DBAI

TECHNICAL REPORT

INSTITUT FÜR INFORMATIONSSYSTEME

ABTEILUNG DATENBANKEN UND ARTIFICIAL INTELLIGENCE

A CSP Hypergraph Library

DBAI-TR-2005-50

Tobias Ganzow **Georg Gottlob** **Nysret Musliu**
Marko Samer

Institut für Informationssysteme
Abteilung Datenbanken und
Artificial Intelligence
Technische Universität Wien
Favoritenstr. 9-11/184-2
A-1040 Vienna, Austria
Tel: +43-1-58801-18403
Fax: +43-1-58801-18492
sek@dbai.tuwien.ac.at
http://www.dbai.tuwien.ac.at

DBAI TECHNICAL REPORT
JUNE 2005

TU
TECHNISCHE UNIVERSITÄT WIEN

DBAI TECHNICAL REPORT

DBAI TECHNICAL REPORT DBAI-TR-2005-50, JUNE 2005

A CSP Hypergraph Library

Tobias Ganzow[1] **Georg Gottlob**[2] **Nysret Musliu**[3]
Marko Samer[4]

Abstract. In this report we describe a collection of hypergraphs used to evaluate our implementations of hypertree decomposition algorithms. Hypertree decomposition was introduced by Gottlob et al. [6] as new structural decomposition method in constraint satisfaction. Since hypergraphs in constraint satisfaction do not have any specific properties, they can also be used for other evaluation purposes.

---

[1]Mathematische Grundlagen der Informatik, RWTH Aachen, D-52056 Aachen, Germany. E-Mail: ganzow@informatik.rwth-aachen.de

[2]Institut für Informationssysteme, Technische Universität Wien, Favoritenstr. 9-11/184-2, A-1040 Wien, Austria. E-Mail: gottlob@dbai.tuwien.ac.at

[3]Institut für Informationssysteme, Technische Universität Wien, Favoritenstr. 9-11/184-2, A-1040 Wien, Austria. E-Mail: musliu@dbai.tuwien.ac.at

[4]Institut für Informationssysteme, Technische Universität Wien, Favoritenstr. 9-11/184-2, A-1040 Wien, Austria. E-Mail: samer@dbai.tuwien.ac.at

1 Introduction

An instance of a constraint satisfaction problem (CSP) [10, 3] consists of a set of variables that range over a domain of values together with a set of constraints that allow certain combinations of values for certain sets of variables. The question is whether one can instantiate the variables in such a way that all constraints are simultaneously satisfied. Since the CSP is NP-complete in general, several approaches have been developed in the literature to identify tractable subclasses. One such direction is based on structural decomposition methods.

By the structure of a CSP instance we understand the dependencies between the variables caused by the constraints. These dependencies can be naturally represented by a hypergraph, that is a generalization of a graph such that the (hyper)edges connect an arbitrary subset of vertices. Each variable of a CSP instance represents a vertex in the hypergraph and each constraint represents a hyperedge connecting the vertices corresponding to the variables in the constraint. The domain values and allowed variable instantiations are ignored when considering only the structure of a CSP instance. The intuitive idea of structural decomposition methods is to decompose a constraint hypergraph into strongly connected components which can be organized as a tree. Examples of such decomposition approaches are *biconnected components* [5], *tree clustering* [4], *cycle cutset* [2], *hinge decomposition* [8], etc.

Gottlob et al. [6] recently introduced a new structural decomposition method called *hypertree decomposition*. They have shown that hypertree decomposition is more appropriate for decomposing constraint hypergraphs than other decomposition methods. The associated *hypertree-width* is a measure for the cyclicity of the underlying hypergraph. In particular, this means that the less the width of a hypertree decomposition the more efficiently the corresponding CSP can be solved. A competitive task is therefore to construct a hypertree decomposition of a given constraint hypergraph with width as small as possible. The minimal width over all hypertree decompositions is called the *hypertree-width* of the hypergraph.

In the scope of a research project [1], we have developed and implemented several algorithms for constructing hypertree decompositions of small width. For evaluating these algorithms, we collected a hypergraph library based on industrial CSPs as well as self-constructed hypergraphs. In this report, we present our hypergraph library; each hypergraph in the library is described in a text file in the following format:

```
Hyperedge_1 (Vertex_1_1, Vertex_1_2, ..., Vertex_1_n₁),
Hyperedge_2 (Vertex_2_1, Vertex_2_2, ..., Vertex_2_n₂),
⋮
Hyperedge_m (Vertex_m_1, Vertex_m_2, ..., Vertex_m_nₘ).
```

Due to this simple structure, we do not formally define the syntax of our format. Just note that the names of hyperedges and vertices may consist of any combination of lower- and uppercase letters, numbers, underscore, colon, etc. Comments start with '%' and continue until the end of the line. Some files may also contain definitions within angle brackets in the header.

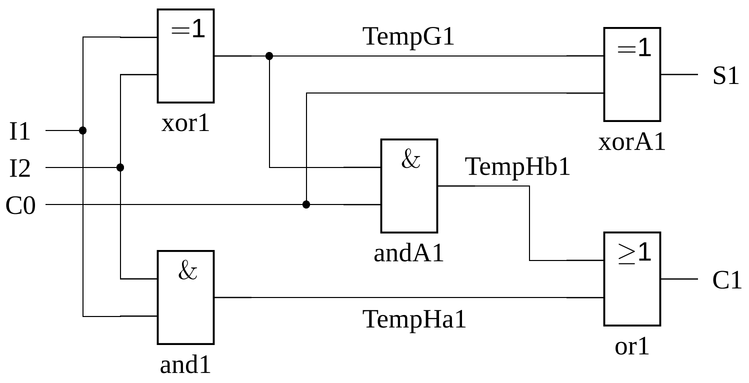

Figure 1: The basic cell of an adder circuit

2 The Hypergraph Library

2.1 DaimlerChrysler

In this section, we describe four classes of hypergraphs obtained from DaimlerChrysler during their cooperation with the Database and AI Group at Vienna University of Technology. In particular, these classes are *adder circuits*, *bridge circuits*, the *NewSystem* class, and the *ATV partial system*.

2.1.1 Adder Circuits

This kind of parameterized examples consists of a certain number of bit adders connected in a line. The basic cell of such an adder circuit is shown in Figure 1. The problem is to find the valid input-output values of such a circuit. For example, an adder circuit consisting of two basic cells connected in a line can be represented by a hypergraph in the following way:

```
init (C0),
and1 (I1, I2, TempHa1),
xor1 (I1, I2, TempG1),
andA1 (C0, TempG1, TempHb1),
or1 (TempHb1, TempHa1, C1),
xorA1 (TempG1, C0, S1),
and2 (I3, I4, TempHa2),
xor2 (I3, I4, TempG2),
andA2 (C1, TempG2, TempHb2),
or2 (TempHb2, TempHa2, C2),
xorA2 (TempG2, C1, S2).
```

Our hypergraph library contains 99 hypergraphs of this kind starting at `adder_1` (6 hyperedges and 8 vertices) extracted from a single basic cell up to `adder_99` (496 hyperedges and 694 vertices) extracted from a circuit consisting of 99 basic cells connected in a line. It is easy to

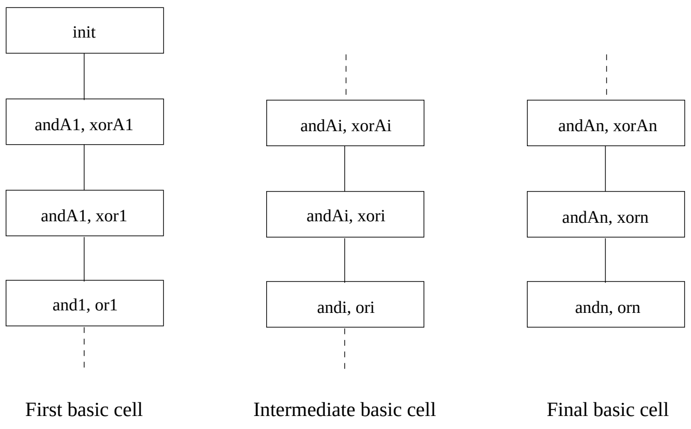

Figure 2: Adder decomposition by hand

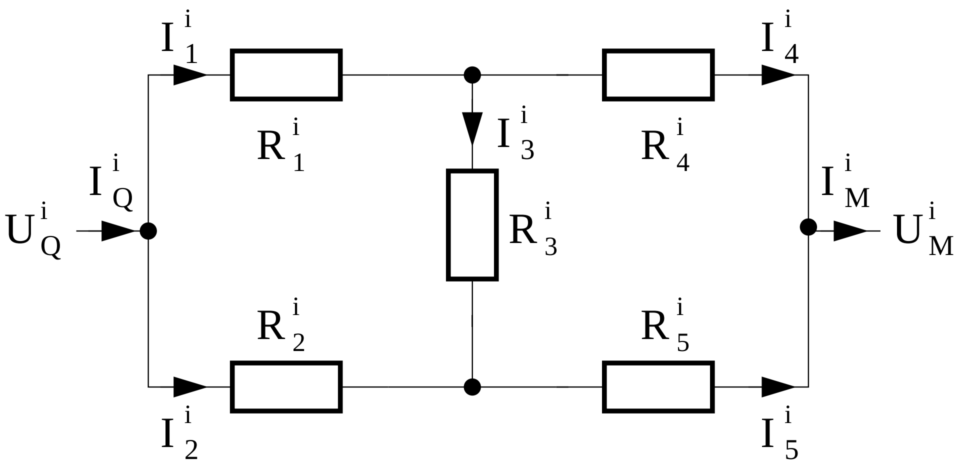

Figure 3: The basic cell of a bridge circuit

construct a hypertree decomposition of width 2 for these simple examples since each of the cells can be easily decomposed of width 2 and the interconnections of the cells are via two hyperedges. See Figure 2 for such a decomposition by hand. The hypertree-width of all these hypergraphs is 2.

2.1.2 Bridge Circuits

This kind of parameterized examples consists of a certain number of bridge cells connected in a line. The basic cell of such a bridge circuit is shown in Figure 3. The problem is again to find the valid input-output values of such a circuit. For example, a bridge circuit consisting of two basic cells connected in a line can be represented by a hypergraph in the following way:

```
Init (Uqv1),
M1v1 (IR1v1, IR2v1, IR3v1),
```

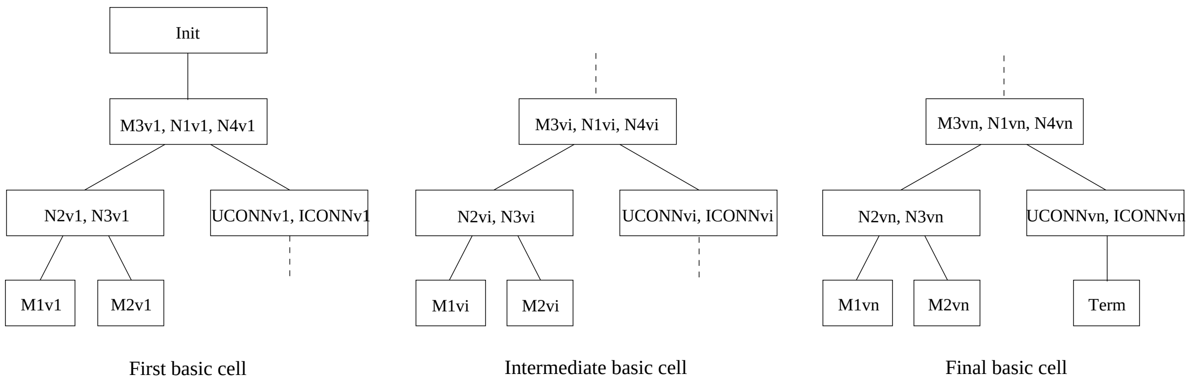

Figure 4: Bridge decomposition by hand

```
M2v1 (IR4v1, IR5v1, IR3v1),
M3v1 (Uqv1, IR1v1, IR4v1, Umv1),
N1v1 (Iqv1, IR1v1, IR2v1),
N2v1 (IR1v1, IR3v1, IR4v1),
N3v1 (IR2v1, IR3v1, IR5v1),
N4v1 (IR4v1, IR5v1, Imv1),
UCONNv1 (Umv1, Uqv2),
ICONNv1 (Imv1, Iqv2),
M1v2 (IR1v2, IR2v2, IR3v2),
M2v2 (IR4v2, IR5v2, IR3v2),
M3v2 (Uqv2, IR1v2, IR4v2, Umv2),
N1v2 (Iqv2, IR1v2, IR2v2),
N2v2 (IR1v2, IR3v2, IR4v2),
N3v2 (IR2v2, IR3v2, IR5v2),
N4v2 (IR4v2, IR5v2, Imv2),
UCONNv2 (Umv2, Uqv3),
ICONNv2 (Imv2, Iqv3),
Term (Uqv3).
```

Our hypergraph library contains 99 hypergraphs of this kind starting at `bridge_1` (11 hyperedges and 11 vertices) extracted from a single basic cell up to `bridge_99` (893 hyperedges and 893 vertices) extracted from a circuit consisting of 99 basic cells connected in a line. It is easy to construct a hypertree decomposition of width 3 for these examples since each of the cells can be easily decomposed of width 3 and the interconnections of the cells are via two hyperedges. See Figure 4 for such a decomposition by hand. In fact, some of our algorithms were able to construct hypertree decompositions of width 2 for all these examples. The hypertree-width of all these hypergraphs is 2.

2.1.3 NewSystem

Another kind of hypergraphs in our hypergraph library consists of four instances of increasing size called “NewSystem”. In particular, `NewSystem1` consists of 84 hyperedges and 142 vertices, `NewSystem2` consists of 200 hyperedges and 345 vertices, `NewSystem3` consists of 278 hyperedges and 474 vertices, and `NewSystem4` consists of 418 hyperedges and 718 vertices. The hypertree-width of `NewSystem1` and `NewSystem2` is 3. In the case of `NewSystem3` and `NewSystem4` we could only find hypertree decompositions of width 4 so far, i.e., their hypertree-width is at most 4.

2.1.4 ATV Partial System

The hypergraph `atv_partial_system` from DaimlerChrysler is a model of a jet propulsion system. It consists of 88 hyperedges and 125 vertices. Its hypertree-width is 3.

2.2 NASA

One of the most challenging examples in our hypergraph library is from NASA. In its original formulation it consists of 680 hyperedges and 579 vertices. However, the corresponding hypergraph is not connected; it contains 8 hyperedges that do not have vertices in common with any other hyperedge. Removing these isolated hyperedges results in a hypergraph consisting of 672 hyperedges and 569 vertices. Of course, the hypertree-width of both variants must be the same; some algorithms, however, require that the input hypergraph is connected.

Several attempts in the past to construct a hypertree decomposition of acceptable width for the NASA example failed. The first successful approach was due to McMahan [9], who achieved a decomposition of width 23. We could improve this result to width 21 by reimplementing McMahan’s algorithm and adding some randomization steps.

2.3 ISCAS

The well-known benchmark library of the IEEE International Symposium on Circuits and Systems (ISCAS) is often used in the literature for testing the performance of algorithms. There are three large ISCAS benchmark suites: ISCAS85, ISCAS89, and ISCAS99. The examples in these benchmarks model several kinds of circuits like ALUs, controllers, multipliers, etc. In order to make them accessible for our purposes, we transformed their representation into our file format.

In this way, we obtained the following 12 hypergraphs from the ISCAS85 benchmark suite: `c432` models a 27-channel interrupt controller and consists of 160 hyperedges and 196 vertices, `c499` models a 32-bit SEC circuit and consists of 202 hyperedges and 243 vertices, `c880` models a 8-bit ALU and consists of 383 hyperedges and 443 vertices, `c1355` models a 32-bit SEC circuit and consists of 546 hyperedges and 587 vertices, `c1908` models a 16-bit SEC/DED circuit and consists of 880 hyperedges and 913 vertices, `c2670` models a 12-bit ALU with controller and consists of 1193 hyperedges and 1350 vertices, `c3540` models a 8-bit ALU and consists

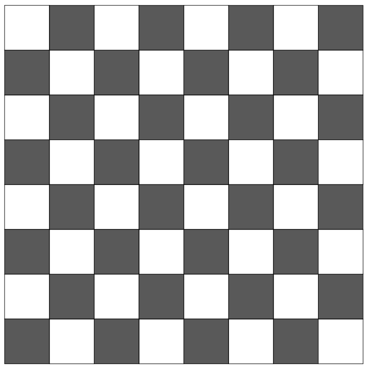
Figure 5: 8×8 Grid

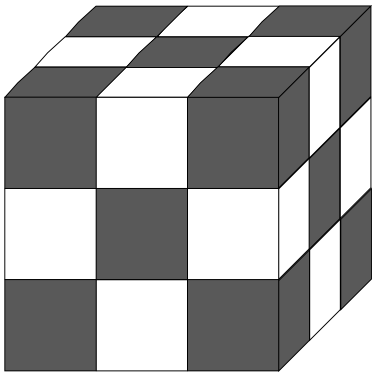
Figure 6: $3 \times 3 \times 3$ Cube

of 1669 hyperedges and 1719 vertices, `c5315` models a 9-bit ALU and consists of 2307 hyperedges and 2485 vertices, `c6288` models a 16×16 multiplier and consists of 2416 hyperedges and 2448 vertices, and `c7552` models a 32-bit adder/comparator and consists of 3512 hyperedges and 3718 vertices. From the ISCAS89 and ISCAS99 benchmark suites, we extracted 30 and 24 hypergraphs respectively starting from a size of 13 hyperedges and 17 vertices up to a size of 39531 hyperedges and 39568 vertices. For example, `s298` models a traffic light controller and consists of 133 hyperedges and 139 vertices and `s349` models a 4×4 add-shift multiplier and consists of 176 hyperedges and 185 vertices.

The widths of hypertree decompositions of the ISCAS hypergraphs vary significantly; in fact, we do not know their hypertree-width. However, they seem to be very useful for comparing various hypertree decomposition algorithms since our experiments have shown that it is very hard to construct hypertree decompositions of small width (if they exist) of these hypergraphs.

2.4 Grids

The grid hypergraphs in our hypergraph library are extracted from pebbling problems of the following form: Given an $n \times n$ chessboard (see Figure 5) where pebbles are arbitrarily placed on the white squares. The question is whether it is possible to put pebbles on the black squares such that for each of the pebbled white squares some of its adjacent black squares are pebbled and for each non-pebbled white square some of its adjacent black squares are not pebbled. This pebbling problem translates naturally into a Boolean CSP: For each black square we choose a variable and for each white square we choose an appropriate constraint on its adjacent black squares. For example, encoding a 4×4 grid in this way results in:

```
C0:1 (X0:0, X1:1, X0:2),
C0:3 (X0:2, X1:3),
C1:0 (X0:0, X2:0, X1:1),
C1:2 (X0:2, X1:1, X2:2, X1:3),
C2:1 (X1:1, X2:0, X3:1, X2:2),
C2:3 (X1:3, X2:2, X3:3),
C3:0 (X2:0, X3:1),
C3:2 (X2:2, X3:1, X3:3).
```

Our hypergraph library contains 98 hypergraphs of this kind starting at `grid2d_2` (2 hyperedges and 2 vertices) extracted from a 2×2 grid up to `grid2d_99` (4900 hyperedges and 4901 vertices) extracted from a 99×99 grid. A special feature of these hypergraphs is that they are highly cyclic, i.e., the hypertree-width of an $n \times n$ grid is at least $\frac{n}{3}$. This can be easily verified by the game-theoretic characterization of hypertree-width [7]. In particular, $\lfloor \frac{n}{3} \rfloor + 1$ marshals have a winning strategy to capture the robber on an $n \times n$ grid by advancing row by row through the grid and monotonically forcing the robber into one of the corners to eventually capture him. It is easy to see that the robber cannot be captured by a smaller number of marshals. Hence, the hypertree-width of a hypergraph encoding an $n \times n$ grid is $\lfloor \frac{n}{3} \rfloor + 1$.

We have also generalized the above idea of extracting hypergraphs from 2-dimensional grids in such a way that we consider 3-dimensional cubes and 4- and 5-dimensional hypercubes. For example, encoding a $3 \times 3 \times 3$ cube (see Figure 6) in the above way results in:

```
C0:0:1 (X0:0:0, X1:0:1, X0:1:1, X0:0:2),
C0:1:0 (X0:0:0, X1:1:0, X0:2:0, X0:1:1),
C0:1:2 (X0:0:2, X0:1:1, X1:1:2, X0:2:2),
C0:2:1 (X0:1:1, X0:2:0, X1:2:1, X0:2:2),
C1:0:0 (X0:0:0, X2:0:0, X1:1:0, X1:0:1),
C1:0:2 (X0:0:2, X1:0:1, X2:0:2, X1:1:2),
C1:1:1 (X0:1:1, X1:0:1, X1:1:0, X2:1:1, X1:2:1, X1:1:2),
C1:2:0 (X0:2:0, X1:1:0, X2:2:0, X1:2:1),
C1:2:2 (X0:2:2, X1:1:2, X1:2:1, X2:2:2),
C2:0:1 (X1:0:1, X2:0:0, X2:1:1, X2:0:2),
C2:1:0 (X1:1:0, X2:0:0, X2:2:0, X2:1:1),
C2:1:2 (X1:1:2, X2:0:2, X2:1:1, X2:2:2),
C2:2:1 (X1:2:1, X2:1:1, X2:2:0, X2:2:2).
```

Our hypergraph library contains 28 hypergraphs extracted from 3-dimensional cubes starting at `grid3d_2` (4 hyperedges and 4 vertices) extracted from a $2 \times 2 \times 2$ cube up to `grid3d_29` (12194 hyperedges and 12195 vertices) extracted from a $29 \times 29 \times 29$ cube. Moreover, our library contains 8 hypergraphs extracted from 4-dimensional hypercubes and 6 hypergraphs extracted from 5-dimensional hypercubes. Unfortunately, in contrast to 2-dimensional grids, it is very hard to determine the hypertree-width of these hypergraphs. In the case of 3-dimensional cubes, however, we were able to identify an interval in which the hypertree-width must lie. The idea is to generalize the above strategy in the Robber & Marshals game in such a way that the marshals move in shape of two adjacent 2-dimensional planes. A careful analysis reveals that

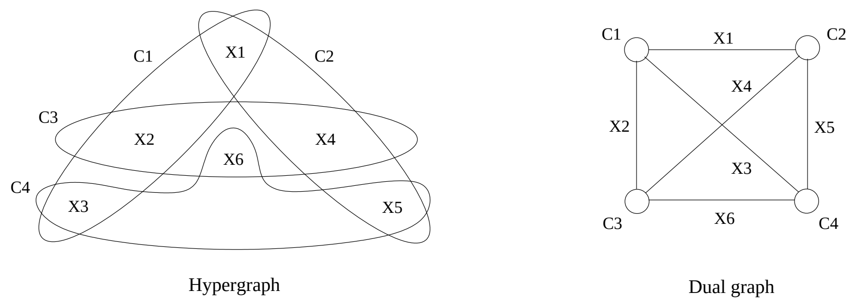

Figure 7: Hypergraph and its dual graph of the `clique_4` example

$M(n)+1$ marshals have a winning strategy in this way, where

$$M(n) = n - 2 + (2n-1)\left\lfloor \frac{n}{5} \right\rfloor - 5\left\lfloor \frac{n}{5} \right\rfloor^2 + \begin{cases} 2 & \text{if } n \equiv 4 \pmod 5 \\ 0 & \text{otherwise.} \end{cases}$$

Moreover, note that $\left\lceil \frac{n^2}{5} \right\rceil$ marshals are necessary to cover a single plane in an $n \times n \times n$ cube, which yields a trivial lower bound of $\left\lceil \frac{n^2}{5} \right\rceil + 1$ marshals to capture the robber. Hence, the hypertree-width of a hypergraph encoding an $n \times n \times n$ cube lies in the interval $[\left\lceil \frac{n^2}{5} \right\rceil + 1, M(n)+1]$.

2.5 Cliques

The clique hypergraphs in our hypergraph library are a parameterized class of examples, each consisting of n hyperedges such that each hyperedge has a vertex in common with each other hyperedge and each vertex occurs in exactly two hyperedges, i.e., the dual graph of the hypergraph is a clique such that each edge in the dual graph corresponds to exactly one vertex in the hypergraph. For example, `clique_4` consisting of 4 hyperedges and 6 vertices has the form:

```
C1 (X1, X2, X3),
C2 (X1, X4, X5),
C3 (X2, X4, X6),
C4 (X3, X5, X6).
```

The hypergraph and its dual graph of the `clique_4` example is shown in Figure 7. Our library contains 98 hypergraphs whose dual graphs are cliques as described above starting at `clique_2` consisting of 2 hyperedges and 1 vertex up to `clique_99` consisting of 99 hyperedges and 4851 vertices. It is easy to see that such clique hypergraphs with n hyperedges have hypertree-width $\lceil \frac{n}{2} \rceil$. This follows immediately from the fact that each hyperedge is connected with each other hyperedge via a unique vertex. The corresponding hypertree consists of only two nodes such that one node is labeled with $\lceil \frac{n}{2} \rceil$ hyperedges and the other node is labeled with

$\lfloor \frac{n}{2} \rfloor$ hyperedges. Since every hypergraph can be decomposed in this way (the conditions of a hypertree decomposition are trivially satisfied), the clique examples represent the worst case concerning hypertree-width. Although there exists only a trivial hypertree decomposition of these hypergraphs, some of our algorithms spent a huge amount of time for finding this decomposition.

3 Conclusion

In this report, we presented our CSP hypergraph library consisting of several classes of hypergraphs. Our library contains industrial examples from DaimlerChrysler, NASA, and the ISCAS circuits as well as self-constructed hypergraphs like Grids and Cliques. The latter ones have the advantage that their hypertree-width is known in advance.

Our experiments have shown that the evaluation results sometimes depend on the order of the hyperedges and vertices in the input file. Therefore, we propose to order hyperedges and vertices randomly when reading them from the file. This guarantees that the obtained results only depend on the hypergraph and not on a particular representation of the hypergraph.

References

[1] Complementary approaches to constraint satisfaction. Project Proposal, Database and AI Group, Vienna University of Technology, 2003.

[2] Rina Dechter. Constraint networks. In Wiley and Sons, editors, *Encyclopedia of Artificial Intelligence*, pages 276–285. Second edition, 1992.

[3] Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[4] Rina Dechter and Judea Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1989.

[5] Eugene C. Freuder. A sufficient condition for backtrack-bounded search. *Journal of the ACM*, 32(4):755–761, 1985.

[6] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decomposition and tractable queries. *Journal of Computer and System Sciences*, 64(3):579–627, 2002.

[7] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Robbers, marshals, and guards: Game theoretic and logical characterizations of hypertree width. *Journal of Computer and System Sciences*, 66(4):775–808, 2003.

[8] Marc Gyssens, Peter G. Jeavons, and David A. Cohen. Decomposing constraint satisfaction problems using database techniques. *Artificial Intelligence*, 66:57–89, 1994.

[9] Ben McMahan. Bucket eliminiation and hypertree decompositions. Implementation Report, Database and AI Group, Vienna University of Technology, 2004.

[10] Edward Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.