Meta-Constraints to Aid Interaction and to Provide Explanations.

Janet van der Linden

Faculty of Mathematics and Computing, The Open University,
Walton Hall, Milton Keynes MK7 6AA, England
j.vanderlinden@open.ac.uk

Abstract. We explore the use of meta-constraints as a way of providing explanations to the user**.** Meta-constraints can provide a summary of the state of the CSP, and thus form a way of leaving out a large amount of detail that would be unhelpful to the user when dealing with a large problem. The ideas are illustrated through the problem of University students selecting modules for their studies.

1. Introduction

When solving a problem, it is often as important to understand *why* a solution is considered a solution, as the solution itself.

In this paper I will propose a strategy for modelling a CSP, where by we deliberately add constraints to the CSP which can function as summaries of particular parts of the CSP. Those constraints then function as points that reflect useful information about the state of the CSP, and thus explain the current solution. In this paper we are concerned with the issue of modelling and representing the CSP: how can we provide explanations which form a useful tool for the users of a constraints based system?

I will discuss how meta-constraints can aid the process of interaction and are also able to provide explanations for the user. The type of explanation provided is different from those presented in other papers [2], [3], [10] because it does not describe all the detailed steps in the process of constructing a solution, but rather provides a high-level summary of the actual current solution. This is possible because meta-constraints can be used to model parts and sub-parts of the problem. This method of explanation is attractive because it provides a representation of the solution in which unnecessary and cluttering details are omitted, and instead the big lines are provided. This is particularly relevant in large problems, where the details of explanations would quickly overwhelm the user.

The approach is explained through the example of the Modular Course System, which is a system developed to assist students to select modules for their degree courses. Interaction and explanation are important aspects of this system, as the student needs to be able to specify their own preferences for modules, they need to understand whether their preferences contradict each other and they need to receive guidance on the parts that are missing from their programme of study.

2. Current Approaches to Explanation in CSP

A number of constraint programming systems produce no clear message in the case where there is no solution to the problem. This leaves the user to wonder what has happened: was the problem specified incorrectly, is there a bug in the solver, or is there genuinely no solution to this problem? Recently, efforts are being made to redress this imbalance, and to provide explanations. Usually, such explanations show the process of arriving at the solution [2], [3].

Jussien [3] distinguishes between two types of explanations: *contradiction explanations*, which explain the conflict between a current constraint and assignments made earlier; and *eliminating explanations*, which specify why values were deleted from a variable's domain. This notions of explanation is very similar to that of *justification* developed by Bessière, [1] in the context of Dynamic CSPs. In Bessière's algorithm, additional information (a justification) is stored each time a value is deleted from a variable's domain, in order to cope with the scenario where constraints are dynamically added or retracted from the constraint network. Jussien uses his *e-constraints* approach, i.e. the approach of *explanation-based constraint programming*, in a similar way in order to improve algorithms by using explanations. Overall, Jussien uses a quite formal notion of explanation, which can be used by algorithms, solvers, and even for co-operation between multiple solvers, but which provides a level of detail which is unlikely to be directly useful to a human user.

Wallace and Freuder raise the issue of the human user, in their paper: *Explanations for Whom*? [10] They identify a gap between the formal explanation, and the way that this can be represented to the user in a useful way, and set out to explore the issue of perspicuity with respect to the representability of explanations. They distinguish between the *explanation*, and the *explanatory sign*, where the explanatory sign is the way that the explanation, or part of the explanation can be conveyed to the user. Examples of explanatory signs are given, such as 'highlighting queens' on a games board for use in a queens problem, colour coding and other graphical methods to present information. It would appear that a considerable amount of information can be left implicit, while still presenting the right explanation.

The ideas on representability of explanations are of interest to the main motivation of this paper, which is to provide explanations that are useful to the user. However, Wallace and Freuder explore the notion of perspicuity with a number of problems, which are small and very homogeneous. The example problem discussed in this paper is the exact opposite : it is very large and very structured and heterogeneous.

3. Background of the Modular Course Problem: the need for Meta-Constraints.

The motivation for this research lies with the problem of students selecting modules for their studies at a university. A proto-type system was built which works interactively, and also gives insight into why certain combinations of modules are not allowed, and whether all the field regulations have been met. We have to remember that modular courses were very much developed in the spirit of giving students freedom to select the courses they actually want to study. Flexibility in study programmes is a key issue, and therefore, flexibility in the system that supports students is equally important. For this reason the system was built as a constraints management system, i.e. when the user inputs information about which modules they explicitly wish to take or avoid, the system will report how this affects certain constraints. It will not automatically try to produce an overall solution, but instead it will present up-to-date views of the state of the problem, giving the user a chance to decide on the next move. In this respect, interaction and explanation are very important features of the system.

The problem itself, i.e. the university regulations with all their peculiarities, turned out to be a very complex one as well as being very large. There are some complicated rules as to which combinations are allowed, and in which order modules are to be taken. In particular there are numerous rules like *'if you want to take this module, then this module should be taken first*', or '*if studying this field then at least 7 of the following modules should be taken, but not more than 10*'. A further characteristic is that only parts of the variables and constraints are involved in any particular instance of the problem (i.e. if studying French, then the constraints and variables related to Computing and Business Studies are not applicable). Finally, constraints appear to fall into groups, where some constraints may be representing the fact that either two single fields need to be selected, or one double field, while other constraints actually represent the field regulations of a certain field, and yet other constraints relate to the existence of prerequisites. In that sense the problem is not a flat problem, but it has structure and appears to consist of several sub-parts, each of which needs solving.

The above characteristics lead to the area of research where configuration and constraint satisfaction meet, in particular that of Dynamic CSP (DCSP) as defined by Mittal and Falkenhainer[4], and recently re-defined as that of Conditional CSP (CCSP) by Sabin and Freuder[5]. In DCSP or CCSP special constraints are introduced which can activate *variables* to become part of the problem. However, in the current problem domain it is more appropriate to have *constraints* that can be activated if they need to be part of the problem. Secondly, in this problem domain it is important to be able to reason about the *groups of constraints* that are satisfied or not. This is much in line with additions to Constraint Logic Programming, in particular that of the cardinality operator by van Hentenryck[8],[9]. This operator works as a meta-structure, ranging over other constraints. Finally, the user should be able to select a section of the problem, resolve that section, and request for the rest of the problem to be solved in line with the section resolved already. This requires that the problem solving process is modelled as an interactive one, allowing a step-by-step approach.

3.1 Some Definitions and Network Representation

In the work for the Modular Course System, a network structure was developed as shown in Fig. 1.

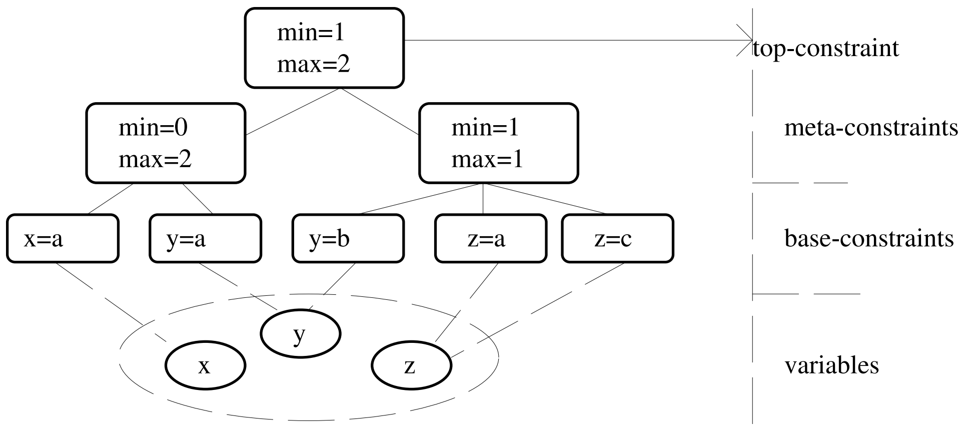

Fig. 1 Meta-Constraints' Network Representation

The variables are stored separately from the constraints and they form an unordered set with no relationships directly between them. The constraints in the problem are linked to each other as in a tree structure, which expresses the fact that they are ordered, and relate to each other as in a parent-child relationship. The constraints and the variables are connected through the base-constraints, which are situated at the bottom of the tree, 'the leaf-nodes'. Because the constraints in these leaf-nodes are connected to variables, the problem as a whole stops being tree-structured, and turns into graph. Therefore, unfortunately, the problem is in essence still like the classic CSP, and belongs to the class of NP-complete problems.

The satisfaction of a base-constraint depends directly on the value assigned to the variable it relates to. Furthermore, a base-constraint cannot itself be a parent to another constraint, and each base-constraint points to one variable only. A meta-constraint has a minimum and a maximum value, and a set of constraints that it ranges over. We say that if at least *min* and at most *max* of the dependent constraints are satisfied, then the meta-constraint is satisfied. The top-constraint represents the problem as a whole.

In order to solve a CSP structured as above, the user can select any constraint, and request for it to be satisfied. This action will cause propagation through the whole network of constraints and variables, and will result in constraints being set to satisfied or unsatisfied (or other satisfaction values) and of values being assigned to variables. The algorithm is described in more detail in [6], [7]. Here it is important to emphasise that unlike in normal CSP approaches, where we ask to solve the whole problem, in the current model we expect that the user wishes to select first one part of the problem to work on and resolve that. They can then either move on to the next, or request the solver to solve the rest of the problem in line with the decisions made earlier. The way a user can select a part of the problem to be resolved, is by picking a meta-constraint (or base-constraint if it is a very small part they wish to resolve) and request for it to be satisfied. In the next section we will see examples of this.

3.2 Examples of Meta-Constraints.

We mentioned earlier that the Modular Course Problem is an example of a problem where constraints appear to be grouped together, under certain headings. That is, the problem has a certain structure to it, which we want to exploit. Here we will show how the structure of the problem can be modelled using meta-constraints.

Within the Modular Course System three important areas can be identified that need to be resolved. First of all, there are numerous and strict regulations that need to be obeyed in the formulation of any programme of study. Secondly, students will have a number of preferences, which may vary from a certain dislike for a certain module, to a preference for a certain time-slot. Thirdly, the actual programme of study must make sense in terms of time-tabling requirements, such that it presents no clashes, and that the modules are spread out over the years in an even manner. Each of these three concerns can be represented by a meta-constraint, as shown in Figure 2.

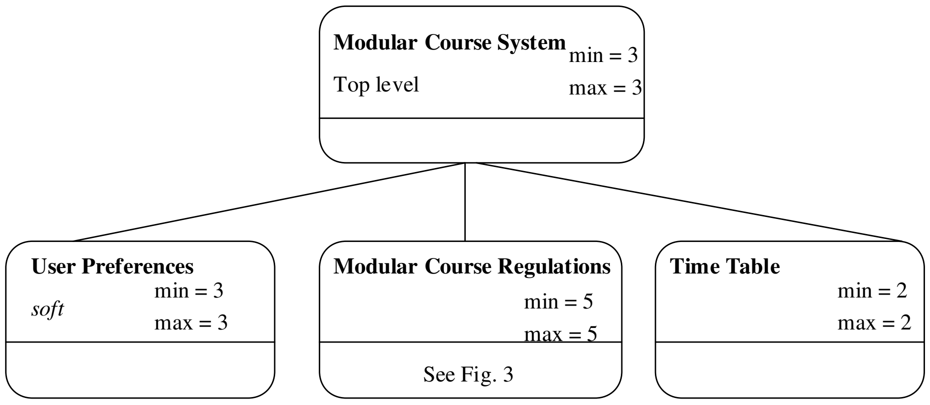

Fig. 2 Top level Constraint Representing the Problem as a Whole.

Note that in Figure 2 we see just the constraints, and we have to imagine the variables at the lowest level, after many layers of constraints. It is also important to keep in mind that the problem is still a graph, and not simply a tree, due to the fact that several constraints may be connected with the same variable.

Through the use of meta-constraints, we are able to model parts of the problem that we wish to be able to discuss and reason about. For example, the *User Preferences* are kept separate, so that we can reason about them separately from what the actual university regulations are.

The constraints relating to the Regulations can be further broken down into areas of concern. For example, there is a set of regulations referring to the fact that students should select either two single fields, or one double field. Another area of concern is that specific numbers of modules should be taken, at the appropriate level, and that this is different for the different types of award.

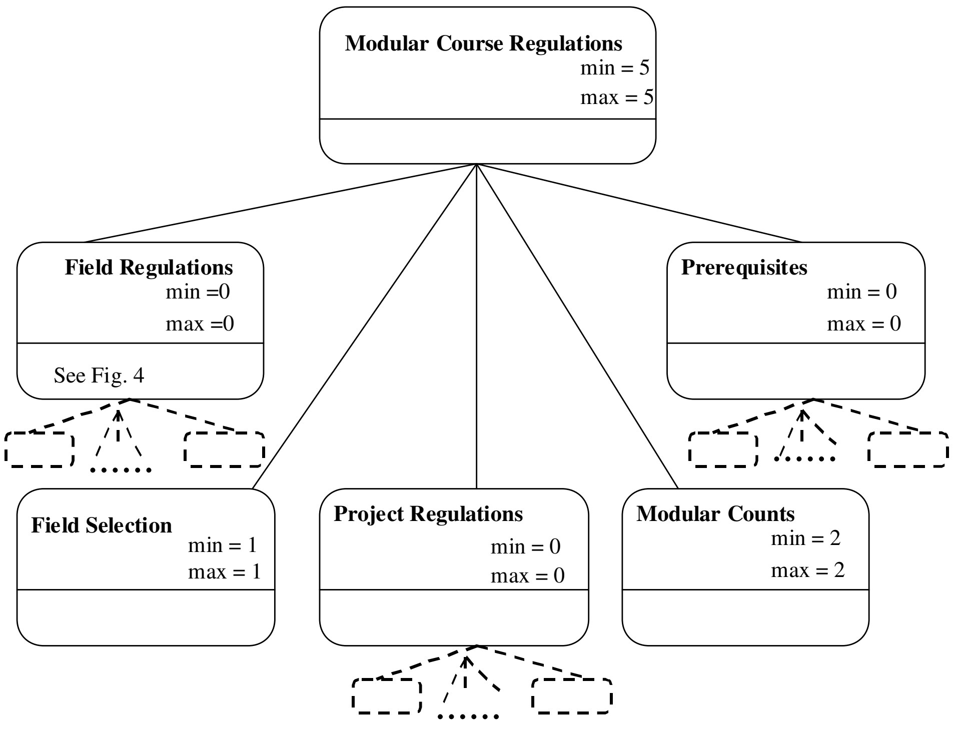

Fig. 3 Main Structure Underlying the Modular Course Regulations.

The structure displayed in Fig. 3 is an expansion of the meta-constraint labelled "Modular Course Regulations" shown in Fig. 2. It shows how the Modular Course

Regulations can be broken down into five sections: those relating to field regulations, the field selection, the actual module counts, the project regulations and the prerequisites.

If a student selects a certain module that they wish to study, then another module, or a combinations of other modules, have to be taken as prerequisites. In such a case, the constraints relating to those prerequisite requirements can be organised under the 'prerequisites' meta-constraint holder. As we can see, the meta-constraints are useful holders, or ways to organise an otherwise huge amounts of little constraints, that apparently have nothing to do with each other, but that now form some coherence by showing to which aspect they belong.

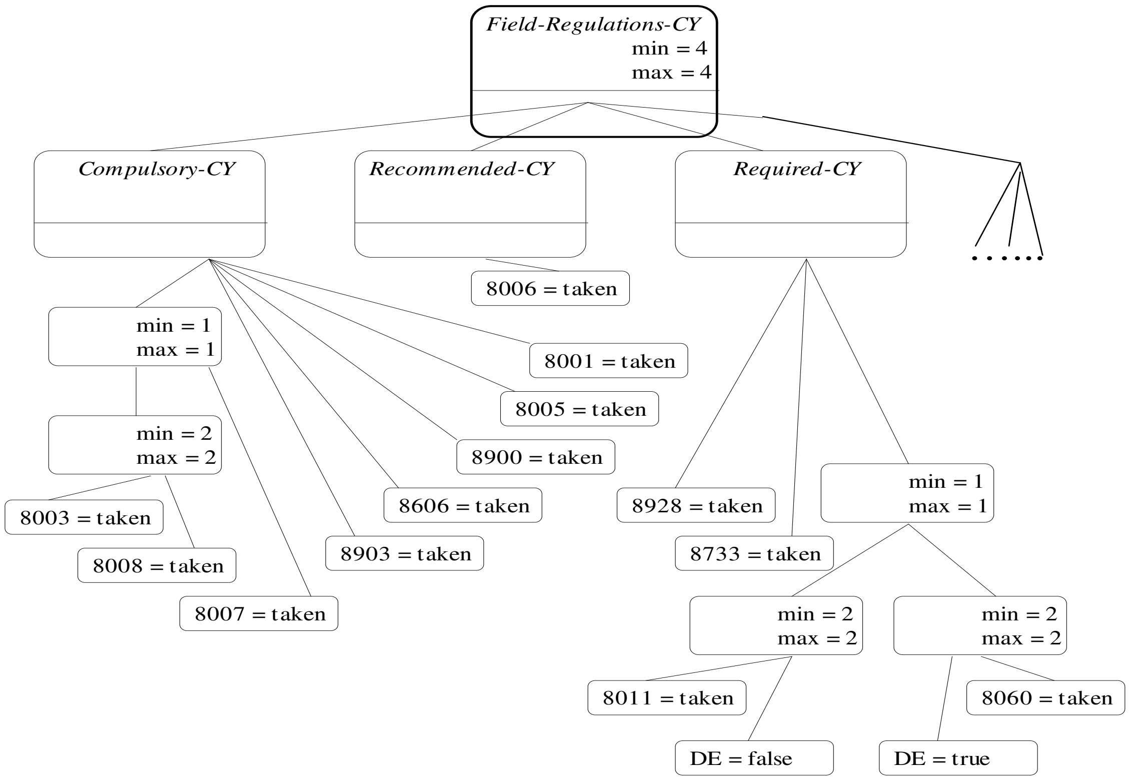

Fig. 4 Field Regulations for the Computing Technology (CY) field.
DE refers to Direct Entry Final Year students.

In Fig. 4 we see the result of the student selecting a certain field as the field they want to study. The example uses the Computer Technology field, which has code CY. Again, the constraints relating to this field can be broken down into categories of constraints. It is not necessary to fully understand the details of this example, but rather to gain a flavour of how after several layers of meta-constraints, we finally arrive at the level of the base constraints. It is at this level that we see an overwhelming amount of detail, and if explanations were to be worded in such terms, they would be quite unhelpful to most users.

4. Meta-Constraints as an Organising Principle: Interaction and Explanation

In the previous section we have seen an example of how the meta-constraints are used as an organising principle, and a way of modelling the problem. Rather than dealing with an otherwise flat CSP consisting of thousands of little constraints, we now see one which has structure. We see meta-constraints functioning as summaries.

Suppose a student expresses her interest for a particular field, and also that she is particularly keen on a number of modules, while wanting to avoid certain others. The feedback from that could either be seen as a huge expanse of little regulations that all need to be met, or it could be seen through the eyes of the meta-constraints that were shown in Fig. 3 and 4. Thus the student may learn that the compulsory modules are not yet incorporated in the schedule, because the constraint representing that section of the problem is still unsatisfied.

The user can also find out that their combination of preferences is an impossible one. They may have stated that they wished to avoid a certain module which is compulsory for the field of their choice The user will notice that there is a contradiction, either by noticing that the top-constraint is unsatisfied, which means that no overall solution can be found for this problem. Alternatively, they may notice that other important constraints, for example the Field Regulations for their chosen field are now unresolvable.

Currently, in the interface to the system the constraints are organized into lists depending on whether they are satisfied or not. This is very unsatisfactory and needs to change. This example shows that an important design issue for the interface will be to associate meaningful names to meta-constraints, because it is through these names, as text labels, that most of the explanation can be provided.

Jussien argues that explanations for basic constraints are easy to provide, but that those for global constraints are much harder to compute correctly. The example he refers to is that of the all-different constraint, for which it is difficult to represent what the exact triggering was that removed a value from a variable. He adds that global constraints require additional work, in order to make it possible to have them explained. We have seen that Jussien’s type of explanation is different from the one put forward here. We could argue, that even if it requires additional work, even if it means deliberately adding constraints, it will be worth the effort if the user has thereby gained a better understanding of state of the constraints in their problem.

The modular course system is a problem that is naturally complex and has a lot of structure. In the paper by Mittal et al[4], the example of the Car Configuration problem is given[1]. This is another example of a problem where constraints could easily be grouped together using meta-constraints: one meta-constraint to gather all the constraints relating to ‘engine features’, another one for ‘sun roof’ issues’ etc. See [6] on how to translate this problem into a structured meta-constraints problem.

Freuder and Wallace base their observations on experiments with the queens problems, and the nine-puzzle. Both are ‘flat’ problems and both are very small. They argue that their method of building explanations works well for small problems, but probably does not scale to larger problems. It is worth contemplating whether the cardinality operator could even be used in such problems. Or, whether the queens problem is unusually ‘flat’, and that most real problems have some type of structure that is waiting to be captured.

5. Future Work and Conclusion.

The approach of using meta-constraints as summaries of parts of the constraints network, can provide the user with information about the current state of the constraints and explain a solution. In this paper it was shown to be a fruitful approach, particularly for large problems which are very structured.

There are a number of directions for future work. First of all, we need to explore how this approach can be translated to be used with a more conventional constraint solver, using the cardinality operator for explanation. Secondly, at the moment the user interface is still very crude, and constraints are organised into different lists according to whether they are satisfied or not. The next step is to work on the representational issues.

References

1. Bessière, C.: Arc Consistency in Dynamic Constraint Satisfaction problems. In : Proceedings AAAI’91, 1991.
2. Freuder, E. C., Likitvivatanavong, C., Wallace, R. J.: Deriving Explanations and Implications for Constraint Satisfaction Problems. In: Proceedings of Principles and Practice of Constraint Programming – CP2001, 7th International Conference, Paphos, Cyprus, LNCS 2239, Springer, pp 585-589.

[1] The Car Configuration problem consists of configuring cars with accessories according to some level of luxury: ‘standard, deluxe and luxury’. Depending on these the car will or will not be fitted with air-conditioner, sun-roofs, etc..

3. Jussien, N.: e-constraints: Explanation-based Constraint Programming. In Workshop Proceedings UICS'01, User Interaction in Constraint Satisfaction, part of CP2001, Cyprus (2001), 59-72.
4. Mittal, S., Falkenhainer, B.: Dynamic Constraint Satisfaction Problems. In: Proceedings AAAI '90, Boston, MA (1990).
5. Sabin, M., Freuder, E. C.: Detecting and Resolving Inconsistency and Redundancy in Conditional Constraint Satisfaction Problems. In: Configuration, Papers from the AAAI Workshop, WS-99-05, AAAI Press, Menlo Park, California (1999) 90-94.
6. van der Linden, J.: Dynamic Meta-Constraints: An Approach to Dealing with Non-Standard Constraint Satisfaction Problems. PhD thesis, Oxford Brookes University, Oxford, England (2000).
7. van der Linden, J. : Assigning Satisfaction Values to Constraints: An Algorithm to Solve Dynamic Meta-Constraints. Proceedings ERCIM WG on Constraints (Prague, June 2001). http://arXiv.org/abs/cs/0110012.
8. van Hentenryck, P., Deville Y.: The Cardinality Operator: A New Logical Connective for Constraint Logic Programming. In: Furakawa, K.(ed.): Proc. of the 8th International Conference, France, (1991) 745-759.
9. van Hentenryck, P., Simonis, H., Dincbas, M.: Constraint Satisfaction using Constraint Logic Programming. In: Artificial Intelligence 58 (1992), 113-159.
10. Wallace R. J., Freuder E. C.: Explanations for Whom? In: Workshop Proceedings UICS'01, User Interaction in Constraint Satisfaction, part of CP2001, Cyprus (2001), 119-129.