means that the zero trend experiment on the non-binary constraint does not influence the problem hardness. Namely, the hardness of the problem is decided by the binary constraints, so removing the non-binary constraints would have no affect on the resumption value; a different problem representation is suggested as a future research direction. The top graph shows that the High Node Degree strategy has overtaken Low Tightness, and manages to reach a resumption profit value of ~20% before it dips to 5%. After this typical dip, which indicates the hardness curve has been crossed and solutions have been introduced, the strategy returns to produce a positive trend. The graph shows High Arity manages to relax the CSP with ~10% gains, a negative trend starts after ~55 constraints are removed, and the negative curve culminates in creating a relaxed CSP that is ~25% harder to solve than the original. The other two strategies, Isolate Low Nodes and Low Tightness, report negative trends.

For our relaxation approach to have a good chance of working on real-world scheduling problems, it is vital that they give good results over a wide spectrum of problem classes. Therefore, extended experiments have been conducted on non-binary problem classes where the proportions of binary/non-binary are varied [21]. The findings indicate that Low Tightness is unaffected by the variation in the proportion of binary/non-binary constraints while the High Node Degree strategy improves as the proportion of non-binary constraints increase. The other strategies do not show any performance changes depending on the non-binary/binary ratio. Although Isolate High Nodes has shown surprisingly good results, Low Tightness still is overall the most profitable of the strategies.

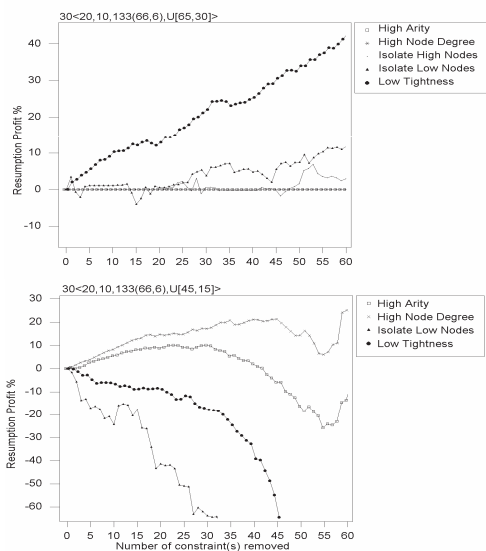


Figure 5. The Most Promising Relaxation Strategies' Resumption Profit on New Non-binary CSPs

The experiment shown in Figure 6 (upper graphs) was conducted to examine whether Low Tightness can avoid creating relaxed CSPs that are harder to solve than the original. The graphs show the search effort for Low Tightness and Random Removal on two problem classes, when all 133 constraints are removed. Starting with the two particular problem classes on the right-hand side of the curve, constraints are removed according to each strategy, and the new problems are solved after each removal. In both cases, the Low Tightness strategy avoids creating relaxed CSPs that are harder than the original. A closer investigation of $30\langle 20,10,133(66,6),U[tu,30]\rangle$ reveals Low Tightness curve's resumption level is always lower than that of the original problem class. The solution transition phase portrayed in Figure 6 (lower graphs) highlights the fact that

Low Tightness does not introduce any solutions earlier than the density curve (represented by Random Removal). Note that if the relaxed CSP has a solution, then the original CSP represents a plausible combination and should be retained for further investigation. The delayed solution transition phase gives us some confidence that we can use our relaxation strategy reliably without introducing new solutions.

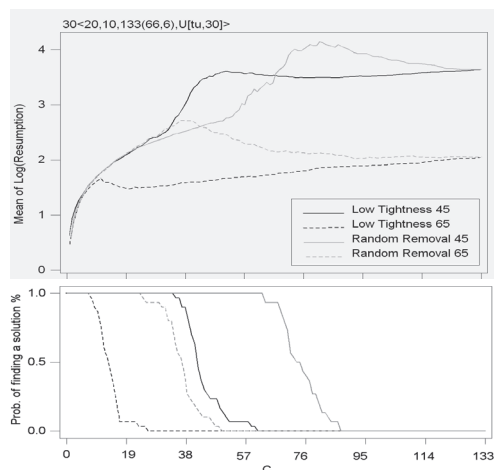


Figure 6. Search Effort & Transition Phase for Random Removal vs. Low Tightness

The above experiments shows that certain focused relaxation strategies do produce simpler problems, and thus relaxation is an effective method for plausibility testing. In particular, the relaxation strategies are effective on randomly generated scheduling problems. Low Tightness strategy can in some cases detect inconsistency using only 60% of the search effort of the original problem. Moreover, for the only problem class that the Low Tightness does not produce a positive resumption profit, the High Arity and High Node Degree strategies perform well.

SUMMARY

This paper has described a novel constraint relaxation approach that contributes to the challenging problem of KB-reuse as it helps answer the question whether combinations of standardised KBs can be reused to solve a new task. We proposed a cheaper plausibility-test, based on constraint relaxation techniques that identify inconsistent KBs-PS combinations, which leaves fewer plausible combinations for the MUSKRAT-Advisor to examine in detail. If a relaxed CSP is demonstrated to be inconsistent then the original problem will be inconsistent as well and can be discarded without an expensive search. Note that for our relaxation strategy to be successful the relaxed problem must be easier to solve than the original. However, relaxing a CSP to produce an easier problem is not that simple; previ-

ous phase transition research on random binary CSPs (of the same size and tightness) has shown that inconsistent CSPs with fewer constraints are harder to solve than those with more constraints. Our task then was to create a relaxation algorithm that cheaply produced relaxed CSPs without introducing any early solution(s). We created a Prolog test Suite designed to generate test-beds of CSPs with real-world scheduling properties, which have helped identify beneficial relaxation strategies. The experimental results presented in this paper have shown not only that some relaxation strategies quickly identify incompatible CSPs without introducing any early solutions but also that the simple strategy of removing constraints of Low Tightness has in most cases been very effective in reducing the time required to detect inconsistencies; in some cases identifying inconsistent CSPs using only 60% of search effort for the original problem. We believe the reason why removing low-tightness constraints gives good results is that constraints with low tightness rule out very few combinations and are therefore more likely to be redundant than tight constraints on inconsistent CSPs, but they require repeated propagation checks for no benefit.

FUTURE WORK

The profitability of our relaxation approach is dependent on the proportion of consistent and inconsistent KB-PS combinations. When a relaxed CSP is found to be consistent this does NOT mean that the original CSP is consistent, it means it is plausible and needs to be investigated. The higher the proportion of inconsistent combinations the more beneficial our approach will be and subsequently the higher the proportion of consistent KB-PS combinations the less profitable it will become. To make our approach more profitable, we suggest reusing the search effort spent on the relaxed CSP (when demonstrated consistent) when investigating the original CSP. All variable instantiations (except the instantiation that introduced the solution) on the relaxed CSP can be ignored when searching for a solution on the original CSP. This means that the search space of the relaxed CSP can be deducted from the original problems search space. This ability to reuse the search would allow our approach to be more profitable for the MUSKRAT-Advisor reuse investigation.

Moreover, we believe that the implemented strategies are not optimal and that combining the existing strategies could produce even better results. For example, some strategies appear to be natural combinations such as Low Tightness and High Arity. Both have performed well individually, in particular the results in the empirical section have shown that they complement each other; where High Arity excels, Low Tightness normally performs badly and vice versa. This combination is worth investigating further.

Furthermore, we appreciate that in general KB to CSP transformation is a very hard task. It can be argued that we simplified the KB-CSP transformation task in our approach by working with standardised scheduling KBs that can eas-

ily be transformed to CSPs by the constraint solver. We suggest the next step in the MUSKRAT framework would be to facilitate KB to CSP transformations.

Finally, when we determine that a particular KBs-PS combination is not viable, it will be useful to investigate whether that is caused by a particular KB(s). In which case that KB(s) would be "black listed" so as to further reduce the number of combinations to be considered (c.f. effective strategies applied to explore search spaces [17]).

ACKNOWLEDGMENTS

This work was supported under the EPSRC's grant number GR/N15764 and the Advanced Knowledge Technologies Interdisciplinary Research Collaboration.

REFERENCES

- [1] AKT, (2003) 'Reuse Knowledge', The Advanced Knowledge Technologies project (AKT) [WWW] Available from: <http://www.aktors.org/publications/reuse/> [Accessed 10 June 2004]
- [2] Bacchus, F., X. Chen, P.v. Beek, and T. Walsh, (2002) 'Binary vs. non-binary constraints', *Artificial Intelligence*, Volume 140, Issue 1-2, September, pp. 1-37
- [3] Bennett, J. and R. Englemore, (1983) 'Experience using EMYCIN. In Rule-Based Expert Systems', E. Shortliffe, Addison-Wesley, London, pp. 314-328
- [4] Carlsson, M., G. Ottosson, and B. Carlsson, (1997) 'An Open-Ended Finite Domain Constraint Solver' in *Programming Languages: Implementations, Logics, and Programs.*, pp. 345-381
- [5] Chandrasekaran, B., J.R. Josephson, and V.R. Benjamins, (1999) 'What Are Ontologies, and Why Do We Need Them?' *IEEE Intelligent Systems*, Volume 14, Issue 1. pp. 20-26
- [6] Chaudhri, V.K., M.E. Stickel, J.F. Thomere, and R.J. Waldinger, (2000) 'Using Prior Knowledge: Problems and Solutions' in *Seventeenth National Conference on Artificial Intelligence and Twelfth Innovative Applications of Artificial Intelligence Conference*, Austin, Texas, USA: AAAI Press/MIT Press, pp. 437
- [7] Cheeseman, P., B. Kanefsky, and W.M. Taylor, (1991) 'Where the really hard problems are' in *Seventh International Joint Conference on Artificial Intelligence*, Sydney, Australia, pp. 331-337
- [8] Dieng, R., O. Corby, A. Giboin, and M. Ribière, (1998) 'Methods and Tools for Corporate Knowledge Management' in *11th Banff Workshop on Knowledge Acquisition, Modelling and Management*, pp. 42
- [9] Gennari, J.H., M.A. Musen, R. Ferguson, and . (2003) 'The Evolution of Protégé: An Environment for Knowledge-Based Systems Development.' *International Journal of Human-Computer Studies*, Volume 58, Issue 1. pp. 89-123

- [10] Hayes-Roth, F., D.A. Waterman, and D.B. Lenat, (1983) 'Building Expert Systems'. Teknowledge series in knowledge engineering ; v. 1, Reading, Mass.: Addison-Wesley, pp. 444
- [11] ILOG Solver, (2003) Version: 5.3, from ILOG Inc., [WWW]: <http://www.ilog.com/>
- [12] Kalfoglou, Y., T. Menzies, E. Motta, and K.-D. Althoff, (2000) 'Metaknowledge in systems design: panacea...or undelivered promise', *The Knowledge Engineering Review*, Volume 15, Issue 4. pp. 381-404
- [13] Kolodner, J.L., (1993) 'Case-Based Reasoning', San Mateo, CA: Morgan Kaufmann Publishers, pp. 668
- [14] MacIntyre, E., P. Prosser, B. Smith, and T. Walsh., (1998) 'Random Constraint Satisfaction: Theory meets Practice' in *Fourth International Conference on Principles and Practice of Constraint Programming*, pp. 325-339
- [15] Newman, D.R., (1998) 'Advantages of KBS', [WWW] Available from: <http://www.qub.ac.uk/mgt/intsys/kbsadvan.html> [Accessed 12 Mars 2004]
- [16] Newman, D.R., (1998) 'What are KBS used for', [WWW] Available from: <http://www.qub.ac.uk/mgt/intsys/kbsused.html> [Accessed 14 Mars 2004]
- [17] Nilsson, N.J., (1971) 'Problem-solving methods in artificial intelligence', New York, McGraw-Hill, pp. 255
- [18] Nordlander, T., (2001) 'AI Surveying: Artificial Intelligence in Business', De Montfort University, Thesis
- [19] Nordlander, T., K. Brown, and D. Sleeman, (2003) 'Identifying inconsistent CSPs by Relaxation' in *Ninth International Conference on Principles and Practice of Constraint Programming*, Cork, Ireland: Springer Verlag, pp. 987
- [20] Nordlander, T., K. Brown, and D. Sleeman, (2003) 'Constraint Relaxation Techniques to Aid the Reuse of Knowledge Bases and Problem Solvers' in *The Twenty-third SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, Cambridge, UK: Springer Verlag, pp. 323-336
- [21] Nordlander, T.E., (2004) 'Constraint Relaxation Techniques & Knowledge Base Reuse', University of Aberdeen, PhD Thesis, pp. 246
- [22] Park, J.Y., J.H. Gennari, and M.A. Musen, (1998) 'Mappings for Reuse in Knowledge-based Systems' in *11 th Workshop on Knowledge Acquisition, Modelling and Management*, Canada: Springer, pp. 1-21
- [23] Prosser, P. and I. Buchanan, (1994) 'Intelligent scheduling: past, present and future', *Intelligent system engineering*, Volume 3, Issue 2. Summer, pp. 67-78
- [24] Puppe, F., (1998) 'Knowledge Reuse among Diagnostic Problem Solving Methods in the Shell-Kit D3', *International Journal of Human-Computer Studies*, Volume 49, Issue 4. pp. 627-649
- [25] Sauer, J. and H.-J. Appelrath, (1997) 'Knowledge-Based Design of Scheduling System' in *World Manufacturing Congress, International Symposium on Manufacturing Systems*, Auckland: ICSC Academic Press, pp. 247-252
- [26] Sauer, J. and R. Bruns, (1997) 'Knowledge-Based Scheduling Systems in Industry and Medicine', *IEEE-Expert*, Volume 12, Issue 1. pp. 24-31
- [27] Schiex, T., H. Fargier, and G. Verfaillie, (1995) 'Valued Constraint Satisfaction Problems: hard and easy problems' in *International Joint Conferences on Artificial Intelligence*, pp. 631-637
- [28] Schreiber, G., H. Akkermans, A. Anjewierden, R.d. Hoog, N. Shadbolt, W.V.d. Velde, and B. Wielinga, (1999) 'KNOWLEDGE ENGINEERING AND MANAGEMENT The CommonKADS Methodology', MIT press, pp. 91
- [29] SICStus Prolog, (2001) Version: 3.10.0, [WWW]: <http://www.sics.se/sicstus/>
- [30] Smith, S.F. and M.A. Becker, (1997) 'An Ontology for Constructing Scheduling Systems' in *Fourteenth National Conference on Artificial Intelligence: Spring Symposium on Ontological Engineering (AAAI'97)*, Providence, Rhode Island, USA: AAAI Press, pp. 1-10
- [31] Stergiou, K. and T. Walsh, (1999) 'The Difference All-Difference Makes' in *Fifteen International Joint Conference on Artificial Intelligence*, Montreal Quebec: Morgan Kaufmann Publishers, Inc, pp. 414-419
- [32] Uschold, M., P. Clark, M. Healy, K. Williamson, and S. Woods, (1998) 'An Experiment in Ontology Reuse' in *11th Banff Knowledge Acquisition Workshop*, Calgary Canada: SRDG Publications, pp. 33
- [33] Wallace, M., (1996) 'Practical application of constraint programming', *Constraints*, Volume 1, Issue 1-2. pp. 139-168
- [34] Walsh, T., (2001) 'Search on high degree graphs' in *Seventeenth International Joint Conference on Artificial Intelligence*, Seattle, WA, pp. 266-274
- [35] White, S. and D. Sleeman, (2000) 'A Constraint-Based Approach to the Description & Detection of Fitness-for-Purpose', *Electronic Transactions on Artificial Intelligence*, Volume 4. pp. 155-183