TU
TECHNISCHE UNIVERSITÄT WIEN

DIPLOMARBEIT

Generalized Hypertree Decomposition based on Hypergraph Partitioning

ausgeführt am

Institut für Informationssysteme
Abteilung für Datenbanken und Artificial Intelligence
der Technischen Universität Wien

unter der Betreuung von

Univ. Prof. Dipl. Ing. Dr. Georg Gottlob
Univ. Ass. Dipl. Ing. Dr. Nysret Musliu

durch

Artan Dermaku
Lorezmüllergasse 1A/5002
1200 Wien

Wien, den 21. Dezember 2006 ....

Acknowledgements

I would like to thank Prof. Georg Gottlob and Dr. Nysret Musliu for supervising and proofreeding of this diploma thesis, and for their useful suggestions and their valuable support.

Special thanks to my parents Ejup and Fahrije Dermaku and all my family for theirs moral and financial support during my studies. I would like also to thank my fiancee Krenare Sogojeva for reading this thesis and for her useful suggestions.

This master thesis is supported by the Austrian Science Fund (FWF) project: Nr. P17222-N04, Complementary Approaches to Constraint Satisfaction (Project time: Sept. 2004-2006).

Abstract

The first deed of humans to solve any real problem in computer science, is the finding of an adequate mathematical model. This model should make possible the simplification of the problems and at the same time tries to make it solvable.

The CSP is one such model for many problems in AI, databases or mathematics. However, it is known that CSPs are in general $NP - complete$ problems and thus intractable. The hypertree decomposition is one of the best decomposition models, which can be used to solve tractable classes of CSP. However, the finding of a minimal generalized hypertree decomposition for bounded width at most k is also $NP - complete$. Thus several heuristic approaches are developed to find an “optimal” or "nearly optimal" hypertree decomposition.

In this diploma thesis we developed two new heuristic algorithms for generalized hypertree decomposition. Both these algorithms are based on hypergraph partitioning.

The first algorithm tries to find recursively the “best local” decomposition of dual graph. In order to achieve “good” decomposition of the hypergraph, we propose a method which finds the cycles of its dual graph and then computes “touch points” of these cycles. The second algorithm uses the library packages of HMETIS partitioning approach, known in the literature as one of the best partitioning algorithm.

The computational results of those both approaches show that the heuristics implemented in this diploma thesis can achieve for many problems good results. For some given benchmark problems from the literature, the proposed heuristics can produce the generalized hypertree decomposition of minimal width.

Kurzfassung

Wenn wir uns in der Informatik mit der Lösung eines realen Problems beschäftigen, versuchen wir als Erstes ein entsprechendes mathematisches Modell für das Problem zu finden. Ein solches Modell sollte uns ermöglichen, eine Vereinfachung des Problems zu erreichen, es verständlicher zu gestalten und gleichzeitig es lösbar zu machen. Für viele Probleme im Bereich der AI, Datenbanken und Mathematik, stellt CSP ein solches Modell dar. Allerdings ist es bekannt, dass das Lösen eines CSP ist im allgemeinen $NP - vollständig$, das heißt nicht in polynomieller Zeit lösbar, ist. Hypertree Decomposition ist die beste Zerlegungsmethode, die wir verwenden können, um Klassen von CS Problemen zu lössen. Trotzdem, das Auffinden einer minimalen Generalzed Hypertree Decomposition für einen "bounded width" k ist auch $NP - vollständig$. Deshalb sind bereits einige heuristische Methoden entwickelt, um eine "optimale" hypertree decomposition zu erzeugen.

In dieser Magisterarbeit entwickelten wir zwei neue heuristische Algorithmen, die eine generalized hypertree decomposition erzielen. Beide diese Algorithmen basieren auf Hypergraph Partitioning. Der erste Algorithmus versucht rekursiv die beste „lokale Decomposition“ von dualem Graph zu finden. Um eine „gute“ Decomposition zu erreichen, wir schlagen eine Methode vor, die einige Zyklen seines duale Graph findet und dann die „Berührungspunkte“ zwischen diese Zyklen berechnet. Der zweite Algorithmus verwendet Bibliotheken des HMETIS Zerlegungs-Methode. Diese Methode ist bekannt in der Literatur als einer des besten Zerlegungsalgorithmen.

Unsere Tests jener beide Methode zeigen dass die Heuristiken teilweise sehr gute Ergebnisse produzieren .

Contents

1 Introduction

In computer sciences, computational problems occurring are classified according to their complexity, because complexity highly influences the solvability of a problem. One of the classifications used is to divide the problems with polynomial running time from problems with non-deterministic polynomial running time. Many authors have worked on this topic and have defined classes of problems such as P, NP, NP hard and $NP-complete$ problems.

Only a few problems in computer science those are important and thus needed to be solved, can be solved in polynomial time, i.e. their worst-case running time is $O(n^k)$ for some constant k. This class of problems in the literature is called P problems. For other problems which have a practical application we can not find an algorithm solving them in polynomial time but only an exponential algorithm. Exponential algorithms however bring difficulties for the computer in cases of rather complex input problems. There exist a number of problems which are theoretically solvable but their running times change enormously, that is exponentially, depending on input size. This kind of problems becomes unsolvable quickly even for marginal differences in problem size. Instances of problems that have only been solved with exponential algorithms so far might still have polynomial time solutions which may be found in future. A number of problems that have been solved only with exponential algorithms however have the property that for a given solution we can verify that solution in polynomial time. The class of problems that have these properties are called NP (Non-Deterministic Polynomial) problems. If solving a NP problem in polynomial time makes it possible to solve all other problems of this class also in polynomial time, we say a problem is $NP-hard$ (Non-Deterministic Polynomial hard). In other words, the algorithm which allows to solve a NP problem in polynomial time can be modified and to solve any another NP problem also in polynomial time. And finally for a problem having both NP and $NP-hard$ properties we say is NP complete problem. Further we say a problem is *tractable* if we find a solution in polynomial time, otherwise it is *intractable*.

The CSPs are the most important class of problems in computer science, which are known to be $NP-complete$. This art of problems occur while we try to solve different problems in AI, Databases and Operation Researches. In order to solve these problems we have to satisfy a certain number of constraints. With other words, to solve any CSP problem it is meant to find such allowed values, which are assigned to variables of CSP-s and which will satisfies all given constraints.

A number of problems which can be modelled as Constraint Satisfaction Problems are Boolean Conjunctive Query (BCQ) , Homomorphism Problem (HOM), Graph k-Colourability, Advanced Planning, N-Queens Problem, and many other problems.

In order to solve CSP-s we represent them graphically. The graphic structure of CSP can be a Tree, Graph or Hypergraph. The Hypergraph structure of CS problems will be in focus of this diploma thesis, because according to [5], several $NP - complete$ problems will be tractable if are restricted to instances with acyclic hypergraphs. Therefore a number of decomposition methods are developed in order to reduce the cyclicity of hypergraph, and if it is possible to generate an acyclic hypergraph. All these decomposition methods defines a concept of width which can be interpreted as measure of cyclicity, such that for each fixed *width* k, all CSPs of width bounded by k are solvable in polynomial time [4]. Further those methods try to decompose a given CSP instance into smaller sub-problems which then can be solved efficiently. The meaning of width k is denoted as the size of the greatest subproblem. The Hypertree Decomposition, developed by Gottlob et al. [5] , is the most powerful method among decomposition methods. There exists an implementation of Hypertree Decomposition, *opt-k-decomp* [11], which for a given constant k checks in the poynomial time if the hypergraph has hypertree width k. The smaller k implies a better solution. This algorithm, for a given hypergraph produces an exact hypertree decomposition. When we solve a CSP, normally we want to find an optimal solution, that is, in our case, to reproduce minimal hypertree decomposition. The problem is however that k appears in the exponent of runtime of this algorithm, thus for large problems it will be quickly unusable.

To overcome this problem several heuristic methods are developed. All these methods try to find a nearly optimal solution within a satisfying running time. Some of these methods are developed by DBAI research group [2]. They are based on vertex ordering as well as on hypergraph partitioning. These methods usually produce a generalized hypertree decompostions. If the fourth condition in definiton of hypertree decompostions is ignored, the corrosponding decomposition is called generalized hypertree decompostion. Note that the first three condions of hypertree decompositons are sufficient to solve the the corresponding CS problem in polynomial time. The fourth condition was added to aid the proof that, for a fixed k, determining if a hypergraph H has hypertree width k can be solved in polynomial time. In this thesis we investigate the genaration of generalized hypertree decompositions.

The purpose of this diploma thesis was finding and implementing of new heuristic approaches for generalized hypertree decomposition based on hypergraph partitioning. These approaches should solve the problems of given benchmarks which *opt-k-decomp* was not able to solve.

We implemented two different heuristics based on partitioning algorithms. The first algorithm tries to find a significant number of intersection points between the cycles of dual graphs. Evaluating of these “touch” points with respect to certain fitness criteria should lead to a “good” local partitioning of the graph. The experimental results show that for some problems, this algorithm achieves partly optimal solutions.

The second algorithm is based on HMETIS partitioning algorithm and uses HMETIS package library. The computational results show that this algorithm yields a very good solution. In comparison with some other existing heuristic algorithm, we conclude that

HMETIS and Bucket Elimination heuristics give the best results.

This thesis is organized as follows. Chapter 1 gives an introduction, and in chapter 2 are given basic definitions. Chapter 3 describes two heuristic methods proposed in this thesis for generation of hypertree decomposition. Furthermore this chapter gives a survey of previous methods used for hypertree decompositions. Computational results are given in chapter 4. In chapter 5 are given the conclusion remarks.

2 Fundamentals

This chapter gives a short overview and some basic definitions about Constraint Satisfaction Problems (2.1), Boolean Conjunctive Queries (2.2) and Homomorphism Problems (2.3). All these classes of problems can be represented by the same structure, that is hypergraph (2.4). Hypertree decomposition is one of the latest methods for decomposing hypergraphs developed by Gottlob et al [5] which tends to reduce the complexity of problems. That can be achived if the cyclicity of these hypergraphs will be restricted that they will become acyclic or nearly acyclic and thus tractable [8]. Following sections of this chapter show the background and basis for understanding the problems and goals of hypertree decomposition.

2.1 Constraint Satisfaction Problems

Many real problems of Articial Intelligence, Database systems, Operation research etc. can be represented as Constraint Satisfaction Problems (CSP). This art of a modeling of problems is very useful and sometimes the only way to make them tractable. In order to describe those mathematical models, below we give some basic definitions which can be found at [5, 6, 15, 20].

For example, the problems of advanced planning and scheduling, network management, theory of graphs, electrical engineering, can be represented in such a way that we define objects as a collection of *variables* $Var = \{X_1, X_2, ..., X_n\}$ which all have to be assigned values, called *domain* D_i.

In general, different variables can have different domains. The relations assumed between the values of variables are called Constraints. For a Constraint C we define its scope, as the set of variables, which are contained in Constraint C. Binary constraint scopes are scopes containing only two variables. The constraints can be given intentionally, i.e. as a formula, extensionally, i.e. as a set, or procedurally, i.e. with a recognising function. An *instance* of CSP is a tripple $\langle Var, D, C\rangle$ where Var is a set of variables, D is a finite domain of values and C is finite set of constraints. The constraint satisfaction problem is to find such value in D_i for variable X_i, $\forall i \in \{1, ..., n\}$, so that all constraints are satisfied.The CSP can be finite or infinite. The finite CSP involves discrete variables which have finite domains.

Further we give some typical examples of finite-domain constraint satisfaction problems which are similar with those found everywhere in the literature.

Example 1 *The 8-queens problem: Only one queen is allowed to be on the same row, column or diagonal .*

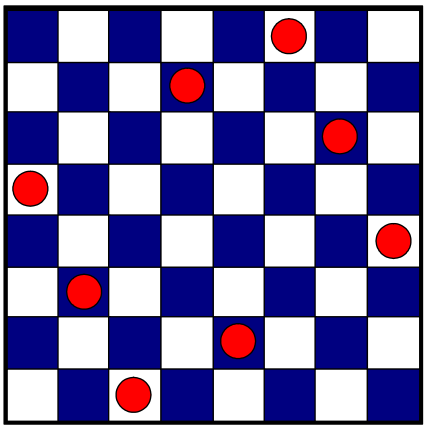

Figure 2.1: 8-queens problem

Example 2 *A crossword puzzle: The puzzle has to be completed (Figure 2.2).*

Given the list of words: HOME, MEAT, EU, GO, EUR, OU, ARITY, ISO, YAF, AG, WOLF. The words can be inserted in the cross-puzzle only at the locations labeled with numbers 1,2,...,12 .

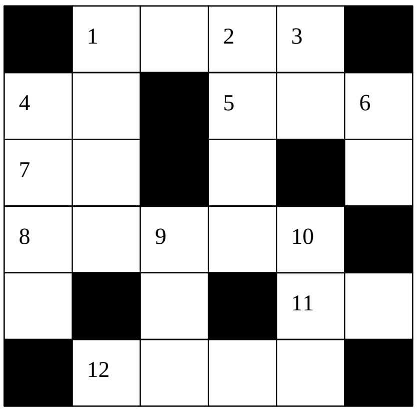

Figure 2.2: A crossword-puzzle

Example 3 *A cryptography problem: Each letter have to be replaced by a distinct digit. The sum have to be correct.*

$$\begin{array}{rl} & TWO \\ + & TWO \\ \hline\hline & F\,O\,U\,R \end{array}$$

Example 4 *A map coloring problem: All neighbour countries (Figure 2.3) should have different colours (green, red or blue).*

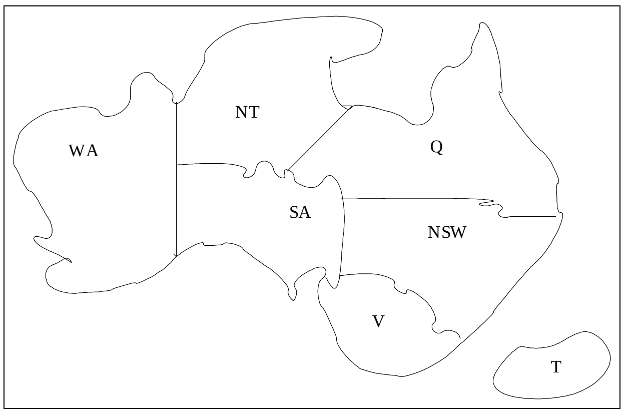

Figure 2.3: A map coloring problem refers to the states of Australia

The definition of constraint satisfaction problems, varies between authors [19, 5, 6, 7]. In this diploma thesis I will adopt the following formal definition

Definition 1 *[15] A constraint satisfaction problem is a sextuple $P = (X, \Delta, \delta, C, \Sigma, \sigma)$ where*

- *δ is a mapping from the finite set of variables X onto the set of sets Δ; for each $x \in X, \delta\,(x)$ is called the domain of x;*
- *σ is a one-two-one mapping from the finite set of constraints C onto the set of sets Σ satysfaing $\cup\Sigma = X$; for each $c \in C, \sigma\,(x)$ is called the scope of c.*

A mapping t from $Y \subseteq X$ into $\cup\Delta$ such that $t(x) \in \delta(x)$, for all $x \in Y$, is called a labeling of Y. Each constraint $c \in C$ is a set of labelings of $\sigma(c)$.

Example 5 *We represent Example 4 in formal notation:*
Let $P = (X, \Delta, \delta, C, \Sigma, \sigma)$ be the constraint satisfaction problem. Then :

- *$X = \{WA, NT, Q, NSW, V, SA, T\}$ where each variable stands for a region.*

- $\Delta = \{\{red, green, blue\}\}$.
- $\delta(X_i) = \{red, green, blue\}$, *for* $i = 0, 1, ..., 6$.
- $C = \{C_1, C_2, C_3, ..., C_{10}\}$ *where*

  $$C_1 \equiv C_2 \equiv C_3 \equiv ... \equiv C_9 = \{\{red, green\}, \{red, blue\}, \{green, red\}, \{green, blue\}, \{blue, red\}, \{blue, green\}\}$$

  $$C_{10} \equiv \{red, green, blue\}.$$
- $\Sigma = \{S_1, S_2, S_3, ..., S_{10}\}$ *where the pair of values each* S_i *are the neighbour countries except* S_{10} *which defines the country without neighbours, T.*

  $S_1 = \{WA, NT\}$ $\quad S_2 = \{WA, SA\}$ $\quad S_3 = \{NT, SA\}$

  $S_4 = \{NT, Q\}$ $\quad S_5 = \{Q, SA\}$ $\quad S_6 = \{Q, NSW\}$

  $S_7 = \{SA, NSW\}$ $\quad S_8 = \{SA, V\}$ $\quad S_9 = \{NSW, V\}$

  $S_{10} = \{T\}$
- $\sigma(C_i) = S_i, i = 1, 2, ..., 10.$

There are many possible solutions, such as:
$\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = blue\}$

Example 6 *We represent Example 3 in formal notation:*
Let $P = (X, \Delta, \delta, C, \Sigma, \sigma)$ *be the constraint satisfaction problem. Then :*

- $X = \{F, U, R, T, W, O\}$ *where each variable stands for a given letter.*
- $\Delta = \{\{0, 1, 2, ..., 9\}\}$.
- $\delta(X_i) = \{0, 1, 2, ..., 9\}$, *for* $i = 0, 1, ..., 9$.
- $C = \{C_1, C_2, C_3, C_4, C_5\}$ *where*
- $C_1 = \{\{1, 2\}, \{2, 4\}, \{3, 6\}, \{4, 8\}, \{5, 0\}, \{6, 2\}, \{7, 4\}, \{8, 6\}, \{9, 8\}\}$

  $C_2 = \{\{0, 0\}, \{0, 1\}, \{1, 2\}, \{1, 3\}, \{2, 4\}, \{2, 5\}, \{3, 6\}, \{3, 7\}, \{4, 8\}, \{4, 9\}, \{5, 0\}, \{5, 1\}, \{6, 2\}, \{6, 3\}, \{7, 4\}, \{7, 5\}, \{8, 6\}, \{8, 7\}, \{9, 8\}, \{9, 9\}\}$

  $C_3 = \{\{5, 1, 0\}, \{5, 1, 1\}, \{6, 1, 2\}, \{6, 1, 3\}, \{7, 1, 4\}, \{7, 1, 5\}, \{8, 1, 6\}, \{8, 1, 7\}, \{9, 1, 8\}, \{9, 1, 9\}\}$

$$C_4 = \{\{5,0,1\},\{5,0,2\},...,\{5,0,9\},\{5,1,0\},\{5,1,1\},...,\{5,1,9\},...,\{5,9,0\},\{5,9,1\},...,\{5,9,9\},\{6,0,1\},...,\{6,9,9\},...,\{9,9,8\}\}$$

$$C_5 = \{\{1,2,3,0\},\{1,2,3,4\},\{1,2,3,6\},\{1,2,3,8\},\{1,2,4,0\},\{1,2,4,6\},\{1,2,4,8\},\{1,2,5,0\},\{1,2,5,4\},\{1,2,5,8\},...,\{1,9,7,8\}\}$$

- $\Sigma = \{S_1, S_2, S_3, S_4, S_5\}$ where
  $S_1 = \{O, R\}$ $S_2 = \{W, U\}$ $S_3 = \{T, F, O\}$ $S_4 = \{T, W, O\}$
  $S_5 = \{F, O, U, R\}$
- $\sigma(C_i) = S_i, i = 1, 2, 3, 4, 5$

The solutions to P :

F	U	R	T	W	O
1	6	8	7	3	4
1	3	0	7	6	5
1	7	2	8	3	6
1	9	2	8	4	6
1	3	4	8	6	7
1	5	6	9	2	8
1	7	6	9	3	8

Definition 2 *[15] Let $P = (X, \Delta, \delta, C, \Sigma, \sigma)$ be a constraint satisfaction problem.*

- *Given any constraint $c \in C$ a labeling t of $\delta(c)$ is said to satisfy c if $t \in c$.*
- *A labeling t of X is said to be a solution to P if, for every $c \in C$,the restriction of t to $\delta(c)$ satisfies c. The set of all solutions to P is denoted $Sol(P)$.*

We illustrate this definition with an example:

Example 7 *Let be $t = \{1, 6, 8, 7, 3, 4\}$ any labeling of X of example 6.*

- *For $c = C_1 = \{\{1,2\},\{2,4\},\{3,6\},\{4,8\},\{5,0\},\{6,2\},\{7,4\},\{8,6\},\{9,8\}\} \Longrightarrow \delta(C_1) = S_1 = \{O, R\}$. The labeling t of $\delta(c)$ is $t_1 = \{4, 8\}$, which satisfy c because $t_1 \in C_1$*

- *A labeling $t = \{1,6,8,7,3,4\}$ is a solution of P, because $\forall c \in C$, restriction of t to $\sigma(c)$ satisfies c.*

  For $S_1 = \{O,R\} \Longrightarrow t_1 = \{4,8\}$ satisfies c_1

  For $S_2 = \{W,U\} \Longrightarrow t_2 = \{3,6\}$ satisfies c_2

  For $S_3 = \{T,F,O\} \Longrightarrow t_3 = \{7,1,4\}$ satisfies c_3

  For $S_4 = \{T,W,O\} \Longrightarrow t_4 = \{7,3,4\}$ satisfies c_4

  For $S_5 = \{F,O,U,R\} \Longrightarrow t_5 = \{1,4,6,8\}$ satisfies c_5

Definition 3 *[15] Let $P = (X,\Delta,\delta,C,\Sigma,\sigma)$ be a constraint satisfaction problem, and let D be any subset of C. The subproblem of P generated by D is the constraint satisfaction problem $P\ |_D = (X\ |_D, \delta(X\ |_D), X\ |_{X|_D}, D, \sigma(D), \sigma\ |_D)$, where $X\ |_D = \cup_{c\in D}\sigma(c)$.*

For every solution t of $P \Longrightarrow$ the restriction of t to $X\ |_D$is a solution to $P\ |_D$ .We can also say that a solution t' to $P\ |_D$ can be *extended* to a solution to P if there exists a solution t to P such that t' is the restriction of t to $X\ |_D$ [15].

Example 8 *Consider $D_1 = \{C_1, C_2\}$ as a subset of the constraints of CSP P of example 6.*

Then $X\ |_{D_1} = \{U,R,W,O\}$. There are 51 possible solutions.
*The set of solutions : $Sol(P\ |_{D_1}) = \{(0,2,5,1),(0,4,5,2),...,(4,2,7,1),...,(6,8,3,4),...,$
$(8,2,4,1),...,(9,6,4,8)\}$*

- *For $t = \{1,6,8,7,3,4\}$, the restriction of t to $X\ |_{D_1}$is $t_1 = \{6,8,3,4\}$. We can verify easily that $t_1$is a solution to $P\ |_{D_1}$.*

- *Conversely a solution to $P\ |_{D_1}$ $t_1 = \{6,8,3,4\}$ can be extended to a solution of to a solution to P, because there exists a solution $t = \{1,6,8,7,3,4\}$ to P and $t_1$is the restriction of t to $X\ |_{D_1}$*

- *But a solution to $P\ |_{D_1}$ $t_2 = \{9,6,4,8\}$ can not be extended to a solution to P, becuse t_2 cannot be extended to any solution to P.*

Two different constraint satisfaction P and P' over the same set of variables are *equivalent* if $Sol(P) = Sol(P')$. [15]

2.2 Databases and Queries

CSP problems in general can have a set of finite relations. In such cases it is very useful to use database methods to solve them. One of most important classes of *database queries* is the class of *conjunctive queries* (CQs) evaluating of which is known to be equivalent to solving of constraint satisfaction problems [6, 7, 15, 23]. First we give some basic concepts and definitions of databases and of query problems (taken from [6]) in detail.

A *relation schema* $R = \{A_1, ..., A_n\}$ is a finite set of *attributes*. Each attribute A_i for $1 \leq i \leq n$ has a *domain* $Dom\{A_i\}$. n is called the *arity* of relation. A *relation instance* r over schema R, is a finite subset of the cartesian product $Dom(A_1) \times ... \times Dom(A_n)$. The elements of this subset are called *tuples*.

A database schema $DS = \{R_1, ..., R_m\}$ is a finite set of relation schemas. A *database instance db* consists of relation instances $r_1, ..., r_m$ for the schemas $R_1, ..., R_m$ respectively. More formally is shown by Gottlob [6]. A conjunctive Query on database consists of a rule of the form

$$Q : ans(u) \leftarrow r_1(u_1) \wedge ... \wedge r_n(u_n)$$

where $n \geq 0; r_{1,...}, r_n$ are relation names of database schema, ans is relation name not in database schema, and $u, u_1, ..., u_n$ are lists of terms(variables). If ans does not contain variables i.e., its arity is 0 than the conjunctive query is a *Boolean conjunctive query* (BCQ). Such a query can evaluate to *true* or *false*.

Conjunctive queries are also equivalent to the SQL queries of the type

$$SELECT\ R_{i_1}A_{j_1}, ..., R_{i_k}A_{j_k}\ FROM\ R_1, ..., R_n\ WHERE\ cond,$$

where cond is a conjunction of conditions of the form $R_iA = R_jB$ or $R_iA = C$, for C constant [6].

Example 9 *We represente Example 4/5 as database*

a	1	2	3
	R	G	B
	R	B	G
	G	R	B
	G	B	R
	B	R	G
	B	G	R

e	1
	R
	G
	B

where $b \equiv c \equiv d \equiv a$

and query :

$$ans \leftarrow a(NT, WA, SA) \wedge b(NT, SA, Q)$$
$$\wedge c(SA, NSW, Q) \wedge d(SA, NSW, V) \wedge e(T)$$

Input of a query problem in general is a database and output is evaluation of query against that database [5]. If a query have a cyclic associated hypergraph than we say that the query is also cyclic otherwise the query is acyclic [5].

Conjunctive querie problems are in general NP-hard. If we join all its relation instances we get actually the set of all solution of his equivalent constraint satisfaction problem [15]. However that requires an exponential time what make it inefficient. But there is an exception, acyclic boolean conjunctive query (ABCQ). Yannakakis [19] developed an algorithm which solved acyclic boolean conjunctive queries in polynomial time. This algorithm is based on solving JTREE using semijoins.

First we define JTREE (join tree)

Definition 4 *Let Q be a query, $T = (atoms(Q), E)$ a tree and $A_1, A_2 \in atoms$. If the same variable X occurs in both A_1 and A_2, and A_1 and A_2 are connected in T which means that variable X occurs in each atom in the path between A_1 and A_2, then a tree T is called join tree.*

Example 10 *Consider the following query:*
$Q\ :\ ans \longleftarrow a(X,Y) \wedge b(X,Z) \wedge c(X,Z,W) \wedge d(X,Y,V,M) \wedge e(X,Y,M) \wedge f(X,V,N) \wedge g(X,Y,V,P)$

A join tree for Q is shown in Figure 2.4

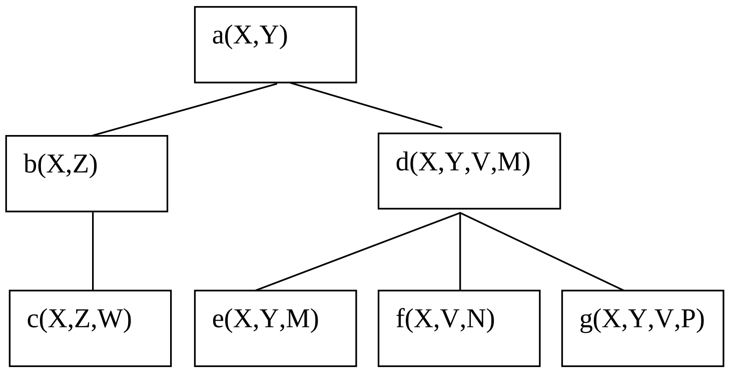

Figure 2.4: A join tree for the acyclic query Q in Example 10

In [6], Gottlob proved that ABCQ and JTREE are both complete for LOGCFL where LOGCFL is the class of problems that are logspace-reducible to a context free Language and highly parallelizable.

There exists a close relationship between Constraint Satisfaction Problems and databases. Where for a given CSP $P = (X, \Delta, \delta, C, \Sigma, \sigma)$, the variables of X can be interpreted as attributes, the domains Δ as the domains of these attributes and a labelling of subsets $Y \subseteq X$ of variables is a tuple of other relation schemes with a set of attributes Y [15].

Definition 5 *[21] Let S be any ordered set of r variables and let $C(S)$ be a constraint on S. For any ordered subset $S' \subseteq S$, let $(i_1, i_2, ..., i_k)$ be the indices of the elements of S' in S. Define the projection of $C(S)$ onto S', denoted $\pi_{S'}(C(S))$, as follows*

$$\pi_{S'}(C(S)) = \{(x_{i_1}, x_{i_2}, ..., x_{i_k}) \mid \exists (x_1, x_2, ..., x_r) \in C(S)\}$$

Definition 6 *[21] For any constraints $C(S_1)$ and $C(S_2)$, the join of $C(S_1)$ and $C(S_2)$, denoted $C(S_1) \bowtie C(S_2)$ is the constraint on $S_1 \cup S_2$ containing all tuples t such that $\pi_{S_1}(\{t\}) \subseteq C(S_1)$ and $\pi_{S_2}(\{t\}) \subseteq C(S_2)$.*

From these definitions we can say that the set of all solutions to CSP is equal to the join of the relation instances corresponding to the constraints denoted as $Sol(P) = C(S_1) \bowtie C(S_2) \bowtie ...C(S_n)$. The constraints of CSP are supersets of the projections of the set of all solutions.

We say a set of constraints is a set of *minimal* constraints if all constraints are equal to respective projections of the set of all solutions [15]. With other words, for each constraint, each member of that constraint can be extended to a solution to the full problem.

Example 11 *Reconsider a constraint satisfaction problem P of Example 6.*
The projection of $Sol(P)$ onto S_1, the scope of C_1, is shown in Table 1.

$R_{\tilde{C}_1}$	
O	R
4	8
5	0
6	2
7	4
8	6

Table 1: The projection of $Sol(P)$ onto S_1

This projection is not equal to C_1, therefore the set of constraints C is not a set of minimal constraints. But a new constraint satisfaction problem $\widetilde{P} = (X, \Delta, \delta, \widetilde{C}, \Sigma, \widetilde{\sigma})$ with $\widetilde{C} = \{\widetilde{C}_1, \widetilde{C}_2, \widetilde{C}_3, \widetilde{C}_4, \widetilde{C}_5\}$ which is equivalent to P and where the scope of $\widetilde{C}_i$ is equal to the scope of C_i and the value of $\widetilde{C}_i$ is equal to the corresponding projection of $Sol(P)$, for all $i = 1, ..., 5$ has a set of minimal constraints.

These minimal constraints are shown in Table 2 as relational database.

$R_{\widetilde{C}_1}$		$R_{\widetilde{C}_2}$		$R_{\widetilde{C}_3}$			$R_{\widetilde{C}_4}$			$R_{\widetilde{C}_5}$			
O	R	W	U	T	F	O	T	W	O	F	O	U	R
4	8	3	6	7	1	4	7	3	4	1	4	6	8
5	0	6	3	7	1	5	7	6	5	1	5	3	0
6	2	3	7	8	1	6	8	3	6	1	6	7	2
7	4	4	9	8	1	7	8	4	6	1	6	9	2
8	6	2	5	9	1	8	8	6	7	1	7	3	4
							9	2	8	1	8	5	6
							9	3	8	1	8	7	6

Table 2: The constraints of $\widetilde{P}$ (Example 10) as a relational database

If the set of relation instances is consistent then the corresponding constraints are minimal. This close relationship between minimality of a set of constraints and the consistency can be found at [15].

2.3 Homomorphism Problem

As we denoted at the beginning of this chapter, the Constraint Satisfaction Problem instances can be viewed as a pair of relational structures whose mappings are nothing else as homomorphism between those relational structures which at same time represented the solutions of the CSP problem [20].

In order to describe Constraint Satisfaction Problems in algebraic terms and to define the homomorphism problem, we first give the definitions of some standard algebraic notations.

Definition 7 *[20] A 'relational structure' is a tuple , $\langle V, E_1, E_2, ..., E_k \rangle$, consisting of a non-empty set, V, called the "universe" of the relational structure, and a list, $E_1, E_2, ..., E_k$, of relations over V.*

Definition 8 *[20] The 'rank function' of a relational structure $\langle V, E_1, E_2, ..., E_k \rangle$, is a function ρ from $\{1, 2, ..., k\}$ to the set of non-negative integers, such that for all $i \in \{1, 2, ..., k\}$, $\rho(i)$ is the arity of E_i.*

A relational structure Σ is 'similar' to a relational structure Σ' if they have identical rank functions.

Definition 9 *[20] Let* $\Sigma = \langle V, E_1, E_2, ..., E_k\rangle$ *and* $\Sigma' = \langle V', E'_1, E'_2, ..., E'_k\rangle$ *be two similar relational structures, and let* ρ *be their common rank function. A 'homomorphism' from* Σ *to* Σ' *is a function* $h : V \rightarrow V'$ *such that, for all* $i \in \{1, 2, ..., k\}$*,*

$\langle v_1, v_2, ..., v_{\rho(i)}\rangle \in E_i \Longrightarrow \langle h(v_1), h(v_2), ..., h(v_{\rho(i)})\rangle \in E'_i$.

Definition 10 *[8] (The Homomorphism Problem HOM). Given two finite structures* Σ *and* Σ'*, decide whether there exists a homomorphism from* Σ *to* Σ'*.*

We denote such an instance of HOM *by* $HOM(\Sigma, \Sigma')$*.*

Proposition 1 *[20] For any constraint satisfaction problem instance* $P = \langle X, \Delta, C\rangle$ *with*

$C = \{\langle s_1, R_1\rangle, \langle s_2, R_2\rangle, ..., \langle s_q, R_q\rangle\}$*, the set of solutions to P equals* $HOM\langle\Sigma, \Sigma'\rangle$*, where* $\Sigma = \langle X, \{s_1\}, \{s_2\}, ..., \{s_q\}\rangle$ *and* $\Sigma' = \langle\Delta, R_1, R_2, ..., R_q\rangle$*.*

Example 12 *We represent Example 4/5 (the map colouring problem) as HOM problem. The solution of the question if the map is three colourable is actually the solving of* $HOM\langle\Sigma, \Sigma'\rangle$ *problem for relational structures* Σ *and* Σ' *where*

$\Sigma = \langle X, \{s_1\}, \{s_2\}, ..., \{s_{10}\}\rangle$

$\Sigma' = \langle\Delta, R_1, R_2, ..., R_{10}\rangle$

where

$X = \{WA, NT, Q, NSW, V, SA, T\}$

$s_1 = \{WA, NT\}, s_2 = \{WA, SA\}, s_3 = \{NT, SA\}, s_4 = \{NT, Q\}, s_5 = \{Q, SA\},$

$s_6 = \{Q, NSW\}, s_7 = \{SA, NSW\}, s_8 = \{SA, V\}, s_9 = \{NSW, V\}, s_{10} = \{T\}$

$R_1 = \{\{red, green\}, \{red, blue\}, \{green, red\}, \{green, blue\}, \{blue, red\}, \{blue, green\}\}$

$R_2 \equiv R_3 \equiv ... \equiv R_9 \equiv R_1$

$R_{10} = \{red, green, blue\}$

In Figure 2.5 we have represented a solution as a homomorphism between two graphs G and K, which corresponds to a three colouring of G.

In [23, 8] was observed that HOM is equivalent to Constraint Satisfaction Problem. What more, HOM, CSP and BCQ are the same and are NP-complete.

2.4 Hypergraphs

In general humans tends to represent the real problems occourred in theirs life as simple as possible and easily understandable. One such way is of course a graphic representing of problems. Therefore it is attempted constraint satisfaction problems to represent graphically. The graphic structure of the constraint satisfaction problems can be a *Tree*,

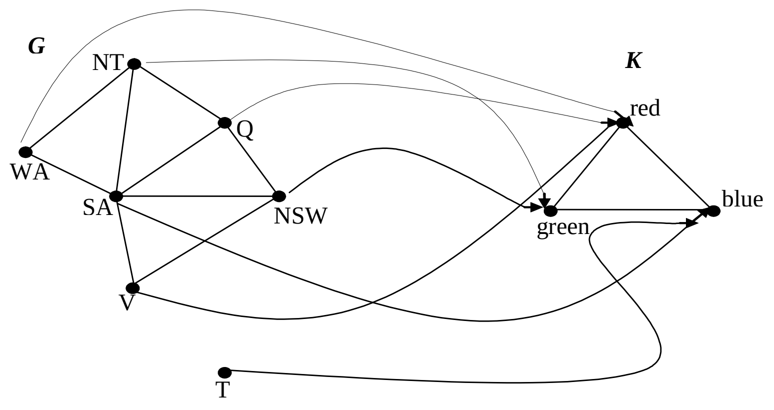

Figure 2.5: A solution to a graph colorability problem instance

Graph or *Hypergraph.* As is well known for CSP with a tree-structure exists very efficient algorithm to solve it, therefore will be tried to convert the CSP of arbitrary structure into pseudo CSP of tree structure. This will be described in detail in the next chapter.

The graphs simplify the problems but not always are the best solutions. There are a large number of problems which structure is better represented by hypergraphs than by graphs. For example several NP complete problems will be tractable if restricted to instances with acyclic hypergraphs [5] .

Definition 11 *[19] A hypergraph H is an ordered pair (V, E) where V is a finite set of vertices and E is a set of edges, each of which is a subset of V.*

Further we give some other basic definitions [19]

Special cases of hypergraphs are the undirected graphs where each edge contains exactly two vertices. From the formal definition of satisfaction problem $P = (X, \Delta, \delta, C, \Sigma, \sigma)$ we can derive the hypergraph $H_p = \{X, \Sigma\}$ which is associated to P the edges of which are the scopes of the constraints and the vertices are the finite set of variables.

For the analogy between CSP and their associated hypergraphs we can say that two vertices are in the same hypergraphs if they occur in the same constraint. If one vertex belongs exactly to one edge then is called *isolated.* If one edge contains excactly one vertex then is called *singleton* [11].

Example 13 *Reconsider the constraint satisfaction problem P of Example 6 (Cryptography problem). The hypergraph $H_p = \{X, \Sigma\}$ associated to P is shown in Figure 2.6*

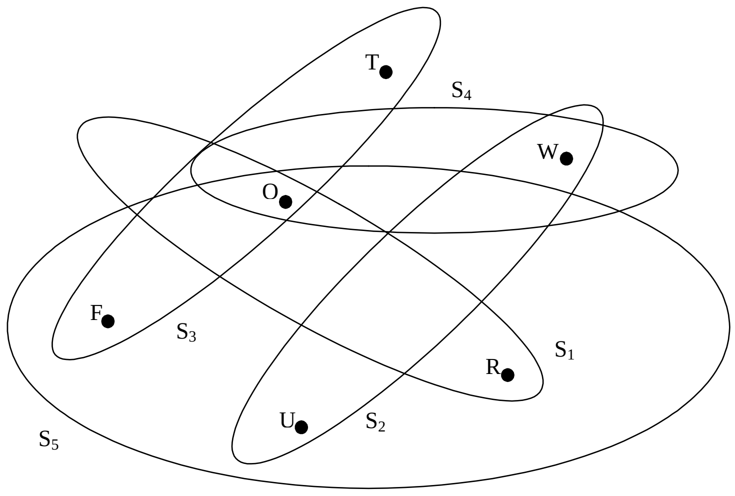

Figure 2.6: The hypergraph associated with P of Example 5 (Cryptography problem)

Let (V, E)be a hypergraph H_p let $H \subseteq E$, let $F \subseteq E - H$, and let be $f_1, f_2 \in F$ any two edges in F. If there exist a sequence of edges $e_1, e_2, ..., e_n \in F$ such that:

1. $e_1 = f_1$
2. for $i = 1, ..., n - 1$ $\quad e_i \cap e_{i+1}$ is not contained in $\cup H$
3. $e_n = f_2$

than F is called *connected with respect* to H [15]. The maximal connected subsets of $E - H$ with respect to H are called the connected components of $E - H$ with respect to H. If H is empty than we speak about connected subsets and connected components of E [15].

Let illustrate that with an example:

Example 14 *Let be $H_p = (V, E)$ a hypergraph from Figure 2.7.*

Then $E = \{H_1, H_2, H_3, H_4, F_0, F_1, F_2, F_3, F_4, F_5\}$. If $H = \{H_1, H_2, H_3, H_4\} \Rightarrow H \subseteq E$, and if $F = \{F_1, F_2, F_3, F_4, F_5\} \Rightarrow F \subseteq E - H$.

We say F is connected with respect to$H$, because $\exists$ a sequence F_1, F_2, F_3, F_4, such that $F_1 \cap F_2$ and $F_2 \cap F_3$ and $F_3 \cap F_4$ are not contained in $\cup H$. But for $F' = \{F_0, F_1, F_2, F_3, F_4, F_5\}$ we say is a connected component of $E - H$.

The connected components are very helpful in process of finding of independent subproblems. The advantage of these subproblems is that they can be solved totally

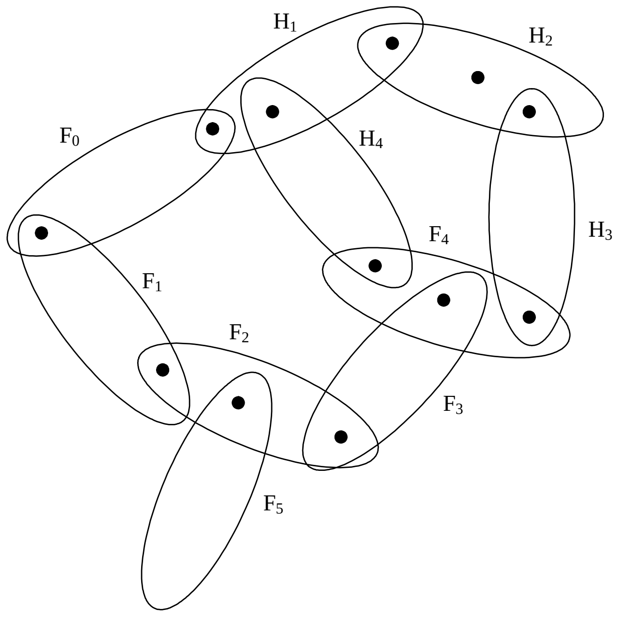

Figure 2.7: The hypergraph H_p of Example 14

independently. We say a hypergraph $H = \{V, E\}$ is *reduced* if none of its edges is properly contained in any other [15]. It will be obtained by removing of each edge that is properly contained in another edge.

Given a hypergraph $H = \{V, E\}$, the $GYO(H)$ (Graham, You and Özsoyoğlu 1979)[6] is the hypergraph obtained from H by repeatedly applying the following rules:

1. Remove the hyperedges that are empty or contained in other hyperedhes;
2. Remove vertices that appear in at most one hyperedge

2.4.1 Acyclicity of Hypergraphs

Acyclicity of Hypergraphs is very important property due to the fact that CSP Problems with acyclic instances are tractable [6]. In the literature exists different degrees

of acyclicity[15, 26] , and in this chapter we will focus over main acyclcities and give an overview of some of these. First we give some theoretical definitions about acyclicity of hypergraphs which can be found at [15].

Let (V, E) be a connected and reduced hypergraph H, and let $F \subseteq E$ be a connected set of edges. Further let f and g be in F, and $q = f \cap g$. We say that a pair $\{f, g\}$ is an *articulation pair* of F and q is *articulation set* of F if after moving q from all edges of F the set of remainder edges is not connected.[15]

We say F is closed if for every edge $e \in E$, $\exists e' \in F$, so that $e \cap (\cup F) \subseteq e'$.

And finally a connected and reduced hypergraph is acyclic if every closed connected set of edges consisting of at least two elements has an articulation set.

Graham gives another definition of acyclicity. A hypergraph H is acyclic if $GYO(H)$ is an empty hypergraph [6].

In [26] are given and compared different characterization of acyclic hypergraph. Here we give a short overview of some of these.

Definition 12 *[26, 11] A reduced hypergraph is α- acyclic if all its blocks are trivial (contains less then two members), otherwise it is α- cyclic.*

Definition 13 *[26, 11] A hypergraph is β - acyclic if all its subhypergraphs are α - acyclic.*

In Figure 2.8. are given different degrees of cyclicity.

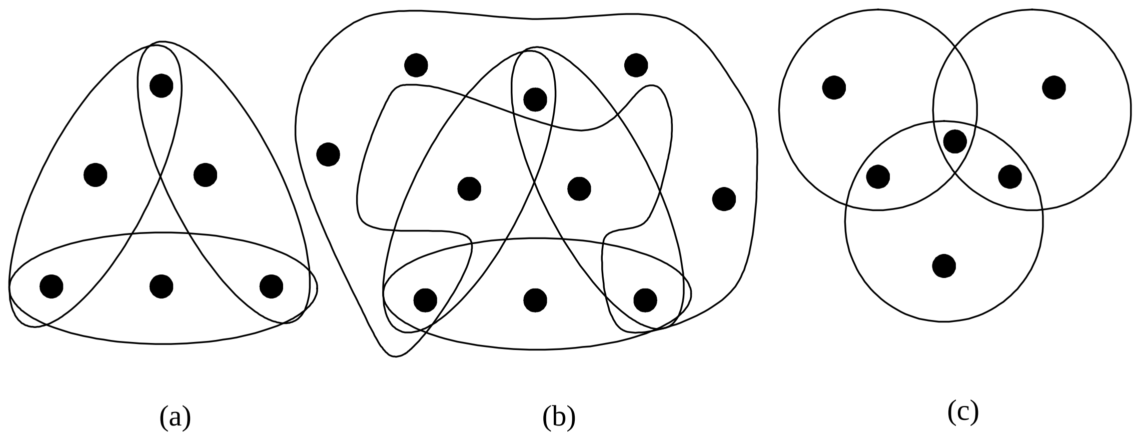

Figure 2.8: Various degrees of cyclicity: (a) $\alpha - acyclic$ hypergraph, (b) $\alpha - acyclic$ but $\beta - cyclic$ and (c) $\beta - acyclic$ [11]

An easy but not so efficient algorithm $(O(|HG|^2)$ of checking of acyclicity of hypergraph$H$ is checking if $GYO(H)$ is empty. There are also some other algorithms, which are linear. Acyclicity of hypergraphs is into a narrow relationship with join trees. We can say:

A hypergraph H is acyclic if it has a join tree [7]. There exists a linear-time algorithm for computing a join tree [7] , therefore acyclicity of hypergraph is efficiently recognizable.

2.4.2 Primal and dual graphs

We differentiate between primal and dual graph of hypergraph. We say that a graph $G(H)$ is a *primal (Geifman)* graph of hypergraph $H = (V, E)$ if for any two vertices of the same hyperedge of hypergraph H, exists an edge in graph $G(H)$ which connects these two vertices. [22]

Formally, $G(H) = (V, E')$, $E' = \{\{x, y\}|x, y \in V, \exists e \in E : \{x, y\} \subseteq E\}$.

Further we say that a graph $D(H)$ is a *dual* graph of hypergraph H if his vertices are the edges of hypergraph H and for any two connected vertices in D(H), exists at least one vertex of H in common.[22]

Formally,$D(H) = (V', E")$, where $V' = E$ and $E" = \{\{x, y\}|x, y \in E, \exists v \in V : v \in x \wedge v \in y\}$.

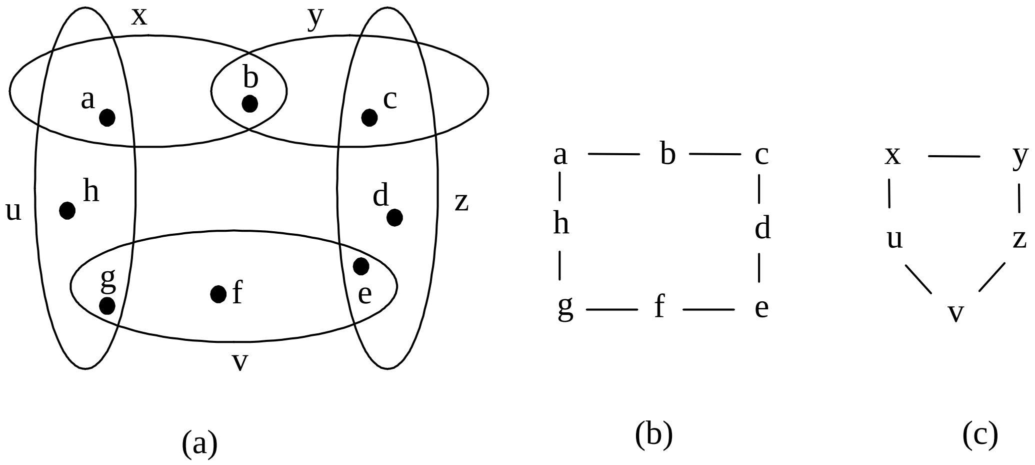

Figure 2.9: A simple hypergraph (a), it's primal (b) and dual (c) graph.

Although converting of hypergraph into primal and dual graphs have some advantages during decomposition using biconected components (see next chapter), they have also disadvantages due to the fact that no unique reversible way exists, i.e. some of information can disappear (see Figure.2.9).

2.5 Decomposition Methods

As previously mentioned the CSP problems are in general NP-hard problems [4], and thus intractable. However there are some classes of CSP problems which are identified as tractable [4]. In order to find such tractable CSP classes different methods are developed, among them a method based on restricting of structure [4]. As we know the structure of CSP problems is much better represented by hypergraphs [5], and the CSP with acyclic constraint hypergraphs are polynomially solvable [4] therefore the decomposition methods attempt to reduce the cyclicity of hypergraph, and what more, if possible to generate an acyclic hypergraph.

Gottlob et al.[4] compares some main decomposition methods. All these decomposition methods defines a concept of width which can be interpreted as measure of cyclicity, such that for each fixed width k, all $CSPs$ of width bounded by k are solvable in polynomial time [4].

In general CSP problems that have an importance in practice are not acyclic CSPs, but they can be transformed into acyclic by applying of some methods which some edges ore part of them put into new one, which make that the cycles disappear and the problems become simpler. The advantage of these methods is that they create new subproblems which then can be solved separately.

In this master's thesis I will give an overview of some of decomposition methods described at [4] and comparison results found by Gottlob . Fore more details, see [4].

2.5.1 Biconected Components (BICOMP)

This decomposition method is first published by Freuder and operates on the primal graph and not in hypergraph directly. First we give some basic definitions

Let $G = (V, E)$ be a graph, and let be $p \in V$ a vertex. We say p is a *separating vertex* of G, if removing him, the graph will be separated into some connected components. If a graph contains no separating vertex then is called *biconnected*.[4]

Definition 14 *[11]The biconnected components of a graph G are the maximal biconnected subgraphs of G.*

A labelled tree $< T, \chi >$ where χ is a bijective function that associates to each vertex of the tree either a biconnected component of G, or a singleton containing a separating vertex for G, is called *BICOMP-decomposition* of G [4].

The *BICOMP* decomposition of hypergraph H is the *BICOMP* decomposition of its primal graph. The biconnected width of a hypergraph H, *BICOMP-width (H)* is the maximum number of vertices over the biconnected components of the primal graph of H.[4]

Example 15 *Figure. 2.10 (a) shows a hypergraph H, and 2.10 (b) its primal graph, and 2.11 the decomposition tree consisting of biconnected components.*

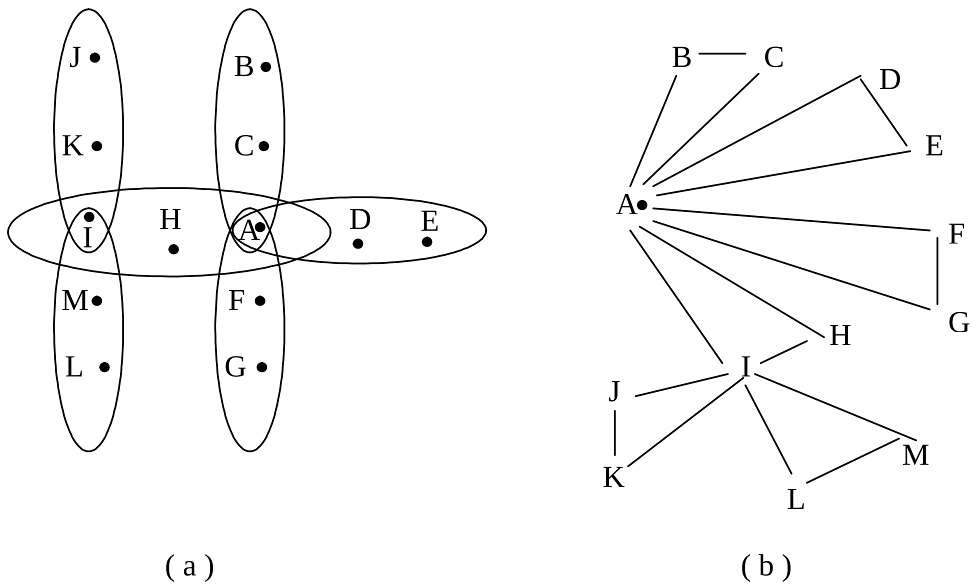

Figure 2.10: The Hypergraph H (a) and its primal graph (b)

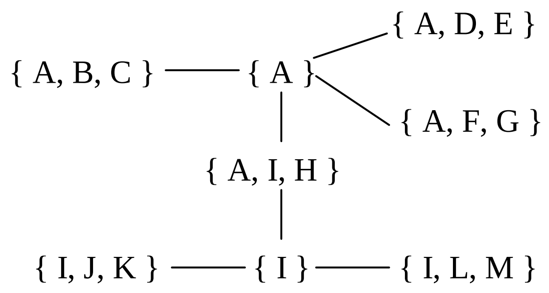

Figure 2.11: The BICOMP-decomposition of the hypergraph H in Figure2.10.

From Example 16 we see that the vertices I and A are the separating vertices of the primal graph (see Figure2.10 (b)). The maximal number of vertices over biconnected components of the graph G is 3, and thus BICOMP-width (H) = 3.

2.5.2 Tree clustering (TCLUSTER)

This method is proposed by Dechter and Pearl, which transforms the primal graph into chordal graph.

A graph is *chordal* if every cycle of length at least 4 has a chord, i.e. an edge joining two non-consecutive vertices along the cycle [15].

Let $G = (V, E)$ be a primal graph, and G' a chordal graph of any CSP instance. The *TCLUSTER-decomposition* is the acyclic hypergraph $H(G')$ having the same set of vertices as G' and the maximal cliques of G' as its hyperedges[4].

A primal graph is *conformal* if each of its maximal cliques corresponds to an edge in the original hypergraph [15]. Due to the fact that hypergraph is acyclic if its primal graph is both chordal and conformal, Dechter and Pearl show that the equivalent CSP is acyclic [15].

They also show that for constraint satisfaction problem $P = (X, \Delta, \delta, C, \Sigma, \sigma)$ complexity-time of finding of solution with this decomposition method is $O(|X^2|) + O(|X|\, k^r r \log k)$ where k is the maximal size of a domain in Δ and r is the number of vertices in the largest maximal clique of the triangulated primal graph [15]. Therefore we can see that the complexity of this technique is exponential in r.

Example 16 *[4]Consider the hypergraph H shown in figure 2.12 (a) . 2.12 (b) shows its primal graph.*

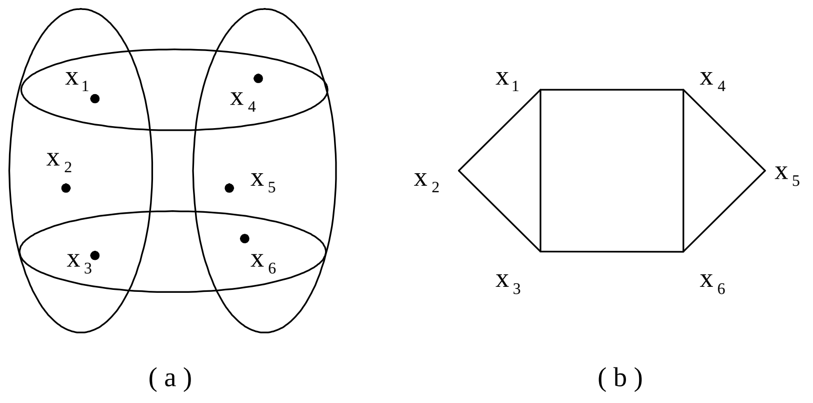

Figure 2.12: The Hypergraph H (a) and its primal graph (b)

This graph can be triangulated as shown in figure 2.13 (a) . If for each maximal clique we associate a hyperedge, we get the acyclic hypergraph shown in figure 2.13 (b) . This hypergraph is at the same time a *TCLUSTER* decomposition of hypergraph H.

2.5.3 Treewidth

This method operates also on the primal graph of hypergraph. In [4] can be find the following definition of *tree decomposition* and *treewidth.*

A *tree decomposition* of a graph $G = (V, E)$ is a pair $\langle T, \chi \rangle$, where $T = (N, F)$ is a tree, and χ is a labelling function associating to each vertex $p \in N$ a set of vertices $\chi(p) \subseteq V$, such that following conditions are satisfied:

1. for each vertex b of G, there exists $p \in N$ such that $b \in \chi(p)$;
2. for each edge $\{b, d\} \in E,$ there exists $p \in N$ such that $\{b, d\} \subseteq \chi(p)$;
3. for each vertex b of G, the set $\{p \in N \mid b \in \chi(p)\}$ induces a (connected) subtree of T .

According to this definition each vertex must occur in the tree, each edge of graph must be covered by some vertices in tree, and finally connectedness condition must be valid.

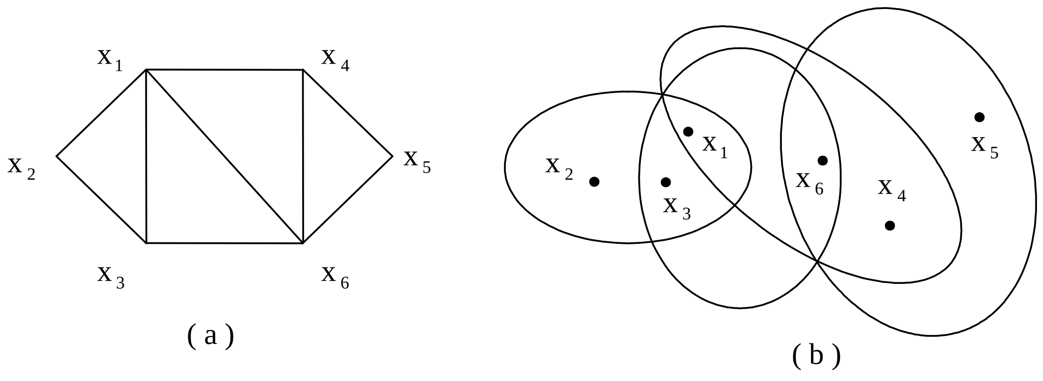

Figure 2.13: A triangulation (a) of the primal graph in Figure 2.12 and itsTCLUSTER-decomposition (b)

The *width* of the tree decomposition $\langle T, \chi\rangle$ is $\max_{p\in N} |\chi(p) - 1|$. The *treewidth* of a graph is the minimum width over all its tree decomposition. If a hypergraph is acyclic, his treewidth has size 1, otherwise equally treewidth of his primal graph [4] . TREEWIDTH and TCLUSTER are in fact two equivalent methods [4].

Example 17 *Consider the hypergraph H in Figure 2.14 (a). Figure 2.14 (b) shows a tree decomposition of H*

From Figure 2.14 (b) we see that tree decomposition of H is 2. As only hypergraphs with acyclic primal graph have treewidth 1 follows that treewidth of Hypergraph is also 2.

2.5.4 Hinge Decomposition

The hinge decomposition is described by Gyssen et al. [15]. Further in [15, 4] can be found the following definitions.

We gave the definitions of *connected components* already in section Hypergraphs. Further let H be a hypergraph, $H \subseteq edges(H)$, and $C_1, ..., C_n$ be the connected components of H with respect *H*. Then H is a *hinge* if, for $i = 1, ..., n$, there exists an edge $h_i \in H$ such that $var(edges(C_i) \cap var(H)) \subseteq h_i$. A hinge is *minimal* if does not contain any other hinge.

A *hinge decomposition* og H is a tree T such that satisfies the following conditions

1. the vertices of T are minimal hinges of H .

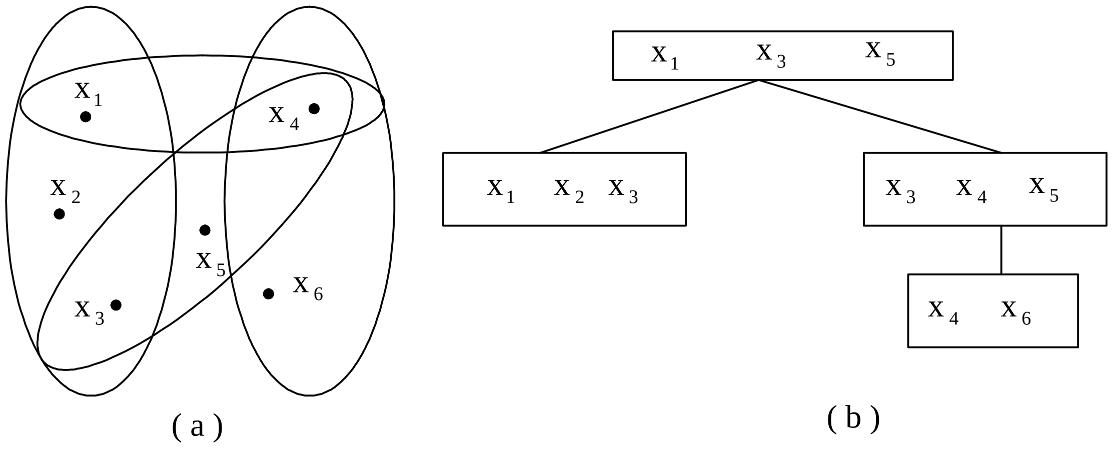

Figure 2.14: A given hypergraph H (a), and a tree decomposition of hypergraph H (b).

2. each edge in $edges(H)$ is contained in at least one vertex of T .
3. two adjecent vertices of T share precisely one edge of H. Moreover, their shared vertices are exactely contained in this edge.
4. the vertices of H shared by two vertices of T are entirely contained within each vertex on their connecting path.

The size of the laregest vertex of T (that is the number of edges contained in vertex) is the Hinge width of H .

Example 18 *Consider the hypergraph in Figure 2.15. The minimal hinges of H are* $H_1 = \{S_1, S_4, S_3, S_2\}$,$H_2 = \{S_2, S_6, S_7, S_5\}$,$H_3 = \{S_5, S_8\}$,$H_4 = \{S_5, S_9\}$. *See Figure 2.16 .*

The cardinality of largest minimal hinges is 4 (hinges H_1 and H_2), therefore the HINGE width of H is 4.

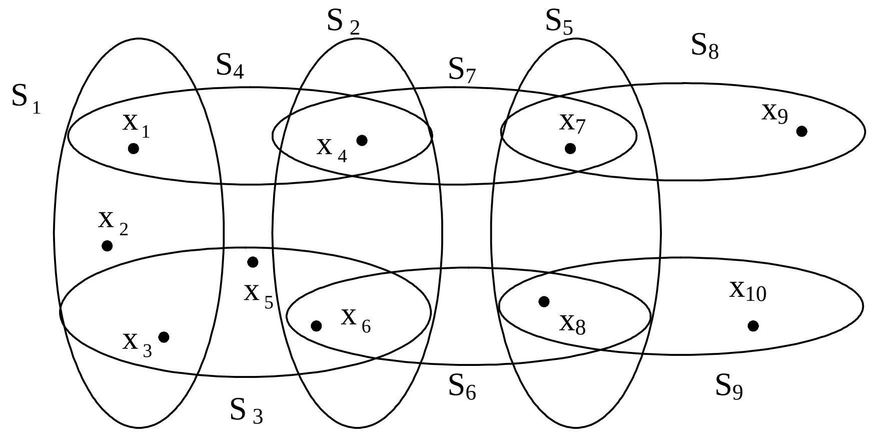

Figure 2.15: The Hypergraph H

2.5.5 Hypertree Decomposition

Another decomposition method is developed by Gottlob et al (see [5]). This method, according to the literature is the best decomposition method, which for a given constant k checks in the polynomial time if the hypergraph has a hypertree width k. In positive case this method can efficient to compute a hypertree decomposition, which actually is LOGCFL complete.

First we give some basic definitions which can be found at [4, 5].

A hypertree for a hypergraph H is a triple $\langle T, \chi, \lambda \rangle$, where $T = (N, E)$ is a rooted tree, and λ and χ are labeling functions where $\lambda(p) \subseteq edges(H)$ and $\chi(p) \subseteq var(H)$.

$vertices(T)$ denotes the set of vertices N of T . Further for any $p \in N$, T_p denotes the subtree of T rooted at p .

Definition 15 *[4, 5] A hypertree decomposition of a hypergraph H is a hypertree $HD = \langle T, \chi, \lambda \rangle$ for H which satisfies the following conditions:*

1. for each edge $h \in edges(H)$, there exists $p \in vertices(T)$ such that $var(h) \subseteq \chi(p)$ (we say p *covers* h);
2. for each variable $Y \in var(H)$, the set $\{p \in vertices(T) \mid Y \in \chi(p)\}$ induces a (connected) subtree of T ;
3. for each $p \in vertices(T), \chi(p) \subseteq var(\lambda(p))$;

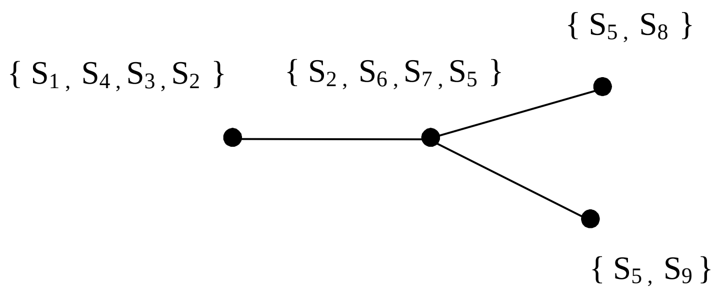
{ S_1, S_4, S_3, S_2 }
{ S_2, S_6, S_7, S_5 }
{ S_5, S_8 }
{ S_5, S_9 }

Figure 2.16: Hinge decomposition of Hypergraph in Figure 2.15

4. for each $p \in vertices(T), var(\lambda(p)) \cap \chi(T_p) \subseteq \chi(p)$.

The first condition meant that each constraint will be present in the tree, at least as covert hyperedge. The second condition (*connectednes condition)* makes possible application of Yannakakis algorithm on hypertree in order to evaluate it. The third condition says that each variable that occurs in a vertex occurs also in one or several edges which are part of this hypertree vertex. Fourth condition is only to ensure the efficient computation of the hypertree decomposition.

If the fourth condition in definiton of hypertree decompostions is ignored, the corrosponding decomposition is called *generalized hypertree decompostion*.

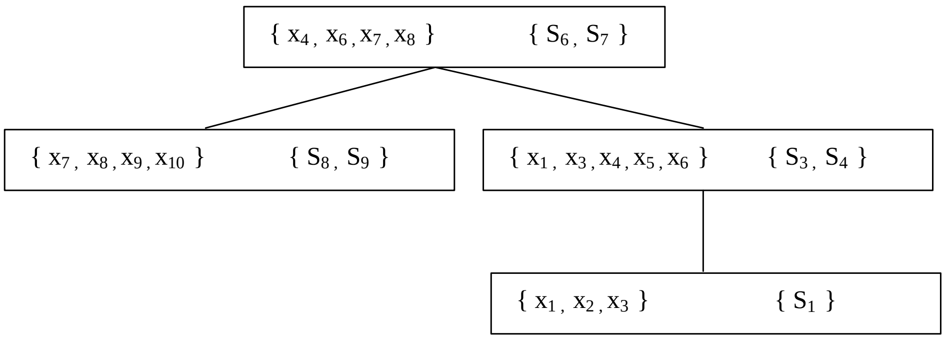
{ x_4, x_6, x_7, x_8 } { S_6, S_7 }
{ x_7, x_8, x_9, x_{10} } { S_8, S_9 }
{ x_1, x_3, x_4, x_5, x_6 } { S_3, S_4 }
{ x_1, x_2, x_3 } { S_1 }

Figure 2.17: A hypertree decomposition of width 2 of hypergraph H in example 19

Further, an edge h is *strongely covered* in hypertree decomposition if there exists a vertex p such that $var(h) \subseteq \chi(p)$ and $h \in \lambda(p)$.

Hypertree decomposition of hypergraph H is a *complete decomposition* of H if every edge of H is strongely covered in hypertree decomposition.

The *width* of hypertree decomposition $\langle T, \chi, \lambda \rangle$ is $\max_{p \in vertices(T)} |\lambda(p)|$. The *hypertree width* $hw(H)$ of H is the minimum width over all its hypertree decompositions. The acyclic hypergraphs have the hypertree width 1.

Example 19 *Consider the hypergraph in Figure 2.15. Figure 2.17 shows a hypertree decomposition of the width 2 of hyprgraph in Example 19. Figure 2.18 shows complete hypertree decomposition.*

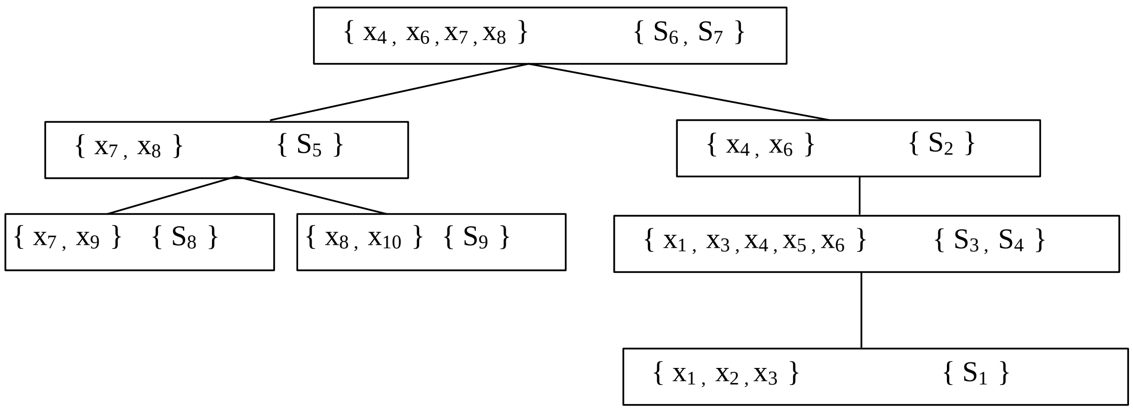

Figure 2.18: A complete hypertree decomposition of width 2 of hypergraph H in example 19

3 Heuristic Algorithms

How efficiently we can solve constraint satisfaction problems, depends in particular from the grade of cyclicity of the problem instances. The smaller grade of cyclicity implies a better and faster solution. However the problem is how to eliminate these cycles. As it has been described in the former chapter, in order to minimize or make the cycles disappears, the different decomposition methods are developed. The newest method developed by Gottlob et al [4][5][6], Hypertree Decomposition, one of most powerfull decomposition method, ensures that for a given constant *k,* it can be decided in polynomial time, whether a given hypergraph has a hypertree decomposition of width equal to or smaller then *k*. The problem is that *k* appears in the exponent of runtime, thus for large problems the algorithm becomes slower and requires lot of memory space and therefore becomes quickly intractable. The implementations of this exact algorithm, *opt-k-decomp,* are already done by the DBAI research group [11] and by the research group of the University of Calabria. Because of these runtime behaviour different heuristic algorithms are developed in sense to make hypertree decomposition useful even for large problems where opt-k-decomp becomes intractable.

In the literature we can find some interesting heuristic approaches which give in partial very good results. They are, however, strongly influenced by the structure of the hypergraphs , and only in few cases achieve a minimal width, so it is very desirable to find approaches, which results deviate minimally from optimality.

In this chapter we will describe some of major heuristic approaches, which are partly developed by DBAI research group (see [2]). Some of these heuristics are based on vertex ordering and generate a tree decomposition of a primal or dual graph which then can be extended to hypertree decomposition. There are also heuristics which are based on hypergraph partitioning.

In this diploma thesis we propose two methods for generating of generalized hypertree decomposition based on hypergraph partitioning. These methods are described in details in section (3.4).

3.1 Bucket Elimination [16]

This heuristic approach was developed primarily to solve CSP problems. The method was applied and modified by McMahan [16]. In this diploma thesis we will give a short description of this method. This algorithm is based on ordering of variables of any CSP problem. For any ordering of n variables of hypergraph of the CSP $x_1, x_2, ..., x_n$ the method creates n buckets, one for each variable. Afterwards for each hyperedge of hypergraph, the method puts the hyperedge variables into the bucket of maximum variable. The maximum variable is the variable with maximal position in already given ordering of variables. The next step is an iteration on all buckets and their elimination. In every bucket i the algorithm computes joins between all relations containing the variable x_i and finally projects out x_i . In case of an empty result the Constraint Satisfaction Problem is also empty, otherweise let j be the largest index in ordering, smaller then i , such that x_j is also a variable of the projected result. Then the results will be moved to bucket j. Finally, the BE creates a tree decomposition. In order to generate a generalized hypertree decomposition from the tree decomposition, McMahan inserts the hyperedges in each tree decomposition vertex, if tree vertex contains its variables. To insert the minimal number of hyperedges in tree vertex, he used different set covering heuristics [16].

The Bucket Elimination heuristic has the property that for an optimal order, BE will produce an optimal width of tree decomposition. Since the choosing of optimal BE order is NP-hard he chooses the order heuristically. For a detailed description of the algorithm see [16].

Although Bucket Elimination has a very simple concept , it gives very good results, in particular the best evaluated results for some benchmark problems.

3.2 Branch Decomposition [27]

This method works on graphs. In this diploma thesis we will give a short description of the method [27]. The first step is building of a star from graph, where each edge of the graph is represented exactly from one edge in the star, additionally labelled with set of vertices. The process of splitting of vertices (see Figure 3.1) of the star where every two resulting vertices are connected by the new edge is called branch decomposition of the graph [27]. This heuristic, used by Sammer (see [27]), first constructs a branch-decomposition for a given hypergraph, then transforms it into tree-decomposition and finally into hypertree-decomposition applying *set covering heuristic*.

Note that for a branch decomposition of the hypergraph of a width k , it is possible to construct a tree-decomposition of width at most $3k/2$ [27]

For a detailed description of the algorithm see [27].

3.3 Hypertree - Decomposition through Hypergraph Partitioning

There are a number of heuristic approaches which are based on hypergraph partitioning. The idea of Hypergraph partitioning is to separate a hypergraph (set of vertices of hypergraph) in two ore more subhypergraphs. At the same time they try to satisfy the given certain rules or definitions, which tends to minimize the number of common

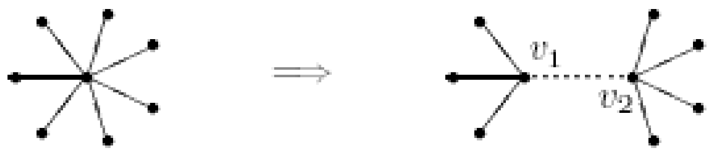

Figure 3.1: Splitting a vertex [27]

hyperedges between clusters (see Figure.3.2). In the literature exists several heuristic approaches which produce hypertree decomposition based on hypergraph partitioning.

One such heuristic, which uses a recursive partitioning where every new cluster is smaller that original one, is developed by Korimort [14]

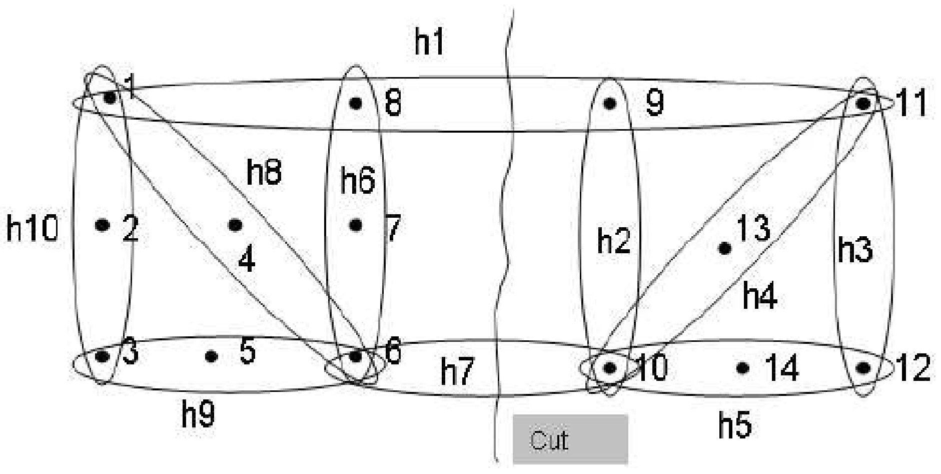

Figure 3.2: Example of partitioning of hypergraph in two parts [2]

3.3.1 Recursive partitioning (Korimort [14])

I will give only brief description of this method. This method was developed by Korimort [14]. In general this heuristic approach is composed by the following steps:

1. Computing of connected components of hypergraph

2. Decomposition of each component
3. Building of the hypertree

The main challenge during the decomposition routine is how to compute such separators (set of hyperedges), such makes a problem smaller and simpler. In order to find a *good separator* Korimort uses different heuristics. I will give briefly some of them.

- "*Dual graph separators*"

First the hypergraph is transformed into weighted dual graph and then the vertex connectivity of the dual graph is determined . Korimort shows that the vertex connectivity of dual graph equals the size of minimal separator which splits the problem into more than one subproblems [14].

- "*Computing the vertex connectivity of the graph*"

This method is based on computing a series of network flows. Indeed, it is computing of the vertex connectivity of primal graph. For more details please see [14].

- "*Eat up separators*"

This method try to produce a subproblem which must be smaller then original one. It can be done if for an arbitrary node of hypergraph we find all hyperedges that intersect this node.

- "*Random separators*"

The hyperedges will be chosed incrementally according to already defined probability distribution over all hyperedges [14].

3.3.2 Partitioning with Fiduccia-Mattheyses algorithm, FM [2]

This method is used and implemented by the DBAI research group [2]. We will give a short description of the method which can be found at [2]. Fiduccia-Myttheyeses algorithm is an iterative refinement heuristic. After the hypergraph is splitt into two parts, a number of moves of vertices to the opposite partitions is done. After each move, a moved vertex will be "*locked*" in order to prevent it from being moved back again. During those moves will be tried to avoid imbalanced partitionings. This will be achived if every moving is done according to the selection criterium. This criterium takes care that the size of the cut decreases. Although the next move isn't always the "best" one, the best solution will be memorised and is taken as the initial solution for the next pass [2].

In order to keep the connectedness condition after each partitioning step (new inserted special hyperedges), our research group has implemented four variants of the Fiduccia-Matteyses algorithm. Those variants are dependent on handling of special hyperedges.There are different handling of special hyperedges. These special hyperedges can be treated as normal hyperedges, as a minor variant of normal hyperedges or as abolition of special hyperedges from separators (which can be done by moving of all nodes contained in that special hyperedges).

For more information about implementing of this heuristic algorithm see [2]

3.3.3 Heurisctic algorithm based on Tabu Search [2]

Another heuristic approach implemented by DBAI research group is based on tabu search. For complete background of tabu search please see [2]. We will give here short description about implementing of heuristic.

In the progress of iterations during the searching of solutions, some information about moves and reverse moves of vertices between partitions will be stored. These stored informations will be usefull to avoid multiple searching of solutions and will be kept for the certain number of iterations. In the implementation of tabu search heuristic some modifications have been done. For instance, during each iterations process, the neighbourhood size will be reduced by moving only of vertices which appear in separators [2]. Another modification is the insertion of some probability rules for one randomly selected separator. We can expect that with some probability the nodes of only this separator will be moved.

The fitness criterium used in this implementing is the sum of weights of all hyperedges which connect two partitions.

3.4 Two new heuristic algorithms based on partitioning

The heuristic approaches which are implemented in this diploma thesis are based on the hypergraph partitioning. The main difference between two proposed heuristics is the method of finding of separators. The first implementation is based on HMETIS algorithm [12], according to the literature one of the best partitioning algortithm. This algorithm works direct on hypergraphs. Please note that the HMETIS algorithm is also a part of hypertree decomposition library, which was implemented by DBAI research group (see [2]). The second approach takes into consideration a large number of possible separators and chooses the best one with respect to certain fitness criteria. This algorithm is based on dual graphs. In this section I will give a detailed description of both algorithms.

3.4.1 Heuristic algorithm based on HMETIS partitioning

This approach which we use in order to achieve a good hypergraph partitioning is based on HMETIS algorithms. HMETIS is a software package for partitioning hypergraphs, developed at the University of Michigan. According to the literature, HMETIS is one of the best available packages for hypergraph partitioning [12, 9, 13]. I will give a brief description of the HMETIS partitioning approach . More information about HMETIS heuristic can be found in [12, 9, 13]. In general, the algorithm comprises of three phases. In the *coarsening* phase the group of vertices of hypergraph will be merged together in order to create the single vertices and smaller hypergraph. In this way the size of large hyperedges will be reduced, and it is very helpful because of the fact that FM algorithm is better than other algorithms when refning smaller hyperedges [9]. There are three possibilities to merge the vertices during the coarsening phase: the finding of a maximal set of vertices which have the common hyperedges (edge coarsening), the merging of vertices within the same hyperedge (hyperedge coarsening), and finally the modified hyperedge coarsening which also merges the vertices within hyperedges that have not yet been contracted [9].

After the coarser hypergraph is created, the next phase called the *initial* partitioning phase computes a bisection of those hypergraphs tending a small cut and a specified balanced constraint. The coarser hypergraph has a small number of vertices, usually less than 200 vertices [9], therefore the partitioning time tends to be small. In order to compute the initial partitioning HMETIS uses two different algorithms followed by the Fiduccia-Mattheyses (FM) refinement algorithm [9].. Because the algorithms are randomised, different runs result in different solutions, and the best initial partitioning will be selected for the next phase.

During the *uncoarsening* phase the partitioning will be successively projected to the next level finer hypergraph and a partitioning refinement algorithm will be used to reduce the cut-set in order to improve the quality of partitioning. HMETIS implements a variety of algorithms that are based on the FM algorithm which repeatedly moves vertices between partitions in order to improve the cut [9, 13]. The hMETIS package offers a stand-alone library which provides the HMETIS_PartRecursive() and HMETIS_PartKway() routines. HMETIS_PartRecursive() routine computes a *k-way* partitioning and is based on recursive partitioning of hypergraph in two partitions (multilevel recursive bisection) [9, 13]. HMETIS_PartKway() routine computes also k-way partitioning and is based on recursive partitioning of hypergraph in more than two partitions (multilevel *k-way* partitioning) [13]. We use both routines in order to achieve appropriate partitions, that lead to a hypertree decomposition of small width. The HMETIS package offers the possibility to change different parameters which have an impact on the quality of partitioning. Therefore we make a series of tests with parameters of different values, and we come to the conlusion that the parameters which mostly impact the quality are the number of desired partitions *nparts*, and the imbalance factor between partitions *ubfactor*. For a complete description of parameters see [12]. The test results show that for *nparts* less than 3 the hypertree decomposition was not necessarily better, and usually higher ubfactors lead to smaller hypertree-widths.

3.4.2 New heuristic algorithm based on dual graph

We will give a short description of the heuristic. Generally our heuristic performs the following steps:

1. Find the so-called *star structures* and decomposes it
2. Find a *good* separator *K* (i.e., a set of edges that divides the hypergraph)
3. Divide the hypergraph into subgraphs (subproblems) where the above found separator *K* is at the same time at hypertree *T* a parent node for the new subproblems
4. For each subproblem continue recursively at 2
5. At each hyperedge put back the hypernodes values that were present before the *star structure* decomposition
6. Check the hypertree T for *star connectedness condition* (After disappear of star structures a connectedness condition can be violated)
7. Return the resulting hypertree

Star structures Nearly at all real practical problems are present star structures, i.e. a set of hyperedges (more than two) that have at least one common nodes. There are two reasons why for our heuristic the decomposing of such structures is very significant and desirable.

- A star structures avoids the finding of cycles in a dual graph
- A star structures makes divide of the hypergraphs into subgraphs more difficult

As we see from Figure 3.4 the common element continues to remain in two hyperedges (usually these two hyperedges are the hyperedges that are connected with most other hyperedges of the hypergraph), in our case let be B and C. The most connected hyperedge is called a *basic star hyperedge*. In order to keep further the hyperedges connectivity we insert new temporal variables (X and Y) into the basic star hyperedge B and into all other star hyperedges that don't contain the common element anymore (A and D).This temporally variables ensures further the connectivity between hyperedges

The hypergraph structure after star decomposition has become more simple. Now the hyperedges A, C and D are connected only with basic star hyperedge B but not directly with each other (see Figure 3.5 (b)). The advantage is that now there are less

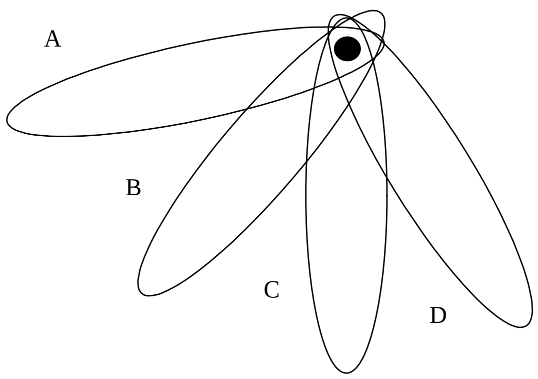

Figure 3.3: A hypergraph star structure

connected hyperedges (i.e. less unnecessary combination during the finding of separator K, which next section describes in detail) and now we can find cycles. Note that although from Figure 3.5 (a) we get a cycle $(A - B - C)$, we see from Figure 3.5 (b) that after star decomposing no such cycle exists anymore. Now they are two differents graphs.

The disadvantage of the star decomposition is the necessity to check the hypertree T for star connectedness condition (see the chapter star connectedness condition)

For the real problems star decomposition is computed quickly (Nasa: 680 Constraints => Star decomposition time = 16 sec).

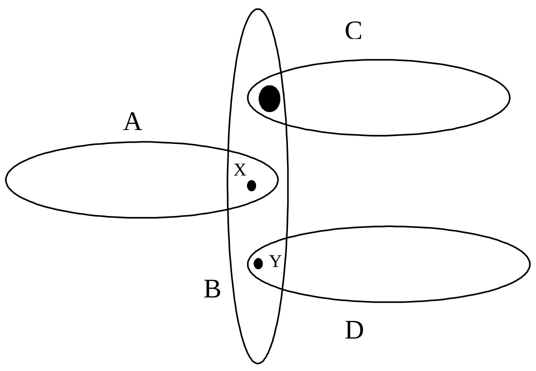

Figure 3.4: Hypergraph structure after star decomposing

Computing of separators The main challenge during the computing of the separator is how to find a *good* separator. A good separator allows a hypertree decomposition of minimal width. Its subgraphs can be further decomposed in such way that the minimal width is not exceeded.

Generally such a (good) separator is very hard to compute because in the worst case we must take in consideration a very large number of possibilities which is almost as one exact decomposition and we risk a very large running time.

In the implementation of the heuristic we perform the following steps:

1. Find a set of separators
2. Evaluate them (i.e. compare them with respect to certain fitness criteria)
3. Chose the one with the best evaluation

In order to restrict the search space for separators , we assume an upper bound for the width of separators. Finding a set of separators consists of a lot of trials.

We differentiate between the finding process of the first separator and the finding process of the others separator. The separators in hypertree decomposition cannot be chosen independently because of connectedness. We begin with converting a hypergraph into a dual graph. Then, beginning from the most connected edge in that dual graph, we find all cyclic (first the smaller cycles) and all acyclic components . After

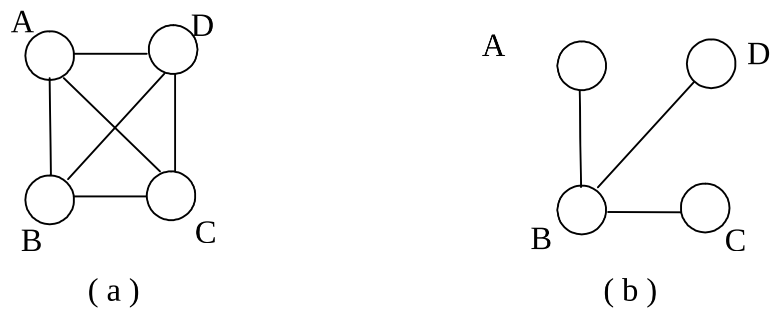

Figure 3.5: The hypergraph star structure in dual form (a), and the hypergraph structure in dual form after star decomposition (b).

that we find all touch-points between those components and thus get a set of separator candidates. The touch-points make it possible to choose such separators whose edges are close together, which increases the probability that these edges can be covered by a small separator in the next step.

Finally, we evaluate the separators. Of course this method gives us no certainty that the best separators can be found, but tells us where the graph can be eventually divided. Sometimes the separator candidates cannot divide the dual graph, so we need to combine them as long as either one such separator is found or the given upper bound is exceeded. However in such cases we can make optimality worse. If however in spite of that, none of separators splites the dual graph and also upper bound is reached, we chose simply the largest found separator.

Let us consider an example to illustrate this process. The hypergraph to be decomposed and its dual graph are given in Figure 3.6 and 3.7 and upper bound for the width of separators is equal to 2. The most connected node with other nodes in the dual graph is D (if no such unique edge exists then choose one randomly). Please note that nodes in dual graph represent the edges of hypergraph. Beginning from D we try to find the first cycle component containing as less as possible nodes. As we see we have two possibilities $D - C - B$ and $D - F - I - H$. Our desired cycle component would be $D - C - B$. We cut out the found cycle component from the dual graph (see Figure 3.8) and recursively find the next cycle components.

From the dual graph in figure 3.8 we find the next cycle component $E - G - I - F$. After we cut out the newly found cycle from the dual graph we finally get two acyclic components A and H (see figure 3.9). We have a set of cycle components $Cyc = \{D - C - B, E - G - I - F\}$and a set of acyclic components $Acyc = \{A, H\}$. The next step is to find the touch-points between those components. As touch-points

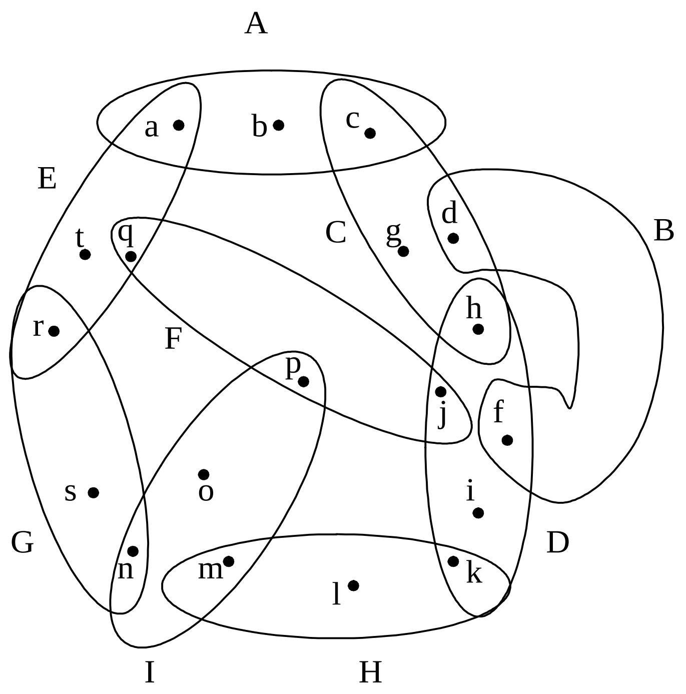

Figure 3.6: The given hypergraph *HG*

we consider only the nodes that are contained in cycle components. A set of touch-points would be $P = \{C, D, F, E, I)$. However no edge from set P alone divides the hypergraph, so we must combine the edges that are contained in different cycle components and are not directly connected. So we get a new set of possible separators $P' = \{CF, CE, CI, DE, DI\}$. The given upper bound for the width of separator is reached.

Generally, for hypergraphs with large number of edges a set of possible separators P' can be very large too, and evaluation time in that case can be a problem. In order to restrict the evaluation time, we restrict the set of possible separators P' heuristically.

The main problem is how to achieve a connectedness by finding of the other separators. Our heuristic uses two different methods. We either try a total covering of a previous separator which sometimes increase rapidly a separators width , or we extend each found possible separator with edges of a previous separator (Figure 3.10).

Suppose AB is a separator. Then the separator will be totaly covered by edges

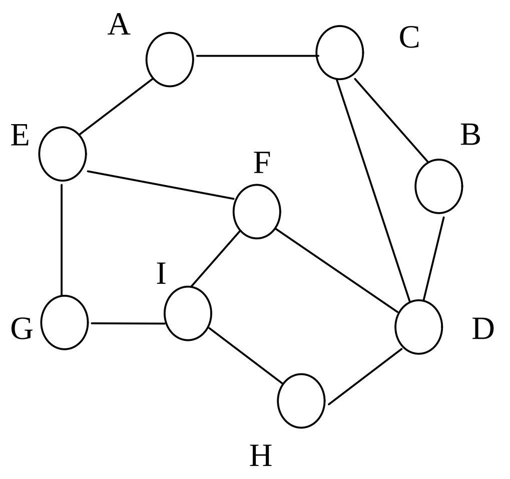

Figure 3.7: It's dual graph

C, D, F . So the next separator will be CDF. This method is easy but increases unnecessary the separators width. The other method is more efficient. We suppose a set of a new possible separators $Poss = \{D, E\}$. Than we extend the possible separators with edges of a previous separator, thus get a new set of a possible separators $Poss' = \{DAB, EAB\}$. And finally we have the unique set of possible of separators $Poss = \{CDF, DAB, EAB\}$.

We have special cases where a dual graph is a cycle or is a acyclic. If the dual graph is a cycle, see Figure 3.11 (a), (the width of separator is exactly 2) than we simply select two not neighbour edges e.g. AD or EB or… CF as e separator and decompose the graph into acyclic graphs.If a dual graph is a acyclic, see Figure 3.11 (b), (the width of separator is exactly 1) than we select only one edge as separator e.g.. D or C.

The evaluating of separators begins with determining one fitness criteria that makes possible to choose separators which divide the hypergraph into subgraphs that are as independent as possible of each other and that are approximately of equal size.

The evaluation algorithm perform the following steps:

1. Find the number of subgraphs $(\#Subgraphs)$

2. Compute the average value of all subgraphs edges (μ)

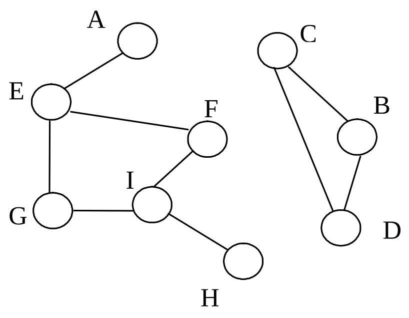

Figure 3.8: The dual graph after the cycle component D-C-B is cut out

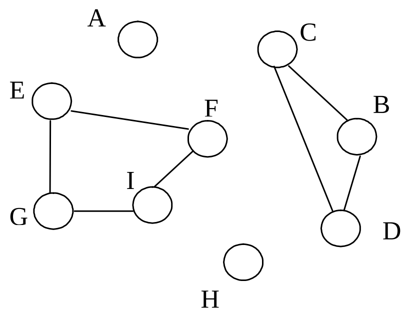

Figure 3.9: The dual graph after the cycle component E-G-I-F is cut out

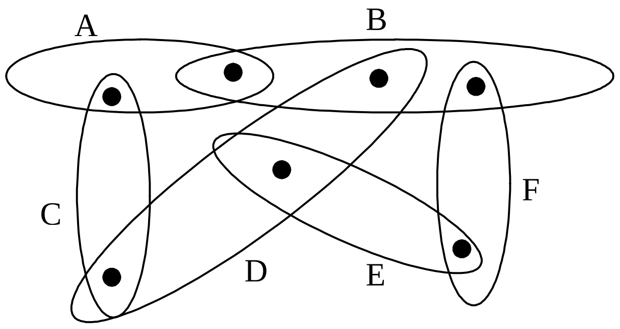

Figure 3.10: The covering example

3. Compute the deviation of the average value for each subgraph
4. Compute the average value of all deviations (MD)
5. Choose the separator with the least average value of all deviations within the set of separators.

where

$$\mu = \frac{\sum_{i=1}^{\#Subgraphs} |\#Subgraph_i|}{\#Subgraphs} \qquad MD = \frac{\sum_{i=1}^{\#Subgraphs} ||\#Subgraph_1| - \mu|}{\#Subgraphs}$$

Let us consider an example to illustrate this process. We compare two possible separators from set P'(Figure 3.7), say DE and CE.

DE divids a hypergraph into subgraphs : $HGIF$ and ACB (see Figure 3.12) $=>$ $\#Subgraphs = 2$

$$\mu = \frac{4+3}{2} = 3.5 \qquad and \qquad MD = \frac{|4-3.5|+|3-3.5|}{2} = \frac{0.5+0.5}{2} = 0.5$$

CE divides a hypergraph into subgraphs : A and $BDFIHG$ (see Figure 3.13) $=> \#Subgraphs = 2$

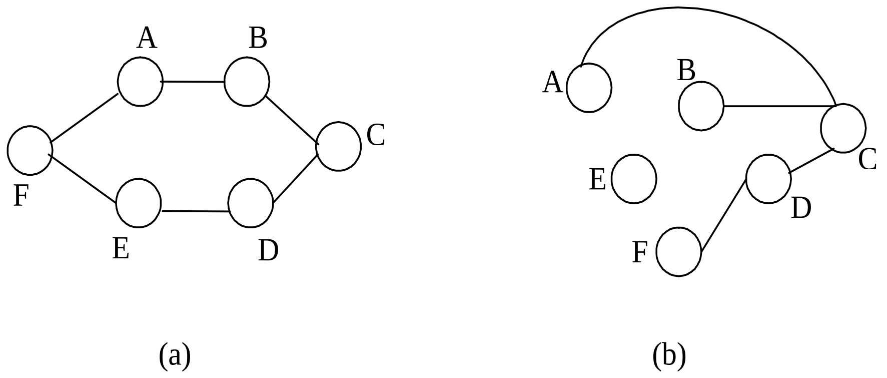

Figure 3.11: A cyclic dual graph (a) and an acyclic graph after decomposing (b)

$$\mu = \frac{1+6}{2} = 3.5 \qquad and \qquad MD = \frac{|1-3.5|+|6-3.5|}{2} = \frac{2.5+2.5}{2} = 2.5$$

As we see the separator DE hase a smaller MD than separator CE which makes it a better separator.

Star connectedness condition The star decomposition process can have wrong connectedness condition as result. From Figure 3.5 we see that after star decomposition the edges A, C, D are not anymore directly connected. The won hypertree would be correct , however only for dual graph after star decomposition, but not for original dual graph. In order to repair such cases, we check the connectedness condition between all star edges. If between two vertexes in hypertree (they must be within a subtree) no such connection exists, we insert common star variables in all vertexes of that subtree.

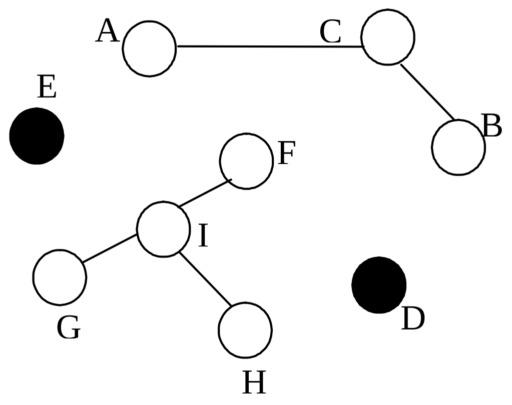

Figure 3.12: The resulting subgraphs for graph in Figure 3.7 from separator DE

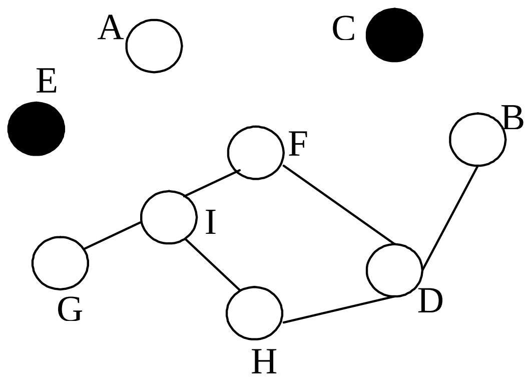

Figure 3.13: The resulting subgraphs for graph in Figure 3.7 from separator CE

Building of generalized hypertree decomposition This heuristic approach is a top-down algorithm, which after every succsessful found separator creates a new hypertree node and inserts it in tree as a child. This will be made recursively until the subgraph is smaller then worst found separator-width .

Let explain it with an example:

Consider the hypergraph in Figure 3.14. We suppose that in first step B is chose as separator, i.e. $Sep = \{B\}$. A new node of hypertree is created, see Figure 3.15 (a). The separator divides hypergraph into two subhypergraphs $HG_1 = \{A, D, E\}$ and $HG_2 = \{G, C, F\}$.

From HG_1 are chosen D and E as separator,i.e. $Sep = \{D, E\}$. A new hypertree node is created and inserted as child in parent node, see Figure 3.15 (b) . Finally we get

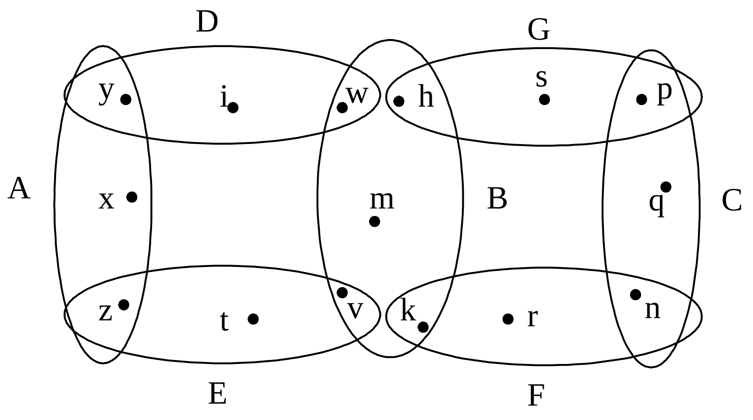

Figure 3.14: Hypergraph *HG*

an acyclical graph, only edge A. A new hypertree node is created and inserted as child in parent node, see Figure 3.15 (c) . Here we break the recursivity, and we begin with HG_2. The won hypertree is shown in Figure 3.15 (d) .

After the hypertree is built we begin with the second(bottom-up) phase. In that phase we try to optimize the variables of all hypertree nodes. Each parent node should have only those variables which are necessary to keep connectivity. If an hyperedge occurs in a hypertree node and can also be found in its parent node, then holds only the variables which can also be found in its child node, otherwise all variables are assigned to the hyperedge. If hyperedge occurs in the root of hypertree then it holds all variables. From Figure 3.16 we see that the hyperedge A occurs at hypertree nodes p and q, therefore at hypertree node q : $var(A) = var(A) \cap var(D)$. However hyperedge A at hypertree node p, hyperedge C and hyperedge D have to hold all its variablen.

Let be p, p_1, p_2, p_3 the hypertree nodes from Figure 3.15 (d), where $\lambda(p) = \{B\}, \lambda(p_1) = \{D, E\}, \lambda(p_2) = \{A\}, \lambda(p_3) = \{G, F\}, \lambda(p_4) = \{C\}$.

We begin with hypertree node p_2 .

Hypertree node p_2: It is a leaf of hypertree, therefore $\chi(p_2) = \{x, y, z\}$.

Hypertree node p_1: It is a parent node of hypertree node p_2 . The edges D and E can not be found in hypertree node p therefore they holds all its variables. It is similar for all other edges.

Finally we get: $\chi(p) = \{w, h, m, v, k\}$, $\chi(p_1) = \{y, i, w, z, t, v\}$, $\chi(p_2) = \{x, y, z\}$, $\chi(p_3) = \{h, s, p, k, r, n\}$, $\chi(p_4) = \{p, q, n\}$.

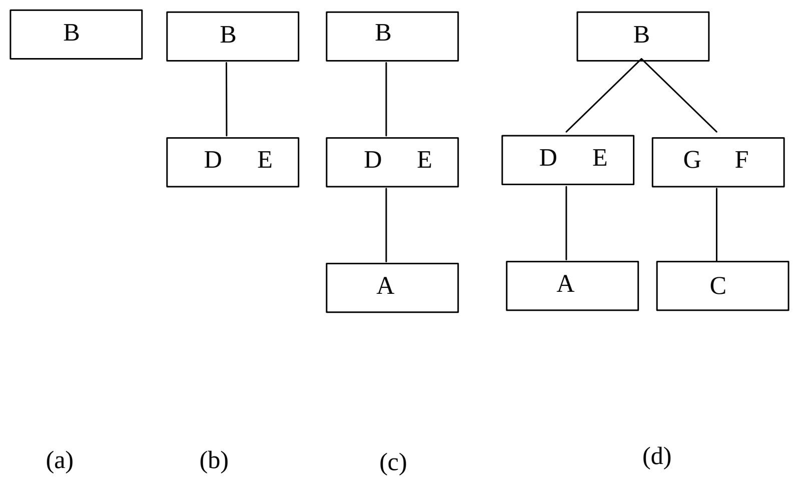

Figure 3.15: Building of hypertree

The Correctness of the Heuristics To prove that our heuristic produces a correct generalized hypertree decomposition, we show that the heuristic satisfies following three conditions :

1. Every constraint must appear in hypertree T or is covered, i.e. $\forall E \in edgess(HG), \exists p \in vertices(T)$ such that $var(E) \subseteq \chi(p)$
2. The connectedness condition, i.e. $\forall x \in var(HG)$, all $p \in vertices(T)$ such that $x \in \chi(p)$ induces a connected subtree
3. That for each variable of the hypertree vertex there exists at least one hyperedge that accours in that vertex which contains that variable, i.e. $\forall p \in vertices(T), \chi(p) \subseteq var(\lambda(p))$

First we give some basic definitions which will help us to understand and explain our heuristic approach and which are necessary to prove the correctness of the method.

Definition 16 *(Subproblem) Let* $HG = (V, E)$ *be a hypergraph of a problem* S. *Then* $S_1 = (V_1, E_1)$ *is a subproblem if*
V_1 *is the finite set of hypernodes*
E_1 *is the finite set of hyperedges*

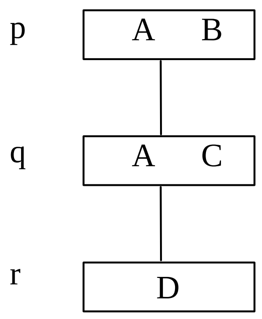

Figure 3.16: Optimization of variables

where $V_1 \subseteq V$ and $E_1 \subseteq E$

Definition 17 *(Separator) Let $S_1 = (V_1, E_1)$ be a subproblem. Then $K = Sep(S_1)$ is its separator if $K \subset E_1$ and $\mid var(K) \mid < \mid V_1 \mid$*

Note that the heuristic accepts separators even then they do not split the hypergraph.

Definition 18 *(ParentSeperator) Let $S = (V, E)$ be a problem, $K = Sep(S)$ its separator and $S_1 = (V_1, E_1)$ a subproblem as result of decomposing by K.*

Then K is ParentSeparator for subproblem $S_{1.}$ Formally $K = ParentSep(S_1)$

Definition 19 *(CommonEdges) All edges of subproblem S_1 having some common variables with its ParentSeparator K are called common edges of subproblem with its ParentSeparator, formally $commEdg(S_1{,}K) = \{E \in E_1 \mid var(E) \cap var(K) \neq \varnothing\}$.*

Analogous $commEdg(K, S_1) = \{E \in edgess(K) \mid var(E) \cap var(S_1) \neq \varnothing\}$ are called common edges of ParentSeparator with its subproblem

Definition 20 *(CommonVariables) All common variables of subproblem with its ParentSeparator are called CommonVariables, formally $commVar(S_1{,}K) = commVar(K, S_1) = var(S_1) \cap var(K)$*

We say $commVar(S_1{,}K)$ is totally covered from $commEdg(S_1{,}K)$.

Now we give a definition of the separators used in our heuristic approach. After every decomposition step we choose as a separator either common edges of subproblem (complet covering of *CommonVariables*) or we find a new separator from subproblem and then we extend this with common edges of *ParentSeparator*.

Formally

$Sep(S_1) = commEdg(S_1,K)$ or $Sep(S_1) = Sep(S_1) \cup commEdg(K, S_1)$

From these definitions we can show and prove the correctness of the heuristic algorithm:

- The first condition is trivial. Because of the fact that we operate on dual graph, it follows that we decompose our hypergraph as long as all constraints either become part of separators and thus accour in a hypertree node, or they become part of an acyclic hypergraph, which is normally a leaf of hypertree.
- From our separator definition follows that every new found separator is a child of previous separator and it either coveres totally the *CommonVariables* or is extended with common edges of previous separator. This art of separators ensures us that all parent-child vertexes have common variables so that no casualties of connectednnes exists. The second condition is also satisfied.
- Third condition is also trivial. A separator is a product of hyperedges. So the variables of separator(vertex of hypertree) is a set of variables of all hyperedges of that vertex. This is exactly the third condition.

A simple example Consider the hypergraph in Figure 3.17. The hyperedges of this hypergraph are:

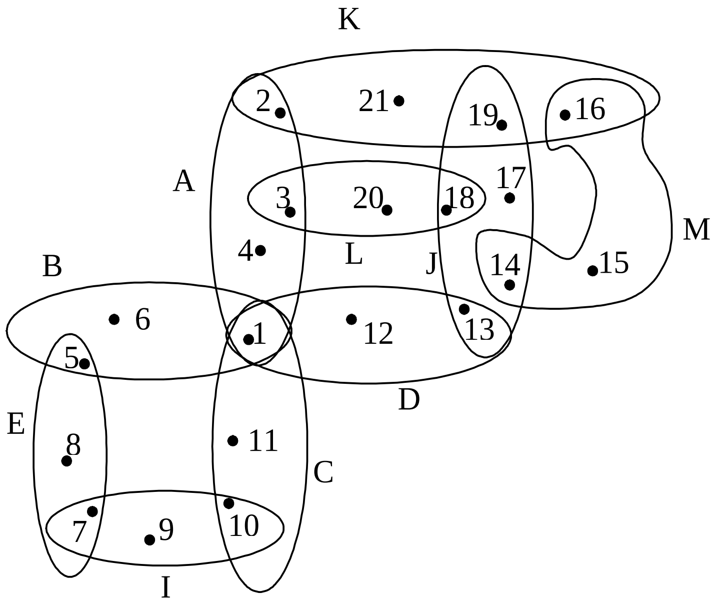

Figure 3.17: Example of a hypergraph

$$
\begin{aligned}
A &= \{1,2,3,4\}\\
B &= \{1,5,6\}\\
C &= \{1,10,11\}\\
D &= \{1,12,13\}\\
E &= \{5,7,8\}\\
I &= \{7,9,10\}\\
K &= \{2,16,19,21\}\\
L &= \{3,18,20\}\\
M &= \{14,15,16\}\\
J &= \{13,14,17,18,19\}
\end{aligned}
$$

Further we suppose a hypertree width 2, therefore we give a start (desired) separator width $k = 2$.

- *Star Decomposition*

First we find all star structures of the hypergraph. We have only one such structure, that is $A - B - C - D$. The edge A is the most connected hyperedge of star structure, therefore we chose it as *basic star hyperedge.*The common variable is 1. This variable remain again in basic star hyperedge, additional we insert new temporally variables x and y. Finally we decompose the star structure, see Figure3.18.

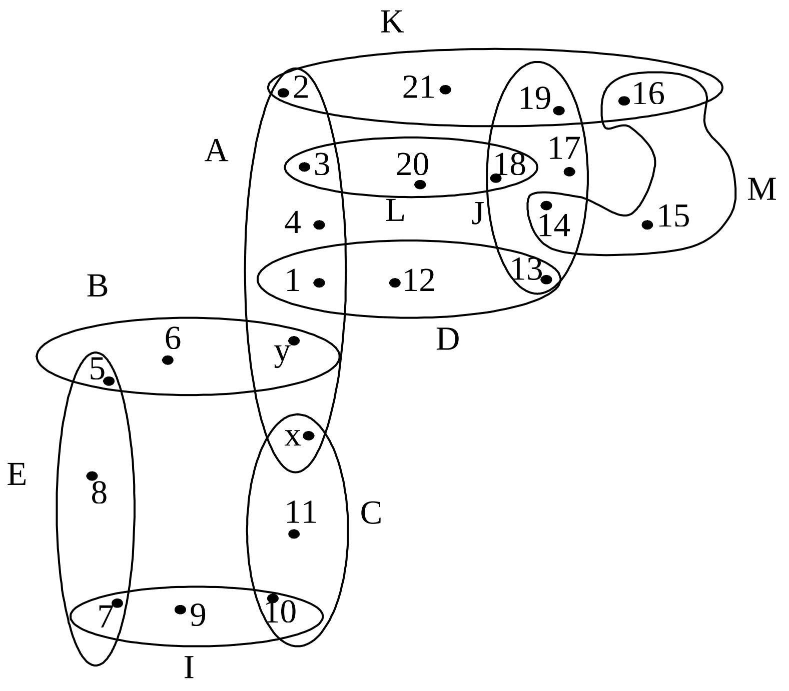

Figure 3.18: Hypergraph after star decomposition

After star decomposition the edges have these variables:

$$A = \{1, 2, 3, 4, x, y\}$$

$$\begin{aligned}
B &= \{5, 6, y\}\\
C &= \{10, 11, x\}\\
D &= \{1, 12, 13\}\\
E &= \{5, 7, 8\}\\
I &= \{7, 9, 10\}\\
K &= \{2, 16, 19, 21\}\\
L &= \{3, 18, 20\}\\
M &= \{14, 15, 16\}
\end{aligned}$$

$$J = \{13, 14, 17, 18, 19\}$$

- *Dual Graph*

  After converting of hypergraph into dual graph (Figure 3.19), we find the cycle components, begining of most connected node, that is A.

  The smallest cycle component is: $Cyc = \{A - D - J - K\}$.After this cycle is cut, we get those acyclic components: $Acyc = \{L, M, B - E - I - C\}$.

  All touch points are: $P = \{A, J, K\}$.

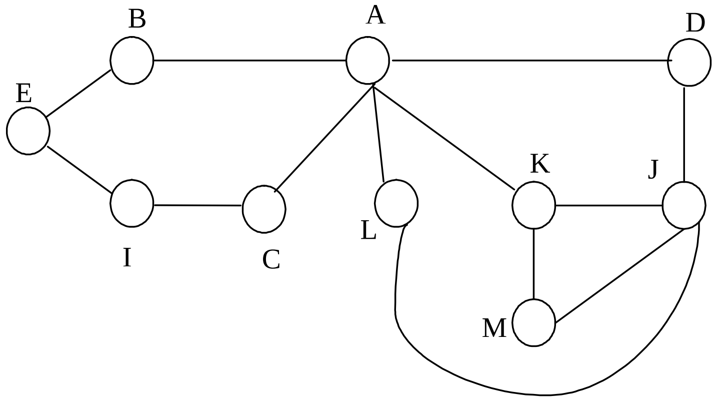

Figure 3.19: Dual graph of given hypergraph

- *Choose of the separators*

  A set of possible separators is: $PossSep = \{A, J, K\}$. We consider also AJ, AK, JK as possible separator,because our start separator width $(= 2)$ allows us, therefore $PossSep = \{A, J, AJ, AK, JK\}$.

  The fitness function choses AJ as separator, that is $K = \{A, J\}$. We create a vertex (root of hypertree). This separator divides the graph in the following subgraphs(subproblems): $S_1 = \{L\}, S_2 = \{D\}, S_3 = \{M, K\}, S_4 = \{B, E, I.C\}$. Up to the subproblem S_4 we have nothing to decompose, but to create new vertexes and to insert them in separator (vertex) K. We decompose further the subhypergraph $S_4 = \{B, E, I.C\}$. The common edges of subhypergraph with parent separator are $commEdg(S_4, K) = \{B, C\}$.Analogous $commEdg(K, S_4) =$

$\{A\}$. We have two possibilities to choose the new separator K_1 of subhypergraph. Either $K_1 = commEdg(S_4, K) = \{B, C\}$ (totally covered), or we get edge E as a separator of subhypergraph, and extend this with $commEdg(K, S_4) = \{A\}$, that is $K_1 = \{A, E\}$. The fitness function chose $\{B, C\}$ vs. $\{A, E\}$, that is $K_1 = \{B, C\}$. We create a new vertex and insert it as a child in separator (vertex) K.

The separator $K_1 = \{B, C\}$ divides further the subhypergraph into $S_5 = \{E, I\}$. Here we break the recursivity and create a new vertex and insert it as a child in separator (vertex) K_1.

The won hypertree (only λ set) is shown in Figure

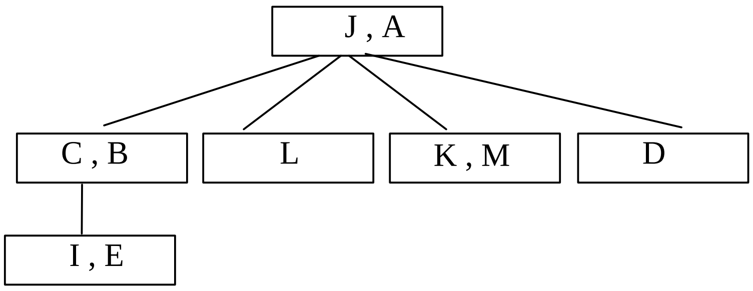

Figure 3.20: Hypertree decomposition of the hypergraph in Figure 3.18

- *Bottom - Up phase*

  All edges present in vertexes of hypertree, do not occur in any another vertex, therefore they holds all variables .

- *Star connectedness condition*

  Star connectedness condition is also satisfied. All star edges $\{A, B, C, D\}$ are connected (see Figure 3.20) therefore no additional proceedings is necessary.

4 Evaluations of heuristics

In this chapter we will show computitional results of two heuristics implemented in this diploma thesis (algorithm based on dual graph and algorithm based on HMETIS partitioning). Additionally we will give a comparison between them and results of other heuristics implemented by DBAI research group [2]). All these heuristc algorithms are tested on different industrial examples from DaimlerChrysler, NASA, ISCAS circuits and syntethicaly generated examples like Grids and Cliques. The description of these problems is given in [18].

The experiments were done on two different machines. Because of the fact that the heuristic algorithm based on dual graph for large problems has not optimal runtime performance we have tested this algorithm for problems having a width size at the most 600 Edges/Notes. These experiments were done in a machine with IntelPentium Proccesor (1,7 GHz, 1GB Memory).

The algorithm based on HMETIS partitioning is tested in a machine with a Intel Xenon (2x) Procesor (2,2GHz, 2GB Memory). The experiments of this algorithm are described also in [2]

4.1 Comparison of heuristics for small problems (600 Edges/Nodes)

In this section we compare the results for two algorithms implemented in this diploma thesis, as well as *opt-k decomp* and Korimort algorithm.

The algorithm based on dual graph needs a start parameter which represents our desired size of separators (see section 2.4.2). Note that, in general for different start parameters the algorithm gives different results (see Tables 4.1 and 4.2). The results for *opt-k-decomp* and Korimort algorithm are taken from [2]. The tables 4.1 and 4.2 show that the algorithm yields better results for start parameter k=1 then for k=2. However the runtime is worse. In general there is no guarantee that decompositions started with smaller desired size of separators yields a better decomposition results. Indeed a found local "*good*" separator is not necessary "*the best*" one.

The HMETIS algorithm is tested for the different number of partition*s* (nparts = 2 and nparts = 3) and for the imbalance factor between partitions (ubfactor = 5, ubfactor = 10, ubfactor = 20, ubfactor = 30, ubfactor = 40, ubfactor = 49). The best found width is taken as the result.

The algorithms are tested on total 101 examples from DaimlerChrysler, NASA, ISCAS and Cliques. The obtained results are shown in tables 4.3, 4.4, 4.5, 4.6, 4.7 and 4.8. The tables have the following structure: The first column represents the name of example and its number of atoms and variables. The results and runtime obtained from *opt-k-decomp* and from the algorithm of Korimort can be found in columns named with "opt-k-decomp" respectively "Korimort". The columns named with "Dual Graph" and "HM" represent the results and runtimes obtained from algorithms implemented in this diploma thesis (algorithm based on dual graph and algorithm based on HMETIS partitioning). Note that the runtime measures are shown in seconds.

	For k=1	
Instance (Atoms / Variables)	Time (sec)	Width
adder_2 (11 / 15)	0	2
adder_3 (16 / 22)	0	2
adder_15 (76 / 106)	25	2
adder_30 (151 / 211)	120	2
bridge_4 (38 / 38)	8	2
bridge_12 (110/ 110)	220	2
bridge_32 (290/ 290)	2780	2
atv_partial_system (88 / 125)	65	4

Table 4.1: The results of algorithm based on dual graph with start parameter k=1

	For k=2	
Instance (Atoms / Variables)	Time (sec)	Width
adder_2 (11 / 15)	0	3
adder_3 (16 / 22)	0	3
adder_15 (76 / 106)	13	3
adder_30 (151 / 211)	87	3
bridge_4 (38 / 38)	3	3
bridge_12 (110/ 110)	80	3
bridge_32 (290/ 290)	830	3
atv_partial_system (88 / 125)	51	4

Table 4.2: The results of algorithm based on dual graph with start parameter k=2

From tables we conclude that *opt-k-decomp* is useful only for small examples. The Korimort algorithm is tested for examples of DaimlerChrysler (adder, bridges, newsystem and atv_partial_system). Although the algorithm for adder and bridges examples yields the optimal results, it is not useful for larger problems instances, see [2]..

The algorithm based on dual graph is tested for examples of DaimlerChrysler, NASA, ISCAS and Cliques. For adder and bridges examples the algorithm yields optimal results. However the larger the problem the worse the result. In comparision to HMETIS algorithm the dual graph algorithm has a very large runtime. Therefore in order to avoid a very large runtimes, this algorithm is prefered only for small problem instances (with at most 600 Edges/Notes) . This algorithm was able to solve all given examples,but not always gives optimal solutions. The total sum of widths of all given examples for Dual Graph algorithm is 1797. The total runtime is 19.37 hours.

The HMETIS algorithm yields the best results among all other algorithms. The total sum of widths of all given examples is 1157. The total runtime is 0.32 hours.

Overall the results show that HMETIS algorithm is much better heuristic approch then the other heuristics shown in this section. Therefore the HMETIS is compaired also with other heuristics developed by DBAI research group (see [2]). The next section shows the results of these comparasons.

		opt-k-decomp		Korimort		Dual Graph			HM	
Instance (Atom / Var)	Min	W	T	W	T	W	T	k	W	T
adder_2 (11 / 15)	2	2	1	2	?	2	0	1	2	0
adder_3 (16 / 22)	2	2	1	2	?	2	0	1	2	0
adder_4 (196 /274)	2	2	?	2	?	2	280	1	2	0
adder_5 (26 / 36)	2	2	3	2	?	2	1	1	2	0
adder_6 (31 / 43)	2	2	5	2	?	2	2	1	2	0
adder_7 (36 / 50)	2	2	12	2	?	2	4	1	2	0
adder_8 (41 / 57)	2	2	17	2	?	2	5	1	2	1
adder_9 (46 / 64)	2	2	21	2	?	2	7	1	2	1
adder_10 (51 / 71)	2	2	28	2	?	2	10	1	2	2
adder_11 (56 / 78)	2	2	29	2	?	2	14	1	2	2
adder_12 (61 / 85)	2	2	32	2	?	2	19	1	2	2
adder_15 (76 / 106)	2	2	39	2	?	2	25	1	2	3
adder_18 (91 / 127)	2	2	42	2	?	2	34	1	2	4
adder_20 (101 / 141)	2	2	49	2	?	2	42	1	2	4
adder_25 (126 / 176)	2	2	52	2	?	2	49	1	2	5
adder_30 (151 / 211)	2	?	?	2	?	2	120	1	2	6
adder_40 (201 / 281)	2	?	?	2	?	2	232	1	2	8
adder_50 (251 / 351)	2	?	?	2	?	2	508	1	2	10
adder_60 (301 / 421)	2	?	?	2	?	2	805	1	2	12
adder_75 (376 / 526)	2	?	?	2	?	2	1791	1	2	14
adder_85 (426 / 596)	2	?	?	2	?	2	2823	1	2	16
adder_90 (451 / 631)	2	?	?	2	?	2	3464	1	2	18
adder_99 (496 / 694)	2	?	?	2	?	2	6200	1	2	20

Table 4.3: Results of Dual Graph and HMETIS for Adder examples

Instance (Atom / Var)	Min	opt-k-decomp		Korimort		Dual Graph			HM	
		W	T	W	T	W	T	k	W	T
bridge_1 (11 / 11)	2	2	?	2	?	2	0	1	2	0
bridge_4 (38 / 38)	2	2	?	2	?	2	8	1	3	1
bridge_8 (74 / 74)	2	2	?	2	?	2	64	1	3	2
bridge_12 (110 / 110)	2	2	?	2	?	2	220	1	4	4
bridge_16 (146 / 146)	2	2	?	2	?	2	520	1	3	6
bridge_20 (182 / 182)	2	2	?	2	?	2	670	1	3	10
bridge_26 (236 / 236)	2	2	?	3	?	3	394	2	4	14
bridge_32 (290 / 290)	2	2	?	3	?	2	2780	1	3	16
bridge_38 (344 / 344)	2	2	?	3	?	3	3273	2	3	18
bridge_46 (416 / 416)	2	2	?	3	?	3	4800	2	4	20
bridge_50 (452 / 452)	2	2	2211	3	174	3	7879	2	4	21

Table 4.4: Results of Dual Graph and HMETIS for Bridge examples

Instance (Atom / Var)	Min	opt-k-decomp		Korimort		Dual Graph			HM	
		W	T	W	T	W	T	k	W	T
clique_10 (10 / 45)	5	5	0	?		7	0	4	5	0
clique_11 (11 / 55)		?		?		8	0	4	6	0
clique_12 (12 / 66)		?		?		9	0	4	6	0
clique_13 (13 / 78)		?		?		10	0	4	7	0
clique_14 (14 / 91)		?		?		11	1	4	8	0
clique_15 (15 / 105)	8	?		?		12	1	4	8	1
clique_16 (16 / 120)		?		?		13	2	4	8	1
clique_18 (18 / 153)		?		?		15	3	4	10	1
clique_20 (20 / 190)	10	?		?		17	4	4	10	1
clique_22 (22 / 231)		?		?		19	6	4	11	2
clique_24 (24 / 276)		?		?		21	8	4	13	2
clique_25 (25 / 300)	13	?		?		22	9	4	14	2
clique_30 (30 / 435)	15	?		?		27	16	4	15	3
clique_35 (35 / 595)	18	?		?		32	28	4	18	7
clique_40 (40 / 780)	20	?		?		37	47	4	20	9
clique_45 (45 / 990)	23	?		?		42	70	4	23	30
clique_50 (50 / 1225)	25	?		?		47	110	4	25	47
clique_60 (60 / 1770)	30	?		?		57	230	4	51	4
clique_70 (70 / 2415)	35	?		?		67	430	4	68	1
clique_80 (80 / 3160)	40	?		?		77	719	4	71	6
clique_90 (90 / 4005)	45	?		?		87	1154	4	89	2
clique_99 (99 / 4851)	50	?		?		96	1666	4	98	3

Table 4.5: Results of Dual Graph and HMETIS for clique examples

Instance (Atom / Var)	Min	opt-k-decomp W	opt-k-decomp T	Korimort W	Korimort T	Dual Graph W	Dual Graph T	Dual Graph k	HM W	HM T
grid2d_5 (12 / 13)		?		?		3	1	2	3	0
grid2d_10 (50 / 50)	4	?		?		8	20	4	6	3
grid2d_12 (72 / 72)		?		?		10	54	4	9	7
grid2d_15 (112 / 113)	6	?		?		16	162	4	11	10
grid2d_16 (128 / 128)		?		?		19	216	4	11	12
grid2d_20 (200 / 200)	7	?		?		24	166	23	15	18
grid2d_25 (312 / 313)	9	?		?		28	459	23	16	38
grid2d_26 (338 / 338)		?		?		33	571	23	15	30
grid2d_28 (392 / 392)		?		?		37	870	23	16	24
grid2d_30 (450 / 450)	11	?		?		38	1277	23	17	52
grid2d_35 (612 / 613)	12	?		?		40	2995	23	20	68
grid2d_40 (800 / 800)	14	?		?		58	6185	23	23	82
grid3d_4 (32 / 32)	5	?		?		11	3	9	6	0
grid3d_5(62 / 63)	[6,8]	?		?		18	72	4	11	2
grid3d_6 (108 / 108)	[9,11]	?		?		38	54	14	16	7
grid3d_7 (171 / 172)	[11,14]	?		?		46	190	14	18	16
grid4d_3 (40 / 41)		?		?		20	7	17	8	1
grid4d_4 (128 / 128)		?		?		54	83	32	20	8

Table 4.6: Results of Dual Graph and HMETIS for grid examples

Instance (Atom / Var)	Min	opt-k-decomp W	opt-k-decomp T	Korimort W	Korimort T	Dual Graph W	Dual Graph T	Dual Graph k	HM W	HM T
NewSystem1 (84 / 142)	3	?		3	31	3	35	2	3	4
NewSystem2 (200 / 345)	3	?		4	88	4	260	2	4	10
NewSystem3 (278 / 474)		?		4	271	5	820	2	5	14
NewSystem4 (418 / 718)		?		4	741	6	1200	5	5	21
atv_part_sys (88 / 125)	3	?		3	47	4	36	3	4	4
NASA (680 / 579)		?		?		63	4230	14	32	68

Table 4.7: Results of Dual Graph and HMETIS for NewSystem, ATV and NASA examples

Instance (Atom / Var)	Min	opt-k-decomp		Korimort		Dual Graph			HM	
		W	T	W	T	W	T	k	W	T
b01 (45 / 47)	>4	?		?		7	4	2	5	2
b02 (26 / 27)	3	3	2	?		4	1	2	4	1
b03 (152 / 156)	>3	?		?		8	112	2	8	12
b06 (48 / 50)	4	?		?		6	11	3	5	2
b07 (432 / 433)	>3	?		?		57	1696	3	31	49
b08 (170 / 179)	>3	?		?		21	250	3	12	18
b09 (168 / 169)	>3	?		?		21	900	3	12	19
b10 (189 / 200)	>3	?		?		27	357	3	16	21
b13 (324 / 352)	>3	?		?		26	1270	3	8	31
s27 (13 / 17)	2	?		?		3	0	2	2	0
s208 (104 / 115)	>3	?		?		8	113	2	7	10
s298 (133 / 139)	>3	?		?		8	115	2	6	11
s344 (175 / 184)	>3	?		?		10	362	2	7	21
s349 (176 / 185)	>3	?		?		13	246	2	7	18
s382 (179 / 182)	>3	?		?		8	257	2	7	14
s386 (165 / 172)		?		?		21	219	7	11	15
s400 (183 / 186)	>3	?		?		8	286	2	7	17
s420 (420 / 231)	>3	?		?		15	812	2	10	22
s444 (202 / 205)	>3	?		?		8	374	4	8	20
s510 (217 / 236)	>3	?		?		41	404	3	27	21
s526 (214 / 217)	>3	?		?		14	437	3	11	23

Table 4.8: Results of Dual Graph and HMETIS for some ISCAS examples

4.2 Comparison of HMETIS algorithm with other algorithms

We give in this section the comparison of comparison of HMETIS algorithm with other heuristic algorithms implemented by DBAI research group [2]. The tables 4.9 and 4.10 show only the part of results obtained by this group and the description of art of comparison. A complete results and tables can be found also at [2].

In this section are given the results for four heuristics approaches described in previous chapter. The results based on use of Fiduccia-Mattheyses algorithm (FM), tabu search algorithm (TS) , HMETIS (HM) as well as Bucket Elimination (BE) are presented. For partitioning algorithms exists different treatments of special hyperedges. The special hyperedges have either the same weight as other hyperedges, or they have weight equal to 2, while other hyperedges have the weight equal to 1, or they have the weight equal to the number of edges of the original hypergraph needed to cover all of its vertices [2] .

Each algorithm was executed 5 times. The best found width is taken as result. The average time of 5 runs was taken as runtime. Comparing these algorithms, we can conclude that the HMETIS and BE algorithms give the best results. The time perfomance of HM is in general better than the perfomance of BE algorithm.[2]

Instance (Atom / Var)	Min	FM		TS		BE		HM	
		W	T	W	T	W	T	W	T
adder_15 (76 / 106)	2	2	0	4	0.2	2	0	2	3
adder_25 (126 / 176)	2	2	1	4	0.2	2	0	2	6
adder_50 (251 / 351)	2	2	6	4	1.2	2	0	2	12
adder_75 (376 / 526)	2	2	21	5	2	2	0	2	19
adder_99 (496 / 694)	2	2	53	5	3.2	2	1	2	25
bridge_15 (137 / 137)	2	8	1	8	0.8	3	0	3	6
bridge_25 (227 / 227)	2	13	1	6	1.4	3	0	3	11
bridge_50 (452 / 452)	2	29	5	10	3.2	3	1	4	22
bridge_75 (677 / 677)	2	44	10	10	5.4	3	1	3	35
bridge_99 (893 / 893)	2	64	18	10	6.8	3	2	4	45
NewSystem1 (84 / 142)	3	4	1	6	0.8	3	0	3	5
NewSystem2 (200 / 345)	3	9	2	6	2.2	4	0	4	13
NewSystem3 (278 / 474)		17	4	11	4	5	1	5	18
NewSystem4 (418 / 718)		22	8	12	6.8	5	2	5	29
atv_part_sys (88 / 125)	3	4	0	5	0.6	3	0	4	6
NASA (680 / 579)		56	20	98	33.6	22	25	32	84
grid2d_10 (50 / 50)	4	5	0	8	0.2	5	0	5	3
grid2d_15 (112 / 113)	6	10	1	12	1.2	8	0	10	10
grid2d_40 (800 / 800)	14	28	38	41	19.8	26	28	22	91
grid2d_70 (2450 / 2450)	24	65	347	65	119	48	474	41	239
grid2d_75 (2812 / 2813)	26	70	504	99	157.8	48	631	44	274
grid3d_10 (500 / 500)	[21,27]	41	20	67	26.4	41	164	31	77
grid3d_14 (864 / 864)	[41,49]	86	161	176	162.2	78	3600	69	196
grid4d_6 (648 / 648)		58	40	140	66.8	68	2153	47	106
grid4d_8 (2048 / 2048)		120	441	310	580.8	148	3600	107	393
grid5d_3 (121 / 122)		18	1	49	3.6	18	5	19	10
grid5d_4 (512 / 512)		49	25	137	56.2	62	2039	46	78
grid5d_5 (1562 / 1563)		118	280	362	474	137	3600	111	319

Table 4.9: Comparison of HMETIS algorithm with other algorithms

Instance (Atom / Var)	Min	FM		TS		BE		HM	
		W	T	W	T	W	T	W	T
clique_10 (10 / 45)	5	5	0	6	0.2	5	0	5	0
clique_30 (30 / 435)	15	30	0	16	16.2	8	4	15	3
clique_50 (50 / 1225)	25	50	1	28	144.6	25	3600	25	13
clique_90 (90 / 4005)	45	90	12	50	2045	45	3600	78	5
clique_99 (99 / 4851)	50	99	19	54	2844.8	50	3600	97	8
c432 (160 / 196)	>3	15	3	24	3.6	9	1	12	19
c499 (202 / 243)	>3	18	3	27	4.6	13	1	17	28
c2670 (1193 / 1350)		66	56	78	45.8	31	9	38	106
c5315 (2307 / 2485)		120	250	157	156.6	44	64	68	214
c7552 (3512 / 3718)		161	514	188	351	38	85	37	309
s27 (13 / 17)	2	2	0	3	0	2	0	2	0
s208 (104 / 115)	>3	7	1	11	0.8	7	0	7	9
s832 (292 / 310)	>3	22	8	71	9.8	12	3	20	39
s1488 (659 / 667)		45	46	148	36	23	18	39	77
s5378 (2958 / 2993)		178	308	169	271	85	141	89	246
b11 (757 / 764)		65	28	98	27.2	30	82	38	79
b12 (1065 / 1070)		38	55	83	38.6	27	19	34	102

Table 4.10: Comparison of HMETIS algorithm with other algorithms

5 Conclusions

As already mentioned in the beginning of this thesis, the solving of Constraint Satisfaction Problems is in general NP – Complete, and thus intractable. Furthermore it is known that the acyclic instances of CSPs are tractable. Therefore follows, if CSP-s have instances with respective acyclic hypergraphs, than they can be solved efficiently. In order to find such tractable CSP classes different decomposition methods are developed. These methods try to generate acyclic hypergraphs from a given CSP. Hypertree Decomposition, developed by Gottlob, has been shown as the best method. The algorithm *opt-k-decomp*, which computes the optimal hypertree decomposition for bounded width at most k, will be quickly unusable because k appears in the exponent of the runtime and in the exponent of the memory space.

In order to make hypertree decomposition usable for large problems, different heuristic algorithms are developed. However, the heuristic methods give no guarantee that an optimal solution can be found.

The purpose of this diploma thesis was to find such heuristic methods which solutions deviate minimally from an optimal solution. We implemented two different heuristics based on partitioning algorithms. In chapter 3 we presented both algorithms. The first algorithm is based on dual graph. The method tries to find a significant number of "touch" points between cycles in a dual graph. Evaluating of these "touch" points with respect to certain fitness criteria should lead to a "good" local partitioning of the dual graph. The fact that algorithms takes in consideration a large number of separator "candidates ", impacts the runtime in negative sense. Therefore we tested the algorithm for problems which have at most 600 Edges/Nodes. The computational results show that this algorithm was able to compute all given examples (see chapter 3). The best results (optimal) were achieved with "Adder" and "Bridge" examples, while the worst results were achieved with "Clique" examples. The novelty of this algorithm is the solving of those problems which *opt-k-decomp* was not able to compute.

The second algorithm implemented in this diploma thesis is based on HMETIS partitioning approach [12]. This approach implements a variety of algorithms that are based on Fidducia – Mattheyses algorithms. In order to find the best possible partitioning, we used and modified the HMETIS package library [9, 13]. The computational results show that this algorithm yields a very good solution. In comparison with some other existing heuristic algorithm, we conclude that HMETIS and Bucket Elimination heuristics give the best results.

An improvement of two heuristics proposed in this thesis can be done in the future. For instance, the removing of the unnecessary "touch " points during the finding of the separator in the algorithm based on dual graph, can improve the runtime . This can be done if we pick a set of "bad touch" points. If this set remains in new partition, it will be excluded from further decomposition routine.

Another possible improvement is the combining of the HMETIS partitioning algorithm with heuristics based on vertex ordering.

References

[1] Arnaud Durand, Etienne Grandjean. "The complexity of acyclic conjunctive queries revisited," (November 2004). In FOCS'98: Proceedings of the 39th Annual Symposium on Foundations of Computer Science.

[2] Artan Dermaku, Tobias Ganzow, Georg Gottlob Ben McMahan Nysret Musliu Marko Samer. "Heursitic Methods for Hypertree Decompositions," (2005). DBAI-TR-2005-53, Technische Universität Wien.

[3] David Cohen, Peter Jeavons and Manolis Koubarakis. "Tractable Disjunctive Constraints,"

[4] Georg Gottlob, Nikola Leone, Francesco Scarcello. "A comparision of structural CSP decomposition methods," (1999). In IJCAI'99: Procedings of the Sixteenth International Joint Conference on Artificial Intelligence.

[5] Georg Gottlob, Nikola Leone, Francesco Scarcello. "Hypertree decompositions and tractable queries," (1999). In PODS'99: Procedings of the Eighteenth ACM Symposium on Principles of Database Systems.

[6] Georg Gottlob, Nikola Leone and Francesco Scarcello. "The Complexity of Acyclic Conjunctive Queries," (1998). In FOCS'98: Proceedings of the 39th Annual Symposium on Foundations of Computer Science.

[7] Georg Gottlob, Nikola Leone and Francesco Scarcello. "On tractable queries and constraints," (1999). In DEXA '99: Database and Expert Systems Applications, 10th International Conference.

[8] Georg Gottlob, Nikola Leone and Francesco Scarcello. "Hypertree Decomposition: A Survey," (2001). In MFCS 2001.

[9] George Karypis, Rajat Aggarwal, Vipin Kumar and Shashi Shekhar. "Multilevel hypergraph partitioning: applications in vlsi domain," (1999). IEEE Trans.Very Large Scale Integr. Syst,7(1):69ñ79.

[10] G.Gottlob. "Computing Cores for Data Exchange: Hard Cases and Practical Solutions," (2001). In IST 2001: INFOMIX: Knowledge-Based Information Integration.

[11] Hutle, Martin. "Constraint Satisfaction Problems - Hybrid Decomposition and Evaluating," (Februar 2002). Institut für Informationssysteme TU Wien: `Diplomarbeit`.

[12] Karypis, G. and V. Kumar. "hMETIS: A hypergraph partitioning package version 1.5.3," (1998).

[13] Karypis, George and Vipin Kumar. "Multilevel k-way hypergraph partitioning," (1999). In DAC '99:`Proceedings of the 36th ACM/IEEE conference on Design automation, pages 343ñ348, New York, NY, USA`.

[14] Korimort, Thomas. "Heuristic Hypertree Decomposition," (2003). Vienna University of Technology.

[15] Marc Gyssens, Peter G.Jeavons, David A.Cohen. "Decomposing Constraint Satisfaction Problems Using Database Techniques," (March 1994). In `Artificial Intelligence`.

[16] McMahan, Ben. "Bucket elimination and hypertree decompositions," (2004). Implementation report, Institut of Information Systems (DBAI),TU Vienna.

[17] M.Yannakakis. "Algorithms for Acyclic Database Schemes," (1981). In (VLDB'81): `Proc. of Int. Conf. on Very Large Data Bases`.

[18] Nysret Musliu, Marko Samer, Tobias Ganzow Georg Gottlob. "A CSP Hypergraph library. Technical report," (2005). DBAI-TR-2005-50, Technische Universität Wien.

[19] Peter Jeavons, David Cohen and Marc Gyssens. "A Structural Decomposition for Hypergraphs," (1991). In : `Mathematics Subject Classification`.

[20] Peter Jeavons, David Cohen and Justin Pearson. "Constraints and Universal Algebra," (September 1998).

[21] Peter Jeavons, Martin Cooper. "Tractable Constraints On Ordered Domains," (Oktober 1995).

[22] PeterHarvey and Adyita Ghose. "Fast Hypertree Decompositions,"

[23] Ph.G.Koalitis and M.Y.Vardi. "Conjunctive-Query Containment and Constraint Satisfaction," (1998). In (PODS'98): `Proc. of Symp. on Principles of Database Systems`.

[24] R.Dechter. “Constraint Networks,” (1992). In: `Encyclopedia of Artificial Intelligence, second edition`.

[25] R.Dechter, J.Pearl. “Tree clustering for constraint networks,” (1989). In: `Artficial Intelligence`.

[26] R.Fagin. “Degrees of Acyclicity for Hypergraphs and Related Database Schemes,” (July 1983). In : `Journal of the ACM`.

[27] Samer, Marko. “Hypertree-decomposition via branch-decomposition,” (2005). In:`19th International Joint Conference on Artificial Intelligence (IJCAI 2005)`,pages 1535-1536.