A New Algorithm for Maintaining Arc Consistency after Constraint Retraction

Pavel Surynek, Roman Barták*

Charles University in Prague, Faculty of Mathematics and Physics
Institute for Theoretical Computer Science
Malostranské nám. 2/25, 118 00 Praha 1, Czech Republic
pavel.surynek@seznam.cz, roman.bartak@mff.cuni.cz

Abstract. Dynamic Constraint Satisfaction Problems play a very important role in modeling and solving real-life problems where the set of constraints is changing. The paper addresses a problem of maintaining arc consistency after removing a constraint from the constraint model. A new dynamic arc consistency algorithm is proposed that improves the practical time efficiency of the existing AC|DC algorithm by using additional data-structures. The algorithm achieves real time efficiency close to the so far fastest DynAC-6 algorithm while keeping the memory consumption low.

1 Introduction

For solving many real-life problems, the traditional static formulation of the constraint satisfaction problem (CSP) is not sufficient because the problem formulation is changing dynamically [5]. To model such problems Dechter and Dechter [4] proposed a notion of *Dynamic Constraint Satisfaction Problem* (DCSP) that is a sequence of static CSPs, where each CSP is a result of addition or retraction of a constraint in the preceding problem. Several techniques have been proposed to solve Dynamic CSP, including searching for robust solutions that are valid after small problem changes, reusing the original solution to produce a new solution, and reusing the reasoning process. A typical representative of the last technique is *maintaining dynamic arc consistency*. The goal of maintaining dynamic arc consistency is keeping the problem arc consistent after constraint addition and constraint retraction. Adding a new constraint is a monotonic process which means that domains can only be pruned and existing arc consistency algorithms can be applied there. When a constraint is retracted from the problem then the problem remains arc consistent. However, some solutions of the new problem might be lost because the values from the original problem that were directly or indirectly inconsistent with the retracted constraint are missing in the domains. Consequently, such values should be returned to the domains after constraint retraction. Then we are speaking about *maximal arc consistency*.

* Supported by the Czech Science Foundation under the contract No. 201/04/1102.

In this paper we address the problem of maintaining maximal arc consistency after constraint retraction. Dynamic versions of popular arc consistency algorithms have been proposed to solve this problem, namely DnAC-4 [2] and DnAC-6 [3] based on extended data structures of AC-4 and AC-6, and AC|DC algorithm [1] as a reverse version of the AC-3 algorithm. DnAC-4 and DnAC-6 are very fast algorithms thanks to minimizing the number of constraint checks but they are also memory consuming and complicated for implementation. On the other hand, AC|DC is easy to implement, it has minimal memory requirements but it is less time efficient. The main reason for its time inefficiency is restoring too many values in the domains that are immediately pruned in the completion stage of the algorithm. In our work, we focused on removing this drawback by keeping additional data structures. Our hope was to improve significantly the practical time efficiency of AC|DC without increasing much the space complexity. This makes our approach different from AC-3.1|DC [6] that only substitutes AC-3.1 for AC-3 in the AC|DC scheme which increases space complexity.

2 Algorithm AC|DC-2

We extend the original AC|DC algorithm with additional data structures that record a justification and a timestamp for every value eliminated from the variable domain. By the *justification* we mean the first neighboring variable in which the eliminated value lost all supports. The *timestamp* describes the time when the value has been removed – a global counter which is incremented after every manipulation of the variable domains is used to model time. Using the timestamps is the original contribution of this paper. To prepare the above data structures (stored in the compound structure `data`) we propose a slightly modified AC-3 based algorithm that we call AC-3' (Figure 1).

```
function propagate-ac3'(P, data, revise)
1    queue := revise
2    while queue not empty do
3    |  select and remove a constraint c from queue
4    |  {u,v} := the variables constrained by c
5    |  (P,data,revise_u) := filter-arc-ac3'(P, data, c, u, v)
6    |  (P,data,revise_v) := filter-arc-ac3'(P, data, c, v, u)
7    |_ queue := queue ∪ revise_u ∪ revise_v
8    return (P,data)

function filter-arc-ac3'(P, data, c, u, v)
1    modified := false
2    for each d in P.D[u] do
3    |  if d has no support in P.D[v] w.r.t. c then
4    |    P.D[v] := P.D[v] - {d}
5    |    data.justif[u,d].var := v
6    |    data.justif[u,d].time := data.time
7    |    data.time := data.time + 1
8    |_   modified := true
9    if not modified then
10     return (P,data,∅)
11   return (P,data,{e in P.C|u is constrained by e and e≠c})
```

Fig. 1. A pseudo code of the modified arc-consistency algorithm AC-3'

When a constraint is retracted from the problem the algorithm uses the justifications and timestamps to determine the set of values which have been removed possibly because of the retracted constraint and that should be restored in the relaxed problem. As in the case of AC|DC algorithm the constraint retraction using AC|DC-2 is carried out in three phases. In the first phase (**initialize-ac|dc2**) the algorithm puts back the values into the variable domains that have been removed directly because of the retracted constraint. The second phase (**propagate-ac|dc2**) consists of a propagation of the initial restorations from the first phase. Finally, in the last phase (**propagate-ac3'**) the algorithm removes the inconsistent values that have been incorrectly restored in the previous phases. Notice that in addition to the description of the current domains (P.D) we keep the original domains of the variables ($P.D_0$). For a detail description of the algorithms we kindly ask the reader to see [7].

```
function retract-constraint-ac|dc2(P, data, c)
1    (P,data,restored_u) :=
2        initialize-ac|dc2(P, data, c, u, v)
3    (P,data,restored_v) :=
4        initialize-ac|dc2(P, data, c, v, u)
5    P.C := P.C - {c}
6    (P,data,revise) :=
7        propagate-ac|dc2(P, data,{restored_u,restored_v})
8    (P,data) := propagate-ac3'(P, data, revise)
9    return (P,data)

function initialize-ac|dc2(P, data, c, u, v)
1    restored_u := ∅
2    time_u := ∞
3    for each d in (P.D0[u] - P.D[u]) do
4      if data.justif[u,d].var = v then
5        P.D[u] := P.D[u] ∪ {d}
6        data.justif[u,d].var := NIL
7        restored_u := restored_u ∪ {d}
8        time_u := min(time_u, data.justif[u,d].time)
9    return (P,data,(u,time_u,restored_u))

function propagate-ac|dc2(P, data, restore)
1    revise := ∅
2    while restore not empty do
3      select and remove (u,time_u,restored_u) from restore
4      for each c in P.C|u is constrained by c do
5        {u,v} := the variables constrained by c
6        restored_v := ∅
7        time_v := ∞
8        for each d in (P.D0[v] - P.D[v]) do
9          if data.justif[v, d].var = u then
10           if data.justif[v, d].time > time_u then
11             if d has a support in restored_u w.r.t. c then
12               P.D[v] := P.D[v] ∪ {d}
13               data.justif[v,d].var := NIL
14               restored_v := restored_v ∪ {d}
15               time_v := min(time_v, data.justif[v,d].time)
16       restore := restore ∪ {(v,time_v,restored_v)}
17     revise := revise ∪ {e in P.C|u is constrained by e})
18   return (P,data,revise)
```

Fig. 2. A pseudo code of the dynamic arc-consistency algorithm AC|DC-2

The algorithm AC|DC-2 performs a correct retraction of a constraint with respect to maximal arc consistency (constraint addition can be easily done via **`propagate-ac3'`**). Moreover, it performs at most as many consistency checks as the existing algorithm AC|DC. Formally, the space complexity of AC|DC-2 is $O(nd+e)$ and the worst case time complexity of AC|DC-2 is $O(ed^3)$, where n is the number of variables, d is the size of the domains of variables, and e is the number of constraints. The formal proofs can be found in [7]. The theoretical complexities are thus identical to the original AC|DC algorithm, but in practice AC|DC-2 is much faster and performs much less consistency checks than AC|DC as the experiments showed.

5 Experimental Results

We compared the new algorithm AC|DC-2 to the existing algorithms AC|DC and DnAC-6 under Red Hat Linux 9.0 on 2 GHz Pentium 4 with 512 MB of memory. We performed the experiments on a set of randomly generated binary constraint satisfaction problems. For each problem, ten random instances were generated and the mean values of runtime are presented here. Constraints were added incrementally to the problem until a given density was reached or an inconsistent state was encountered. After this step, 10% of randomly selected constraints were retracted from the problem. Figure 3 shows the total runtime; AC|DC-2 is much faster than AC|DC and almost as fast as DnAC-6. A similar result was obtained for the number of constraint checks [7].

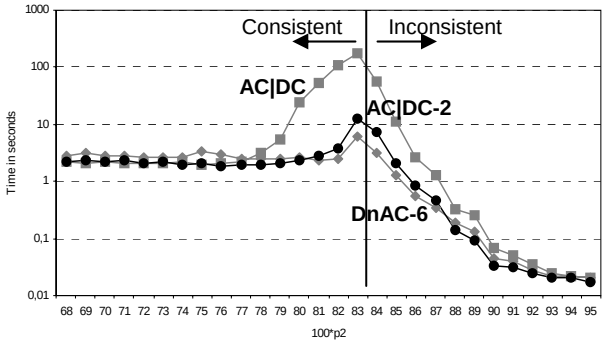

Fig. 3. Runtime (logarithmic scale in seconds) as a function of tightness for random constraint satisfaction problems $\langle 100, 50, 0.3, p_2 \rangle$.

We also compared the memory consumption of the algorithms. We measured the memory consumption of the data structures specific for the algorithms so the representation of the constraints is not included. The memory consumption was measured at the time point just before the constraint causing inconsistency was added because the memory consumption is largest there. The results (Table 1) show that the memory consumption of AC|DC-2 and AC|DC is neglecting and it is much smaller than the memory consumption of DnAC-6.

Table 1. Memory consumption depending on the size of variable domains for random constraint satisfaction problems $\langle 100, d, 0.3, p_2 \rangle$.

Domain size (*d*)	**20**	**30**	**40**	**50**	**60**	**70**	**80**
100*p_2	**68%**	**77%**	**80%**	**82%**	**84%**	**85%**	**85%**
AC|DC	<1MB	<1MB	<1MB	<1MB	<1MB	<1MB	<1MB
DnAC-6	21MB	32MB	49MB	64MB	77MB	94MB	120MB
AC|DC-2	<1MB	<1MB	<1MB	<1MB	<1MB	<1MB	<1MB

5 Conclusions

In the paper, we proposed a new algorithm AC|DC-2 for maintaining maximal arc consistency in dynamic environments where the constraints are added and retracted incrementally. As the experiments showed we improved significantly the practical time efficiency of the AC|DC algorithm and the time is now comparable to the so far fastest DnAC-6 algorithm. Moreover, the additional data structures did not increase much the space complexity of AC|DC which remains much smaller than for DnAC-6. Additionally, AC|DC is easier to implement than DnAC-6. We did not perform yet the experiments with AC-3.1|DC but we expect behavior similar to DnAC-6.

References

1. Berlandier, P. and Neveu, B.: Arc-Consistency for Dynamic Constraint Satisfaction Problems: a RMS free approach. In Proceedings of the ECAI-94 Workshop on "Constraint Satisfaction Issues Raised by Practical Applications", Amsterdam, The Netherlands, 1994.
2. Bessière Ch.: Arc-Consistency in Dynamic Constraint Satisfaction Problems. In Proc. of the 9th National Conference on Artificial Intelligence (AAAI-91), Anaheim, CA, USA. AAAI Press, 1991, 221–226.
3. Debruyne R.: Arc-Consistency in Dynamic CSPs is no more prohibitive. In Proc. of the 8th IEEE International Conference on Tools with Artificial Intelligence (ICTAI-96), Toulouse, France, 1996, 239–267.
4. Dechter, R. and Dechter, A.: Belief Maintenance in Dynamic Constraint Networks. In Proc. of the 7th National Conference on Artificial Intelligence (AAAI-88), St. Paul, MN, USA. AAAI Press, 1988, 37–42.
5. Kocjan, W.: Dynamic scheduling: State of the art report, SICS Technical Report T2002:28, ISSN 100-3154, 2002.
6. Mouhoub, M.: Arc Consistency for Dynamic CSPs. In Vasile Palade, Robert J. Howlett, Lakhmi C. Jain (Eds.): Proceedings of the 7th International Conference on Knowledge-Based Intelligent Information and Engineering Systems – Part I (KES 2003), Oxford, UK. Springer Verlag LNCS 2773, 2003, 393–400.
7. Surynek, P. and Barták, R. A New Algorithm for Maintaining Arc Consistency after Constraint Retraction. ITI Series 2004-205, Institute for Theoretical Computer Science, 2004.