Backtracking Procedures for Hypertree, HyperSpread and Connected Hypertree Decomposition of CSPs

Sathiamoorthy Subbarayan and Henrik Reif Andersen
The IT University of Copenhagen
Denmark
sathi,hra@itu.dk

Abstract

Hypertree decomposition has been shown to be the most general CSP decomposition method. However, so far the exact methods are not able to find optimal hypertree decompositions of realistic instances. We present a backtracking procedure which, along with isomorphic component detection, results in optimal hypertree decompositions. We also make the procedure generic; variations of which results in two new tractable decompositions: hyperspread and connected hypertree. We show that the hyperspread width is bounded by both the hypertree width and the spread cut width, which solves a recently stated open problem. In our experiments on several realistic instances, our methods find many optimal decompositions, while the previous methods can find at most one.

1 Introduction

A CSP decomposition method *DM* is *tractable* if, for a fixed k, determining whether the *DM-width* of a CSP instance is at most k takes polynomial time and all the CSP instances having *DM-width* at most k can be solved in polynomial time. A method DM_1 is *more general* than an another method DM_2 if all the CSP instance classes having bounded DM_2*-width* also have bounded DM_1*-width* but not vice-versa. Likewise, DM_1 is *better* than DM_2 if the DM_1*-width* of any instance is at most its DM_2*-width*, and for an instance its DM_1*-width* is smaller than its DM_2*-width*. Among several tractable decomposition methods [Freuder, 1985; Dechter, 2003; Gyssens *et al.*, 1994; Gottlob *et al.*, 2000], the hypertree decomposition has been shown to be the most general [Gottlob *et al.*, 2000]. But no exact procedure for hypertree decomposition has been able to handle instances of realistic sizes. This is mainly due to huge preprocessing in the existing exact methods [Gottlob *et al.*, 1999; Harvey and Ghose, 2003].

We first present a backtracking procedure for hypertree decomposition. We identify isomorphic components of a hypergraph, and use it to speed-up the procedure. We generalize the procedure, the variations of which results in two new tractable decomposition methods: HyperSpread and Connected Hypertree. Finding a tractable decomposition better than hypertree decomposition has been recently stated as an open problem [MAT, 2006]. We show that hyperspread width is bounded by both hypertree width and spread cut width, which gives a simple positive answer to the open problem. In the experiments on several instances our hypertree and connected hypertree decomposition methods result in finding optimal decompositions of many instances, while the previous methods can find optimal decompositions of at most one instance. In our experiments with a given time limit, for all the instances we are able to obtain a connected hypertree decomposition of width at least as small as the obtained hypertree decomposition. We finally state as an open problem whether the connected hypertree width of an instance equals the hypertree width.

The next section lists necessary definitions. The Section 3 presents the backtracking procedure and defines isomorphic components. The generic procedure and its variants are presented in the Section 4. The Section 5 presents the experimental results. The Section 6 concludes the paper.

2 Definitions

A *constraint satisfaction problem instance* (CSP) is a triple (V,D,C), where V is a set of variables, D is a set of domains and C is a set of constraints. A *variable* $v_i \in V$ can take values in the finite *domain* $d_i \in D$. A *constraint* $c_i \in C$ is a pair (s_i,r_i), where $s_i \subseteq V$ is the *scope* of the constraint and r_i is a *relation* over s_i. The r_i restricts the allowed combination of values the variables in s_i can be assigned. A *solution* to a CSP is an assignment of values to all the variables in the CSP, such that the assignment restricted to a scope s_i belongs to the relation r_i. A CSP is *satisfiable* if there is a solution for the CSP.

A *constraint hypergraph* of a CSP (V,D,C) is a hypergraph H = (V,E), where the set of nodes in the hypergraph V is the same as the set of variables in the CSP. For each (s_i,r_i) $\in C$, there will be an *edge* (also called a *hyperedge*) $e_i \in E$, such that e_i = s_i, i.e., $E = \{s_i \mid (s_i, r_i) \in C\}$. Hereafter, just the term hypergraph refers to a constraint hypergraph. For a set of edges $K \subseteq E$, let the term *vars*(K) = $\bigcup_{e \in K} e$, i.e., the union of variables occurring in the edges K. For a set of variables $L \subseteq V$, let the term *edgeVars*(L) = *vars*($\{e \mid e \in E, e \cap L \neq \emptyset\}$), i.e., all the variables occurring in a set of edges, where each edge in the set contains at least one variable in L.

A *tree decomposition* [Robertson and Seymour, 1986; Dechter, 2003] of a hypergraph (V,E) is a pair *TD* = (T,γ), where T = (N,A) is a tree and γ is a mapping which for each node $n \in N$ associates $\gamma(n) \subseteq V$. Additionally, a tree decomposition has to satisfy two properties: (1) $\forall e \in E.\ \exists n \in N.\ e \subseteq \gamma(n)$, and (2) $\forall v \in V$, the nodes $\{n \mid n \in N, v \in \gamma(n)\}$ induces a *connected subtree* in T. A *hypertree* [Gottlob *et al.*, 2000] of a hypergraph (V,E) is a triple (T,χ,λ), where T = (N,A) is a *rooted* tree. The χ and λ are mappings that associate for each node $n \in N$, $\chi(n) \subseteq V$ and $\lambda(n) \subseteq E$. If T' = (N', A') is a subtree of T, then $\chi(T')$ = $\bigcup_{n \in N'} \chi(n)$. Let the term T_n denote the subtree rooted at the node $n \in N$. Let the term *nodes*(T) = N. The *width* of the hypertree (T,χ,λ) is *max* $\{|\lambda(n)| \mid n \in$ *nodes*$(T)\}$.

A *hypertree decomposition* [Gottlob *et al.*, 2000] of a hypergraph H is a hypertree *HD* = (T,χ,λ), which has to satisfy the three properties: (1) (T,χ) is a tree decomposition of H, (2) $\forall n \in$ *nodes*(T), $\chi(n) \subseteq$ *vars*($\lambda(n)$), and (3) $\forall n \in$ *nodes*(T), $\chi(T_n) \cap$ *vars*($\lambda(n)$) $\subseteq \chi(n)$. The *hypertree width* of H, denoted *htw*(H), is the minimum width of all possible hypertree decompositions of H.

To simplify the presentation, we assume that H is connected. Given a set of variables $P \subseteq V$, and two variables $m,n \in V$, the variables m and n are *[P]-adjacent* if $\exists e \in E.\ m,n \in (e \backslash P)$. A set of variables $Q \subseteq V$ is *[P]-connected*, if for any two variables $m,n \in Q$, there is a sequence of variables $m = l_1, l_2, \ldots, l_{r-1}, l_r = n$, such that, for each $i \in [1, r-1]$, l_i and l_{i+1} are [P]-adjacent. A set of variables $Q \subseteq V$ is a *[P]-component* (or *[P]-comp* in short), if it is a maximal [P]-connected subset of V. Let *[P]-components* (or *[P]-comps* in short) denote the set containing every *[P]-component*.

Given an integer k and $\alpha \subseteq E$, α is a *k-edge*, if $|\alpha| \leq k$. Let the set *k-edges* = $\{\alpha \mid \alpha \subseteq E, |\alpha| \leq k\}$, i.e, the set containing every k-edge. Given a k-edge α, an *α-comp* is a [*vars*(α)]-comp. The size of *α-comps*, which contains every α-comp, is at most $|E|$, as for every α-comp *edgeVars*(α-comp) will uniquely contain at least one edge $e \in E$. Given a set of variables $Q \subseteq V$, the *sub-hypergraph* induced by Q is the hypergraph (V',E'), where V' = Q and E' = $\{e \cap Q \mid e \in E\}$. Let a pair of the form (α,α-comp), where α is a k-edge, be called a *k-edge-component*.

3 The Backtracking Procedure

Our procedure is shown in Figure 1. The main procedure *IsDecomposable* takes the pair ((V,E), k) as input. The output is a hypertree decomposition if *htw*((V,E)) $\leq k$, otherwise the message 'Not-k-decomposable'.

Given a k-edge-component (α,α-comp), we say the pair (α,α-comp) is *k-decomposable* iff the sub-hypergraph induced by the nodes in *edgeVars*(α-comp) has a hypertree decomposition HD' of width at most k, such that, if the root node of HD' is r, then *edgeVars*(α-comp) $\setminus$ α-comp $\subseteq \chi(r)$. Essentially, the notion k-decomposable tells us whether the sub-hypergraph can be decomposed into a HD' of width at most k, such that HD' can be used as a block in building a decomposition for the whole hypergraph. Hence, the pair ($\emptyset$,V) is k-decomposable iff the *htw*((V,E)) $\leq$ k. We say that the (α,α-comp) is k-decomposable *by using* the k-edge β if there exists a HD' with $\lambda(r) = \beta$.

The procedure *AddGood* is used to add a tuple (α,α-comp,β,cCs) to the set *goods*, if the pair (α,α-comp) is k-decomposable by using the k-edge β. The set *cCs* in the tuple, denotes the elements of β-comps contained in α-comp. Similarly, the procedure *AddNoGood* is used to add a pair (α,α-comp) to the set *noGoods*, if the pair (α,α-comp) is not k-decomposable. The procedure *IsGood*(α,α-comp) returns true iff a tuple of the form (α,α-comp,β,cCs) exists in the set *goods*. Similarly, the procedure *IsNoGood*(α,α-comp) returns true iff the pair (α,α-comp) exists in the set *noGoods*.

The procedure *Decompose* takes a (α,α-comp) pair as input and returns true iff (α,α-comp) is k-decomposable. The line(1) of the procedure finds *cv*, the set of variables through which the α-comp is connected to the rest of the variables in the hypergraph. The loop that begins at line(2) iterates for every k-edge β, such that (*vars*(β) $\cap$ α-comp $\neq \emptyset$) and (cv $\subseteq$ *vars*(β)). That is the β should be such that *vars*(β) contains at least one variable in α-comp, and *cv* is contained in *vars*(β). The set *cCs* in line(3) is the set of elements of β-comps, which are contained in α-comp and hence necessary for consideration by the *Decompose* procedure.

The internal loop starting at line(5) of *Decompose* checks whether a (β,cC) pair is k-decomposable. The variable *success* will be true at the line(10) iff, for each cC $\in$ cCs, the pair (β,cC) is k-decomposable. A call to *Decompose*(β,cC) at line(8) is made iff both the calls to *IsNoGood*(β,cC) and *IsGood*(β,cC) return false. This makes sure that for any (β,β-comp) pair, the call *Decompose*(β,β-comp) is made at most once. At line(10), if the variable *success* is set to true, then the tuple (α,α-comp,β,cCs) is added to the set *goods* and the procedure returns true. Otherwise, the loop at line(2) continues with an another k-edge choice for β. If all choices for β at line(2) is exhausted without any success, then the procedure adds the (α,α-comp) pair to the set *noGoods* at line(11) and then returns false.

In case the call *Decompose*($\emptyset$,V) at line(1) of the procedure *IsDecomposable* returns true, then a hypertree decomposition (HD) is built and returned. Otherwise, a message 'Not-k-decomposable', meaning the hypertree width of (V,E) is larger than k, is returned. At line(2) of the procedure a root node r is created and a tuple of the form ($\emptyset$,V,β,cCs) is obtained from the set *goods*. The third entry β in the tuple corresponds to the k-edge used during the first call of the recursive *Decompose* procedure. Hence, $\lambda(r)$ is set to β. By the definition of hypertree decomposition, for a root node r, $\chi(r)$ = *vars*($\lambda(r)$). Hence, the $\chi(r)$ is set to *vars*(β). In the lines(4-5), for each cC $\in$ cCs, a child node of r is created by the call *BuildHypertree* (β,cC,r). The procedure *BuildHypertree* in fact recursively calls itself and extracts, from the set *goods*, the subtree rooted at a child vertex of r. At line(6), the procedure returns a hypertree decomposition.

Theorem 3.1. *IsDecomposable(H,k) returns a hypertree decomposition iff* htw(H) $\leq k$.

The proof of the above theorem follows from the fact that, our procedure is essentially a backtracking version of the *opt-k-decomp* [Gottlob *et al.*, 1999] procedure for hypertree de-

```
Set goods = $\emptyset$;  Set noGoods = $\emptyset$;  Hypertree HD = ($T$,$\lambda$,$\chi$)

Decompose ($\alpha$,$\alpha$-comp)
1:  cv = vars($\alpha$) $\cap$ edgeVars($\alpha$-comp)
2:  $\forall\beta \in k$-edges. (vars($\beta$) $\cap$ $\alpha$-comp $\neq \emptyset$). (cv $\subseteq$ vars($\beta$))
3:       cCs = {cC | cC $\in$ $\beta$-comps, cC $\subseteq$ $\alpha$-comp }
4:       success = true
5:       $\forall$ cC $\in$ cCs
6:            if (IsNoGood($\beta$,cC)) then success = false
7:            else if (IsGood($\beta$,cC)) then continue
8:            else if (Decompose($\beta$,cC)) then continue
9:            else success = false
10:      if (success) then AddGood($\alpha$,$\alpha$-comp,$\beta$,cCs);  return true
11:  AddNoGood($\alpha$,$\alpha$-comp);  return false

IsDecomposable (($V$,$E$),$k$)
1:  if (Decompose($\emptyset$,$V$)) then
2:       Add root node $r$ to $T$;  Find ($\emptyset$,$V$,$\beta$,cCs) $\in$ goods
3:       $\lambda(r) = \beta$;  $\chi(r)$ = vars($\beta$)
4:       $\forall$ cC $\in$ cCs
5:            BuildHypertree ($\beta$,cC,$r$)
6:       return HD
7:  else  return 'Not-$k$-Decomposable'

IsGood ($t_1$,$t_2$)
1:  $\forall$ ($t_w$,$t_x$,$t_y$,$t_z$) $\in$ goods
2:       if ($t_1$,$t_2$) = ($t_w$,$t_x$) then return true
3:  return false

IsNoGood ($t_1$,$t_2$)
1:  $\forall$ ($t_x$,$t_y$) $\in$ noGoods
2:       if ($t_1$,$t_2$) = ($t_x$,$t_y$) then return true
3:  return false

AddGood ($t_1$,$t_2$,$t_3$,$t_4$)
1:  goods = goods $\cup$ $\{(t_1, t_2, t_3, t_4)\}$

AddNoGood ($t_1$,$t_2$)
1:  noGoods = noGoods $\cup$ $\{(t_1, t_2)\}$

BuildHypertree ($\alpha$,$\alpha$-comp,$s$)
1:  Find ($\alpha$,$\alpha$-comp,$\beta$,cCs) $\in$ goods
2:   Add a node $t$ to $T$ as a child node of $s$
3:  $\lambda(t) = \beta$;  $\chi(t)$ = (vars($\beta$) $\cap$ edgeVars($\alpha$-comp))
4:  $\forall$ cC $\in$ cCs
5:       BuildHypertree ($\beta$,cC,$t$)
```

Figure 1: The pseudo code of the backtracking procedure for hypertree decomposition.

composition. The opt-k-decomp procedure suffers from a huge pre-processing cost. For a given k, the opt-k-decomp implicitly finds all possible decompositions of width at most k and extracts the optimal one. But our backtracking procedure stops as soon as it discovers a decomposition of width at most k. Hence, our procedure will be able to stop quickly when there are several decompositions of width at most k.

Theorem 3.2. *The time complexity of* IsDecomposable*(H,k) is O($|E|^{3k+3}$).*

Proof. The time complexity is dominated by *Decompose*. During a call to *Decompose*, the loop over choices for β iterates at most O($|E|^k$). The $|cCs|$ is at most $|E|$. Since each k-edge results in at most $|E|$ k-edge-components, the total number of k-edge-components is O($|E|^{k+1}$). The cost of each call to *IsNoGood/IsGood* will take O($|E|^{k+1}$) time, as the size of goods/noGoods can be at most the number of k-edge-components. We can ignore the cost for calls to *AddGood/AddNoGood* as their contribution to the complexity is dominated. Hence, the cost for each call to *Decompose* is O($|E|^{2k+2}$). There can be at most O($|E|^{k+1}$) calls to *Decompose*, at most once for each k-edge-component. Hence, the time complexity of the procedure *IsDecomposable* is O($|E|^{3k+3}$). □

Given two k-edge-components (α,comp) and (α',comp'), they are *isomorphic* if comp = comp'.

Theorem 3.3. *Given two isomorphic pairs (α,comp) and (α',comp'), (α,comp) is k-decomposable iff (α',comp') is k-decomposable.*

Proof. By definition comp = comp'. Hence, the only difference between the calls *Decompose*(α,comp) and *Decompose*(α',comp') are the first parameters: α and α'. In *Decompose* the distinction between α and α' is of importance only at line(2), where the set *cv* is calculated. We now show that *cv* is independent of α. Since (α,comp) is a k-edge-component, by definition, $\forall e \in E.\ e \subseteq$ *edgeVars*(comp) $\Rightarrow (e\backslash\text{comp}) \subseteq \alpha$. Also, comp $\cap$ *vars*(α) $= \emptyset$. This implies that, during the call *Decompose*(α,comp), the set *cv* = *edgeVars*(comp)$\backslash$(comp). Therefore, the set *cv* remains the same during both the calls *Decompose*(α,comp) and *Decompose*(α',comp'). Hence, the proof. □

For each set of isomorphic components we need to make at most one call to *Decompose*. Hence, by using isomorphism the number of calls to *Decompose* may reduce significantly.

4 Variants of Hypertree Decomposition

The Figure 2 presents the pseudo code for a generic decomposition procedure *IsDecomposableGe*, based on the hypertree framework. There are two changes in the generic procedure from the earlier procedure in Figure 1. First change is that, the generic version is able to detect isomorphism, so that a call to *Decompose* is made at most once for a set of isomorphic components. The second change is in line(2) of the procedure *Decompose*, where the code $\beta \in k$-edges in the old procedure has been replaced with $(\beta, Q_\beta) \in \Omega$ in the generic version. Each element in the set Ω is a pair of the form (β, Q_β), where $\beta \in k$-edges and $Q_\beta \subseteq$ *vars*(β). In all the places where *vars*(β) was used in the old procedure, Q_β is used in the generic procedure. The set Ω defines, for each $\beta \in k$-edges, the allowed subsets of *vars*(β) which can be used for decomposition. Essentially, by allowing subsets of *vars*(β) to be considered for obtaining a decomposition we obtain a generic procedure. Note, the procedures *IsGood*, *AddGood*, *IsNoGood*, and *AddNoGood* are not listed in Figure 2 as the old versions of them can be naturally modified to work in the generic framework. Let the set $\Omega_{HTD} = \{(\alpha, vars(\alpha)) \mid \alpha \in k\text{-edges}\}$. We have the following theorem.

Theorem 4.1. *When* $\Omega = \Omega_{\text{HTD}}$, IsDecomposableGe*($H$,$k$) returns a hypertree decomposition iff* htw(H) $\leq k$.

```
Set goods = $\emptyset$; Set noGoods = $\emptyset$; Hypertree D = ($T$,$\lambda$,$\chi$)

Decompose (comp)
1:  cv = edgeVars(comp) $\setminus$ comp
2:  $\forall(\beta, Q_\beta) \in \Omega$. ($Q_\beta \cap$ comp $\neq \emptyset$). (cv $\subseteq Q_\beta$)
3:      cCs = {cC | cC $\in [Q_\beta]$-comps, cC $\subseteq$ comp }
4:      success = true
5:      $\forall$ cC $\in$ cCs
6:          if (IsNoGood(cC)) then success = false
7:          else if (IsGood(cC)) then continue
8:          else if (Decompose(cC)) then continue
9:          else success = false
10:     if (success) then AddGood(comp,$\beta$,$Q_\beta$,cCs); return true
11: AddNoGood(comp); return false
```

```
IsDecomposableGe (($V$,$E$),$k$)
1: if (Decompose($V$)) then
2:     Add root node $r$ to $T$; Find ($V$,$\beta$,$Q_\beta$,cCs) $\in$ goods
3:     $\lambda(r) = \beta$; $\chi(r) = Q_\beta$
4:     $\forall$ cC $\in$ cCs
5:         BuildHypertree (cC,$r$)
6:     return D
7: else return 'Not-$k$-Decomposable'

BuildHypertree (comp,$s$)
1: Find (comp,$\beta$,$Q_\beta$,cCs) $\in$ goods
2:  Add a node $t$ to $T$ as a child node of $s$
3: $\lambda(t) = \beta$; $\chi(t)$ = ($Q_\beta \cap$ edgeVars(comp))
4: $\forall$ cC $\in$ cCs
5:     BuildHypertree (cC,$t$)
```

Figure 2: The pseudo code of the generic procedure for decomposition in hypertree framework.

4.1 HyperSpread Decomposition

An *unbroken guarded block (ug-block)* [Cohen *et al.*, 2005] is a pair (α,Q_α), where $\alpha \subseteq E$ and $Q_\alpha \subseteq$ *vars*(α), satisfying two conditions: (1) $\forall e_1, e_2 \in \alpha.\ e_1 \cap e_2 \subseteq Q_\alpha$, and (2) each $[Q_\alpha]$-comp has non-empty intersection with at most one [*vars*(α)]-comp. The size of an ug-block (α,Q_α) is $|\alpha|$.

For some $\alpha \subseteq E$ and $v \in V$, let the *label*, $L_\alpha(v) = \{$cC $|$ cC $\in [\alpha]$-comps, $v \in$ *edgeVars*(cC)$\}$. An ug-block (α,Q_α) is *canonical* if $\forall e \in \alpha.\ \forall v_1, v_2 \in e \backslash Q_\alpha.\ L_\alpha(v_1) = L_\alpha(v_2)$. Let Ω_{HSD} be the set of all the at most k sized canonical ug-blocks of (V,E). Note, by the definition of canonical ug-blocks, $\Omega_{HTD} \subseteq \Omega_{HSD}$. Given k and a canonical ug-block (α,Q_α) with $|\alpha| \leq k$, let a triple of the form (α,Q_α,[Q_α]-comp) be called a *k-spread-component*. It has been shown in [Cohen *et al.*, 2005] that, there exists at most $(|E|+1)^{k+2}$ k-spread-components. The notion of *isomorphism* can be naturally extended to k-spread-components. Given two k-spread-components (α,Q_α,comp) and (α',$Q_{\alpha'}$,comp'), they are *isomorphic* when comp = comp'.

A *spread cut* [Cohen *et al.*, 2005] of a hypergraph H is a hypertree SC = (T,λ,χ), satisfying the two conditions: (1) (T,χ) is a tree decomposition of H, and (2) $\forall n \in$ *nodes*(T), ($\lambda(n)$,$\chi(n)$) is a canonical ug-block. The *spread cut width* of H, denoted *scw*(H), is the minimum width of all possible spread cuts of H.

A *hyperspread decomposition* of a hypergraph H is a hypertree HSD = (T,λ,χ), satisfying the two conditions: (1) (T,χ) is a tree decomposition of H, and (2) $\forall n \in$ *nodes*(T). $\exists(\alpha, Q_\alpha) \in \Omega_{HSD}$. $\lambda(n) = \alpha$. $\chi(n) = Q_\alpha \cap \chi(T_n)$. The *hyperspread width* of H, denoted *hsw*(H), is the minimum width of all possible hyperspread decompositions of H.

Due to a variant of *IsDecomposableGe*, we have:

Theorem 4.2. *When* $\Omega = \Omega_{\text{HSD}}$, IsDecomposableGe*(H,k) returns a hyperspread decomposition iff* hsw(H) $\leq k$.

By arguments similar to those of Theorem 3.2, we have:

Theorem 4.3. *When* $\Omega = \Omega_{\text{HSD}}$*, the time complexity of* IsDecomposableGe*(H,k) is O(*$(|E|+1)^{3k+7}$*).*

Given a hyperspread decomposition of a CSP, the CSP can be solved by the same algorithm for CSP solving using hypertree decompositions [Gottlob *et al.*, 2000]. Also, for a fixed k, determining whether *hsw* of CSP is $\leq k$ takes polynomial time. Hence, the hyperspread decomposition is tractable.

We now state that the hyperspread width is bounded by both hypertree width and the spread cut width.

Theorem 4.4. hsw(H) $\leq$ htw(H) *and* hsw(H) $\leq$ scw(H)

Proof. The last two conditions in the definition of hypertree decomposition implies that: $\forall n \in$ *nodes*(T). $\exists(\alpha, Q_\alpha) \in \Omega_{HTD}$. $\lambda(n) = \alpha$. $\chi(n) = Q_\alpha \cap \chi(T_n)$. Since for any k, $\Omega_{HTD} \subseteq \Omega_{HSD}$, each hypertree decomposition is also a hyperspread decomposition. Hence, *hsw*(H) $\leq$ *htw*(H).

The second condition in the definition of spread cut implies that: $\forall n \in$ *nodes*(T). $\exists(\alpha, Q_\alpha) \in \Omega_{HSD}$. $\lambda(n) = \alpha$. $\chi(n) = Q_\alpha$. Therefore, each spread cut is also a hyperspread decomposition. Hence, *hsw*(H) $\leq$ *scw*(H). □

Recently in [MAT, 2006], the existence of a tractable decomposition better than hypertree decomposition was stated as an open problem. In [Cohen *et al.*, 2005], a family of hypergraphs H_n, for $n = 1, 2, \ldots$, was presented for which the hypertree width of H_n is at least $3n$, while the spread cut width is at most $2n$. Note, it is not known whether for any H, *scw*(H) $\leq$ *htw*(H). Since for any H, *hsw*(H) $\leq$ *htw*(H) and *hsw*(H_n) $\leq$ *scw*(H_n) $<$ *htw*(H_n) , hyperspread decomposition gives a simple positive answer to the open problem.

4.2 Connected Hypertree Decomposition

A *connected hypertree decomposition* is a hypertree decomposition CHTD = (T,λ,χ), satisfying two conditions: (1) for the root node r of T, $|\lambda(r)| = 1$, and (2) if $c \in$ *nodes*(T) is a non-root node and the parent node of c is p, then $\forall e \in \lambda(c)$. $\exists v \in \chi(c) \cap \chi(p)$. $v \in e$. The second condition states that for any non-root node c with parent node p, each edge e in $\lambda(c)$ should have at least one variable that is common to both $\chi(c)$ and $\chi(p)$. The two conditions in the definition results in a significant reduction of the branching choices in the procedure *Decompose*. We show empirical results in the next section to support such reduction. Let the *connected hypertree width* of a hypergraph H, denoted *chtw*(H), be the minimum width of the all possible connected hypertree decompositions of H. By definitions, *htw*(H) $\leq$ *chtw*(H). It is an open problem whether *chtw*(H) = *htw*(H).

Let $\Omega_{CHTD}(cv) = \{(\alpha, vars(\alpha)) \mid \alpha \in k\text{-edges}, (cv = \emptyset) \Rightarrow (|\alpha| = 1), (cv \neq \emptyset) \Rightarrow (\forall e \in \alpha.\ \exists v \in cv.\ v \in e)\}$,

Instance			Exact Methods					Heuristic
	Size	Arity	CHTD	CHTD-NoIso	BE+CHTD	HTD	HTD-NoIso	BE
Name	$\lvert V\rvert, \lvert E\rvert$	μ, Max	Time,Width	Time, Width	Time, Width	Time, Width	Time, Width	μT, μW, MinW
1-32.1.cp	110, 222	3.15, 8	64, 4*	503, 4*	64, 4*	1800, 4	1800, 4	0.12, 4, 4
FISCHER1-1-fair.smt	308, 284	2.89, 3	1800, 64	1800, 64	1800, 17	1800, 82	1800, 82	1, 18.4, 17
FISCHER1-6-fair.smt	1523, 1419	2.90, 3	1800, 208	1800, 208	1800, 64	1800, 456	1800, 456	66, 67.4, 64
baobab1.dag	145, 84	3.61, 6	1800, 6	1800, 6	1800, 6	1800, 6	1800, 6	0.14, 8, 8
baobab3.dag	187, 107	3.55, 6	660, 6*	1800, 6	660, 6*	1800, 6	1800, 6	0.17, 7, 7
complex.cp	414, 849	3.46, 17	1800, 10	1800, 10	1800, 9	1800, 13	1800, 13	4, 9.4, 9
das9201.dag	204, 82	4, 7	4, 5*	21, 5*	4, 5*	1139, 5*	1800, 5	0.14, 6, 6
isp9601.dag	247, 104	3.56, 7	0.33, 4*	0.7, 4*	0.35, 4*	59, 4*	312, 4*	0.12, 6, 6
isp9603.dag	186, 95	3.55, 7	196, 7*	1800, 7	196, 7*	1800, 7	1800, 8	0.15, 9, 9
large-partial.cp	184, 371	3.19, 10	5, 3*	11, 3*	5, 3*	50, 3*	172, 3*	0.35, 4, 4
large2.cp	77, 130	3.16, 8	22, 4*	159, 4*	22, 4*	308, 4*	1800, 4	0.08, 5, 5
renault.cp	99, 113	4.91, 10	0.31, 2*	0.38, 2*	0.4, 2*	0.19, 2*	0.25, 2*	0.28, 2.6, 2
# optimal			**8**	6	**8**	5	3	n/a

Table 1: Experimental results. The mean (μ) and the maximum arity of the instances are also listed.

where *cv* $\subseteq V$ corresponds to the connection variables in the procedure *Decompose*. As an another variant of the generic decomposition procedure, we state the following theorem.

Theorem 4.5. *When* $\Omega = \Omega_{\text{CHTD}}(\text{cv})$*, a connected hypertree decomposition is returned by* IsDecomposableGe(H, k) *iff* chtw$(H) \leq k$.

Unlike the previous two variants of the generic decomposition procedure, when $\Omega = \Omega_{CHTD}(cv)$, the set Ω is newly defined during each call to *Decompose* based on the connection variables *cv*.

Since connected hypertree decompositions are subsets of hypertree decompositions, connected hypertree decomposition is tractable.

5 Experiments

We have implemented the presented backtracking procedures for both hypertree decomposition and connected hypertree decomposition. We have not implemented the hyperspread decomposition procedure, which does not seem easy to implement. We plan to do it in the future. All of our experiments are done on an Intel Xeon 3.2Ghz machine, running Linux and with 4GB RAM. We use 12 instances in our experiments. They are constraint hypergraphs from different sources: five from configuration problems (*.cp), five from fault trees (*.dag), and two from SMT instances (*.smt). Our tools and instances are available online[1].

In our implementation, for a given Ω and $H = (V, E)$, we find an optimal decomposition by a sequence of queries to the procedure *IsDecomposableGe*(*H*,*k*). The parameter *k* will take values in the sequence $k_1, k_2, \ldots, k_{l-1}, k_l$. In the sequence, $k_1 = |E|$. If the query *IsDecomposableGe*(*H*,k_i) returns a decomposition of width k', then $k_{i+1} = k' - 1$. If the query *IsDecomposableGe*(*H*,k_i) returns the message 'Not-k-Decomposable', then $l = i$. The optimal width will be $k_l + 1$. We found that the l is often very small due to jumps the sequence could make. The results of our experiments are listed in Table 1. All the experiments in the table are done with a time limit of 1800 CPU seconds. The legend 'CHTD' ('HTD') refers to the experiments with $\Omega = \Omega_{CHTD}$ ($\Omega = \Omega_{HTD}$). The legend 'CHTD-NoIso' ('HTD-NoIso') refers the 'CHTD' ('HTD') procedure without detection of isomorphic components. For each method, we list the best width obtained. In the cases we are able to prove optimality, the entries in the width columns will be of the form $k*$. We also tested the heuristic 'BE' in the *HTDECOMP*[2] tool. The BE uses tree decomposition heuristics and set covering heuristics to obtain a heuristic *Generalized Hypertree Decomposition* (GHTD) [Gottlob *et al.*, 2003]. In case of a GHTD, the third condition in the definition of hypertree decomposition may be violated. Since BE is randomized, we ran experiments using BE five times, and list in the table the mean time (μT), the mean width (μW) and the best width (MinW). We also did experiments on a hybrid BE+CHTD, in which five runs of BE will be used to first obtain five GHTDs. If the best width of the GHTDs so obtained is k_{be}, then the CHTD procedure will be invoked with $k_1 = k_{be}$. Note, the best width decomposition found by the hybrid BE+CHTD may be a GHTD, whose width can be smaller than *htw*. Although, the HTDECOMP tool has heuristic choices other than BE, we choose BE since we think it is the best choice. The BE+CHTD combination can be naturally extended with any initialization heuristic. The last row, #*optimal* of the table lists the number of proven optimal width decompositions obtained by each method. Note, to prove optimality, the query corresponding to k_l needs to return 'Not-k-decomposable' within the time limit.

All the exact methods listed in the table are our contributions. The experiments using the implementations[3] of the previous exact methods *opt-k-decomp*, and an improvement over opt-k-decomp, *red-k-decomp* [Harvey and Ghose, 2003], either exceeded time limit or aborted in all cases except renault.cp, in which red-k-decomp took 1257 seconds to find an optimal decomposition. In fact, both the experiments using the opt-k-decomp and red-k-decomp tools consisted of only one query to the tool. For an instance H, if the 'CHTD' finds a decomposition of width k' in 1800 seconds, then the opt-k-decomp and red-k-decomp are just asked to find a decomposition of width k'. Hence, the experiments are done in a favorable setting to the previous methods.

From the table, we can observe that the CHTD procedure

[1]http://www.itu.dk/people/sathi/connected-hypertree/

[2]http://www.dbai.tuwien.ac.at/proj/hypertree/downloads.html

[3]http://www.dsl.uow.edu.au/~harvey/research_hypertrees.shtml

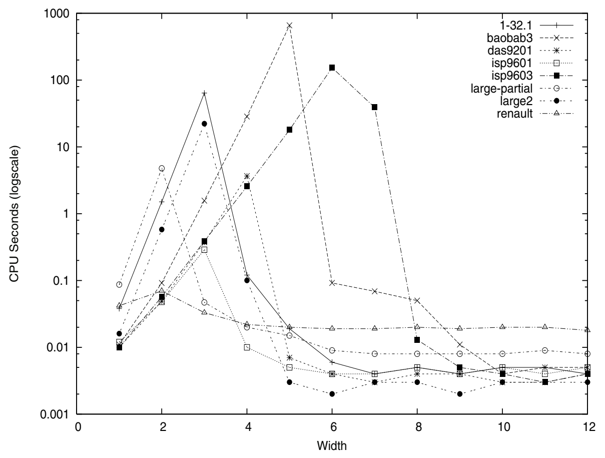

Figure 3: Width vs. CPU-time (y axis in logscale). The empirical complexity peaks at $k * -1$.

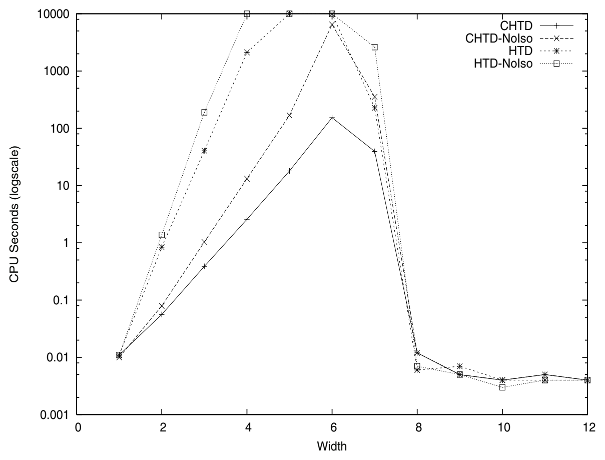

Figure 4: The effect of detecting isomorphism and CHTD vs. HTD in *isp9603* instance (time limit 10000 seconds).

gives more optimal decompositions than others. In none of the instances the HTD could find a better decomposition than that of CHTD. In three instances the BE finds GHTDs of width smaller than those by the exact methods. For use in practice the hybrid BE+CHTD seems the best choice, as it results in best decomposition in all the cases.

In Figure 3, we plot the time taken for the queries *IsDecomposableGe*(H,k) with $\Omega = \Omega_{CHTD}$, $k \in \{1, 2, \ldots, 11, 12\}$ and H being one among the eight instances for which CHTD succeeds in obtaining optimal decompositions within 1800 seconds. We can observe that, for each instance H, the cost of queries until $k * -1$, where $k*$ is *chtw*(H), increases exponentially. The cost suddenly decreases at $k*$ and continues to decrease exponentially.

In Figure 4, we plot the results of queries on the four different exact procedures for the instance *isp9603*, with time limit 10000 seconds. The plot shows a huge benefit of identifying isomorphism. The plot also shows an exponential difference between the runtime of CHTD and HTD which is due to the branching restrictions in CHTD. The HTD and HTD-NoIso timed out for some queries.

6 Conclusion

We have presented a backtracking procedure for hypertree decomposition and defined isomorphism to speed-up the procedure. We generalized the procedure, the variants of which results in two new tractable decompositions: hyperspread and connected hypertree. We have shown that hyperspread width is bounded by both hypertree width and spread cut width, which solves a recently stated open problem. The experiments using our methods are able to find optimal decompositions of several instances.

Avenues for future work include: implementation of hyperspread decomposition, determining *chtw* $=$ *htw*?, identifying other tractable variants, and comparison with exact methods for tree decomposition like [Gogate and Dechter, 2004].

References

[Cohen *et al.*, 2005] D. Cohen, P. Jeavons, and M. Gyssens. A unified theory of structural tractability for constraint satisfaction and spread cut decomposition. In *IJCAI*, pages 72–77, 2005.

[Dechter, 2003] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

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

[Gogate and Dechter, 2004] V. Gogate and R. Dechter. A complete anytime algorithm for treewidth. In *UAI*, pages 201–208, 2004.

[Gottlob *et al.*, 1999] G. Gottlob, N. Leone, and F. Scarcello. On tractable queries and constraints. In *DEXA*, pages 1–15, 1999.

[Gottlob *et al.*, 2000] G. Gottlob, N. Leone, and F. Scarcello. A comparison of structural CSP decomposition methods. *AIJ*, 124(2):243–282, 2000.

[Gottlob *et al.*, 2003] G. Gottlob, N. Leone, and F. Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *Journal of computer and system sciences*, 66(4):775–808, 2003.

[Gyssens *et al.*, 1994] M. Gyssens, P. G. Jeavons, and D. A. Cohen. Decomposing constraint satisfaction problems using database techniques. *AIJ*, 66(1):57–89, 1994.

[Harvey and Ghose, 2003] P. Harvey and A. Ghose. Reducing redundancy in the hypertree decomposition scheme. In *ICTAI*, pages 474–481, 2003.

[MAT, 2006] Open Problems List - MathsCSP Workshop (version 0.3), Oxford. 2006.

[Robertson and Seymour, 1986] N. Robertson and PD Seymour. Graph minors. II: Algorithmic aspects of tree-width. *Journal of algorithms*, 7(3):309–322, 1986.