Open, Interactive and Dynamic CSP[⋆]

Santiago Macho González and Pedro Meseguer

Institut d'Investigació en Intel.ligència Artificial
Consejo Superior de Investigaciones Científicas
Campus UAB, 08193 Bellaterra, Catalonia, Spain.
{smacho|pedro}@iiia.csic.es

Abstract. In previous work, the notions of Open, Interactive and Dynamic CSP have been independently defined. *Open* constraint satisfaction is a new model where values are incrementally gathered during problem solving. Domains are assumed unbounded. *Interactive* constraint satisfaction also deals with partially known domains, assuming implicitly that domains are finite. *Dynamic* constraint satisfaction deals with problems of dynamic nature (as configuration design or model composition) where variables, domains and constraints are subject to frequent changes. In this paper, we study the relationship between these three models, showing that Interactive CSP can be seen as a particular case of Open and Dynamic. We have applied two algorithms, FOCSP (developed for Open) and LC (developed for Dynamic) to solve Interactive CSP. We provide experimental results of this evaluation.

1 Introduction

In previous work, a new model called Open CSP [2], integrates information gathering and problem solving. This model starts solving a CSP from a state where all domains are possibly empty, and it dynamically asks for values while the CSP has not been solved. This process stops as soon as a solution is found. With a similar motivation, the Interactive CSP model [3] deals with problems with partially defined domains. It is implicitly assumed that the domains are finite, while in Open CSP domains remain unbounded. In a dynamic environment, tasks usually change. As a consequence, CSPs that represent these tasks evolve, and variables, domains and constraints may change over time. The Dynamic CSP model [1] was defined to solve CSP in such dynamic environments.

This paper considers the relation among Open, Interactive and Dynamic CSP approaches. We show that an Interactive CSP can be seen as a particular case of Open CSP, and also as a particular case of Dynamic CSP. Therefore, algorithms developed for Open or Dynamic models can be used for Interactive CSP.

The paper is organized as follows. First, we briefly describe Open, Interactive and Dynamic CSPs. Then, we show how Interactive CSPs can be seen as particular cases of Open and Dynamic CSPs, respectively. We provide an experimental evaluation of Interactive CSP using three algorithms, one from each class, on random problems, in terms of number of queries and search effort.

[⋆] Partially supported by the Spanish REPLI project TIC-2002-04470-C03-03.

2 Open, Interactive and Dynamic CSPs

Open CSP. Imagine you want to configure a PC using web data sources. Querying all the possible PC parts in all data sources on the web is just not feasible. We are interested in querying the minimum amount of information until finding a solution. Since the classical CSP approach (querying all values before search starts) is not applicable here, the new Open CSP approach [2] was proposed. Solving an Open CSP implies obtaining values for the variables, one by one. If the collected information does not permit to solve the problem, new values are requested. The process stops when a solution is found.

The *failure Open CSP (FOCSP)* algorithm [2] solves Open CSP, based on the idea that we gather new values only when the current known part of the problem has no solution. In that case, it contains a smaller subproblem that already has no solution, and the whole problem can be made solvable only by creating a solution to that subproblem. Then, an additional value should be found for one variable (the *failed* variable) of that subproblem, and search restarts. This algorithm is proven correct and complete, no matter unbounded domains.

Interactive CSP. Very related with Open CSP is the Interactive CSP model introduced by [3]. An Interactive CSP (ICSP) has partially known domains for its variables. When solving an Interactive CSP, new values are requested, until finding a solution or proving that no solution exists. The main difference with Open CSP is that Interactive CSP implicitly assumes that variable domains are finite.

In this model, it is possible to use heuristically some of the known constraints to guide the acquisition of new values. This feature depends on the concrete application to solve, but no specific condition is requested on the basic model. The *interactive forward checking (IFC)* algorithm is proposed. When a domain become empty, it launches a specific request for additional values that would satisfy the constraint on that variable. In this process, the algorithm may request all values of a variable, assuming finite domains.

Dynamic CSP. A Dynamic CSP [1] is a finite sequence $\langle$CSP(0), CSP(1),...$\rangle$ of CSP instances, where each CSP(i) differs from the previous one by the addition or removal of some constraints (all possible changes of a CSP can be expressed in terms of constraint additions or removals). Solving a Dynamic CSP implies solving each instance of the sequence. The first instance is solved from scratch, and it is always possible to apply this method to any subsequent one. However, this approach is inefficient and may cause instability between solutions of consecutive instances. For this reason, when solving dynamic CSP one wants to reuse as much as possible the solving episodes of previous instances.

There are several solving approaches for Dynamic CSP solving. We consider the *local changes (LC)* algorithm [4], which tries to repair a previous solution. When the assignment of a variable becomes inconsistent with a previous solution, the inconsistent part of that solution is modified, keeping the assignment of that variable, until consistency is restored. If this is not possible, other values for the considered variable are tried.

3 Relationship between models

In this section we study the relationship between Open, Interactive and Dynamic CSP models. The relation between these 3 models is shown in figure 1.

Interactive CSP as Dynamic CSP. An Interactive CSP can be seen as a particular case of Dynamic CSP, as follows. In Interactive CSP, the operation that passes from a problem instance to the next one is *acquire_value*, getting a new value for a particular variable. Then, the variable domain is extended with that value, and the relational part of constraints involving such variable are enlarged with the allowed tuples that contain the new value. This process can be modelled in Dynamic CSP as follows. Adding a new value is equivalent to removing a unary constraint which disallowed this value in the domain of the corresponding variable, so that value is now available. Enlarging the constraints in which the variable is involved is equivalent to replacing (removing plus adding) the previous constraints by the enlarged ones.

At first glance one may think that this approach requires to know all the values of the domains from the beginning, to form the variable domains of the Dynamic CSP. However, this is not the case. It is enough to know the maximum number of values for each variable, say d_i for v_i. Initially, the problem state is as follows. The domain of v_i is a set of d_i dummy values $\{dummy_1, \dots, dummy_{d_i}\}$. When value a is found, it replaces a dummy value, say $dummy_1$, in the variable domain (that now becomes $\{a, dummy_2, \dots, dummy_{d_i}\}$), and in the constraints.

Strictly speaking, this model is an extension of the standard model of Dynamic CSP, where all domains are known from the beginning. The existence of dummy values which are replaced by real values as search progresses is not a big issue for the standard Dynamic CSP model, because the domain size does not change, and dummy values are replaced by real ones only once. This is the only extension that the standard Dynamic CSP model requires to include Interactive CSP.

Interactive CSP as Open CSP. An Interactive CSP can be seen as a particular case of Open CSP. Similarly to the Open CSP model, Interactive CSP uses partially defined data, where domains are acquired incrementally from external agents. The main difference between these models is that Interactive CSP does not address the problems of an open environment, in particular it limits itself to finite domains whilst Open CSP works with unbounded domains. Thus Interactive CSP instances can be solved with Open CSP algorithms.

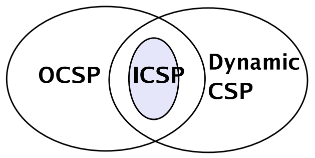

Fig. 1. *Relation between Open CSP, Interactive CSP and Dynamic CSP*

4 Experimental Results

We applied the LC algorithm [4] defined for solving Dynamic CSPs to solve Interactive CSP instances. Also we compare the FOCSP algorithm [2] defined for solving Open CSP problems which solves Interactive CSP problems. Both LC and FOCSP algorithms are compared with the IFC algorithm[3] developed for solving Interactive CSP problems.

To compare the algorithms, we are interested in the number of checks needed to solve the Interactive CSP and the number of accesses to information sources until a solution is found. We generated 1000000 random Interactive CSPs, with between 5 to 18 variables and 4 to 12 values per variable, with random constraints, forcing the graph to be at least connected and at most complete.

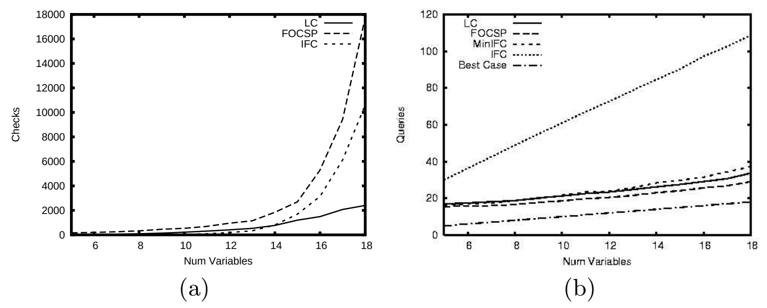

Fig. 2. *(a) Comparison of the number of checks against the number of variables. (b) Comparison of the number of queries against the number of variables*

Figure 2(a) shows the number of checks against the number of variables, studying the performance of the algorithms when we increase the number of variables. It is shown that the LC algorithm has a much better performance that the FOCSP and the IFC algorithms. The FOCSP algorithm redoes again the same solving process every time a new value is added, the IFC is just a backtracking with forward checking, while the LC algorithm uses the information from the previous assignments (compatible assignments, incompatible assignments) for solving without solving again the CSP from scratch as the FOCSP algorithm does. The high number of checks of the IFC algorithm is due in part to the acquire_value phase, where the algorithm checks for a compatible value.

Figure 2(b) shows the number of queries needed to find a solution against the number of variables. We assume that a query retrieve just a single value. We can see that LC and FOCSP algorithms have a much better performance than the IFC algorithm which needs to collect all values in a mediator for doing the forward checking. Also we included the Minimal Interactive Forward Checking [3] algorithm (MinIFC) which queries compatible values without collect the complete variable domain. We notice that LC, MinIFC and FOCSP algorithms query nearly the same number of values to find a solution. We compare all algorithms with the case where we found a solution just querying a value for every variable. We included this situation in the figure 2(b) with the line *Best Case*.

Figures 2(a) and 2(b) show the improvements of the LC algorithm applied to solve Interactive CSP problems. Empirically it is shown in figure 2(a) that reuse previous work on failed branches is better on average than detect the failed variable and redo the problem as the FOCSP algorithm does. Also the LC is better than backtracking with forward checking as the IFC algorithm does. It is interesting to analyze that the number of queries of LC algorithm is always slightly higher than the number of queries of FOCSP. This may be related with the way values are queried by both algorithms. While consecutive queries of FOCSP ask for values of different variables, consecutive queries of LC may ask the complete domain of a variable. Therefore, in some cases LC may ask more than needed to find a solution. This point is subject to current research.

5 Conclusions

In this paper we have analyzed the relation among Open, Interactive and Dynamic CSP. These models, different from the classical CSP, have appeared in different moments motivated by different applications. We have shown that Interactive CSP can be seen as a particular class of Open CSP (restricted to finite domains). In addition, we have also shown that Interactive CSP can be seen as a particular class of Dynamic CSP. As consequence, algorithms used to solve Open CSP and Dynamic CSP can be used to solve Interactive CSP. Based on this relationship, we have applied the FOCSP algorithm (initially developed for Open CSP) and the LC algorithm (initially developed for Dynamic CSP) comparing them with the IFC algorithm (developed for Interactive CSP) when they solve Interactive CSP instances. We have found that the LC algorithm reduces dramatically the number of checks with respect to FOCSP and IFC, just slightly increasing the number of queries needed to find a solution with respect to the FOCSP algorithm.

We think that this relationship between Open, Interactive and Dynamic CSP is a promising avenue for research, that we will further investigate.

References

1. Rina Dechter and Avi Dechter. Belief maintenance in dynamic constraint networks. In *Proceedings of the Seventh Annual Conference of the American Association of Artificial Intelligence*, pages 37–42, 1988.
2. Boi Faltings and Santiago Macho-Gonzalez. Open constraint programming. *Artificial Intelligence*, 161:181–208, 2005.
3. Evelina Lamma, Paola Mello, Michela Milano, Rita Cucchiara, Marco Gavanelli, and Massimo Piccardi. Constraint propagation and value acquisition: Why we should do it interactively. In *IJCAI*, pages 468–477, 1999.
4. Gérard Verfaillie and Thomas Schiex. Solution reuse in dynamic constraint satisfaction problems. In *Proceedings of the Twelfth Conference of the American Association of Artificial Intelligence*, pages 307–312, 1994.