An Incremental Consistency Algorithm for Adaptive Constraint Satisfaction

Evelina Lamma[1], Paola Mello[2], Michela Milano[1]

[1] DEIS Università di Bologna
Viale Risorgimento 2, 40136 Bologna, Italy
Fax +39 51 6443073, {elamma, mmilano}@deis.unibo.it
[2] Dipartimento di Ingegneria
Università di Ferrara
Via Saragat 1, 41100 Ferrara, Italy
pmello@ing.unife.it

Abstract. Constraint Satisfaction Problems (CSPs) play a central role in many Artificial Intelligence fields. Many different approaches have been proposed for solving CSPs ranging from pure backtracking to sophisticated methodologies. However, many efforts have concentrated on the use of a single algorithm for solving all problems. Recent results have shown that a single algorithm is not always the best choice for all classes of problems. Thus, *adaptive consistency* suggests to dynamically switch the consistency algorithm during the computation. In this paper, we propose an incremental and modular algorithm for achieving any degree of consistency by combining different, weaker consistency algorithms. In this way, for reaching higher levels of consistency we do not have to switch the algorithm, but we have to add as many uniform levels as we need, depending on the algorithm performed at each level.

1 Introduction

Constraint Satisfaction algorithms have been widely studied in the field of Artificial Intelligence and Computer Science [5, 7, 9, 10, 11]. These algorithms are intended to solve a particular aspect of the backtracking thrashing behavior [6], like node, arc, path consistency and the general version of k-consistency [3].

Conventional CSP solvers like CHIP [2] use a single search technique regardless of the problem they have to solve. They commit to a fixed search technique which is "generally good" for a variety of problems. Since the choice of the most appropriate algorithm for a given problem is an outstanding problem [4, 14, 15] in the field of constraint satisfaction, the choice of a fixed algorithm is clearly error prone.

Adaptive constraint satisfaction [1, 12] is a technique which relaxes the commitment to a single algorithm, and allows for the dynamic modification of the consistency algorithm during the search process. The basic structure of a system performing adaptive constraint satisfaction consists in (i) a set of algorithms, (ii) an heuristics module which selects the best algorithm for a given problem and (iii) the constraint solver which applies the algorithm on the problem or a part of it. In this paper we concentrate on parts (i) and (iii): instead of a set of *ad hoc* algorithms for each consistency check which can be chosen, we propose a way of incrementally combining a fixed consistency algorithm, say arc-consistency, in order to obtain higher degrees of consistency. Adaptive constraint satisfaction can be useful for example in the field of temporal reasoning where different models of time (say points or intervals) require different consistency techniques [14, 15].

The paper is organized as follows. In Section 2, we recall some results on consistency algorithms and on meta-graphs. Section 3 presents the basic idea underlying the incremental consistency algorithm and some results. Section 4 describes the algorithm. Conclusion and future works follow.

2 Preliminaries

Constraint Satisfaction Problems (CSPs) play a central role in many fields of Artificial Intelligence and Computer Science in general. A CSP can be defined on a set of variables $X_1, \ldots, X_n$ ranging on finite domains $D_1, \ldots, D_n$, representing the possible values that variables can assume, and a set of constraints, representing the relations among variables and, therefore, limiting the values that variables can simultaneously assume. Each CSP can be represented as a graph where nodes are variables and arcs are binary constraints. Each node has an associated domain representing the domain of the variable. In the following we refer to nodes as x_i for $i = 1$ to n, to their domain as D_i and to arcs with the associated constraint. If an arc links variables (nodes) x_i and x_j, the associated constraint is c_{ij}.

A number of consistency techniques have been proposed like node, arc, path consistency [5, 7, 9, 10, 11], and the general technique of k-consistency proposed by Freuder [3]. The general definition of k-consistency is the following: given values of $k-1$ variables that satisfy all the constraints among these variables, then chose any k-th variable. A value for this variable exists that satisfies all the constraints among these k variables. We propose an algorithm for reaching k consistency for each k based on the concept of meta-graphs, see [13].

A meta-graph MG=(A,MC) can be defined starting from a corresponding graph G=(V,A). In the graph G, V is a set of variables (defined by the problem) represented by nodes and A is a set of binary constraints between variables corresponding to arcs between nodes. Moreover, variables (nodes) are subject to unary constraints that represent set of values allowed for the variables. In meta-graph MG, nodes correspond to arcs of the object level graph, i.e., to constraints of the object level. Meta-variables range on a set of n-tuples of values allowed by constraints of the underlying graph. Meta arcs MC are relations (constraints) between meta nodes and should be defined in accordance with the application to be solved (see section 3.1).

3 Consistency Techniques on (Meta) Graphs

With this background, we present the main idea of this paper. If we want to achieve a certain level of consistency on a graph, we have to apply to the graph the corresponding consistency algorithm. Therefore, adaptive constraint satisfaction approach proposes to have a number of different algorithms which can be selected by an heuristic module according to the problem to be solved. The purpose of this paper is to show that we need only one algorithm, e.g., arc-consistency, and we can apply it iteratively to the starting graph and to meta-graphs. By adding higher level graphs which propagate constraints of the underlying level, we can achieve k-consistency for any k.

The main idea is the following: in each meta-graph, nodes represent constraints (admitted values for the variables of the original graph), and arcs represent paths of the underlying graph. A lower degree of consistency algorithm performed on meta-graph corresponds to a stronger consistency degree on the object level graph. Intuitively, if meta-nodes represent arcs of the original graph, node consistency on the meta-graph corresponds to arc-consistency on the original graph. Again, if meta-arcs represent paths of the original graph, arc-consistency on the meta-graph corresponds to path-consistency on the original graph an so on.

The algorithm proposed is based on the following conceptual steps. Given a graph G=(V,A) representing a CSP problem

1. Apply an i-consistency on G;
2. Create a meta-graph MG starting from G;
3. Apply a k-consistency on MG.

The consistency achieved by steps 1, 2 and 3 on the initial graph is $i+k-1$. First, we have to show how to build the meta-graph MG (steps 1 and 2). Then, we prove that the consistency achieved is $i+k-1$. The algorithm can be reused starting with MG as initial graph and incrementally achieving the desired consistency. In section 4 we specialize this algorithm for composing arc-consistency algorithms.

3.1 How to build the Meta-graph

When performing step 1, the algorithm checks if given values for $i-1$ variables there exists at least one value for each i-th variable which is compatible with constraints. Traditional algorithms, say arc and path consistency, check the compatibility but do not store the results of the check. Our algorithm, while checking compatibility, stores the i-consistent tuples in a meta variable domain. Suppose that nodes (variables) $x_1 \dots x_i$ have an i-consistent assignment $v_1 \dots v_i$. This assignment can be stored in a meta-variable domain, say $X_{1..i}$ with domain D_X, without any additional overhead.

Step 2 concerns the addition of meta-constraints that link meta-variables created at step 1. The meta-graph is not fully connected. Suppose variables $x_1,\dots,$ x_i correspond to a meta-variable $MX_{1..i}$ and variables $y_1,\dots,$ y_i correspond to a meta-variable $MY_{1..i}$. The two meta-variables are linked by a meta-constraint if and only if they refer to a same subset of variables, i.e., if they share $i-1$ variables: $x_l = y_p, l, p = 1..i, l \neq m, p \neq n$. The meta arc is labeled with the constraint that holds in the object graph between the two unshared variables. This meta constraint holds if the shared variables have the same values and the constraint between the two unshared variables holds.

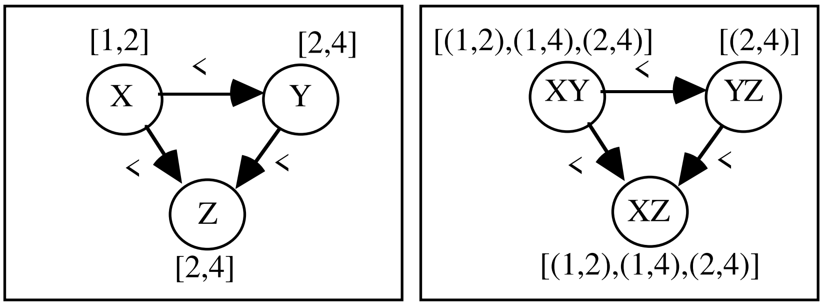

Fig. 1. Example of graph and meta-graph constraints

Figure 1 depicts an example of a graph and the corresponding meta-graph. The meta-graph contains three meta-variables, namely XY corresponding to X and Y, YZ corresponding to Y and Z and XZ corresponding to X and Z. Meta-variable domains contain arc-consistent couples for the underlying graph. Meta-constraints are built as follows: XY and YZ are connected by a meta constraint because they share one $(i-1)$ object level variable Y. The meta-arc is labeled by the constraint $<$ because the two unshared variables X and Z are linked by the constraint $<$ in the underlying graph. This meta-constraint holds if there exist a value (couple (x, y)) in the domain of XY and a couple (y, z) in the domain of YZ such that the value for Y is the same and values for X and Z are linked by the constraint $x < z$.

3.2 Results

Once we have performed an i-consistency $(i > 1)$ on the graph G, built the meta-graph MG, and applied a k-consistency on MG, we have to prove that the overall consistency achieved is $i + k - 1$.

Theorem 1. *Given a module (possibly composed by m levels) resulting in an i-consistent graph, and a meta-graph whose node domains contain consistent object level i-tuples and arcs link meta-variables according to object level constraints, if the meta-graph is k-consistent, the overall consistency achieved by the system is a j-consistency $(j \leq n)$ where $j = i + k - 1$*

Proof: The proof can be given by induction. The first step concerns two levels performing arc consistency: the overall consistency reached is a path-consistency

$(2+2-1=3)$. We then show that a path-consistency algorithm on the original graph is equivalent to an arc-consistency algorithm on the meta-graph.

By definition a 2-length path through the nodes (X, Y, Z) is path consistent iff for any pair of values $x \in D_X$ and $y \in D_Y$ such that $c(x, y)$ is consistent, a node-consistent value $z \in D_Z$ exists such that the constraints $c(x, z)$ and $c(z, y)$ hold. Therefore, path-consistency can be splitted in two steps: the first concerns the selection of a couple of consistent values, while the second concerns the search for a third value consistent with the previous two values.

Suppose we have three variables X, Y and Z ranging on the domains D_X, D_Y and D_Z and an arc-consistency algorithm. In the meta graph we have three meta-variables XY, XZ and YZ ranging on the following domains:

$$D_{XY} = \{(x, y) | x \in D_X, y \in D_Y,\ c(x, y)\ holds\}$$

$$D_{XZ} = \{(x, z) | x \in D_X, z \in D_Z,\ c(x, z)\ holds\}$$

$$D_{YZ} = \{(y, z) | y \in D_Y, z \in D_Z,\ c(y, z)\ holds\}$$

A constraint linking the two meta-variables represents a constraints on the path linking the three variables X, Y and Z.

An arc-consistency on the meta-variables XY, YZ and XZ ensures that for each value (x, y) of XY (first step of path-consistency) there exists one value for (y, z) for YZ consistent with $c_m((x, y), (y, z))$ (second step of path-consistency). This constraint holds iff the component corresponding to the "shared" variable y is equal, while the other two components respect the relation c_m (corresponding to the relation between x and z in the object system).

In the next step we suppose that, given m levels, performing a consistency i, we build a meta-graph whose nodes range on consistent i-tuples. If we perform a 1-consistency on this meta-graph, the overall consistency reached is again i. In fact, the architecture of m levels, at this step, can be considered as a single level performing a i-consistency. If we add a further level performing a 1-consistency, the overall consistency will be:

$$i' = i + 1 - 1 = i.$$

The proof of this step is straightforward: in fact, a level performing a i consistency passes to the higher level consistent i-tuples. The higher level, performing a 1-consistency checks if the i-tuples, belonging to the domain of the meta-variables are node consistent, performing again a i-consistency. $\square$

Now we give a complexity result for arc- and path-consistency. This proof can be easily generalized for higher consistency degrees (this is a work in progress).

Theorem 2. *Given the graph G=(V,A) and a meta-graph MG=(A,MA), performing an arc consistency algorithm on the meta graph MG has the same complexity as that of performing a path consistency on the variables of the original graph.*

Proof. Given a complete graph G=(V,A) let us suppose that n is the number of variables, e is the number of binary constraints and a is the edge degree of each vertex[3]. The complexity of performing the path consistency algorithm PC-2, see [9, 10], is $O(n^3a^3)$.

Now we analyze the behavior of the arc consistency algorithm on the meta graph.

If n is the number of original variables, the meta-graph will contain $\frac{1}{2}n(n-1)$ nodes. The distance graph is not complete. In particular, for each distance node XY we have $2(n-2)$ arcs, corresponding to the 3-length paths containing the original variables X and Y. Therefore, in the distance graph we have the following number of binary constraints:

$$e = \frac{1}{2}(\frac{1}{2}n(n-1))(2(n-2))$$

The complexity of performing the arc consistency algorithm AC-4, see [10], is $O(ea^2)$, but e is proportional to n^3. Therefore, the overall complexity is the same. □

The peculiarity of the framework is the flexibility. By iterating the steps of the algorithm, we increment the degree of consistency desired. This corresponds to the extension of the frameworks to many levels (meta-meta-graphs and so on). If each level performs a certain degree of consistency, these degrees can be added in order to reach a stronger level of consistency.

In particular, the overall consistency is reached incrementally by multi-level graphs. At the n-th level, if the consistency reached since then is m, each level performing an i-consistency further added to the architecture increases the overall consistency of a factor $i-1$. Intuitively, one level performs consistency on the already consistent m-tuples of values of the underlying system (which performs a m-consistency). Therefore, if we have n levels each performing a c_i consistency the overall consistency reached is $k = (((c_1 + c_2 - 1) + c_3 - 1) + c_4 - 1 \ldots)) \ldots)$ which can be written as:

$$k = c_1 + (\sum_{i=2}^{n} c_i - 1).$$

In the next section we present a specialization of this idea where we repeatedly perform arc-consistency algorithm, thus increasing at each iteration the overall consistency of a value 1.

4 Implementation

We now present the algorithm for incrementally achieving any level of consistency by composing arc-consistency algorithms, in particular the AC-4 algorithm by Mohr and Henderson [10]. We use that algorithm since it is based on the notion of *support*. A value in a domain of a variable is supported if there exists a

[3] We suppose that each vertex has the same number of values in its domain.

```
procedure consistency(inputs : K D, C, output :S)
begin
AC-4(D,C,S); Count := K − 1;
if (Count ≠ 1) then
       begin
       domain-meta(MD, S);
       constraint-meta(C,MC,MD);
       Count := Count − 1;
       consistency(Count, MD,MC,S);
       end
end
```

Fig. 2. The incremental algorithm

```
procedure domain-meta(D, S)
begin
for each S⟨i,b⟩ in S do
       for each ⟨j,c⟩ in S⟨i,b⟩ do
             if i < j then /* remove symmetric couples */
                   D_ij := D_ij + {(b,c)}
end
```

Fig. 3. The creation of meta domains procedure

compatible value in every other variable domains. If the support becomes empty, the value can be deleted. In the appendix, we sketch the AC-4 algorithm with slight modifications. This algorithm can be considered as the basic module of our incremental algorithm.

At the end of the algorithm the support set S can be used in order to build the arc-consistent couples of values. Each support set can be represented as follows:

$S_{\langle j,c\rangle} = \{(i,b)|c_{i,j}(b,c)\ holds\}$.

Therefore, in the meta graph we have one variable X_{ij} for each constraint of

```
procedure constraint-meta(C_obj, C_Meta, D_Meta)
begin
for each C_i,j in C_obj do
       for each D_i,k and D_k,j in D do
             C_Meta_ij,jk := C_i,j
end
```

Fig. 4. The creation of meta constraints procedure

the underlying graph $c_{i,j}$. The meta variable domain D_{ij} contains arc-consistent couples (c, b). Constraints in the meta graph can be built in the following way: two meta variables X_{ik} and X_{kj} are linked by a meta constraint that states that the component corresponding to the common elements k is equal and elements i and k are linked by the object graph constraint $c_{i,j}$, as shown in section 3.1.

In the same way, the result of the AC-4 algorithm on the meta graph produces support sets:

$S_{\langle(i,j),(b,c)\rangle} = \{((j,k),(c,d)) | c_{ij,jk}((b,c),(c,d))\ holds\}$

which can be used in order to determine the variables for the meta meta graph and so on. Note that when arc-consistency terminates, the support sets are in some sense redundant since if a couple a, b for variables X_i and X_j is consistent, we find the couple both in the set $S_{\langle j,b\rangle}$ and in $S_{\langle i,a\rangle}$. In the algorithm, we consider only one symmetric couple.

Let us see the incremental consistency algorithm which uses AC-4 and produces any level of consistency K. By composing arc-consistency algorithm at each layer we increase the consistency level of a factor 1 by each added layer. Therefore, if we want to achieve a consistency K we have to apply the algorithm AC-4 $K-1$ times. The resulting algorithm is reported in figure 2.

Now we consider the two procedures that create meta domains (figure 3) and meta constraints (figure 4). Note that, for instance, if meta variable $X_{i,j}$ contains couple (a, b), meta variable $X_{j,i}$ contains couple (b, a). In the meta-graph, we have only one variable for representing the two.

The nature of the algorithm is incremental since it maintains an internal state and process inputs depending only on this state.

For proving this claim, we should consider the algorithm as a transition system on *states*. A state is represented by a couple $\langle k, G\rangle$ where k is the consistency achieved since then, and G is a graph where nodes contain consistent k-tuples (see section 3). The input is a Δk, i.e., an increment of consistency which has to be applied to G in order to obtain a meta-graph MG and a consistency $k+\Delta k-1$.

The initial state is $\langle 1, G\rangle$ if we suppose that the initial graph is node consistent. G is the original graph where variables range on 1-consistent values (1-tuples). The final state is $\langle n, MG_n\rangle$ where n is the level of consistency desired and MG_n is the meta-graph containing all the n-tuples consistent with constraints. If the original problem has n variables the final meta-graph MG_n is composed by a single node whose domain contains the solution of the original problem. An intermediate transition determined by the application of the input Δk to the state $\langle k, G\rangle$ is:

$$\langle k, G\rangle \overset{\Delta k}{\rightarrow} \langle k + \Delta k - 1, MG\rangle$$

5 Conclusions

In this paper, we presented a general algorithm which achieves the desired level of consistency not by designing a special purpose algorithm for it, but by adding several levels (meta-graphs) and performing lower degrees of consistency on them. Several advantages can be underlined:

- The algorithm is incremental since it maintains an internal state and uses this state for processing a new inputs;
- The algorithm is modular and flexible, since it can be implemented in whatever architecture by defining the base module and the algorithm at disposal;
- At the end of the computation, we have achieved a certain level of consistency, say k. The output of the algorithm is the final state which contains a meta-graph where nodes contain consistent k-tuples (partial or total solution). We do not have to perform any search in the search space in order to find these k-tuples.

We are currently testing the performances of the algorithm in order to effectively test the flexibility of the algorithm in the field of temporal reasoning.

References

1. J.E.Borrett, M.G.Wallace, E.B.Richards, "Adaptive Constraint Satisfaction: the Quickest First Principle", in Proceedings of ECAI96, pp.160–164, 1996.
2. M.Dincbas, P.Van Hentenryck, M.Simonis, A.Aggoun, T.Graf, F.Berthier. The Constraint Logic Programming Language CHIP. in *Proceedings of the International Conference on Fifth Generation Computer System (FGCS88)*, pp.693–702,1988.
3. E.C. Freuder, "Synthesizing Constraint Expressions", in *Journal of the ACM*, vol. 21, No11, 1978, pp. 958–966.
4. E.C. Freuder, "A Sufficient Condition for Backtrack Free Search", in *Communication of the ACM*, vol. 29, No1, 1982, pp. 24–32.
5. E.C. Freuder, A.K. Mackworth, "The Complexity of some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems", in *Artificial Intelligence*, vol.25, 1885, pp.65–74.
6. J.Gaschnig, "Performance Measurement and Analysis of Certain Search Algorithms", PhD dissertation 1979, Carnegie Mellon University.
7. V. Kumar, "Algorithms for Constraint-Satisfaction Problems: A Survey", in *AI Magazine*, vol. 13, 1992, pp. 32–44.
8. E.Lamma, P.Mello, M.Milano, "A Multi-Level ACP Architecture for Consistency Tecniques", in *Proceedings of International Workshop on Constraint-Based Reasoning*, in conjunction with FLAIRS'96, pp. 75–83, 1996.
9. A.K. Mackworth, "Consistency in Networks of Relations", in *Artificial Intelligence*, vol.8, 1977, pp. 99–118.
10. R.Mohr, T.C.Henderson, "Arc and Path Consistency Revisited", in *Artificial Intelligence*, vol.28, 1986, pp. 225–233.
11. A.Montanari, "Networks of Constraints: Fundamental Properties and Applications to Picture Processing", in *Information Sciences*, vol.7, 1974, pp. 95–132.
12. H.El Sakkout, M.G. Wallace, E.B.Richards, "An Instance of Adaptive Constraint Propagation", Proceedings of 2nd International COnference of Principles and Practice of Constraint Programming", pp.164–178, 1996.
13. E.P.K.Tsang, "The Consistent Labeling Problem in Temporal Reasoning", in *Proceedings of AAAI87*, 1987, pp.251–255.

14. M.B.Vilain, H.Kautz, P. Van Beek, "Constraint Propagation Algorithms for Temporal Reasoning: A Revised Report", in *Readings in Qualitative Reasoning about Physical Systems*, D.S.Weld, J.De Kleer, Morgan Kaufmann, 1990, pp.373–381.
15. P.Van Beek, "Approximation Algorithms for Temporal Reasoning", in *Proceedings of IJCAI89*, 1989.

Appendix

We here sketch the algorithm AC-4 as presented in [10]. The parameters of the procedure are a set of domain variables D, a set of constraints C, and a support set S.

```
procedure AC-4(inputs : Z, C, output :S)
begin
/* Step 1: Construction of the data structures */
M :=0; S:=∅;
for each C_ij in C do
    for each b in D_i do
        begin
        Total:=0;
        for each c in D_j do
            if satisfies((b,c), C_ij) then
                begin
                Total := Total +1;
                S_⟨j,c⟩ := S_⟨j,c⟩ + ⟨i,b⟩; end;
            if (Total = 0) then /* reject ⟨i,b⟩ */
                begin
                M[i,b]:=1; D_i := D_i - {b};end;
            else Counter[(i,j),b] := Total; end
List := {⟨i,b⟩|M[i,b] = 1}; /*set of rejected labels*/
/* Step 2: remove unsupported labels */
while List ≠ ∅ do
    begin
    choose ⟨j,c⟩ from List;
    List := List - {⟨j,c⟩};
    for each ⟨i,b⟩ in S_⟨j,c⟩ do
        begin
        Counter[(i,j),b] := Counter[(i,j),b] - 1;
        if (Counter[(i,j),b]=0) and (M[i,b] = 0) then
            begin
            List := List + ⟨i,b⟩;
            M[i,b]:=1; D_i := D_i - {b};
            end; end
    S := S - S_⟨j,c⟩
    end return (D,C,S); end
```