An Approximation of Generalized Arc-Consistency for TCSPs

Lin XU and Berthe Y. CHOUEIRY
Constraint Systems Laboratory
Department of Computer Science and Engineering
University of Nebraska-Lincoln University of Nebraska-Lincoln, Lincoln NE, 68588-0115
lxu | choueiry@cse.unl.edu

Abstract

Dechter [2003] proposed to solve the Temporal Constraint Satisfaction Problem (TCSP) by modeling it as a meta-CSP, which is a CSP with a unique global constraint. The size of this global constraint is exponential in the size of the TCSP, and generalized-arc consistency is equivalent to solving the TCSP problem, which is NP-hard. We reformulate the meta-CSP by replacing its unique global constraint with a polynomial number of polynomial-size ternary constraints. We define $\triangle$AC as a generalized arc-consistency algorithm for these ternary constraints. Thus, $\triangle$AC approximates generalized arc-consistency on the TCSP. We use $\triangle$AC as a preprocessing step for solving the meta-CSP and show that it dramatically reduces the size of this meta-CSP and significantly enhances the performance of search for solving the TCSP.

1 Introduction

A Simple Temporal Problem (STP) is defined by a graph $G = (V, E, I)$ (see Figure 1, left) where

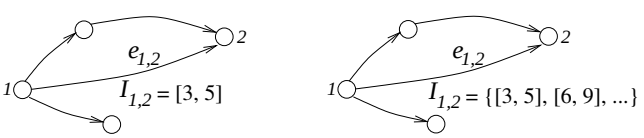

Figure 1: *Left:* STP. *Right:* TCSP.

- V is a set of vertices i representing time points;
- E is a set of edges $e_{i,j}$ representing constraints between two time points i and j; and
- I is a set of constraint labels for the edges. A constraint label $I_{i,j}$ of edge $e_{i,j}$ is a *unique* interval $[a, b]$, $a, b \in \mathbb{R}$, and denotes a constraint of bounded difference $a \leq (j - i) \leq b$.

A Temporal Constraint Satisfaction Problem (TCSP) is defined by a similar graph $G = (V, E, I)$, where each edge label $I_{i,j} = \{l_{ij}^{(1)}, l_{ij}^{(2)}, \ldots, l_{ij}^{(k)}\}$ is a *set* of disjoint intervals denoting a disjunction of constraints of bounded differences between i and j (see Figure 1 right). We assume that the intervals in a label are ordered in a canonical way.

Dechter [2003] described a backtrack search procedure for solving a TCSP, which is NP-hard. To this end, the TCSP is expressed as a 'meta' Constraint Satisfaction Problem (CSP), or meta-CSP:

- The variables of the meta-CSP are the edges $e_{i,j}$ of G. Their number $|E|$ depends on the density of the temporal graph and may reach $\frac{n(n-1)}{2}$, where n is the number of nodes in the TCSP.
- The domain of a variable $e_{i,j}$, denoted $\texttt{Domain}(e_{i,j})$, is its label, $I_{i,j} = \{l_{ij}^{(1)}, l_{ij}^{(2)}, \ldots, l_{ij}^{(k)}\}$.
- The only constraint in the meta-CSP is a global constraint that requires the variable-value pairs (vvps) $\{(e_{i,j}, l_{ij}^{(h)})\}$ for all the variables $e_{i,j} \in G$ to form a consistent STP. The size of this constraint (i.e., number of possible tuples) is $k^{|E|}$ and can reach $k^{\frac{n(n-1)}{2}}$. It is thus exponential in the size of the TCSP.

The backtrack search on the meta-CSP requires solving an STP at every node in the search. Its complexity is thus $O(n^3 k^{|E|})$ [Dechter, 2003]. Given the definition of the unique global constraint, running a generalized arc-consistency algorithm [Mohr, 1988] on the meta-CSP is prohibitively expensive.

Proposition 1.1. *Generalized arc-consistency on the meta-CSP is NP-hard.*

Proof: The only constraint in the meta-CSP is a global constraint. Its allowed tuples are all consistent STPs that are solutions to the meta-CSP. Finding its definition to enforce generalized arc-consistency is thus equivalent to solving the meta-CSP, which is NP-hard [Dechter, 2003]. □

We propose to approximate this problem by replacing the exponential-size global constraint in the meta-CSP with a polynomial number of polynomial-size ternary constraints. We define an efficient generalized arc-consistency algorithm specialized for these ternary constraints, which we call $\triangle$AC. The complexity of $\triangle$AC is $O(degree(G) \times |E| \times k^3) = O(n|E|k^3)$, resulting in an approximation of the generalized arc-consistency of the meta-CSP. We propose to investigate a slight improvement of $\triangle$AC, which may allow us to establish the optimality of this algorithm. In order to demonstrate the effectiveness of our approximation, we test and report the performance of $\triangle$AC as a preprocessing step to search, showing a dramatic reduction in the size of the meta-CSP, which

is a product of the domain sizes of its variables $\Pi_{e_{i,j} \in E} |I_{i,j}|$. We also report the performance improvement of the backtrack search for solving the meta-CSP with and without this preprocessing in terms of CPU time and number of constraint checks CC.

To the best of our knowledge, the only other work reported in the literature on applying consistency algorithms to the meta-CSP is a study by Schwalb and Dechter [1997; 2003]. They attempt to apply a path consistency algorithm (PC) to the labels of variables of the meta-CSP. Given the disjunctive intervals, this closure algorithm causes a fragmentation problem, which increases the number of intervals per label and makes the resulting meta-CSP even harder to solve by a search algorithm. To avoid this fragmentation problem, Schwalb and Dechter introduced the Upper-Lower Tightening algorithm (ULT) [1997]. ULT computes looser networks than those resulting from enforcing full path-consistency, but results in the same upper and lower bounds as PC.

Our approach is neither like path-consistency nor like ULT. We consider each interval as an independent value in the domain of a variable. Our goal is to remove inconsistent individual intervals from the labels, not to tighten these intervals, which may not terminate in the general case and is prohibitively expensive in the integral case.

This paper is structured as follows. Section 2 describes the reformulation of the meta-CSP and the $\triangle$AC algorithm. Section 3 describes our experiments and observations. Finally, Section 4 concludes this paper.

2 The label filtering algorithm

We reformulate the meta-CSP by replacing its unique global constraint with a ternary constraint $\triangle[e_{i,j}, e_{i,k}, e_{j,k}]$ among every variable $e_{i,j}$, $e_{i,k}$, and $e_{j,k}$ of the meta-CSP that forms an *existing* triangle in the temporal network G. Note that we do not triangulate the temporal network, nor do we make it a complete graph. Below, we define the $\triangle$arc-consistency property as the generalized arc-consistency of this constraint and describe the $\triangle$AC algorithm to achieve it.

2.1 $\triangle$arc-consistency

An STP is solved by computing the transitive closure under composition and intersection of the intervals of its edges. The transitive closure of an STP results in a complete temporal graph:

- The composition $l_{ik} = l_{ij} \circ l_{jk}$ of the intervals $l_{ij} = [a, b]$ and $l_{jk} = [c, d]$ labeling the respective edges $e_{i,j}$ and $e_{j,k}$ is a new interval $l_{ik} = [a+c, b+d]$ labeling the edge $e_{i,k}$.
- The intersection $l_{i,k} = l'_{i,k} \cap l''_{i,k}$ of the intervals $l'_{ik} = [a, b]$ and $l''_{ik} = [c, d]$ labeling the respective $e'_{i,k}$ and $e''_{i,k}$ is a new interval l_{ik} = [maximum(a, c), minimum(b, d)] labeling the edge $e_{i,k}$.

To define the concept of $\triangle$AC of a meta-CSP, we use the above two operations.

For each triangle ijk in the original temporal network we define a ternary constraint in the meta-CSP $\triangle[e_{i,j}, e_{i,k}, e_{j,k}]$. Given three variable-value pairs $(e_{i,j}, l_{ij})$, $(e_{i,k}, l_{ik})$, and $(e_{j,k}, l_{jk})$ of the meta-CSP, with $i \neq j \neq k$, we say that the labeled triangle $\triangle[(e_{i,j}, l_{ij}), (e_{i,k}, l_{ik}), (e_{j,k}, l_{jk})]$ is a consistent triangle if and only if $(l_{ij} \circ l_{jk}) \cap l_{ik} \neq \emptyset$. Figure 2 shows a consistent triangle $\triangle[(e_{i,j}, [3, 5]), (e_{i,k}, [4, 9]), (e_{j,k}, [2, 6])]$. We also say that each variable-value pair in the

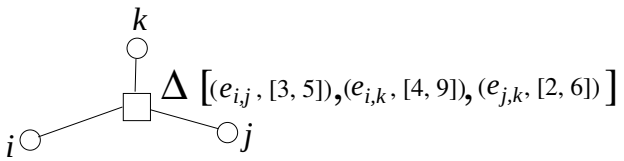

Figure 2: *A consistent triangle.*

triangle is supported by the two other variable-value pairs. We introduce the following definitions:

- The ternary constraint $\triangle[e_{i,j}, e_{i,k}, e_{j,k}]$ is $\triangle$AC relative to the meta-CSP variable $e_{i,j}$ if and only if for every interval $l_{ij}^{(x)} \in$ Domain$(e_{i,j})$ there exist an interval $l_{ik}^{(y)} \in$ Domain$(e_{i,k})$ and an interval $l_{kj}^{(z)} \in$ Domain$(e_{k,j})$ such that $(l_{ik}^{(y)} \circ l_{kj}^{(z)}) \cap l_{ij}^{(x)} \neq \emptyset$.
- The ternary constraint $\triangle[e_{i,j}, e_{i,k}, e_{j,k}]$ is $\triangle$AC if and only if it is $\triangle$AC relative to the variables $e_{i,j}$, $e_{i,k}$, and $e_{j,k}$.
- Finally, the meta-CSP is $\triangle$AC if and only if all its ternary constraints are $\triangle$AC.

We identify all the existing triangles in the temporal network and replace each of them by a ternary triangle constraint. The number of these new constraints is in $O(|E| degree(G)) = O(|E|n)$. The size of each constraint is at most k^3. Note that we do not add any edges to the temporal network to make it a complete graph or to triangulate it.

2.2 $\triangle$AC algorithm

The $\triangle$AC algorithm, shown in Figure 5, removes the intervals in the domain of an $e_{i,j}$ that do not have a support in any triangle in which $e_{i,j}$ appears in the temporal graph. It implements mechanisms for consistency checking that are reminiscent of AC-4 [Mohr and Henderson, 1986] and AC-2001 [Bessière and Régin, 2001] in that it tries to optimize the effort for consistency checking. It uses the procedures First-support of Figure 3 and Initialize-support of Figure 4. The Push and Delete operations we use are destructive stack operations.

It operates by looking at every combination of a vvp $(e_{i,j}, l_{ij})$ and the triangles ijk in which it appears, denoted $\langle (e_{i,j}, l_{ij}), ijk \rangle$. The support of $\langle (e_{i,j}, l_{ij}), ijk \rangle$ is the first element in the domains of $e_{i,k}$ and $e_{j,k}$ that yields a consistent triangle. (Note that domains and variables are ordered canonically.) Intervals in the domain of a variable that are not supported in any triangle are removed from the domain. When an interval is removed, some vvps may lose their support. $\triangle$AC tries to find the next acceptable support. The process is repeated until all vvps have a valid support in every relevant triangle.

We use a hash-table Supported-by to keep track of the support of each vvp $(e_{i,j}, l_{ij})$ in a triangle ijk. A key in this

```
First-support(⟨(e_{i,j}, l_{ij}), ijk⟩)
t_{ijk} ← △[(e_{i,j}, l_{ij}), (e_{i,k}, l^r_{ik}), (e_{j,k}, l^s_{jk})] ← Supported-by(⟨(e_{i,j}, l_{i,j}), ijk⟩)
Unless t_{ijk} Then r ← 1, s ← 0
For m from (s + 1) to |Domain(e_{j,k})|
    Unless (l^r_{ik} ∘ l^m_{jk}) ∩ l_{ij} = nil Return △[(e_{i,j}, l_{ij}), (e_{i,k}, l^r_{ik}), (e_{j,k}, l^m_{jk})]
If r=|Domain(e_{i,k})|
   Then Return nil
   Else For n from (r + 1) to |Domain(e_{i,k})|
            For t from 1 to |Domain(e_{j,k})|
                Unless (l^n_{ik} ∘ l^t_{jk}) ∩ l_{ij} = nil Return △[(e_{i,j}, l_{ij}), (e_{i,k}, l^n_{ik}), (e_{j,k}, l^t_{jk})]
Return nil
```

Figure 3: First-support.

```
Initialize-support(G)
Support-by, Supports: two empty hash-tables
Q ← {(e_{i,j}, l_{ij})}, set of all vvps in the meta-CSP
Q' ← nil, Consistency ← t
While Q ∧ Consistency do
   (e_{i,j}, l_{ij}) ← Pop(Q)
   Forall k such that ijk is a subgraph of G do
     t_{ijk} ← △[(e_{i,j}, l_{ij}), (e_{i,k}, l_{ik}), (e_{j,k}, l_{jk})] ← First-support(⟨(e_{i,j}, l_{ij}), ijk⟩)
     If t_{ijk}, Then Supported-by[(e_{i,j}, l_{ij}), ijk] ← t_{ijk}
                      Push(⟨e_{i,j}, l_{ij}, ijk⟩, Supports[(e_{i,k}, l_{ik})])
                      Push(⟨e_{i,j}, l_{ij}, ijk⟩, Supports[(e_{j,k}, l_{jk})])
                Else  Domain(e_{i,j}) ← Domain(e_{i,j}) \ {l_{ij}}
                      Push((e_{i,j}, l_{ij}), Q')
                      Unless Domain(e_{i,j}) Then Consistency ← f
Return Q', Supported-by, Supports, Consistency
```

Figure 4: Initialize-support.

hash-table is a tuple $\langle(e_{i,j},\ l_{ij}),\ ijk\rangle$; its value is a consistent triangle $\triangle[(e_{i,j}, l_{ij}), (e_{i,k}, l_{ik}), (e_{j,k}, l_{jk})]$. The size of Supported-by is $O(|E|k\ degree(G))$. We also use a hash-table Supports to keep track of what a given vvp supports in Supported-by. The key is a vvp $(e_{i,j}, l_{ij})$ and the value is a list of the keys of Supported-by that this vvp supports.

The procedure Initialize-support shows how these data-structures are initialized. By construction, Supports has $O(|E|k)$ keys and a total of $O(|E|k\ degree(G))$ elements.

In addition to these hash-tables, Initialize-support returns the list Q' of vvps deleted from the domains of the meta-CSP at the initialization step. $\triangle$AC, shown in Figure 5, iterates over the vvps that have been deleted and retracts them from supporting entries in Supported-by.

We can prove that $\triangle$AC terminates, does not remove any consistent intervals (i.e., is sound), and is in $O(degree(G)|E|k^3) = O(n|E|k^3)$. We can further improve its performance and reduce the number of constraint checks by exploiting the convexity property of interval intersection, which we suspect may result in an optimal algorithm.

3 Experimental results

We conducted empirical evaluations on randomly generated TCSPs. Below we describe our random generator, the characteristics of the experiments we conducted, and our observations based on the results.

3.1 Random generator

We designed a generator of random TCSP instances that guarantees that the temporal network is connected and that a specified percentage of the generated instances is solvable. Our generator is designed as follows. It takes as input:

- The number of nodes n in the temporal network, which is the number of time points in the TCSP.
- The density d of temporal graph G. This determines the number of edges $|E|$ in the temporal graph. Naturally, $|E| \leq \frac{n(n-1)}{2}$.
- The maximal number of intervals in the label of an edge k. The actual number of intervals per label is chosen randomly in a uniform manner between 1 and k.
- The range of the nodes selected from $R = [1, r]$, with $r \in \mathbb{N}$.
- The percentage p_c of solvable problems.

We generate a random TCSP example by following the steps below:

1. We select values in R to correspond to positions of time points in this interval. We enforce that the first node of the graph has position 1, and the last node in the graph

$\triangle$**AC**(G)
Q, `Supported-by`, `Supports`, Consistency $\leftarrow$ `Initialize-support`(G)
While $Q \wedge$ Consistency **do**
  $(e_{i,k}, l_{ik}) \leftarrow$ `Pop`(Q)
  Forall each $\langle e_{i,j}, l_{ij}, ijk\rangle \in$ `Supports`$[(e_{i,k}, l_{ik})]$)
    $t_{ijk} \leftarrow \triangle[(e_{i,j}, l_{ij}), (e_{i,k}, l_{ik}), (e_{j,k}, l_{jk})] \leftarrow$ `Supported-by`$(\langle(e_{i,j}, l_{ij}), ijk\rangle)$
    `Delete`$(\langle(e_{i,j}, l_{ij}), ijk\rangle$, `Supports`$[(e_{i,k}, l_{ik})])$
    `Delete`$(\langle(e_{i,j}, l_{ij}), ijk\rangle$, `Supports`$[(e_{j,k}, l_{jk})])$
    $t'_{ijk} \leftarrow \triangle[(e_{i,j}, l_{ij}), (e_{i,k}, l'_{ik}), (e_{j,k}, l'_{jk})] \leftarrow$ `First-support`$(\langle(e_{i,j}, l_{ij}), ijk\rangle)$
    If t'_{ijk}
      Then `Supported-by`$[(e_{i,j}, l_{ij}), ijk] \leftarrow t'_{ijk}$
      `Push`$(\langle e_{i,j}, l_{ij}, ijk\rangle$, `Supports`$[(e_{i,k}, l'_{ik})])$
        `Push`$(\langle e_{i,j}, l_{ij}, ijk\rangle$, `Supports`$[(e_{j,k}, l'_{jk})])$
      Else `Domain`$(e_{i,j}) \leftarrow$ `Domain`$(e_{i,j}) \setminus \{l_{ij}\}$
        `Push`$((e_{i,j}, l_{ij}), Q)$
        Unless `Domain`$(e_{i,j})$ **Then** Consistency $\leftarrow f$
Return $\{$ `Domain`$(e_{i,j})\}$

Figure 5: $\triangle$*AC*.

has position r. Then we select randomly $(n-2)$ distinct points within the given interval R, excluding the extremities of R.

2. We use the $\frac{n(n-1)}{2}$ combinations of two time points i and j (with $i < j$) generated above to generate a list L of edges $e_{i,j}$. Then we build the list E of edges by edges randomly selecting $|E|$ edges from L.
3. Measuring the distance $\delta = (j - i)$ for each edge $e_{i,j}$ in E, we label $e_{i,j}$ with a random number of intervals in $[1, k]$ while ensuring that there is at least one interval $[a, b]$ such that $\delta \in [a, b]$. This ensures that the resulting TCSP has a solution.
4. With probability $(1 - p_c)$, we swap the labels of two random edges in the graph.
5. Finally, we test the graph for connectivity and discard the unconnected graphs.

3.2 Experiments conducted

We tested $\triangle$AC on the randomly generated connected problems of Table 1. Our generator guarantees that at least 80% of these problems have at least one solution. We average the results over 100 samples.

In order to demonstrate the filtering power of $\triangle$AC, the comparison of the average size of the meta-CSP before and after filtering is shown in Figure 6 for TCSP I and Figure 7 for TCSP II. The numerical values reported in Table 2 and Table 3.

In order to demonstrate the advantages of $\triangle$AC, we report the cost of solving the meta-CSP with and without this preprocessing. This also allows us to verify the correctness of our implementation since both processes must yield exactly the same solutions. To solve the meta-CSP we use the basic chronological backtrack search described in [Dechter, 2003]. This search process requires solving an STP at each node expansion. To this end, we use the Directional Path-Consistency algorithm `DPC` also of Dechter [2003]. In our experiments, this algorithm was significantly more efficient than the Floyd-Warshall algorithm in determining the consistency of the STP (although it does not necessarily yield the minimal STP).

Solving the meta-CSP requires finding all its solutions. Figure 6 also shows the number of the solutions of the meta-CSP for TCSP I. We are not able to run the search without filtering for the TCSP II given the size of the meta-CSP and the necessity to find *all* its solutions.

The results of solving the meta-CSP in terms of CPU time and constraint checks `CC` for TCSP I are shown in Figure 8 and Figure 9, and the numerical values are reported in Table 2. In this table, we also report the cost of running $\triangle$AC although it is already included in the cost of search in order to demonstrate that the overhead due to filtering is practically negligible.

3.3 Observations

The comparison of Figure 6 and Figure 7 shows that the pruning power of $\triangle$AC increases with the size of the problem. It also shows that $\triangle$AC dramatically reduces the size of meta-CSP especially when density is high, which is typical of consistency filtering techniques used as a preprocessing step to search. More importantly, Figure 6 shows that the size of meta-CSP obtained after filtering by $\triangle$AC is close to the number of solutions for high-density networks.

Figure 8 and Figure 9 show the cost of solving the meta-CSP with and without preprocessing with $\triangle$AC in terms of constraint checks and CPU time, respectively. The figures show that preprocessing does not negatively affect the cost of search under low density and is tremendously effective in reducing the total cost under high density. Indeed, the cost of search is almost negligible when density is high. In contrast, search without preprocessing with $\triangle$AC is prohibitively expensive when density is high.

When density is low, the temporal graph has few edges, hence the meta-CSP has relatively few variables and its size is small. When density increases, the number of edges in the temporal graph, and hence the number of variables in the meta-CSP, increase, yield exponentially larger problems.

Experiment	TCSP					Samples per point	Results
	n	k	Density Range	Density Step	$\|E\|$ Range		
TCSP I	8	[1, 5]	[0.02, 0.1]	0.02	[7, 9]	100	Figure 6, Figure 8,
	8	[1, 5]	[0.2, 0.9]	0.1	[11, 26]	100	Figure 9 and Table 2
TCSP II	20	[1, 5]	[0.02, 0.1]	0.02	[22, 36]	100	Figure 7 and
	20	[1, 5]	[0.2, 0.9]	0.1	[53, 173]	100	Table 3

Table 1: *Problems tested.*

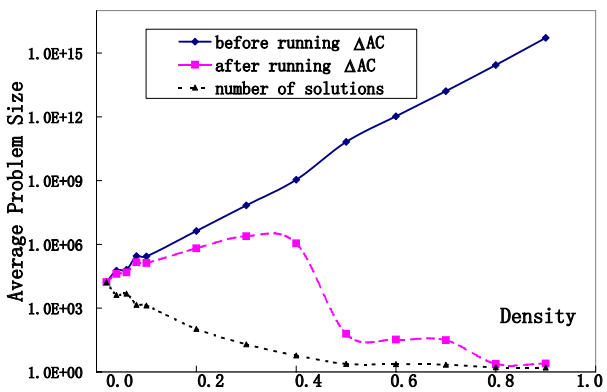

Figure 6: *Reduction of problem size of TCSP I.*

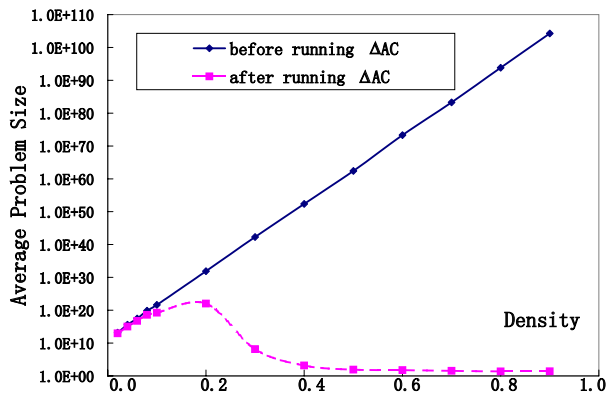

Figure 7: *Reduction of problem size of TCSP II.*

However, this increases the number of triangles in the temporal graph and enhances the filtering power of △AC, which removes most intervals. In all cases, the experiments strongly support using △AC when solving a TCSP.

Moreover, the use of △AC seems to uncover the potential existence of a phase transition around density $= 0.09$. We plan to investigate this more thoroughly in the future.

4 Conclusions

From the experimental results reported in the previous section, we draw the following conclusions:

1. △AC is sound in the sense that it never eliminates intervals that may appear in a solution of the meta-CSP.
2. △AC can dramatically reduce the size of the meta-CSP, especially when density is high. Hence it helps to improve the performance of the backtrack search to solve meta-CSP.
3. The cost of △AC is negligible compared with the cost of the search for solving the meta-CSP.

This establishes that the △AC is a cheap and effective consistency algorithm and should become part of any standard preprocessing technique for solving TCSP.

One interesting direction for future research is to investigate how △AC can be used to improve the performance of the `ULT` algorithm of Schwalb and Dechter [1997] since the two approaches are orthogonal and, to the best of our knowledge, the only reported consistency filtering techniques for TCSPs.

Acknowledgments:

This work is supported by a NASA-Nebraska grant and the CAREER Award #0133568 from the National Science Foundation. We are indebted to anonymous reviewers of a draft of this paper for their comments, which helped up improve our presentation.

References

[Bessière and Régin, 2001] Christian Bessière and Jean-Charles Régin. Refining the Basic Constraint Propagation Algorithm. In *Proc. of the 17 th IJCAI*, pages 309–315, Seatte, WA, 2001.

[Dechter, 2003] Rina Dechter. Constraint Processing. Manuscript, forthcoming, 2003.

[Mohr and Henderson, 1986] R. Mohr and T. C. Henderson. Arc and Path Consistency Revisited. *Artificial Intelligence*, 28:225–233, 1986.

[Mohr, 1988] R. Mohr. Good Old Discrete Relaxation. In *European Conference on Artificial Intelligence (ECAI-88)*, Munich, W. Germany, 1988.

[Schwalb and Dechter, 1997] Eddie Schwalb and Rina Dechter. Processing Disjunctions in Temporal Constraint Networks. *Artificial Intelligence*, 93:29–61, 1997.

Graph density	Number of variables in meta-CSP	Size of meta-CSP		Number of solutions	Cost of search without △AC		Cost of search with △AC		Cost of △AC	
		Original	Filtered		CPU [s]	CC	CPU [s]	CC	CPU [s]	CC
0.02	7	16701.67	16701.67	16701.67	13.6	518463.66	13.62	518463.66	5.00E-04	0
0.04	8	58448.44	40831.72	4176.91	21.6	843112.7	17.86	712777.7 5	0.0011	55.53
0.06	8	64780.24	48399.24	4837.69	25.03	965354.3	22.02	868557	0.0012	50.98
0.08	9	282427.3	142638.28	1437.01	24.23	1008288.4	18.14	782634.6	0.0022	122.7
0.1	9	271254.2	132758.27	1331.86	26.08	1103695.6	17.83	793677.7	0.0017	134.14
0.2	11	4257366	653949	105.88	23.95	1105540.5	6.43	335393.7	0.0033	324.44
0.3	13	6.81E+07	2424326.7	20.02	16.32	866010.3	2.1	117963.0 5	0.005	575.8
0.4	15	1.10E+09	1117395.5	5.97	22.13	1320010.5	0.49	29187.06 8	0.0075	880.23
0.5	18	6.64E+10	62.07	2.4	26.11	1630835.2	0.07	3654.7	0.0115	1383.8
0.6	20	1.06E+12	33.21	2.35	29.25	1932359.2	0.07	3821	0.015	1711.11
0.7	22	1.61E+13	31.16	2.19	34.87	2297002.5	0.077	3607.89	0.0192	2059.18
0.8	24	2.74E+14	2.41	1.66	57.13	3946315	0.07	3226.7	0.0217	2393.2
0.9	26	5.23E+15	2.48	1.6	74.39	5128653	0.08	3851.71	0.0262	2839.48

Table 2: *Performance of* △*AC* on TCSP I

Graph density	Number of variable in meta-CSP	Size of meta-CSP		Cost of △AC	
		Original	Filtered	CPU [s]	CC
0.02	22	1.51E+13	9.31E+12	4.10E-03	86.01
0.04	26	4.16E+15	1.05E+15	0.0064	253.1
0.06	29	2.97E+17	5.66E+16	0.008	362.02
0.08	33	7.27E+19	3.94E+18	0.0111	558.49
0.1	36	4.45E+21	1.72E+19	0.014	811.03
0.2	53	7.86E+31	1.11E+22	0.0362	2581.44
0.3	70	2.00E+42	1.48E+08	0.072	5268.13
0.4	87	2.23E+52	1545.05	0.114	8047.06
0.5	105	2.62E+62	79.69	0.168	11324.46
0.6	122	1.96E+73	60.2	0.254	15446.33
0.7	139	1.90E+83	37.11	0.332	20522.24
0.8	156	6.46E+93	23.55	0.433	26050
0.9	173	1.88E+104	24.6	0.554	33139.41

Table 3: *Performance of* △*AC* on TCSP II

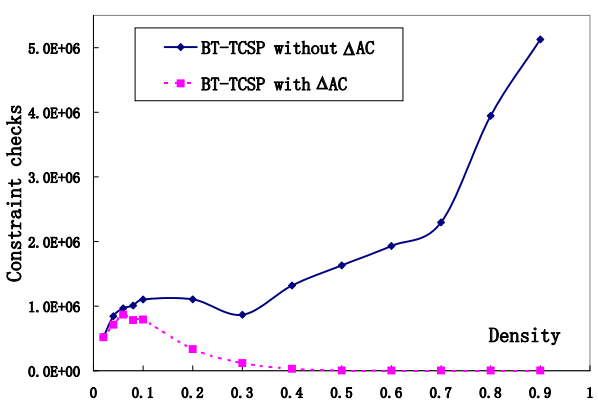

Figure 8: *Constraint checks for solving TCSP I.*

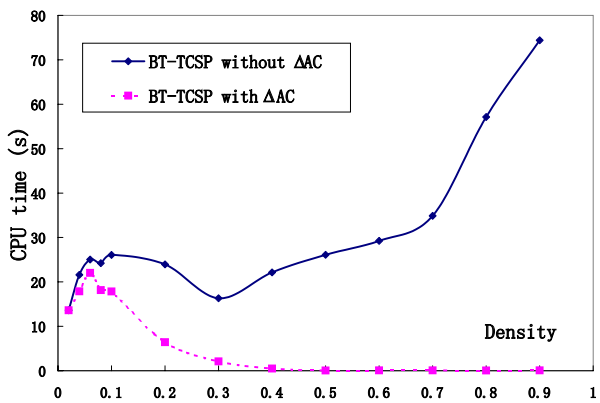

Figure 9: *CPU time for solving TCSP I.*