

Assisting Domain Experts To Formulate & Solve Constraint Satisfaction Problems

Derek Sleeman and Stuart Chalmers

Department of Computing Science, University of Aberdeen,
Aberdeen, AB24 3UE, Scotland, UK
{sleeman, schalmer}@csd.abdn.ac.uk

Abstract. Constraint satisfaction is a powerful approach to solving a wide class of problems. However, as many non-experts have difficulties formulating tasks as Constraint Satisfaction Problems (CSPs), we have built a number of interfaces for particular kinds of CSPs, including crypt-arithmetic problems, map-colouring problems, and scheduling tasks, which ask highly focused questions of the user, c.f., the earlier MOLE/MORE, and SALT knowledge acquisition systems. Information from each of these interfaces is then transformed initially into a structured format which is semantic web compliant and is secondly transformed into the format required by the generic constraint satisfaction problem solver. When this problem solver is run, the user is either provided with solution(s) or feedback that the problem is underspecified (when many solutions are feasible) or over-specified (when no solution is possible). The system has 3 distinct phases, namely; information capture, transformation of the information to that used by a standard problem solver, and thirdly the solving and user feedback phase.

1 Introduction

Constraint Satisfaction constitutes a powerful approach to problem solving, and over the last decade or so a range of tools have been implemented [4, 7]. However, there are distinct skills involved in formulating tasks as CSP problems, the so called modelling problem. The format required by the solvers is deceptively simple: one has to specify a set of relevant variables, their domain values, and constraints between the variables. However the modelling process is still seen as challenging because the problem is often specified in a verbal form (eg as brain teasers), and it is the modeller's task to decide the relevant variables, the ranges associated with each variable, and most challengingly the constraints (relationships) which exist between the identified variables [8].

This paper addresses the modelling problem mentioned early by implementing a series of interfaces which ask for specific information about particular types of CSPs, e.g. map-colouring, cryptarithmic problems and scheduling. This information is then transformed into representational schema currently used on the web, namely an OWL¹ ontology (for the domain knowledge) and CIF/SWRL² rules for the constraints. As we shall see this is further transformed into the representation required by a generic constraint satisfaction solver.

¹ www.w3.org/TR/owl-features/

² www.csd.abdn.ac.uk/research/akt/cif/

One of the aims of this project is to produce a UI which corresponds to each of the types of CSPs (see section 4.2 for details). We then plan to produce a more generic UI which will handle a number of different CSP task types.

The overall architecture of the system implemented has 3 components. The role of the first is to CAPTURE information about a particular task, the second is to transform that information / knowledge into a form which a generic Constraint Problem Solver can use, and the third solves the task and reports the results to the user. We made a conscious decision to use OWL and SWRL as representational formalisms, as then it is, in principle, possible to augment the information created in the interfaces by other knowledge sources available on the web.

The rest of the paper is organized as follows: section 2 gives an introduction to CSPs; Section 3 gives a conceptual overview of the 3-staged system implemented; section 4 discusses a classification of CSPs and describes Knowledge Acquisition (KA) interfaces built for some of these types/classes; section 5 describes the implementation of a CSP solver; section 6 outlines future work and describes some related work.

2 Formulating Tasks as CSPs

CSPs have 3 aspects, namely (i) variables which are associated with (ii) domains (i.e. ranges of values) and (iii) the actual constraint expressions. Below we give examples of all 3 components:

$$a \mapsto D_1\{0..5\}, b \mapsto D_2\{0..5\}, c \mapsto D_3\{0..5\}$$

The above states that the variables a , b and c can be assigned any of the corresponding values given in the domains D_1 , D_2 , D_3 respectively. The following constraints, $C1$ and $C2$, restrict the possible assignments that the variables can take.

$$C1: a-b > c, C2: a*c < b$$

Given these constraints, the assignments of possible values are now restricted to:

$$a \mapsto D_1\{2..5\}, b \mapsto D_2\{1..4\}, c \mapsto D_3\{0..1\}$$

In general, a constraint satisfaction problem (CSP) is the process of satisfying a given set of statements by restricting the assignments of a given set of variables.

3 Main aspects of the system: a Conceptual Architecture

Previous projects in constraint modelling (see section 6) have focused on creating new meta-languages for specifying problems [4] or have required the user to model the CSP task using an existing representation formalism e.g. UML and OCL [7]. Here we are focusing on classifying the types of CSP problems which exist, with a view to creating an interface for each type to enable the non-CSP specialist to communicate the essence of their task. For some CSP tasks it is clear the nature of the interface required, e.g. Map-colouring problems or Cryptarithmic problems (of the form SEND + MORE = MONEY) whereas the differences between Scheduling / Configuration / Assignment /

Constraints / Positioning are in general much more subtle, and will need some further analysis before helpful distinctions can be made. (In fact, as noted in section 4.2, the Cryptarithmic and Map-colouring tasks are both members of the Assignment class).

The general inspiration for this approach is the Knowledge Acquisition (KA) work done in the early 80's when several groups realised that KA could be made more focused if one acquires knowledge for a particular purpose. The MOLE [2] and MORE [5] systems, for instance, acquired knowledge which would support only classification/diagnosis and thus only needed to capture the several diagnostic classes and the corresponding diagnostic rules. Similarly, the SALT KA system [6] was designed to acquire knowledge to support the propose-and-revise algorithm and so elicited 3 types of knowledge/information from the domain expert namely:

- procedural knowledge to specify how an existing entity (such as a motor) could be enhanced/changed
- constraints to specify relationships which must or must not hold between variables
- fixes: what to do if a particular constraint is violated.

So the argument then is that a KA interface will be implemented for each of the classes of CSPs which will ask relevant focused questions. (We will address later the taxing question of how a non-CSP expert can choose the relevant KA interface for a particular problem they wish to solve.) In fact as figure 1 shows, we have conceptualised the task of formulating and solving CSP as three phases:

- CAPTURING the essence of the task (outlined above & discussed in section 4)
- TRANSFORMING the task from the information collected from each interface to a common formalism. In fact, we have chosen SWRL and OWL, emerging WWW standards, so that the information acquired can, in principle, be enhanced by other Web based Knowledge Sources.
- SOLVING & Providing User Feedback. The final phase attempts to solve the task and either provides the user with a result or information which indicates that the task is over- or under - specified.

4 Capturing Information: Creating KA interfaces

4.1 Model Generation vs. Model Selection

Systems such as a CONJURE [4] provide the non-CSP expert with a high-level language in which to formulate the task s/he wishes to solve. This approach has the advantage of allowing a wide range of problems to be specified (in principle) but the disadvantage that it gives the user little guidance. We refer to this as the Model Generation approach. By contrast we refer to the approach being followed here as the model selection approach. Conceptually, at least we can think of this approach as providing the user with a series of templates (one corresponding to each of the CSP classes or perhaps subclasses). Each template has a number of slots which the user has to complete. If a user is able to provide a consistent set of responses to all the slots associated

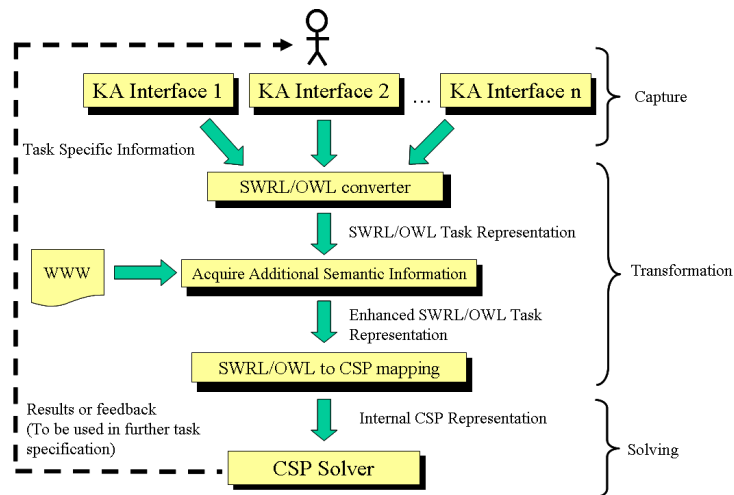


Fig. 1. Conceptual System Design showing 3 principal phases

with a template, then we shall say that this task corresponds to that CSP class (e.g. Assignment). In practice we believe that this part of the system will be implemented as a series of “linked templates”, so that at any stage of the specification of a task, it will be possible to specify:

- which classes have been ruled out (because certain information is *not* available)
- which classes are currently completely satisfied (could be null)
- which information is needed, to satisfy the remaining possible classes

4.2 Classes of CSP tasks

As mentioned in section 3, we are interested in describing the number of CSPs in sufficient detail, so that we can identify distinctive classes and sub-classes. Miguel et. al³, in their work with ESSENCE and CONJURE, have introduced a classification and suggest the following definitions:

- Scheduling - characterised by assigning start times to a series of tasks that have to be performed by some deadline with the possibility of precedence constraints between them (e.g. process A must be completed before process B can start).
- Configuration - where the problem involves assigning a unique value to a variable according to constraints between the values and their variables.
- Assignment - similar to configuration problems, but the assignment of values to variables is not a one-to-one relationship. Subsets of this class include permutation and partitioning problems.

³ <http://www.cs.york.ac.uk/aig/constraints/AutoModel/Essence/Tree/>

- Construction - the object is to construct a set of variables according to a goal (such as maximising the values assigned to the set of variables). Constraints here can be on the membership of this set, and on the position of the variable in that set.
- Positioning - involve arranging objects according to spatial/geometric constraints. Typically all objects must lie within a boundary and objects are not permitted to overlap (where these restrictions are specified as a series of constraints).

Generally, we can classify each problem type by the relationship between the Variables (described as Objects by Miguel) and Assigned Domain Values (defined as Labels). For instance, in permutation problems each label is used only once (effectively saying that the values assigned to a set of variables must all be different), whereas with partitioning problems the actual assignment of the label value is unimportant, but what is important are the groups of variables which have the same values.

The screenshot shows a web interface for a map-colouring task. It is divided into three main sections:

- Domain Names(seperated by ',')**: A text input field containing 'a,b,c' and a 'Submit Query' button.
- Positions**: A section with two dropdown menus, one containing 'a' and the other 'b', with the text 'Next To' between them. Below this is a checked checkbox labeled 'More Positions' and a 'Go' button.
- Input Modification**: A table with two rows:

aNextTob	Del
aNextToc	Del

 Below the table is an [Add](#) link and a [Reload Colour Mapping](#) link.

Fig. 2. The Map-colouring KA Interface

4.3 Implemented KA interfaces

In the previous sub-section we discussed in some detail a classification for CSPs. Here we outline 2 User Interfaces which we have recently developed, namely:

- A Mapping task (this being a subset of the assignment class)
- A Cryptarithmic Problem (which is a particular kind of assignment task)

A screen shot for the Mapping interface is given in Figure 2. The map colouring task specifies a number of physical areas (eg countries, counties, areas of a town) and specifies that adjacent areas should have **different** colours. So the task involves specifying the set of colours available, and the adjacency relationships between the several areas. A solution to the problem is one where all the adjacent areas have different colours. So for example if the objects to be assigned a colour are the 4 counties of the SouthWest of England, namely: Cornwall, Devon, Somerset & Dorset, and the relationships between the 4 objects are as given in the left hand diagram in figure 3, and the 3 colours to be

assigned are white, grey & black, then the right hand figure would be an acceptable solution. The UI in figure 2 used to collect information about the task, initially asks the user to provide the names of the 4 objects (top left hand corner), then the user is asked to indicate the spatial relationships between the objects (this is done in the window called Positions), at which point the system allows the user to revise the relationships provided until they are happy with the resulting model. Finally, the system asks the user for the available colours, and then the complete task is passed to the problem solver.

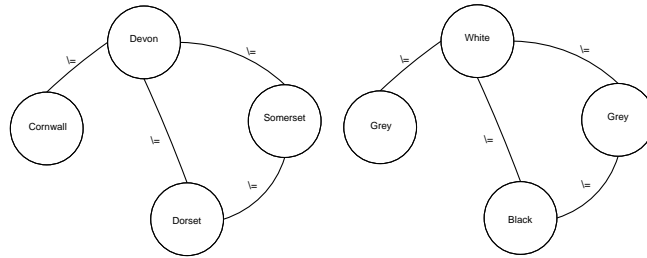


Fig. 3. An example Map Colouring Specification and solution

Cryptarithmic problems are of the form:

$$\begin{array}{r}
 \text{CROSS} + \\
 \text{ROADS} \\
 \text{-----} \\
 \text{DANGER} \\
 \text{-----}
 \end{array}$$

Where one is told that each of the letters is in the range 0 . . 9, each letter has a distinct value, and $D \neq 0$. So it is usual to formulate the problem as a series of variables (in the case of this task: C, R, O, S, A, D, N, G & E). Note in this case the 2 entities are to be **added** to give the result DANGER. In formulating the main constraint for this problem we need to remember the significance/semantics of a column in an arithmetic task. The right hand column consists of units, the adjacent column represents units of 10s, and the next column represents 100s, etc. Bearing this in mind the main constraint can be expressed as⁴: $10000 * C + 10000 * R + 1000 * R + 1000 * O + 100 * O + 100 * A + 10 * S + 10 * D + S + S = 100000 * D + 10000 * A + 1000 * N + 100 * G + 10 * E + R$

The KA interface which we have implemented for this type of problem allows the user to input a wide range of cryptarithmic problems; to date we have implemented both addition & subtraction tasks.

5 Solving: The CSP Implementation

A major design decision behind the implementation of this solver was to make the solving process freely available as a re-usable component. To this end, we have devel-

⁴ There are 2 further constraints: $C > 0, R > 0$

oped the CSP as a web service with an open queryable interface. We use AXIS⁵ as the platform, with the interface defined using a combination of OWL-lite to represent the variables and their domains and CIF/SWRL to represent the actual constraints⁶. We also implement a number of parsing algorithms, to support the following information flow:

KA Interface \mapsto CIF/SWRL + OWL \mapsto CHOCO Solver data structures

For the principal classes of CSPs covered here, we have written a generalised finite domain constraint solver in CHOCO⁷. To formulate a problem in CHOCO we create a finite domain CSP instance. We then use mapping rules for each CIF/SWRL construct to transform the CIF/SWRL representation into CSP constraints.

6 Discussion and Future Work

A number of projects have looked at the problem of CSP modelling. An approach taken by [4] is to provide a high-level language, ESSENCE, for specifying CSP problems. This language is then translated into a CSP by a system, CONJURE, that refines the specification. Renker [7] uses UML and OCL to provide a modelling framework for constraints. Alternative approaches include Fish et. al [1] who look at diagrammatic modelling of constraints using “spider diagrams”, a formalisation based on Venn diagrams.

We plan to evaluate the web-based KA interfaces for the cryptarithmic and mapping problems. We will compare the numbers of tasks formulated and the average time taken by two groups namely one doing the task using pen-and-paper and the other using the CSP interfaces. We will also administer a questionnaire about how the interfaces could be enhanced. A further aim of our future work is to implement a KA interface for each of the CSP types identified in section 4.2, and then to link them so as to reduce the number of questions that a user needs to answer.

Another major issue we wish to explore is that of providing user feedback on problems to help with remodelling and repair. When a user specifies a set of constraints on a problem, there may be no solution returned for that set as the problem is over-constrained. In such circumstances, constraint relaxation [3] techniques aim to partially solve a given problem by maximizing the number of constraints applied. Alternatively, we may find a problem under-defined (i.e. there are a vast number of possible valid solutions) where we may wish to elicit more information from the user to constrain the problem further.

Our CSP implementation is based on a simple generalised finite domain solver, so we are not able to solve the range of problems that a language such as ESSENCE can support. While it is not our intention to do this, we do wish to classify the subset of problems which can be solved as well as explore the representation of CSPs using existing semantic web technology. As discussed earlier in the paper, there were several motivations for using a semantic-web compliant representation for the CSP. The

⁵ <http://ws.apache.org/axis/>

⁶ For a detailed description of the design and capabilities of the semantic interface implementation, see Aberdeen University Computing Science Technical Report AUCS/TR0604

⁷ <http://choco.sourceforge.net/>

principal issues still to be investigated, is whether it is possible to exploit the fact that the intermediary representation of the CSPs is OWL and CIF/SWRL, by enhancing the initial user-provided information about a task with relevant information available from the semantic web [9].

In this paper, we have described a set of user interfaces to aid in the formulation of tasks as CSPs. We have described our representation of the CSPs using semantic web technology, namely OWL for variable and domain representation and CIF/SWRL for representing constraints. We have also described the flow of information, from the KA interfaces, to a semantic web representation and finally to the native constraint solving language.

Acknowledgments

This work is supported by the Advanced Knowledge Technologies (AKT)⁸ Interdisciplinary Research Collaboration (IRC), which is sponsored by the UK Engineering and Physical Sciences Research Council (EPSRC). The AKT IRC comprises the Universities of Aberdeen, Edinburgh, Sheffield, Southampton, and the Open University. The authors are also grateful to Lin Lin and Xuezhou Yuan, who provided the implementation of the web service gateway plus the CIF/SWRL transformation tools.

References

1. Fish A. and Flower J. Investigating reasoning with constraint diagrams. In *VLFM04, Visual Languages and Formal Methods*, volume 127, pages 53–69. Elsevier, Rome, April 2005.
2. L Eshelman. Mole: a knowledge-acquisition tool for cover-and-differentiate systems. In S. Marcus (Ed.), editor, *Automating Knowledge Acquisition for Expert Systems*, pages 37–80. Kluwer Academic, Norwood, Mass, 1988.
3. Eugene C. Freuder. Partial Constraint Satisfaction. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, IJCAI-89, Detroit, Michigan, USA*, pages 278–283, 1989.
4. A.M. Frisch, M. Grum, C. Jefferson, B. Martinez-Hernandez, and I. Miguel. The essence of essence: A constraint language for specifying combinatorial problems. In *Proceedings of the 4th International Workshop on Modelling and Reformulating Constraint Satisfaction Problems*, pages 73–88, 2005.
5. G Kahn. More: From observing knowledge engineers to automating knowledge acquisition. In S. Marcus, editor, *Automating Knowledge acquisition for Expert Systems*, pages 7–35. Kluwer Academic, 1988.
6. Sandra Marcus and John McDermott. Salt: a knowledge acquisition language for propose-and-revise systems. *Artif. Intell.*, 39(1):1–37, 1989.
7. Gerrit Renker. A modeling framework for constraints. In *8th International Conference on Constraint Programming*, pages 8–13, September 2002.
8. Barbara Smith. A tutorial on constraint programming. Technical Report 95.14, School of Computing Research Report, University of Leeds, April April 1995.
9. Yi Zhang, Wamberto Vasconcelos, and Derek Sleeman. Ontosearch: An ontology search engine. In *The Twenty-fourth SGA International Conference on Innovative Techniques and Applications of Artificial Intelligence*, Cambridge, 2004.

⁸ <http://www.aktors.org>