On Tractability and Hypertree Width

Yuliya Zabiyaka and Adnan Darwiche

Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095-1596, USA,
{yuliaz,darwiche}@cs.ucla.edu

Abstract. We investigate in this paper the notion of hypertree width as a parameter for bounding the complexity of CSPs, especially those whose constraints can be represented compactly, such as SAT problems. We first identify a simple condition which is necessary for hypertree width to provide better complexity bounds than treewidth. We then observe that SAT problems do not satisfy this condition and, hence, hypertree width cannot directly provide tighter bounds than those obtained by treewidth on these problems. We next identify a simple class of SAT problems which may contain tractable subsets, yet neither hypertree width nor treewidth bounds can recognize such tractability. Hence our final contribution is a technique for introducing the auxiliary variables into the problems that allows us to recognize the tractability of this class of problems.

1 Introduction

The interaction (primal) graph of a CSP, together with its treewidth [14], have been classical tools for bounding the complexity of CSPs [5, 2]. The interaction graph, like other models of problem structure, is an abstraction which can lose problem details that are necessary for distinguishing between tractable and intractable classes of problems. The hypergraph of a CSP is a more refined abstraction of problem structure [7], allowing one to retain some of the details missed by the interaction graph. In fact, the hypergraph and its associated notion of hypertree width allow one to provide tighter complexity bounds [8], leading one to recognize tractable classes that cannot be recognized as such based on the treewidth of their interaction graphs.

This paper is based on investigating the fundamental reasons allowing hypertree width to provide better bounds than treewidth, with the goal of applying these results to SAT problems, a specific type of CSPs. Our first observation is that the key difference between hypergraphs and interaction graphs lies in the ability of the first to distinguish between multi–variable interactions that result from a single constraint, versus those that result from multiple constraints. This ability is indeed exclusive to hypergraph abstractions and is lost when a CSP is abstracted into an interaction graph. This observation, which may seem obvious in hindsight, has an important implication: one would obtain an asymptotic difference between the bounds based on hypertree width and those based

on treewidth only in the presence of constraints whose scope is not bounded by a constant. In fact, and perhaps more consequential, the difference can only be attained in the presence of a constraint that has a scope not bounded by a constant, yet a bounded number of tuples in its corresponding relation.

Our interest in this study stems from our interest in SAT problems, a special class of CSPs with the following property. Each constraint is a clause, which can have an unbounded number of variables. Yet, a clause, viewed as a relation, has a number of tuples that is exponential in its number of variables. This means that SAT problems do not satisfy the condition which is necessary for hypertree width and treewidth to yield asymptotically different bounds. Therefore, the advantages of hypertree width over treewidth are not immediately applicable to SAT problems.

The situation is even more interesting when one considers the standard assumptions about how CSP/SAT problems are represented and how their sizes are measured. In particular, the bounds based on hypertree width assume that a CSP is represented by explicitly enumerating the tuples of each constraint [8, 4]. Therefore, the size of a CSP is always linear in the number of tuples that define constraints. This is not how the size of a SAT problem is typically measured though, as one would represent a constraint by a single clause instead of listing all truth assignments that satisfy the constraint. Hence, the size of a SAT problem is measured by the length of its clauses. Following is the implication of this important distinction. If we assume that a SAT problem is represented by explicitly listing the tuples of each clause, hypertree width (and the associated algorithm of [9]) may claim a problem to be solvable in time polynomial in the size of the problem. Yet if we express these bounds in terms of the classical size of a SAT problem (length of its clauses), we may find the bounds to be no longer polynomial.

Even more strongly: hypertree width cannot recognize the tractability of the class of SAT problems containing clauses of the sizes that are not bounded by the constant, assuming that the size of a SAT problem is measured using clause lengths. It is also well known that treewidth cannot recognize the tractability of such classes of problems either, as the treewidth cannot be smaller than the size of the largest clause minus one. We present a method, however, that helps in addressing this problem and is based on introducing auxiliary variables. The proposed method can only improve the treewidth bounds of a given SAT problem, allowing one to recognize the tractability of a class of problems which cannot be recognized based on either the hypertree width or treewidth of the original problems.

2 On the relation between treewidth and hypertree width

Consider Figure 1 which depicts the *interaction graph* and the *hypergraph* for a CSP with three constraints:

1. c_1 over variables A, B, C, called the *scope* of c_1.
2. c_2 over variables A, B, D.

3. c_3 over variables C, D.

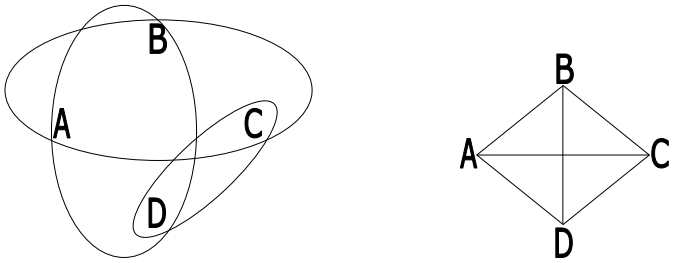

Fig. 1. A hypergraph (left) and an interaction graph (right) for a CSP over variables A, B, C, D with three constraints: c_1 (over A, B, C), c_2 (over A, B, D) and c_3 (over C, D)

Both graphs have CSP variables as their nodes. There is an edge in the interaction graph between any two variables participating in the same constraint, while there is a hyperedge in the hypergraph corresponding to every constraint.

The treewidth of a CSP is defined on its interaction graph. This can be defined in a number of ways, one of which as the width of its best *jointree* [4]. The hypertree width of a CSP is defined on its hypergraph, as the width of its best *hypertree decomposition* [10]. Figure 2 depicts a hypertree decomposition and a jointree for the CSP discussed above. A hypertree decomposition can be viewed as a refinement on a jointree. In particular, a jointree is a tree in which each node is marked by a set of variables that satisfy certain properties; see the formal definition in the Appendix. The *width* of a jointree is the size of its largest variable marking minus one. The jointree in Figure 2 has width 3. A hypertree decomposition can be defined as a jointree in which each node is also marked by a set of constraints. It is required that the scope of constraints

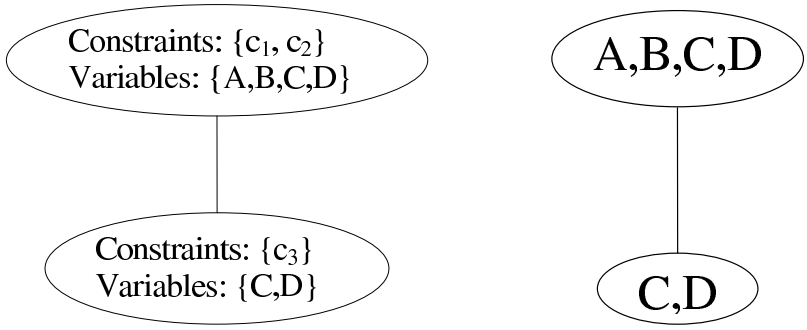

Fig. 2. Hypertree decomposition (left) and jointree (right) that corresponds to the hypertree decomposition for the problem depicted in Figure 1.

marking a node n cover all variables that mark node n. There is an additional requirement on constraint markings which leads to a distinction between *generalized* hypertree decompositions and *standard* hypertree decompositions, but this distinction is not important for our current discussion [11].[1] The *width* of a hypertree decomposition is the size of its largest constraint marking. The width of the hypertree decomposition in Figure 2 is then 2. The formal definition of hypergraph decomposition is given in the Appendix.

It then follows that every hypertree decomposition embeds a jointree. In particular, one can obtain a jointree from a hypertree decomposition by simply ignoring the constraint marking of a hypertree decomposition. Similarly, one can easily convert a jointree into a hypertree decomposition by adding a constraint marking. These observations are key to identifying the conditions under which the hypertree width can be significantly smaller than the treewidth. To identify these conditions, let us examine again the requirements on the constraint markings of a hypertree decomposition.

Let n be a node in a hypertree decomposition and let $cM(n)$ and $vM(n)$ be its constraint and variable markings, respectively. It is required that every variable X in the variable marking $vM(n)$ appear in the scope of some constraint c in $cM(n)$.[2] Given this requirement, the only case in which the width of a hypertree decomposition may be significantly better then the width of its embedded jointree is when we have constraints that involve many variables (constraints of bounded and unbounded scopes were formally studied in [12]). In that case, a few constraints will be able to cover a large variable marking. These observations are made more formal by the following result.

Theorem 1. *For a particular CSP P, let m be the largest number of variables involved in any constraint. Let w_t be the treewidth of P's interaction graph and let w_h be the hypertree width of P's hypergraph. We than have*

$$m - 1 \leq w_t \leq m \cdot w_h - 1.$$

Proof.

- $m-1 \leq w_t$: Given that we have a constraint with m variables, the interaction graph must have a clique over these m variables. Hence, the treewidth of the interaction graph must be no less than $m - 1$.
- $w_t \leq m \cdot w_h - 1$: Suppose we have an optimal hypertree decomposition and let ${w_t}'$ be the width of its embedded jointree. We must then have ${w_t}' \geq w_t$. Let n be a node in the hypertree decomposition with variable marking $vM(n)$ and constraint marking $cM(n)$. It then follows that $vM(n)$ must have at most ${w_t}' + 1$ variables, $|vM(n)| \leq {w_t}' + 1$. Since each one of these variables must be covered by a constraint in $cM(n)$, the minimum number of constraints in

[1] The additional requirement is formalized in the second condition of Definition 5.

[2] Again, this defines a generalized hypertree decomposition. For regular hypertree decomposition, an additional requirement (second condition of Definition 5) needs to be imposed on the constraint marking $cM(n)$.

$cM(n)$ is $({w_t}' + 1)/m$, $|cM(n)| \geq ({w_t}' + 1)/m$. Since $w_h \geq |cM(n)|$, we have $w_h \geq ({w_t}' + 1)/m$ and $m \cdot w_h - 1 \geq {w_t}'$.

The above result has a key implication: The only way to get an asymptotic difference between treewidth and hypertree width is to have constraints with unbounded scopes. In fact, as we show next, even though the existence of constraints with unbounded scope is necessary for inducing an asymptotic difference between treewidth and hypertree width, it is not sufficient for inducing an asymptotic difference between the *complexity bounds* based on treewidth and those based on hypertree width. To see this, let us first note that a CSP is typically represented by enumerating the tuples of each constraint; see Definition 1 in the Appendix. Consider for example SAT problems, where each constraint is a clause $l_1 \vee l_2 \vee \ldots \vee l_m$ with l_i being a literal (a variable or its negation). A clause of size m has $2^m - 1$ tuples. If we have n clauses, the size of the CSP is then $O(n2^m)$. The classical complexity bound based on treewidth w_t is $O(n2^{w_t})$. The classical complexity bound based on hypertree width has the form $O(s^{w_h} \log s)$, where s is the size of the CSP [8, 9]. Since $s = O(n2^m)$ in this case, the bound is then $O(n^{w_h} 2^{m \cdot w_h}(m + \log n))$. Since $w_t < m \cdot w_h$, the treewidth bound is indeed tighter in this case regardless of whether m is bounded or not.

The key point however is that the general treewidth bound is independent of the number of tuples for each constraint. That is, the bound remains the same even if each constraint has a bounded number of tuples. This is not the case for hypertree width. In particular, if each constraint has a bounded number of tuples, the size of a problem will then be $O(nm)$ and the corresponding complexity bound based on hypertree width can be tighter. Hence, to obtain an asymptotic difference between the complexity bounds based on treewidth and those based on hypertree width, one needs to have constraints that have unbounded scope, yet a non exponential number of tuples.

We now consider another important case, where the constraints can have scope and number of tuples not bounded by a constant, yet they can be represented compactly in space linear in their scope size. The prototypical example of this class of CSPs is SAT, where a constraint is represented by a clause. As we shall see next, neither bounds based on treewidth nor those based on hypertree width can recognize the tractability of the subclass of these problems containing clauses of the length that is not bounded by a constant. In particular, the size of a SAT problem is typically taken as $O(m \cdot n)$, where m is the size of the largest clause and n is the number of clauses. As we have seen earlier, the treewidth complexity bound for such a problem is $O(n2^{w_t})$, where $w_t \geq m - 1$. Moreover, the complexity bound based on hypertree width is $O(n^{w_h} 2^{m \cdot w_h}(m + \log n))$. Both bounds are indeed exponential in m, the size of the largest clause, even when the hypertree width is bounded by a constant. The same analysis applies to any CSP that has constraints with neither scope nor number of tuples bounded by a constant, yet the constraints can be represented compactly as discussed above.

We close this section by noting that the complexity bound based on hypertree width depends in a fundamental way on representing each constraint as a set

of tuples, as this representation is essential for the working of algorithms that prove the bound [9, 2].

3 Recognizing tractability using auxiliary variables

We propose in this section the use of a classical technique for converting any SAT problem into one where no clause has more than three literals. We also show that the conversion can never worsen the complexity bounds based on treewidth, but can lead to converting a problem with treewidth not bounded by any constant into one with bounded treewidth. Proposed conversion produces the problem no more than seven times larger than the original problem, while linearly increasing the number of variables and constraints.[3] The significance of this result is two fold. First, since the resulting CNF has clauses with bounded size, it will correspond to a CSP in which hypertree width cannot be asymptotically better than treewidth. Hence, we can restrict our attention to complexity bounds based on treewidth without loss of generality on these SAT problems. Second, since the conversion is proven not to worsen treewidth bounds, the proposed conversion can only help in recognizing easy problems. In fact, for the reasons discussed thus far, this method leads to bounds that are tighter than those based on treewidth or hypertree width (of the original problem), and can therefore recognize tractable classes that neither method can.

Since Charles Sanders Pierce [13] there is a tradition of introducing auxiliary variables in order to control constraint scope [1, 3, 15]. However, the main purpose of introducing variables in the constraint satisfaction community was to make n-ary constraints into binary constraints to enable usage of the methods developed for binary constraints. The standard technique for introducing auxiliary variables into a CSP insists on preserving the satisfaction of the given CSP. That is, the original CSP is solvable iff the augmented CSP is also solvable [3].[4] The method for introducing auxiliary variables that we shall adopt provides an even stronger guarantee: the new and augmented problems have the same number of solutions.

The classical technique [6] for reducing a clause $c = x_1 \vee \ldots \vee x_k$, $k > 3$, is to introduce an auxiliary variable y and replace the clause c by the two clauses:

$$(x_1 \vee x_2 \vee \neg y) \wedge (y \vee x_3 \vee \ldots \vee x_k).$$

The clause $(y \vee x_3 \vee \ldots \vee x_k)$ can then be reduced similarly, by introducing a new auxiliary variable, and the process repeated until every clause has no more than three literals. The above technique will guarantee the following. If Δ is the

[3] If the original problem has n variables and q constraints, with m being the scope size of the largest one, then the new problem will have no more than $3qm$ constraints and $n + qm$ variables.

[4] The method proposed in [3] is not applicable for our purpose, however, since it would require for each unbounded relation either auxiliary variables with unbounded domains (exponential in the number of variables in the relation) or if we insist on bounded domains, an unbounded number of auxiliary variables.

original CNF, and Δ' is the resulting CNF from introducing auxiliary variables $y_1, \ldots, y_n$, we will then have

$$\exists y_1, \ldots, y_n \Delta' \equiv \Delta.$$

This does not, however, preserve the model count of the original CNF. A stronger method [16], which preserves the model count is to replace each clause $x_1 \vee \ldots \vee x_k$, $k > 3$, by the following four clauses:

$$(\neg x_1 \vee y) \wedge (\neg x_2 \vee y) \wedge (x_1 \vee x_2 \vee \neg y) \wedge (y \vee x_3 \vee \ldots \vee x_k). \tag{1}$$

Note that the first three clauses correspond to the constraint $y \equiv (x_1 \vee x_2)$.

Following is the guarantee on the proposed conversion method.

Theorem 2. *Let Δ be a CNF with treewidth w, and let Δ' be a CNF which results from reducing the clauses of Δ as given by (1), until every clause has no more than three literals. The CNF Δ' will then have treewidth $w' \leq w + 1$.*

Proof. By recursively decomposing all the clauses of length greater than three in Δ, we will decompose clause $c = X_1 \vee \ldots \vee X_k$ into the set of clauses that correspond to the following equations:

$$\begin{array}{rcl} Y_1 & \equiv & X_1 \vee X_2 \\ Y_2 & \equiv & Y_1 \vee X_3 \\ Y_3 & \equiv & Y_2 \vee X_4 \\ & \ldots & \\ Y_{k-3} & \equiv & Y_{k-4} \vee X_{k-2} \\ Y_{k-3} & \vee & X_{k-1} \vee X_k \end{array} \tag{2}$$

Figure 3 depicts a jointree T_c that correspond to all the clauses in (2).

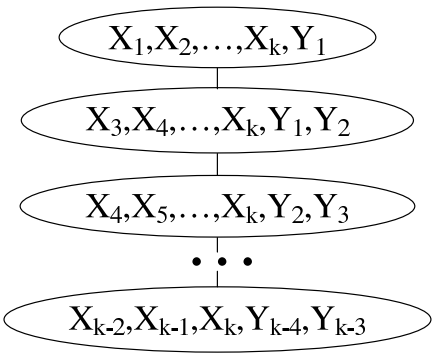

Fig. 3. A jointree corresponding to the clauses that result form decomposing a clause $c = X_1 \vee \ldots \vee X_k$ into clauses with no more than three literals.

Since the treewidth of Δ is w, there must exist a jointree with the width w, let T be one of those jointrees. By definition of a jointree, T must have a node

N_c that contain all the variables mentioned in clause c. We will attach jointree T_c for the set of clauses from equations (2) to the optimal jointree T for Δ, by connecting by an edge node N_c form T with the node labeled $\{X_1, \ldots, X_k, Y_1\}$ from T_c.

Analogously, we will construct jointrees for the sets of clauses that result form reducing all other clauses in Δ; and will attach those jointrees to T. The resulting tree T' is a jointree for CNF Δ' with width

$$w' \leq \max(m, w) \leq w + 1,$$

where m is the length of the longest clause in Δ. This should be obvious since the largest cluster in jointrees that we have constructed has size $m + 1$.

4 Example and discussion

We will consider in this section a class of CNFs with bounded hypertree width and unbounded treewidth. This immediately implies that the CNF must have some clause of unbounded length. Since a clause, viewed as a constraint, has an exponential number of tuples, neither treewidth not hypertree width can be used to recognize the tractability of this class of CNFs.

The class of CNF has k clauses, each of which has size $n + 1$:

$$\Delta = \bigwedge_{i=0\ldots k-1} C_i, \ \textit{where } C_i = X_{i \cdot n} \vee \ldots \vee X_{i \cdot n+n}.$$

If $k = 3$ and $n = 5$ then

$$\begin{aligned} \Delta &= C_0 \wedge C_1 \wedge C_2, \ \textit{where} \\ C_0 &= X_0 \vee X_1 \vee X_2 \vee X_3 \vee X_4 \vee X_5 \\ C_1 &= X_5 \vee X_6 \vee X_7 \vee X_8 \vee X_9 \vee X_{10} \\ C_2 &= X_{10} \vee X_{11} \vee X_{12} \vee X_{13} \vee X_{14} \vee X_{15} \end{aligned}$$

The optimal hypertree decomposition and optimal jointree in the general case look respectively as in Figure 4. The problem has the following parameters:

- Hypertree width $w_h = 1$
- Treewidth $w_t = n$
- Number of clauses (constraints) k
- Maximum clause length $n + 1$
- Size of a CSP (tuples enumerated) $I_{size} = k \cdot 2^{n+1} - k$
- Size of a SAT instance $k \cdot (n + 1)$

Therefore, the complexity bound induced by the hypertree decomposition is $O(k \cdot 2^{(n+1)}(\log k + n))$ and the complexity bound induced by the jointree is $O(k \cdot 2^n)$.

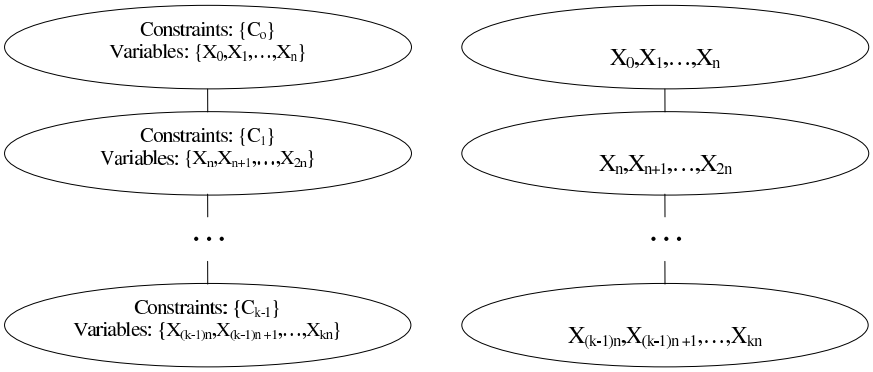

Fig. 4. Optimal hypertree decomposition(left) and optimal jointree decomposition(right) for CNF $\Delta = \bigwedge_{i=0\ldots k-1} C_i$, *where* $C_i = X_{i\cdot n} \vee \ldots \vee X_{i\cdot n+n}$.

If we introduce auxiliary variables as in Theorem 2, we will increase the number of clauses to $k(3n-5)$, but we will decrease treewidth w_t to 2, and therefore the extended problem is solvable in $O(kn)$.

Figure 5 depicts a jointree for the problem with width 2. The augmented CNF was obtained by substituting each clause C_i with a set of clauses as follows:

$$C_i = X_{i\cdot n} \vee \ldots \vee X_{i\cdot n+n} \; becomes \begin{cases} Y_{i\cdot n+1} \equiv X_{i\cdot n} \vee X_{i\cdot n+1} \\ Y_{i\cdot n+2} \equiv Y_{i\cdot n+1} \vee X_{i\cdot n+2} \\ Y_{i\cdot n+3} \equiv Y_{i\cdot n+2} \vee X_{i\cdot n+3} \\ \ldots \\ Y_{i\cdot n+n-2} \equiv Y_{i\cdot n+n-3} \vee X_{i\cdot n+n-2} \\ Y_{i\cdot n+n-2} \vee X_{i\cdot n+n-1} \vee X_{i\cdot n+n} \end{cases} \tag{3}$$

For readability we didn't expand equivalences (each equivalence will produce three clauses) and introduced auxiliary variables, skipping Y's with indexes $i \cdot n$ and $i \cdot n - 1$.

5 Conclusion

We investigated the relationship between treewidth and hypertree width as the basis for bounds on the complexity of solving CSPs. Our conclusion is that an asymptotical difference between these bounds is only possible in the presence of constraints over number of variables that are not bounded by a constant and whose number of tuples is bounded by a constant. We have also observed that neither measure of complexity can recognize class of problems containing constraints with the scope not bounded by a constant and an exponential number of tuples (this includes SAT) as tractable. We have finally proposed a method for introducing auxiliary variables into a CNF which allows us to recognize the tractability of a class of problems that could not be recognized based on either treewidth or hypertree width (of the original CNFs).

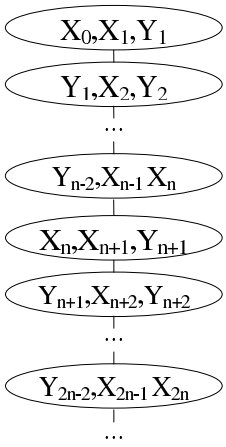

Fig. 5. Jointree that corresponds to the problem in Figure 4 after introduction of the auxiliary variables according to equation (3).

Appendix

Definition 1. *An* instance of a CSP *is a triple* $I = (Var, D, C)$*, where* $Var = \{x_1, \ldots, x_n\}$ *is a finite set of variables,* $D = \{D_1, \ldots, D_n\}$ *respective finite domains of values, and* $C = \{C_1, C_2, \ldots C_q\}$ *is a finite set of constraints.*

Definition 2. *Each constraint* C_i *is a pair* (S_i, R_i)*.* $S_i = (x_{i_1}, ..., x_{i_{m_i}})$ *is the* constraint scope*. A* constraint relation R_i *is a subset of the Cartesian product* $D_{i_1} \times \cdots \times D_{i_{m_i}}$ *denoting the variables' simultaneous legal value assignments.*

Definition 3. *[9] Given a CSP* I*, the* size *of* I *is* I_{size}*, the number of bits needed to encode* I *by listing for each constraint in* I*, its constraint scope and all tuples occurring in its constraint relation.*

$$I_{size} = \sum_{i=1}^{q} (m_i + |r_i|). \tag{4}$$

Definition 4. *A jointree for a CSP* I *is a tree, every node of which is marked with the set of variables that satisfy the following conditions:*

1. *If a variable appears in the marking of two nodes* N_1 *and* N_2*, it should appear in the marking of every node on the path connecting* N_1 *and* N_2*.*
2. *Every set of variables* S_i *that belong to a constraint* C_i *must appear in the marking of at least one of the jointree nodes.*

The *width* of a jointree is the size of its largest cluster minus one. The *treewidth* of the CSP is the width of its best jointree.

We can define a hypertree decomposition on top of a *rooted* jointree: a jointree where some node is chosen as root, and each node is a child of its neighbor closest to the root.

Definition 5. *A hypertree decomposition is a rooted jointree in which each node n is also labeled with a set of constraints, $cM(n)$. Assuming that $variables(cM(n))$ denotes the variables appearing in the constraints of $cM(n)$, the constraint marking $cM(n)$ should satisfy the following two conditions at every node n:*

1. $vM(n) \subseteq variables(cM(n))$
2. $variables(cM(n)) \cap vars(n) \equiv vM(n)$

Here, $vars(n) = vM(n)$ for a leaf node n, and $vars(n) = \bigcup var(n^{ch})$ for an internal node n, where n^{ch} is a child of n in the oriented tree.

The *width* of a hypertree decomposition is measured by the maximum number of constraints in $cM(n)$ labeling among all nodes. The *hypertree width* of a CSP is the minimum width over all its hypertree decompositions.

References

1. F. Bacchus and P. van Beek. On the conversion between non-binary and binary constraint satisfaction problems. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and of the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, pages 311–318, Menlo Park, 26–30 1998. AAAI Press.
2. R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1989.
3. Rina Dechter. On the expressiveness of networks with hidden variables. In *AAAI*, pages 556–562, 1990.
4. Rina Dechter. *Constraint Processing.* Morgan Kaufmann Publishers, Inc., San Mateo, California, 2003.
5. Rina Dechter and Judea Pearl. Network–based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34:1–38, 1987.
6. Michael R. Garay and David S. Johnson. *Computers and Intractability.* W. H. Freeman and Company, New York, 1991.
7. G. Gottlob, L. Leone, and F. Scarcello. On tractable queries and constraints. In *Proceedings Conference on Database and Expert Systems Applications, DEXA-99, Florence*, volume 1677 of *Lecture Notes in Computer Science*, page 115. Springer-Verlag, Berlin, 1999.
8. G. Gottlob, L. Leone, and F. Scarcello. A comparison of structural csp decomposition methods. *Artificial Intelligence*, 124(2):243–282, 2000.
9. G. Gottlob, L. Leone, and F. Scarcello. Hypertree decompositions: A survey. In *Proceedings 26th International Symposium on Mathematical Foundations of Computer Science, MFCS01*, volume 2136 of *Lecture Notes in Computer Science*, pages 37–57. Springer-Verlag, 2001.
10. G. Gottlob, L. Leone, and F. Scarcello. Hypertree decomposition and tractable queries. *Journal of Computer and System Sciences*, 64(3):579–627, 2002.
11. Georg Gottlob, Nicola Leone, and Francesco Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *J. Comput. Syst. Sci.*, 66(4):775–808, 2003.
12. Martin Grohe. The structure of tractable constraint satisfaction problems. In *Proceedings 31st International Symposium on Mathematical Foundations of Computer Science, MFCS06*, volume 4162 of *Lecture Notes in Computer Science*, pages 58–72. Springer-Verlag, 2006.

13. Charles Hartshorne and Paul Weiss, editors. *Collected papers of Charles Sanders Peirce*, volume III. Cambridge : Harvard University Press, 1931-66.
14. N. Robertson and P. D. Seymour. Graph minors II: Algorithmic aspects of tree-width. *J. Algorithms*, 7:309–322, 1986.
15. Francesca Rossi, Charles Petrie, and Vasant Dhar. On the equivalence of constraint satisfaction problems. In Luigia Carlucci Aiello, editor, *ECAI'90: Proceedings of the 9th European Conference on Artificial Intelligence*, pages 550–556, Stockholm, 1990. Pitman.
16. L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8:410–421, 1979.