PERIODICA POLYTECHNICA SER. EL. ENG. VOL. 39, NO. 1, PP. 63-71 (1995)

OPTIMUM PROBLEMS IN CONSTRAINT SATISFACTION

Gyula ROMÁN and Tadeusz P. DOBROWIECKI

Department of Measurement and Instrument Engineering
Technical University of Budapest
H-1521 Budapest, Hungary

Received: March 5, 1995

Abstract

With the advanced spread of Expert Systems (ES) more and more emphasis has been put on efficient knowledge representation. Constraints, a fast developing field in Artificial Intelligence (AI), try to cope with the new requirements imposed on the knowledge representation tools in practical applications. Efficient algorithms operating on constraint networks have been developed [4, 7, 10], and recently some constraint systems have been constructed [3], which are capable of solving relatively complicated problems stated in a declarative way. However, these systems put emphasis mainly on finding a feasible solution, they do not rank the different solutions. In some cases this will not suffice. When optimum criteria is meaningful to define, an optimal solution can be of importance.

The present paper gives a brief overview of constraints, emphasizes the importance of searching for an optimal solution; and gives an application area where the optimal solution would highly improve the performance of the embedding system.

Keywords: knowledge representation, constraints, Constraint Optimisation Problem (COP).

1. Introduction

In the last few decades great efforts have been invested in AI research and as a result quite a few ESs have been developed and put into use. The first approach to represent knowledge in such systems was the rule-based description of the available information. However, it became shortly obvious that in more complex systems the knowledge involved is too diverse to encapsulate it in a single formal approach; and there have been strong demands for knowledge representation tools capable of expressing information in a declarative way. In the late 60's constraints entered the stage of AI and offered solution for the above problem. Recently, the field is dynamically expanding, new efficient algorithms operating on the constraint network have been presented, and the Constraint Satisfaction Problem (CSP) has been mathematically founded. Some constraint systems have been presented in the literature [3], which are capable of solving relatively complicated tasks that are introduced to the system in a declarative way. These systems mainly concentrated on finding a feasible solution, no matter of the quality. In some of the cases when optimum criteria of some kind are meaningful to define the performance of the system can be improved by adding these optimum criteria to the system and searching for an optimal solution.

The following parts give a brief overview of the CSPs and deal with optimum problems in CSPs.

2. The Constraint Satisfaction Problem – A Survey

A constraint satisfaction problem involves a set of variables $X_1, \dots, X_n$; all variables are associated with their domains $D_1, \dots, D_n$ containing the allowed values $R_1, \dots, R_n$; and a set of constraints C. The constraint $C_i(X_{i1}, \dots, X_{ij})$ is a subset of the Cartesian product $R_{i1} \times R_{ij}$, which 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 [8]. A number of methods has been developed utilising this concept, some of them are more general, some exploit specific properties of the net [4,7,10].

There are two major classes based upon the type of variables present in the system:

(1) discrete (finite or countable set of possible values), and
(2) continuous (domains are subsets of the continuum).

The goal of the Constraint Satisfaction is usually set as:

(1) to determine whether a solution exists at all,
(2) to find a single solution,
(3) to find all solutions, or
(4) to find an optimal solution of some kind.

The most time consuming worst case is faced when there is no solution at all. To have a simple method then to determine the existence or rather the lack of a solution might come handy. In cases when cost criteria are meaningful to define, an optimal variable setting might be of importance. The situation seems simpler, than it really is. Developed systems implement either discrete or continuous variables and most of them use incomplete inference techniques. Problems arise when constraints are applied to model real-life situations. Here typically both types of variables are present and mixed methods capable of handling them are needed. A description of such implementation can be found in [3]. Another problem is that constraints are often implicit and the overall system could be underdefined. In such cases heuristic decisions have to be made that lack required basics to be surely correct. In addition, methods are also needed to handle the incorrect choices and to make the overall method complete, i.e. to guarantee a solution if one exists, and to declare the non-existence of it if there is none.

Although the main emphasis is on finding a solution, resources required for that action cannot be neglected. MONTANARI showed that the general CSP is NP complete [16]. It follows then that time complexity of finding a solution in the worst case is exponential in the number of variables. This phenomenon is also met in the field of AI and in graph theory, in connection with graph traversing, and is often referred to as combinatorial explosion. Many researchers dealt with methods which under special conditions decrease the complexity of finding the solution [9,5]. For efficiency reasons fast local propagation techniques are often used. The price of speed is, however, incompleteness, and these methods not always can produce a solution or determine that system of constraints is unsolvable. Such techniques should be augmented with a kind of backtracking search, which however costly, can guarantee a solution. Efforts are made to make the search backtrack-free. The rationale is that generating exact solution by backtrack-free search is linear (with respect to the number of variables) [12], which draws attention to various pre-processing techniques, minimising backtracking, or to intelligent backtracking methods [6,13,14,15].

Recently significant interest has been shown in Constraint Logic Programming (CLP) languages. Many implementations can be found in the literature (see e.g. [1,2] for a survey). They incorporate some kind of local propagation technique based on backtracking method already present in Prolog. Also recently a system was presented, which utilised the power of Nondeterministic Lisp [3]. It allows for both discrete and continuous variables and incorporates functions to handle the underdefined constraints. The constraint network is built incrementally, always checked for consistency. That way the recognition of inconsistency is faster, since it is grasped exactly in the moment it is introduced to the system.

Despite considerable difficulties indicated above constraints constitute knowledge-representation very useful in numerous practical applications [17]. Constraints are advantageous due to:

(1) good representational efficiency, constraints:
   - support hierarchical description,
   - make it possible to integrate details modelled at different levels of abstraction into a single body,
   - are easy to integrate with other knowledge representation schemes, e. g. object oriented tools, semantic nets, frames, etc.,
   - support inference making,

- give good representation for 'deep' models, considering that a part of physical laws and systemic descriptions are just constraints on state variables,

(2) good degradation under time limitations, furthermore constraints:
(3) are well suited for incremental system development,
(4) achieve global consistency through local computations.

3. Optimisation in Constraint Networks

Constraint networks have so far been applied to the task of finding feasible solutions, because in many AI applications preferences play only a minor role and finding a feasible solution suffices. However, in some of the practical applications meaningful optimum criteria do exist, and the optimality, the quality of the solution has very strong influence, far reaching effect on the performance of the overall system. In such cases the optimal or at least a sub-optimal solution has to be sought for.

Different possible approaches exist to find the optimum:
(1) to search for all the solutions and to choose the best at the end;
(2) to integrate the optimum criteria into the constraint network;
(3) to tune a feasible solution towards the optimum.

The first approach seems the easiest, though there are certain difficulties. The system has to guarantee that it finds all the solutions and has to keep track of the process not to find the same solution several times. The unnecessary computations might measurably decrease efficiency.

On the other hand, the optimisation task does not require an exhaustive search among all consistent solutions but rather can be incorporated naturally into the process of finding consistent solutions. In many problem instances the interaction between the optimum criteria and the constraints does not add any computational complexity to the task of finding a consistent solution, and when it does, the extra complexity can be estimated beforehand, so the system can decide whether to use an exact or a heuristic approach to optimisation [11].

In some of the cases, when the criteria are simple, they can be incorporated into the constraint network. The main problem with this approach is that it is difficult to formalise minimum and maximum conditions in a constraint network; it might need significant efforts to tailor the tools handling the network to the minimum conditions.

3.1 Solution Tuning

The process of the solution tuning is as follows:

1. Find a feasible solution
2. Evaluate the cost function
3. Choose a variable to tune (based on the weight of the variable in the cost function)
4. Alter its value
5. Repeat till there is improvement

The concept underlying solution tuning is that a feasible solution is sought first, neglecting the optimum criteria, and this solution is modified – tuned – to find the optimum. The main advantage of this approach is that at every step of the process a solution is available, so if the process is interrupted due to resource limitations, the task is still partly accomplished.

3.2 Problems with the Approach

Though the above approach seems promising there are some problems to face. The most crucial is when the problem-space is partitioned. In this case there are independent sets of solutions, and it is not possible to step from one set into another one by tuning. (Usually singularities break up some of the variables, and it is not possible to continuously tune those variables without violating some of the constraints.) Due to this property the system might find a local minimum of the cost function instead of the global one. *Fig. 1* demonstrates this problem. If a feasible solution is found in which v_1 is instantiated to u_1, then $\min_1$ is reached though the global minimum of the cost function is $\min_2$. If the starting point had been u_2, the global minimum would have been found. This draws attention to the importance of finding the partition containing the global minimum.

On the other hand, it is also a delicate question which variable to tune and how to tune it. The speed of convergence of driving the solution into the optimum strongly depends on the above two selections. If only polynomial cost functions are allowed then a good idea can be to tune the variables in the order of their exponents; the higher exponent represents greater weight in the cost function. However, great care must be taken, because the weights and the values of the variables also have to be taken into account. (E. g. a variable v_1 has an exponent of 10 but its value is small, 0.1, and another variable v_2 has a smaller exponent of 2, but its value is large, 10, then the second variable has greater weight in the cost function, so its value has to be tuned first.)

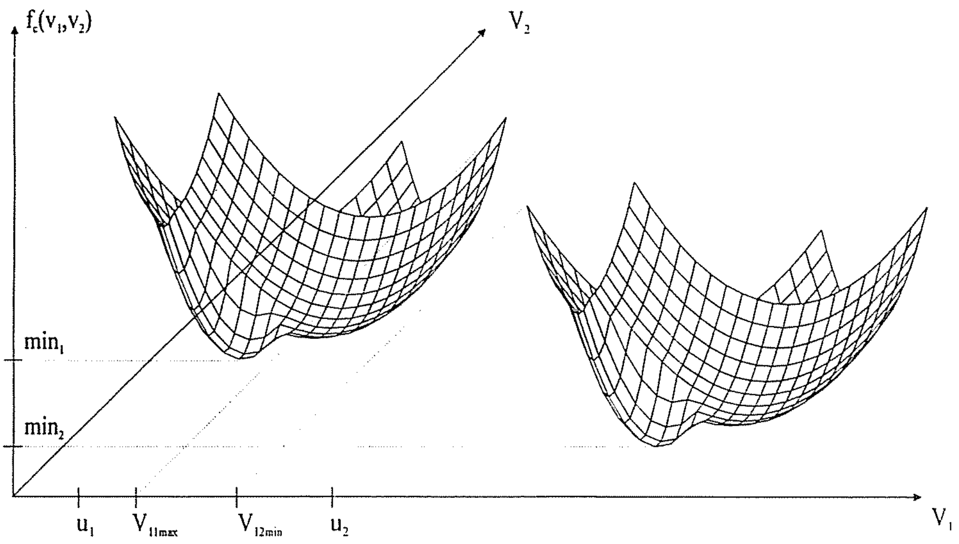

Fig. 1. Partitioned problem-space with local minimum

4. Preferences – The User's Subjective Goals

It occurs very often that in addition to the above optimum criteria the user wants to integrate his subjective goal into the system. These goals have to be respected, though these are weaker conditions than the constraints themselves; no constraint can be sacrificed to achieve the subjective goal, but if there are more solutions satisfying the constraints, the solution has to be chosen that matches the subjective goal better. To express these subjective goals preferences are proposed. The original domain of the variables can be organised into hierarchies, increasing levels representing loss in quality. The sub-domains at the top of the hierarchy are the tightest, those at the bottom are the widest, they are the same as the original ones. The constraint solver tries to instantiate the variables from the sub-domains at the top and it drops a level only if some of the constraints are violated, and they cannot be satisfied with values from the actual level. *Fig. 2* shows the hierarchical structure belonging to the variables with preferences. There are two variables; in case of A the original domain is [0, 30] and the higher values are preferred; while in the case of B the original domain is [15, 45] and the lower values are preferred. An ordering is present between the variables in a form of a constraint: $A < B$. The different levels represent different quality instantiations. With increasing level the quality decreases. So if B_1 is assigned to B (from Level 1, best quality), no value can be selected for A from Level 1, a level has to be dropped. However, if B is instantiated to B_2, the constraint can be satisfied with assigning value to A from Level 1, thus providing a better quality solution.

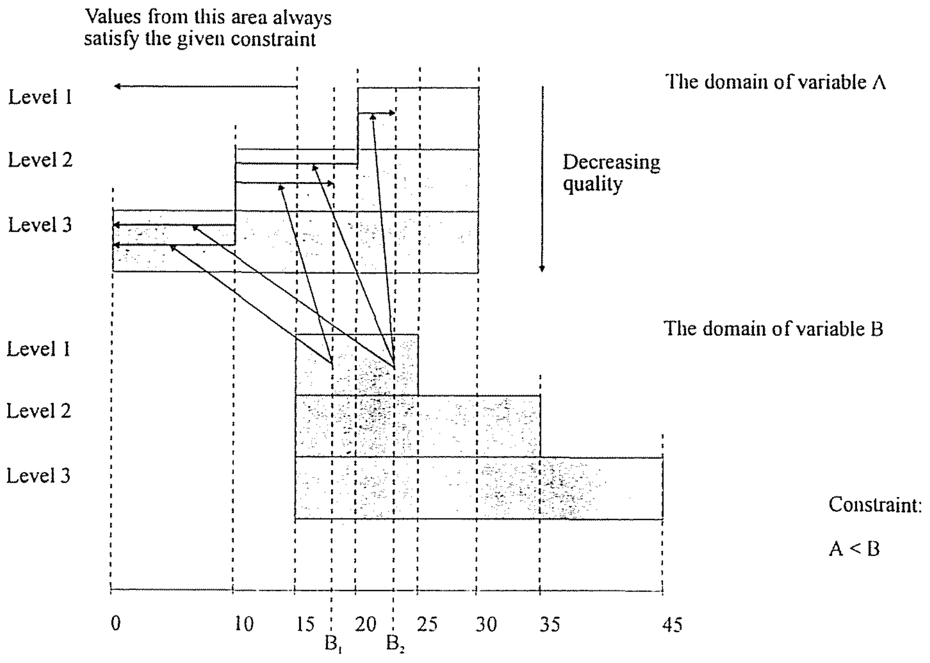

Fig. 2. The preferential treatment of the variables

5. Place of Optimisation in CSPs – An Example

In several practical applications the optimum criteria can naturally be derived from the problem itself. Measurement technical applications constitute an important application area where optimisation has great significance. When designing an ES to solve measurement technical problems, some parts of the system can be efficiently described with constraints (e.g. the parameter setting of an optimal set-up can be naturally mapped onto a Constraint Optimisation Problem). In this case the optimum criteria involves certain parameters of the set-up. *Fig. 3* shows a simple set-up for system identification. Several parameters are associated to all of the blocks

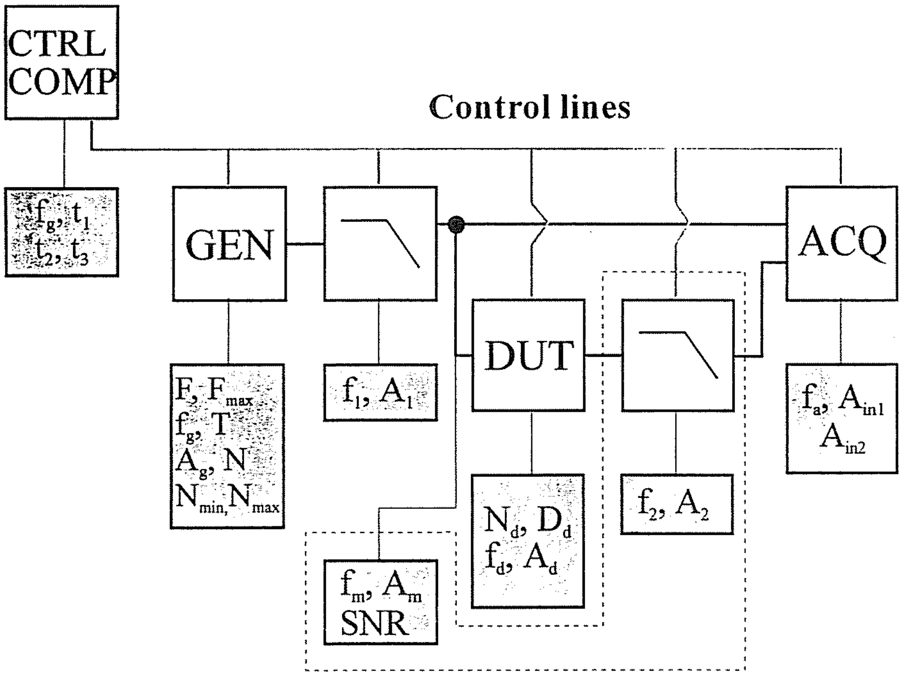

Fig. 3. A simple set-up for system identification

in the set-up. The constraints of an optimal set-up are imposed on these variables and the cost function is composed of a subset of these variables.

Conclusion

Most of the problems have several solutions and a significant part of these problems is sensitive to the quality of the solution. In such cases it is very important to find the optimal or a sub-optimal solution based on the optimum criteria associated with the task. In addition to the optimum criteria preferences can be associated to the variables present in the constraints, which differ in nature from the optimum criteria, but express important aspects of the task. If the constraint network is augmented with optimum criteria and with the preferential treatment of the variables, a wide class of new problems becomes solvable and can be handled with the help of constraints. Though the optimisation might increase the required computations, the gain in performance balances this extra resource requirement.

References

1. JAFFAR. – LASSEZ, J.: Constraint Logic Programming. *Proc. of the 14th ACM Symposium on the Principles of Programming Languages*, 1987, pp. 111-119.
2. VAN HENTENRYCK, P.: Constraint Satisfaction in Logic Programming. MIT Press, Cambridge, MA, 1989.
3. SISKIND, J. M., – MCALLESTER, D. A.: Nondeterministic Lisp as a Substrate for Constraint Logic Programming, *Proc. of the 11th National Conference on Artificial Intelligence*, AAAI-93, 1993.
4. DECHTER, A., – DECHTER, R.: Removing Redundancies in Constraint Networks, *Proc. AAAI-87*, pp. 105-109.
5. DECHTER, R., – PEARL, J.: Directed Constraint Networks: A Relational Framework for Causal Modelling, *Proc. IJCAI-91*, pp. 1164-1170.
6. DECHTER, R., – MEIRI, I.: Experimental Evaluation of Prepro-Cessing Techniques in Constraint Satisfaction Problems, *Proc. IJCAI-89*, pp. 271-277.
7. DECHTER, R., – PEARL, J.: The Anatomy of Easy Problems: A Constraint Satisfaction Formulation, *Proc. IJCAI-85*, pp. 1066-1072.
8. DECHTER, R., – PEARL, J.: Tree-Clustering Schemes for Constraint Processing, *Proc. AAAI-88*, pp. 150-154.
9. FREUDER, E. C.: A Sufficient Condition of Backtrack-Free Search, *J. ACM 29* Vol. (1), 1982, pp. 24-32.
10. FREUDER, E. C.: Partial Constraint Satisfaction, *Proc. of the 11th IJCAI*, 1989, pp. 278-283.
11. DECHTER, R., – DECHTER, A., – PEARL, J.: Influence Diagrams, Belief Nets and Decision Analisis, John Wiley & Sons Ltd. 1990, Chapter 18.
12. HYVNEN, E.: Constraint Reasoning with Incomplete Knowledge. The Tolerance Propagation Approach, Technical Research Center of Finland, Publications 72, VTT, ESPOO 1991.
13. MACKWORTH, A. K.: Consistency in Networks of Relations. Artificial Intelligence 8, 1977 pp. 99-118.
14. MACKWORTH, A. K., – FREUDER, E. C.: The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems, *Artificial Intelligence*, Vol. 25, 1985, pp. 65-74.
15. MOHR, R., – HENDERSON, C.: Arc and Path Consistency Revisited, *Artificial Intelligence*, Vol. 28, 1986, pp. 225-233.
16. MONTANARI, U.: Networks of Constraints: Fundamental Properties and Application to Picture Processing, *Inf. Sci.* Vol. 7, 1974, pp. 95-132.
17. RICH, E., – KNIGHT, K.: Artificial Intelligence, McGraw-Hill, 1992.