PERIODICA POLYTECHNICA SER. EL. ENG. VOL. 37, NO. 2, PP. 131-142 (1993)

CONSTRAINTS: A PROGRAMMING PARADIGM AND A MODELLING METHODOLOGY

Gyula ROMÁN and Tadeusz DOBROWIECKI

Department of Measurement and Instrument Engineering
Technical University of Budapest
H-1521 Budapest, Hungary
email: roman@mmt.bme.hu
Phone and Fax: +36 1 166-4938

Received: September 20, 1993

Abstract

Constraints are often used as a formal approach to problems, because the very essence of the problem can be grasped by them. A lot of problems can be viewed as a set of variables and a set of relations on them. From this point of view the problem can be mapped naturally to a constraint network (the nodes of the network represent the variables; and the constraints in the network represent the relations between the variables of the problem); and this gives great significance to the research on constraints. An additional advantage is that they achieve global consistency through local computations.

Constraints and the Constraint Satisfaction Problem (CSP) can be classified by various criteria. The most significant classification is based on the type of the values assigned to the nodes.

Another possible classification of CSP is based on the kind of the required solution.

Significant effort was invested in developing general constraint programming languages (CPL) to provide an environment where the only thing a user has to do is to declare what she/he wants, not bothering how it is done. Though these languages aimed at generality, due to the limited ability of data abstraction and higher order constraints they could not fully achieve their goal. If the main stress is on the efficiency, dedicated solutions claim their place with their unique data structures and specialised constraint satisfaction algorithms.

The main goal of this paper is to give an overview of constraints as a flexible knowledge representation tool; to draw attention to the problems of representation and to methods of finding the solutions of the different types of constraint networks.

Keywords: constraint, CSP, discrete and continuous domain, optimal solution.

1. Introduction

In the early 60's constraints were introduced in the field of Artificial Intelligence (AI) as a new knowledge representation tool and were used mainly in graphical applications and in solving numerical problems. Since the first applications proved fruitful, more and more effort was invested in research of the field and it was found that though CSPs share common properties in general, they strongly differ in nature depending on their purpose and the type of values they handle.

In addition to its good modelling power, CSPs show other useful properties:

1. they degrade gradually under time limitations; interrupting the process before normal termination gives partial but useful information;
2. they are well suited to incremental system development; constraints may be added incrementally by incorporating it in the network, updating its arguments and propagating its effects (DAVIS, 1987);
3. they achieve global consistency through local computation.

The general definition of CSPs and some important definitions are given in Section 2. In Section 3 CSPs are classified by various criteria; the problems of each class of CSPs are outlined and some algorithms are presented. Section 4 deals with methods that try to solve very difficult or time consuming CSPs. In Section 5 the constraint programming languages are compared to dedicated solutions and Section 6 suggests some directions of development.

2. Basic Notions

Constraint satisfaction systems represent a declarative approach to problem solving. In such systems only the desired goal is specified, and the underlying constraint satisfaction algorithm is responsible for finding the solution. As a result, it is easy to represent problems, and as an additional advantage, this nonprocedural nature has potential for parallel implementation. Before the exact definition of the CSP consider a simple example of a constraint network. The statement

$$C = A + B \tag{1}$$

from a procedural point of view means that the sum of A and B is assigned to C. This equation holds just for the moment of assignment, though C will not follow subsequent changes of A and B. In procedural languages there are two different operators for the assignment and the logical equation expressing the different nature of the two operations. From a declarative point of view this difference does not exist, the system ensures that the constraints are satisfied. In the case of (1), it is rather a relation among three variables and the system tries to maintain the consistency of values or it reports a value conflict.

2.1. The Constraint Satisfaction Problem

A constraint satisfaction problem involves a set of n variables $X_1, \ldots, X_n$; all variables are associated with their domains $D_1, \ldots, D_n$ containing the allowed values $R_1, \ldots, R_n$; and a set of constraints C. The constraint $C_i(X_{i1}, \ldots, X_{ij})$ is a subset of the Cartesian product $R_{i1} \times \cdots \times R_{ij}$, that specifies which values of the variables are compatible with each other. (C_i is a k-ary constraint if it is imposed on k variables.) A solution is an assignment of values to the variables so that all the constraints are satisfied, that is, the constraint network is globally consistent (DECHTER et al, 1988). Consistency can also be defined on a smaller scale, for parts of the network. The following definitions are given by (MACKWORTH, 1977):

1. Node consistency
   node i is *node consistent* iff for any value $x \in D_i$, $P_i(x)$ holds, where $P_i(x)$ is a unary constraint.
2. Arc consistency
   Arc (i, j) is *arc consistent* iff for any value $x \in D_i$ such that $P_i(x)$, there is a value $y \in D_j$ such that $P_j(y)$ and $P_{ij}(x, y)$ where $P_{ij}(x, y)$ is a binary constraint between x and y.
3. Path consistency
   A path of length m through nodes $(i_0, i_1, \ldots, i_m)$ is *path consistent* iff for any values $x \in D_{i_0}$ and $y \in D_{i_m}$ such that $P_{i_0}(x)$ and $P_{i_m}(y)$ and $P_{i_0 i_m}(x, y)$, there is a sequence of values $z_1 \in D_1, \ldots, z_{m-1} \in D_{m-1}$ such that
   (i) $P_{i_1}(z_1)$ and $\ldots P_{i_{m-1}}(z_{m-1})$,
   (ii) $P_{i_0 i_1}(x, z_1)$ and $P_{i_1 i_2}(z_1, z_2)$ and $\ldots$ and $P_{i_{m-1} i_m}(z_{m-1}, y)$.

2.2. Representation

The representation of the problem is very important, because it greatly influences the efficiency of finding the solution. Networks of binary constraints are often represented by graphs where the nodes are the variables, and the arcs are the constraints. Networks consisting of k-ary constraints can be represented by hypergraphs (MONTANARI et al, 1991).

An ordered constraint graph is a constraint graph in which the nodes are linearly ordered to reflect the sequence of variable assignments executed by the backtrack search (DECHTER et al, 1985). Quite a few theorems have been stated and proved for constraint graphs of which some are of great importance:

1. An acyclic graph is globally consistent iff it is locally consistent (HYVÖNEN, 1991).
2. If the constraint graph has width 1 (it is a tree), and if it is arc consistent than the solution can be found without backtracking.
3. If the width of the constraint graph is 2 and it is arc and path consistent than the solution can be found without backtracking (FREUDER, 1982).

The importance of these theorems is that if the above conditions are met, or their cycles can be eliminated, than through applying existing efficient methods to accomplish local consistency (which can be done in polynomial time), global consistency can be achieved. With (1) and (2) the main problem is that the algorithms achieving consistency may increase the width of the graph.

3. Classification of CSPs

Constraint satisfaction problems can be classified by many criteria, however, the most important ones are:

- the type of solution sought;
- the type of the variable domains (finite, infinite);
- the type of values it handles (symbolic, numeric).

Depending on the nature of the CSP different algorithms are used to find the solution and usually these algorithms must be tailored to the actual problem to gain performance.

3.1. The Types of Solution

Formulating the problem as CSP the user may have different goals. In most of the cases only a single consistent solution is required. In some cases it is not sufficient, and more complicated criteria must be satisfied. Depending on the type of solution required the user faces the following problems (HYVÖNEN, 1991):

- satisfiability problem – the user's goal is to know whether a solution exists;
- finding one (any) solution;
- finding all the solutions (not ordered, but exhaustive);
- finding an optimal solution – denoted as the Constraint Optimisation Problem (COP).

3.2. Variables with Finite and Infinite Domains

Various real life problems lead to different representations. The difference is usually incorporated in the variables. In some cases the number of possible value assignments is limited; in other cases the domains of the variables are intervals containing a continuum of possible values. It is obvious that these two types of CSPs cannot be solved with the same algorithms. For problems with finite domain variables the backtrack algorithm has been developed. The backtrack search traverses the variables in a predetermined order, assigning consistent values to a subsequence $(X_1, \ldots, X_i)$ of variables and attempts to assign a value to a new variable X_{i+1} such that the whole set is consistent (DECHTER et al, 1987a). If no such value can be chosen the algorithm backtracks to the most recently instantiated variable and tries to assign a new value to it. This procedure is carried out until all the variables are instantiated and a consistent state is reached or the possible choices are exhausted. In the latter case the system reports failure. Considering the graph representation of CSPs this algorithm operates as the depth-first algorithm well known from the AI literature. To improve backtracking during the search two main modifications were made (HYVÖNEN, 1991):

Look-ahead scheme

a) variable ordering: instantiate first the variable that constrains the possible solutions maximally (the variable participating in the highest number of constraints);
b) value ordering: choose the value that maximises the number of options;

Look-back scheme

a) go back to the source of failure: find the first variable that has influence on the failure and choose the next value for that variable;
b) constraint recording (learning): the reasons of the failure are recorded, the repeated appearance of the situation can be recognized, the same problem can be avoided.

The four methods above could be grouped on the bases of the information they use: the first three utilise information coded in the structure, while the last uses heuristics to guide its operation. The most important problem area that involves continuous infinite domains is numeric equation solving, which can also be considered as a CSP. When solving an equation two major approaches are used:

1. exact value, and
2. interval based systems.

The exact value approach has some significant limitations (HYVÖNEN, 1991):

- a lot of problems have noisy, uncertain inputs, and the result should reflect this property;
- in many cases it is impossible to distinguish between inputs and outputs beforehand, the variables should be handled dynamically;
- an exact value system cannot handle families of solutions (e.g. an interval);
- numerical constraint propagation systems based on exact values are often incomplete, the solution is not guaranteed even if one exists, or only one solution is found, though there exist more.

Interval based systems offer solution to the last problem, though it also has some drawbacks:

- too many and too detailed solutions;
- computational inefficiency;
- it deals with infinite domains, which introduces additional difficulties.

From these points it follows that the notion of solution has to be abstracted to an interval solution, i.e. the solution also has to be sought as an interval (handling the groups of solutions).

The problems involving symbolic data processing can be mapped to CSPs with finite, discrete variable domain; the algorithms discussed above can be applied.

4. Algorithms and Methods

To solve a given CSP, different algorithms are needed. Though the main stress is on finding a solution, the resources needed for the action cannot be neglected. Montanari showed that the general CSP is NP complete (MONTANARI, 1974). Then it follows that the worst case time complexity of finding a solution is exponential in the number of variables. This phenomenon is also known in the field of AI and in graph theory in connection with graph traversing; and is often denoted as combinatorial explosion. Researchers aim at finding methods that under special circumstances, decrease the complexity of finding a solution (FREUDER, 1982), (DECHTER et al., 1991). Efforts are made to make the search backtrack-free. The rationale for this goal is that the complexity of generating the exact-value solutions by a backtrack-free search is linear with respect to the number of variables (HYVÖNEN, 1991).

4.1. Consistency Algorithms

While the goal of search algorithms is to find consistent solutions, the consistency algorithms aim at the removing of inconsistencies from the network, because such inconsistencies may result in costly recomputations. First Mackworth drew the attention to the importance of the consistency algorithms and gave the definitions of node, arc and path consistency (MONTANARI, 1974), (MACKWORTH, 1977) and developed algorithms to exclude inconsistent situations. These algorithms are pre-processing techniques that transform a given constraint network into a more explicit representation before it is subjected to a backtrack algorithm (DECHTER et al., 1989). Using consistency algorithms a lot of unnecessary recomputation can be avoided, speeding up the solution of the problem. Since in numerous applications the use of consistency algorithms proved to be useful, a lot of effort was directed toward the improvement of their efficiency (MACKWORTH et al., 1985), (MOHR et al., 1986). If the graph is ordered, the 'directed' version of the consistency algorithms can be used, that is, the consistency is only checked along the given ordering. If the order in which the backtrack algorithm will eventually search the problem is taken into account, the processing of many constraints which are unnecessary for the search can be avoided (DECHTER et al., 1989).

4.2. Using Easy Problems to Formulate and Solve More Difficult Ones

Sometimes the user faces problems which are not solvable or impractical to solve due to the huge amount of time they require. In such cases the user may decide on solving a 'weaker' version of the problem (Partial CSP), hoping to receive some aid to solve the original problem (FREUDER, 1989). This draws attention to a space of alternative problems, some of which may be both solvable, and 'close enough' to the original one. For this approach a metric has to be defined on the problem space to be able to decide whether a solution in the alternative space is close enough to the original problem's solution. On the other hand, it should be possible to manipulate the representation of a difficult problem until it is transformed into a simpler one, solve that problem, then use the solution to guide the search for the original problem. Before going further, the easy problem has to be defined more precisely.

In general, a problem is considered easy when its representation permits a solution in polynomial time. Since variable instantiation and preprocessing can be done in polynomial time, but backtracking cannot, problems that can be solved without backtracking are considered (DECHTER et

al., 1987a). This property is strongly dependent on the topology of the constraint graph.

To utilise the easy problem approach three major steps are required:

- simplification;
- solution;
- advice generation.

4.3. Redundancies in the Network

Though consistency algorithms remove inconsistencies from the network, they introduce redundancies to the system (there is more than one piece of data in the data base prohibiting certain actions). This has two major disadvantages:

1. such representations might be excessive in storage space,
2. redundancies might hide some important properties of the constraint graph that could be exploited in the search for solutions.

The connective structure of the constraint graph is of particular importance possessing great influence on the tractability of the problem (DECHTER et al., 1987). The structure of the constraint graph and the complexity of the solution are strongly related. Freuder proved that if the graph is a tree then the backtrack-free search is guaranteed (FREUDER, 1982).

Thus, if the problem, (i.e. its graph) shows some desirable properties, it might be beneficial to preserve them to guarantee a simpler solution. It was found that the amount of improvement is directly related to the sparseness of the constraint graph. So, the removal of redundant constraints, (i.e. their corresponding edges in the graph) has a potential for improving the performance of backjumping. Though the advantage of this method is obvious, its application generally increases the search space, which may override its benefits.

5. Applications of Constraints: Constraint Languages and Dedicated Systems

With the development of constraint systems it was recognised that there are classes of problems with similar structure which can be solved by the same algorithms. This idea has led to the implementation of constraint programming languages, which aim at specific problem areas, but provide the user with general tools, and permit him formulate his problem in a declarative

way. Quite a few languages were developed (e.g. Sutherland's Sketchpad, Borning's ThingLab, Nelson's Juno or Van Wyk's IDEAL (LELER, 1988)). There are several advantages of CPLs for the user. Since a CPL is declarative in nature, the programmer just specifies his goal and lets the underlying algorithm to accomplish the task. As a result, constraint programs are easy to build and modify, no traditional programming expertise is needed. Though with such languages it is easy to specify the problems correctly, it might be difficult to solve the CSP. To overcome this problem the constraint satisfaction algorithm must be designed with great care. With imperative languages when a problem is formulated, it is easy to solve, but in complicated cases it might be very difficult to formulate it.

5.1. Constraint Satisfaction Mechanisms Used in CPLs

In existing CPLs different mechanisms are used to find the solution. The most important ones are (LELER, 1988):

- Local propagation – if there is enough information available (values) to inference a new value then a rule is fired, the new value is propagated through the network. If there are more rules available these constitute a conflict set, a conflict resolution algorithm has to be used to decide which one to fire. A significant advantage of this method is that the system can keep trace of fired rules; this information can provide useful help to generate explanation. However, this method lacks the ability to recognize cycles, because no global information is available about the network.
- Relaxation – this method was developed for objects with continuous numeric values, and error estimates are used to guide the process. To make a new guess the system perturbs the value of each object in turn and watches the effect of this perturbation on error estimates. This method can be seen as a kind of heuristic search in continuous numeric domain. The Newton–Raphson method is widely used to iteratively compute new values.
- Propagating degrees of freedom – in some cases the value of a variable can easily be computed if another one, whose value is difficult to determine, is already known. In such cases the constraint solver does not deal with the former variable first, it determines the values of the 'difficult' variables (these might be part of a cycle), and finally the values of the 'easy' variables are simply computed. Parts of the constraint network containing such 'easy' variables can be precompiled to speed up the computation.

- Redundant views – with introducing redundant views into the representation of the problem global information is introduced, which can guide the search when the information locally available is not sufficient.
- Graph transformation (term rewriting) – the constraints are represented as a graph. Cycles present in the graph can be eliminated by applying term rewriting rules such as substituting $2X$ for $X + X$ ($X + X$ might have introduced a cycle in the graph).

5.2. Limitations of CPLs

Problem-solving systems are typically very difficult to implement, and constraint satisfaction systems are no exceptions. Though constraint languages were around for over twenty years, relatively few systems have been built in that time. Furthermore, the existing systems are very application specific, though some of them provide some tools to extend their abilities. However, in most of these systems the user needs programming skills since he has to drop one level and use the implementational language's primitives to achieve the desired extensions.

The main problems, the existing CPLs suffer from, are (LELER, 1988):

- General problem-solving techniques are weak, and constraint satisfaction systems must use application specific techniques. It is usually difficult to modify the implemented algorithms to suit the needs of other, significantly different applications. Most of the systems capable of adjusting their algorithms to serve different problems do it by dropping one level of abstraction and by using the primitives of the implementational language (e.g. Borning's ThingLab).
- Some of the languages do not support extendability, it is inconvenient to introduce new aspects.

Many constraint satisfaction systems use iterative numeric techniques such as relaxation. These techniques may have stability problems; such a system might fail to terminate even when there is a solution. The answer might depend on the order in which the constraints are solved, which effectively destroys any declarative semantics.

To overcome these limitations a general purpose language called Bertran was developed, which allows new constraints to be defined and it also has a form of abstract data type (LELER, 1988).

5.3. Dedicated Solutions

General problem solving tools are very efficient when a new problem is to be solved because the effort needed for the solution in terms of pro-

gramming is small, though the user has to make the abstractions to raise the problem's abstraction level to match that of the general constraint solver. However, in the case of problems where the time required to find the solution is limited and this time limit is very strict, dedicated solutions still claim their place. In an algorithm fully tailored to the actual problem, the special properties can be fully exploited (e.g. the causal relations (DECHTER et al., 1991)) thus speeding up the solution and making the operation of the system more effective. For example, in test generation for digital combinational circuits no numerical computation is needed, so a general problem solver capable of handling numeric values is redundant, the general data representation does not allow to reach the maximal performance. A test generator for combinational circuits tailored to the problem is described in (TILLY et al, 1993).

6. Conclusion

In the last three decades a lot of effort has been invested in the research of constraint satisfaction problems, and quite a few efficient algorithms have been developed. However, in most of the cases researchers concentrated on problems with purely finite or infinite domain variables. Although the algorithms developed so far promise good results in certain applications, most of them cannot cope with engineering problems, because in real life situations the problems to be solved are complex, and they cannot be mapped to CSPs with variables of one type, or such a mapping needs significant simplification, which degrades the solution. Moreover, in some cases the CSP constitutes a compact system, and it is a *part of a larger system*; the resources available for it are limited due to the fact that the CSP is just a part of the problem to be solved and the resources are allocated to the global problem. In such cases one has to be satisfied with an *approximate solution* though emphasis has to be put on the quality of approximation. The advantageous property of CSPs, i.e. that they degrade well under resource limitations, should be better exploited. In addition to the above problem of complex systems *different knowledge representations have to be efficiently integrated into a uniform system* which needs further exploration, because the overall performance of the system strongly depends on the interaction between the parts.

Another problem that should be addressed is the *constraint optimisation problem.* To find an optimal solution by searching for all possible solutions and then ordering them is very time consuming. Dechter investigated this problem and found that the optimisation task does not require exhaustive search among all consistent solutions, but can rather be natu-

rally incorporated into the process of finding a consistent solution. In many cases the computational complexity of finding a solution is not increased by the objective function; and when it is, this increased complexity can be estimated beforehand (DECHTER, 1990).

Developing solutions to the above problems need further research, however, these solutions can be beneficial in future engineer applications.

References

DAVIS, E. (1987): Constraint Propagation with Interval Labels, *Artificial Intelligence*, Vol. 32, pp. 281–231.

DECHTER, R. – PEARL, J. (1987a): Network-based Heuristic for Constraint Satisfaction Problems. *Artificial Intelligence*, Vol. 34, pp. 1–38.

DECHTER, R. (1990): Optimisation in Constraint Networks. Influence Diagrams, Belief Nets and Decision Analysis, Chapter 18, John Wiley & Sons Ltd.

DECHTER, A. – DECHTER, R. (1987b): Removing Redundancies in Constraint Networks. *Proc. AAAI–87*, pp. 105–109.

DECHTER, R. – PEARL, J. (1991): Directed Constraint Networks: A Relational Framework for Causal Modelling. *Proc. IJCAI–91*, pp. 1164–1170.

DECHTER, R. – MEIRI, I. (1989): Experimental Evaluation of Preprocessing Techniques in Constraint Satisfaction Problems, *Proc. IJCAI–89*, pp. 271–277.

DECHTER, R. – PEARL, J. (1985): The Anatomy of Easy Problems: A Constraint Satisfaction Formulation, *Proc. IJCAI–85*, pp. 1066–1072.

DECHTER, R. – PEARL, J.(1988): Tree-clustering Schemes for Constraint Processing, *Proc. AAAI–88*, pp. 150–154.

FREUDER, E. C. (1982): A sufficient condition of backtrack-free search. *J. ACM* Vol. 29 (1) pp. 24–32.

FREUDER, E. C., (1989): Partial Constraint Satisfaction, *Proc. of the 11th IJCAI*, pp. 278–283.

HYVÖNEN, E., (1991): Constraint Reasoning with Incomplete Knowledge. The Tolerance Propagation Approach. Technical Research Center of Finland, Publications 72, VTT, Espoo, 1991.

LELER, W. (1988): Constraint Programming Languages, Addison-Wesley Publishing Company.

MACKWORTH, A. K. (1977): Consistency in Networks of Relations, *Artificial Intelligence*, Vol. 8, pp. 99–118.

MACKWORTH, A. K. – FREUDER, E. C. (1985): The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems. *Artificial Intelligence*, Vol. 25, pp. 65–74.

MOHR, R. – HENDERSON, C. (1986): Arc and Path Consistency Revisited. *Artificial Intelligence*, Vol. 28, pp. 225–233.

MONTANARI, U. (1974): Networks of Constraints: Fundamental Properties and Application to Picture Processing. *Inf. Sci.* Vol. 7, pp. 95–132.

MONTANARI, U. – ROSSI, F. (1991): Constraint Relaxation may be Perfect. *Artificial Intelligence*, Vol. 48, pp. 143–170.

TILLY, K. – SURJÁN, L. – ROMÁN, Gy. (1993): Automatic Test Pattern Generation can be Solved as a Constraint Satisfaction Problem, *Proc. of EUROMICRO '93*, pp. 715–722, Barcelona, 1993.