Journal of Engineering and Technology for Industrial Applications

ITEGAM-JETIA

ISSN ONLINE: 2447-0228

Manaus, v.11 n.51, p. 72-79. January/February., 2025.
DOI: https://doi.org/10.5935/jetia.v11i51.1415

RESEARCH ARTICLE

OPEN ACCESS

SOLVING NON-BINARY CONSTRAINTS SATISFACTION PROBLEMS USING GHD AND RESTART

Fatima Ait Hatrit[1] and Kamal Amroun[2]

[1,2]Université de Bejaia, Faculté des Sciences Exactes, Laboratoire d'Informatique Médicale et des Environnements Dynamiques et intelligents (LIMED), Algeria.

[1]http://orcid.org/0000-0002-0072-1348 , [2]http://orcid.org/0000-0002-4259-2783

Email: fatima.aithatrit@univ-bejaia.dz, kamal.amroun@univ-bejaia.dz

ARTICLE INFO

Article History
Received: November 19, 2024
Revised: December 20, 2024
Accepted: January 15, 2025
Published: January 30, 2025

Keywords:
Constraint Satisfaction Problems, Generalized Hypertree Decomposition, Restart-FC-GHD+NG+DR, Solving.

ABSTRACT

The non-binary instances of the Constraint Satisfaction Problem (CSP) could be efficiently solved if their constraint hypergraphs have small generalized hypertree widths. Several algorithms based on Generalized Hypertree Decomposition (GHD) have been proposed in the literature to solve instances of CSPs. One of these algorithms, called Forward Checking based on Generalized Hypertree Decomposition (FC-GHD+NG+DR), combines the advantages of an enumerative search algorithm with those of Generalized Hypertree Decomposition. However, like all structural decomposition methods, FC-GHD+NG+DR depends on the order in which the clusters are processed. In this paper, we propose a new version of the FC-GHD+NG+DR algorithm with a restart technique that allows changing the order of the nodes of GHD to improve performance. The experiments carried out are very promising, particularly on the satisfiable instances where we achieved better results using the restart method in 52.63% of the modified Renault satisfiable benchmarks and an average time resolution of ≈ 0 for the normalized Pret and normalized Dubois benchmarks.

I. INTRODUCTION

Constraint Satisfaction Problems (CSPs) are a fundamental class of problems in artificial intelligence and operations research. They involve a set of variables, each associated with a domain of possible values, and a set of constraints that restrict the simultaneous assignment of these values. Solving CSPs requires finding an assignment that satisfies all constraints. These problems are widely applied in domains such as activity planning and scheduling problems [1] and allocation problem [2]. CSPs also play a pivotal role in computational complexity research, serving as a foundation for classifying the complexity of problems in algebraic and logical frameworks [3], [4].

Despite their importance, CSPs are inherently challenging due to their NP-complete nature, often requiring an exhaustive search of the solution space. The standard method for solving CSPs is backtracking, which systematically explores a search tree to find solutions. While backtracking guarantees correctness, its exponential time complexity in the worst case makes it impractical for large or complex problem instances.

To address these limitations, researchers have developed structural decomposition methods, which aim to divide a CSP into smaller, independent sub-problems. Techniques such as bounded fractional hypertree width [5] and hybrid width parameters [6] have proven effective in reducing computational complexity. Generalized Hypertree Decomposition (GHD)-based algorithms are particularly noteworthy, leveraging problem structure to guide the exploration of solution spaces [7-9]. Among these, the Forward Checking guided by GHD, FC-GHD algorithm has been widely studied. Extensions such as FC-GHD+NG (exploiting structural NoGoods) and FC-GHD+NG+DR (introducing dynamic subtree reordering) have significantly improved its performance [7]. Another promising strategies for enhancing CSP solvers is exploiting data mining techniques for compressing table constraints [10], the use of restart methods, which periodically restart the search process after a certain number of failures. These methods adaptively manage variable and node ordering, as shown in [11], where restart sequences were used to optimize the selection of heuristics.

Inspired by the success of FC-GHD+NG+DR, we propose the Restart-FC-GHD+NG+DR algorithm, which dynamically adjusts cluster orders based on the number of backtracks generated. This approach mitigates excessive backtracking,

reduces unnecessary exploration, and improves solver efficiency, especially for complex and large-scale CSP instances.
Our contribution builds on the theoretical foundations of structural decomposition methods by integrating restart strategies to enhance adaptability and efficiency. The proposed algorithm optimizes the order of clusters dynamically, offering significant improvements in computational performance for diverse applications. Moreover, this work lays the foundation for integrating machine learning techniques into structural decomposition methods, enabling future solvers to predict optimal cluster orders based on problem characteristics, thereby further improving efficiency and adaptability.

The rest of the paper is organized as follows: Section II presents the technical background; Section III introduces the Restart-FC-GHD+NG+DR method; Section IV presents the experimental results; and Section V concludes the paper.

II. BACKGROUND

The notion of Constraint Satisfaction Problem (CSP) has been formally defined by [12]. A CSP instance is defined as a triplet $P = \langle X, D, C \rangle$. Where $X = \{X_i, \ldots, X_n\}$ is a finite set of n variables and $D = \{D_i, \ldots, D_n\}$ is a set of finite domains. Each variable X_i takes its value from its domain D_i. $C = \{C_1, \ldots, C_m\}$ is a set of m constraints. A constraint $C_i \in C$ on an ordered subset of variables, $C_i = (X_{i_1}, \ldots, X_{i_{a_i}})$ (a_i is called the arity of the constraint C_i), is defined by an associated relation $R_i \in \mathrm{R}$ of allowed combinations of values for the variables in C_i. Note that we take the same notation for the constraint C_i and its scope. Binary CSPs are those defined where each constraint involves only two variables, that is $\forall \mathrm{i} \in \{1, \ldots, \mathrm{m}\}$, $|C_i| = 2$. Constraints of arity greater than 2 are called non binary or n-ary. A CSP with at least one n-ary constraint is called non binary or n-ary CSP. A tuple $t \in R_i$ is a list of values $\left(v_{i_1}, \ldots, v_{i_{a_i}}\right)$ where:

$$a_i = |C_i| : v_{i_j} \in D_{i_j} \ \forall j \in \{1, \ldots, a_i\} \tag{1}$$

A solution to a CSP is an assignment of values to all the variables in X such that for each constraint C_i the assignment restricted to C_i belongs to $\mathrm{R_i}$. The constraint hypergraph associated with a CSP instance $P = \langle X, D, C \rangle$ is the hypergraph $\mathrm{H} = \langle V, E \rangle$ where the set of vertices V is the set of variables X and the set of hyperedges E are the set of constraint scopes in C. For any hyperedge $h \in E$, we denote by $var(h)$ the set of vertices of h and for any subset of hyperedges $K \subseteq E$

$$var(K) = \bigcup_{h \in K} var(h) \tag{2}$$

We denote by $var(\mathrm{H})$ the set of vertices V and by $edges(\mathrm{H})$ the set of $hyperedges\, E$. (We use the term var because the vertices of the hypergraph correspond to the variables of the CSP).

Definition 1: Hypertree

Let $\mathrm{H} = \langle V, E \rangle$ be a hypergraph. A hypertree [13] for H is a triple $\langle T, \chi, \lambda \rangle$ where $T = (N, F)$ is *a* rooted tree, and χ and λ are labelling functions which associate each vertex $p \in N$ with two sets $\chi(p) \subseteq V$ and $\lambda(p) \subseteq E$. If $T' = (N', F')$ is a subtree of T we define:

$$\chi(T') = \cup_{v \in N'} \chi(v) \tag{3}$$

We denote the set of vertices N of T by $vertices(T)$ and the root of T by $root\ (T)$. T_p denotes the subtree of T rooted at the node p and $Parent(p)$ is the parent node of p in T.

Definition 2: Hypertree Decomposition

A Hypertree Decomposition [14] of a hypergraph $\mathrm{H} = \langle V, E \rangle$ is a hypertree $HD = \langle T, \chi, \lambda \rangle$ which satisfies the following conditions:

i. For each edge $h \in E$, there exists $p \in vertices(T)$ such that:

$$var(h) \subseteq \chi(p) \tag{4}$$

ii. For each vertex $v \in V$, the set

$$\{p \in vertices\ (T) | v \in \chi(p)\} \tag{5}$$

induces a connected subtree of T;

iii. For each vertex

$$p \in vertices(T), \chi(p) \subseteq var\big(\lambda(p)\big) \tag{6}$$

iv. For each

$$p \in vertices(T), var\big(\lambda(p)\big) \cap \chi(Tp) \subseteq \chi(p) \tag{7}$$

The width of a hypertree $HD = \langle T, \chi, \lambda \rangle$ is equal to $max_{\,p \in vertices\ (T)} |\lambda(p)|$.The hypertree-width ($hw(\mathrm{H})$) of a hypergraph H is the minimum width over all its hypertree decompositions.

A hyperedge h of a hypergraph $\mathrm{H} = \langle V, E \rangle$ is strongly covered in $HD = \langle T, \chi, \lambda \rangle$ if there exists $p \in vertices(T)$ such that the vertices of h are contained in $\chi(p)$ and $h \in \lambda(p)$.

A hypertree decomposition $HD = \langle T, \chi, \lambda \rangle$ of a hypergraph H is complete if every hyperedge h of H is strongly covered in HD.

A hypertree $HD = \langle T, \chi, \lambda \rangle$ is called a Generalized Hypertree Decomposition (GHD) [15], [16] if the conditions (i), (ii) and (iii) of Definition 2 hold. The width of a Generalized Hypertree Decomposition $HD = \langle T, \chi, \lambda \rangle$ is equal to $max_{\,p \in vertices\ (T)} |\lambda(p)|$. The generalized-hypertree-width ($ghw(\mathrm{H})$) of a hypergraph H is the minimum width over all its generalized hypertree decompositions.

Remark 1. The terms node and vertex will be used interchangeably to refer to a vertex of T.

Example 1. Let $P = \langle X, D, C \rangle$ be a CSP instance defined as follows.

- $X = \{X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, X_{10}, X_{11}, X_{12}, X_{13}, X_{14}, X_{15}, X_{16}, X_{17}\}$ is the set of variables,
-
    - $D = \{D_1, \ldots, D_{17}\}$ where $D_i = \{0,1\}$ is the domain of the variable X_i $\forall i \in \{1, \ldots, 17\}$,
    - $C = \{C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9, C_{10}\}$ is the set of constraints.

Figure 1 is the constraint hypergraph associated with P and Figure 2 is one of its Generalized hypertree decompositions. The width of the decomposition is 3.

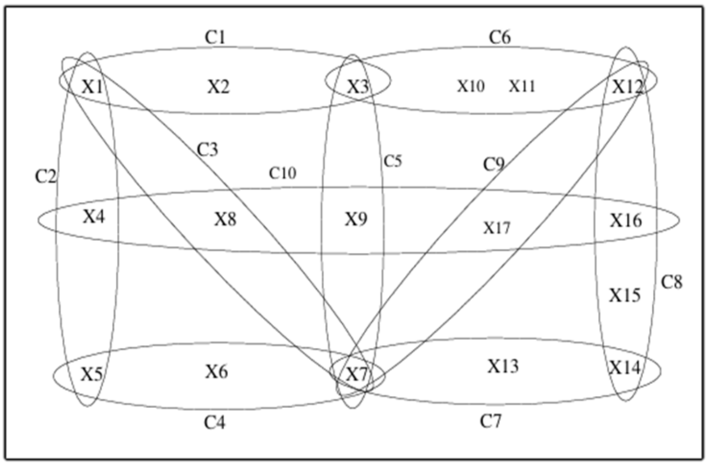

Figure 1: The constraint hypergraph of the CSP instance of Example 1.
Source: Authors, (2025).

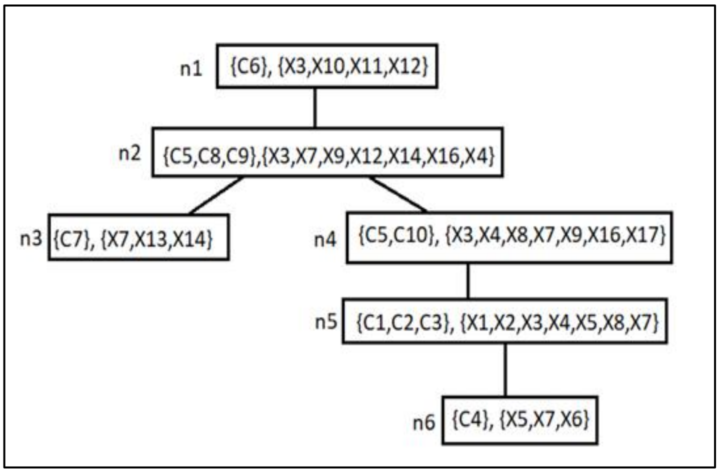

Figure 2: A 3-width generalized hyper tree decomposition of the constraint hyper graph of Example1.
Source: Authors, (2025).,333333

Definition 3: Nogood
A Nogood [17] is an inconsistent partial assignment that cannot be extended to a global solution. A minimal Nogood is any Nogood that is not itself composed of another Nogood.

Definition 4: Subproblem
Let $\mathrm{n_i}$ be a node of a GHD. The subproblem [7] associated with $\mathrm{n_i}$ is a CSP $< \mathrm{X_{n_i}}, \mathrm{D_{n_i}}, \mathrm{C_{n_i}} >$ where $\mathrm{X_{n_i}} = \chi_{\mathrm{n_i}}$, $\mathrm{D_{n_i}}$ is the set of domains defined in the original CSP for the variables in $\mathrm{X_{n_i}}$ and

$$\mathrm{C_{n_i}} = \lambda_{\mathrm{n_i}} . \mathrm{P_{n_i}} \tag{8}$$

Is denotes the subproblem associated with n_i and $sol(P_{n_i})$ denotes the current solution of P_{n_i}.

II.1 THE FC-GHD+NG+DR ALGORITHM

The FC-GHD+NG+DR algorithm [7] searches for a solution for the subproblem associated with the root node and it tries to extend this solution to the other subproblems induced by the nodes of the GHD in a depth-first manner. If a subproblem Pni has no solution, then FC-GHD+NG+DR, reorders the subtrees rooted at children of the current node, backtracks to the subproblem P_{n_j} such that n_j is the parent node of n_i in T, it computes another solution for P_{n_j} and continues from there. FC-GHD+NG+DR is described by (Algorithm 5).

It takes as input a complete $GHD = \langle T, \chi, \lambda \rangle$ associated with a CSP instance $P = \langle X, D, C \rangle$. The nodes of T are organized in a list σ according to the depth-first (preorder) traversal. The subproblems are solved sequentially by the function $Solve\_subpb$ according to σ. If P_{n_i} has a (another) solution then the procedure $Filter - NG$ (Algorithm 4) checks the consistency of the constraints at descendant nodes of n_i. If all these constraints are satisfied and if the current solution is not a Nogood, then all the constraint relations at each child node of n_i are filtered and the subproblem associated with the next node in σ is processed. In the negative case, another solution is computed for P_{n_i} if it exists.

If there is no (other) solution for P_{n_i}, then FC-GHD+NG+DR calls the procedure $BackTrack - DR$ (Algorithm 3) for restoring the tuples removed by the process of filtering, recording a Nogood using the procedure $Record\_nogood$ (Algorithm 1), reordering sub-trees with procedure $Reorder\_hypertree$ (Algorithm 2) such that all nodes of the sub-tree rooted at n_i are inserted between $Parent(n_i)$ and the nodes following n_i in σ noted by $Succ(Parent(n_i))$ and to backtrack to $Parent(n_i)$. FC-GHD+NG+DR stops in two cases:

1. All the subproblems are successfully solved, and then a global solution for the whole CSP instance is computed (line 16).
2. There is no other solution for the subproblem associated with the root node and then the CSP instance is unsatisfiable.

Algorithm 1 *Procedure Record_nogood*

Input n_i : node

$S \leftarrow \chi(Parent(n_i)) \cap \chi(n_i)$
$nogoods(Parent(n_i)) \longleftarrow nogoods(Parent(n_i)) \cup sol(P_{Parent(n_i)})[S]$

Figure 3: Record_nogood Procedure.
Source: [7].

Algorithm 2 *Procedure Reorder_hypertree*

In n_i : node,
Inout σ : list

$cn_1 \longleftarrow Parent(n_i)$
$cn_2 \longleftarrow n_i$
while $cn_2 \neq \emptyset$ **do**
  if $cn_2 \notin nodes(T_{n_i}$ **then**
    Break
  end if
  $cn_3 \longleftarrow succ(cn_2)$
  insert cn_2 just after cn_1 in σ
  $cn_2 \longleftarrow cn_3$
  $cn_1 \longleftarrow succ(cn_1)$
end while

Figure 4: Procedure Reorder_hypertree.
Source: [7].

```
Algorithm 3 Procedure Backtrack-DR
  Inout n_i : node,
  Inout σ : list
  Restore_removed_tuples(n_i)
  Record_nogood(n_i)
  Reorder_hypertree(n_i)
  n_i ⟵ Parent(n_i)
```

Figure 5: Procedure Backtrack-DR.
Source: [7].

```
Algorithm 4 Procedure Filter-NG
  Inout n_i : node
  if ¬Nogood(sol(P_{n_i})) then
    if C_i ∈ λ(T_{n_i}, compatible(sol(P_{n_i})), Rel(C_i)) then
      Filter_child_nodes(n_i)
      n_i ⟵ succ(n_i)
    end if
  end if
```

Figure 6: Procedure Filter-NG.
Source: [7].

```
Algorithm 5 FC-GHD+NG+DR
  Input: a complete GHD = ⟨T, χ, λ⟩ associated with a CSP instance P
  Output: a solution A of P if it exists
  σ ← (n_1, n_2, ..., n_e) /* σ is a depth-first (pre-order) traversal of T with n_1 its root */
  n_i ⟵ n_1
  while n_i ≠ ∅ do
    sol(P_{n_i}) ← Solve_subpb(P_{n_i})
    if sol(P_{n_i}) = ∅ then
      if n_i = n_1 then
        A ← ∅
        exit /* P is unsatisfiable */
      else
        Backtrack-DR(n_i)
      end if
    else
      Filter-NG(n_i)
    end if
  end while
  A ← ⋈_{i=1}^{i=e} sol(P_{n_i})
  return A
```

Figure 7: FC-GHD+NG+DR Algorithm.
Source: [7].

III. RESTART-FC-GHD+NG+DR

In this section, we present $Restart - FC - GHD + NG + DR$ which is a new version of FC-GHD+NG+DR. As all the structural methods, FC-GHD+NG+DR depends in the quality of the decomposition and in the first node (root) considered to process the GHD decomposition. Since finding an appropriate root for processing a GHD is a very hard task [18], we propose to introduce the restart technique in order to consider another root for the hypertree, for this we consider all possible orders (with respect to depth first traversal-pre-order). So, the set of possible order s obtained are represented by $ORDERS$, they are partitioned into many subsets $\sigma_1, \dots, \sigma_r$ such that $\sigma_1 \cup \dots \cup \sigma_r = ORDERS$ where r is the number of orders. For the purpose of improving the performances, we introduce the restart techniques to the $FC - GHD + NG + DR$. The main steps of this techniques are:

1. Select the initial order $\sigma_1 \in ORDERS$ and initiate the resolution with $FC - GHD + NG + DR$;

2. At each time the number of backtracks reaches a threshold $limit\_backtracks$ which is updated at each iteration by a constant factor $param$, we apply a restart;

3. Restart allows us to choose another order from the set of $ORDERS$ already defined, and restart the resolution.

III.1 ALGORITHM

$Restart - FC - GHD + NG + DR$ is formally described by Algorithm 6.

```
Algorithm 6 Restart-FC-GHD+NG+DR
    Input: a complete GHD ⟨T, χ, λ⟩ associated with the CSP instance
    Output: a solution A of P if it exists
1:  σ ← (n_1, n_2, ..., n_e) /* σ is a depth-first (pre-order) traversal of T with n_1 its root */
2:  n_i ⟵ n_1
3:  nb_backtracks ← 0
4:  restart ← 1
5:  while restart = 1 do
6:    while n_i ≠ ∅ do
7:      sol(P_{n_i}) ← Solve_subpb(P_{n_i})
8:      if sol(P_{n_i}) = ∅ then
9:        if n_i = n_1 then
10:         A ← ∅
11:         restart ← 0
12:         exit /* P is unsatisfiable */
13:       else
14:         nb_backtracks++
15:         if nb_backtracks < limit_backtracks then
16:           Backtrack(n_i)
17:         else
18:           restart ← 0
19:           Break
20:         end if
21:       end if
22:     else
23:       Filter-NG(n_i)
24:     end if
25:   end while
26:   if rstart = 1 then
27:     A ← ⋈_{i=1}^{i=e} sol(P_{n_i})
28:     return A
29:   else
30:     print "P is unsatisfiable"
31:   end if
32: end while
```

Figure 8: Restart-FC-GHD+NG+DR Algorithm.
Source: Authors, (2025).

It takes as input a complete GHD associated with the CSP, and returns a solution of the CSP if it exists. First, (line 1) the algorithm commences by establishing an initial order $\sigma_1 = (n_1, \dots, n_e)$ where n_1 is the root node. This order is obtained with respect to the depth-first search strategy. At each node n_1 the algorithm tries to solve the associated sub-problem P_{n_i} using the function $Solvesubpb(P_{n_i})$ (line 7). If P_{n_i} has a solution, we use the procedure $Filter - NG$ (line 24) to filter the relations of constraints at the λ label of each child node of n_i, then solves the next subproblems P_{n_j} associated with the node n_j . In cases where P_{n_i} is inconsistent and n_i is the first node, then the problem P has no solution (line 11). Otherwise, it involves increment the number of backtracks $nb\_backtracks$ and checks the $limit\_backtracks$ (lines 14, 15). If the number of backtracks does not exceed the $limit\_backtracks$, it performs a backtrack (line 16) in order to compute another solution for the subproblem associated with the node $Parent(n_i)$; otherwise, it restarts (line 18), where the algorithm considers an new root for the GHD and adopts with a new order $\sigma_2 = (n_2, \dots, n_e)$ according to the depth-first strategy.

Example2. Consider the GHD in Figure 2.

Initially the order σ is defined as follows: $\sigma_j = (n_1, n_2, n_3, n_4, n_5, n_6)$ with $n_1$as root of the hypertree. We consider the $limit\_backtracks = 3$.

First, we start the resolution with the first subproblem P_{n_1} associated with the node n_1 which is considered as a root of the hypertree. If P_{n_1} has no solution then we stop the resolution and the problem P has no solution, else we filter all the constraints in the λ label of each child of node n_1 (n_2 and n_3) and then we move to the next node n_2, we look for $sol(P_{n_2})$ which is compatible with $sol\ (P_{n_1})$. If P_{n_2} is consistent, we filter all the constraints in the λ label of each child of the node n_1 (n_2 and n_3). Else, $nb\_backtracks$ is incremented and a backtracking occurs from n_2 to n_1 (if $limit\_backtracks$ is not reached) to calculate another solution forP_{n_1} if it exists. When the solution computed to P_{n_1} is consistent we move to P_{n_2} . If the solution $sol(P_{n_2})$ is inconsistent, $nb\_backtracks$ is incremented to 2 and a backtracking occurs from n_2 to n_1 , then it generates another solution to P_{n_1} if it exists. If the solution $sol(P_{n_2})$ is consistent then move to the next subproblem.

At this stage, if P_{n_3} or another P_{n_i} is inconsistent, the $nb\_backtracks$ is incremented and if $limit\_Backtracks$, it restarts from the new root n_2 of the order $\sigma_2 = (n_2, n_3, n_4, n_5, n_6, n_1)$ (see Figure 9).

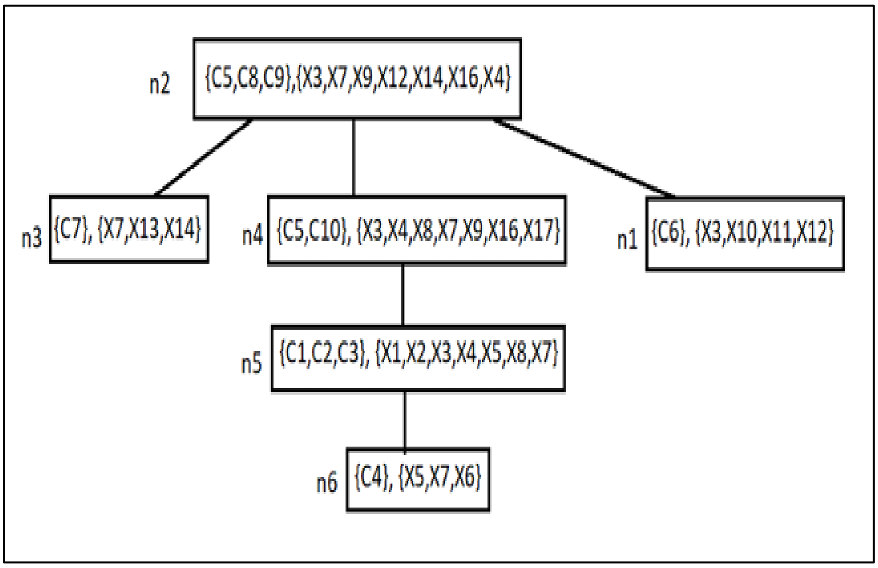

Figure 9: The GHD of Example 1 after reordering nodes.
Source: Authors, (2025).

IV. EXPERIMENTS

This section presents the experiments carried out in order to evaluate the performances of the $RestartFC - GHD + NG + DR$ method. $Restart - FC - FGD + NG + DR$ has been implemented in MPI C++ and run on a Core (TM) 2 Duo CPU T5670 @ 1.80 GHZ with 2GB of RAM under Linux Debian. The tests have been executed on benchmarks selected for the CSP Solver international Competitions CPAI'08 and CPAI'09.1.

For each instance, the time out (TO) is fixed to 1,800 seconds. The Memory Out (MO) is fixed to 2GB.

For computing the GHD Decomposition we used the Bucket Elimination (BE) algorithm [19] which is one of the best algorithms giving nearly optimal generalized hypertree decompositions within a reasonable CPU time [19][2]

In all the following tables of results, $|X|$ is the number of variables, $|C|$ is the number of constraints, w is the width of the GHD decomposition returned by BE and $time$ is the CPU run time needed to solve the instance of the considered series. The results in bold are the lowest (best) of each row. All CPU times are given in seconds.

They include the time for computing a GHD using BE (unless otherwise stated), in addition to the time for completing the GHD and solving the problem. In all the tables, the symbol $'/'$ indicates unknown values.Note that the reported times for each instance are average runtime sover 5 executions because of the random nature of the BE algorithm, giving possible different GHD decompositions for one given instance. For this study, we have used the following benchmarks: Renault series, Renault Modified series, Pret series, Dubois series and VarDimacs which are described in Subsection 4.1.

IV.1 DESCRIPTION OF BENCHMARKS

Structured Instances: Both the Renault series and the Renault-mod series consist of multiples instances related to the Renault Megane configuration problem. These instances are represented in different forms:

- Renault Series: contains 2 structured instances coming from the original Renault Megane configuration problem appearing under two forms: normalized and simple form. Both instances involve large constraint relations of high arity and the largest relation contains 48,721 tuples.
- Renault-mod Series: this class (Modified Renault) contains 50 structured instances involving domains with up to 42 possible values. The largest constraint relation contains 48,721 tuples.

Quasi random instances (random plus a small structure):

- Boolean instances (each variable domain is {0,1}):

  - Pret series: contains 8 instances encoding 2-coloring problems forced to be unsatisfiable with either 60 or 150 variables. The maximum arity of the constraints is 3 (3-SAT) and each constraint relation contains 4 tuples.
  - Dubois series: contains 13 randomly generated unsatisfiable 3-SAT instances. For each instance, each constraint relation contains 4 tuples.
  - VarDimacs series: comes from the original Sat formalization of Circuit fault analysis: Bridge Fault (BF): 4 unsatisfiable instances, and from the well-known Pigeon-hole problem: 5 unsatisfiable instances. The maximum arity of the constraints is greater than 2 and the largest constraint relation contains 1,023 tuples (normalized-hole-10_ext).

IV.2 COMPARING $RESTART - FC - GHD + NG + DR$ WITH $FC - GHD + NG + DR$

This subsection gives the comparative results of Restart-FC-GHD+NG+DR and FC-GHD+NG+DR on all the considered series.

IV.2.1 on normalized renault

Table 1 presents the comparison results of FC-GHD+NG+DR and Restart-FC GHD+NG+DR on the two instances of Renault series. The two algorithms have almost similar performances with little advantage to Restart-FC-GHD+NG+DR. The two instances of Renault series are very structured and come from real applications. This explains the good time results of the two methods.

Table 1: Comparison between FC-GHD+NG+DR and Restart-FC-GHD+NG+DR: Renault series.

Problems normalized	**Size**			W	**FC − GHD + NG + DR**	**Restart − FC − GHD + NG + DR**
	$\|X\|$	$\|C\|$	r		**Time**	**Time**
renault ext	101	134	48,721	3	0.83	**0.6**
renault-mgd ext	101	113	48,721	2	0.96	**0.7**

Source: Authors, (2025).

IV.2.2 On Modified Renault.

Table 2 presents the comparison results of the two algorithms on the Renault-mod series. It shows that $Restart - FC - GHD + NG + DR$ clearly improves $FC - GHD + NG + DR$ in terms of CPU time for both consistent and inconsistent instances. We can observe that the $FC - GHD + NG + DR$ is better than the $Restart$ one on few instances. This is due to the restart technique which needs more deeper study in order to fix the $limit\_backtaracks$.

Table 2: Comparison between FC-GHD+NG+DR and Restart-FC-GHD+NG+DR on Renault-mod series.

Problems normalized Renault-mod	**Size**			W	$FC - GHD + NG + DR$	$Restart - FC - GHD + NG + DR$	**Consistency**
	$\|X\|$	$\|C\|$	r		**Time**	**Time**	
-0_ext	111	154	48,721	4	1.32	**0.86**	Consistent
-1_ext	111	154	48,721	3	7.73	18.87	Inconsistent
-2_ext	111	154	48,721	5	1.59	**1.21**	Consistent
-3_ext	111	154	48,721	3	6.02	**5.98**	Inconsistent
-4_ext	111	154	48,721	4	1.49	**1.11**	Consistent
-5_ext	111	154	48,721	3	13.97	40.72	Inconsistent
-6_ext	111	154	48,721	3	0.85	**0.83**	Inconsistent
-7_ext	111	154	48,721	4	1.93	3.07	Consistent
-8_ext	111	154	48,721	3	0.83	**0.80**	Inconsistent
-9_ext	111	154	48,721	3	1.09	**1.08**	Consistent
-10_ext	111	154	48,721	3	7.57	**7.22**	Inconsistent
-11_ext	111	154	48,721	3	1.28	**1.25**	Consistent
-12_ext	111	154	48,721	3	32.04	107.73	Inconsistent
-13_ext	111	154	48,721	3	1.03	**1.00**	Consistent
-14_ext	111	154	48,721	3	6.91	18.04	Inconsistent
-15_ext	111	154	48,721	3	4.36	12.60	Inconsistent
-16_ext	111	154	48,721	3	11.58	**11.19**	Inconsistent
-17_ext	111	154	48,721	3	2.11	**1.84**	Inconsistent
-18_ext	111	154	48,721	3	206.13	**1.69**	Inconsistent
-19_ext	111	154	48,721	3	1.06	**0.95**	Inconsistent
-20_ext	111	154	48,721	3	9.71	9.74	Inconsistent
-21_ext	111	154	48,721	3	52.02	421.08	Inconsistent
-22_ext	111	154	48,721	3	26.45	28.05	Inconsistent
-23_ext	111	154	48,721	4	2.21	**1.82**	Inconsistent
-24_ext	111	154	48,721	4	3.37	**3.29**	Inconsistent
-25_ext	111	154	48,721	3	40.01	107.94	Inconsistent
-26_ext	111	154	48,721	3	MO	MO	Inconsistent
-27_ext	111	154	48,721	3	2.14	**1.96**	Inconsistent
-28_ext	111	154	48,721	3	74.04	76.61	Inconsistent
-29_ext	111	154	48,721	4	14.18	**13.82**	Inconsistent
-30_ext	111	154	48,721	3	4.78	10.45	Inconsistent
-31_ext	111	154	48,721	3	1.65	**1.63**	Consistent
-32_ext	111	154	48,721	4	5.09	14.41	Consistent
-33_ext	111	154	48,721	5	12.40	12.41	Inconsistent
-34_ext	111	154	48,721	4	6.13	9.76	Consistent
-35_ext	111	154	48,721	3	19.72	50.41	Inconsistent
-36_ext	111	154	48,721	4	21.08	45.90	Consistent
-37_ext	111	154	48,721	4	11.02	27.67	Inconsistent
-38_ext	111	154	48,721	4	1.70	2.58	Consistent
-39_ext	111	154	48,721	4	51.71	638.17	Inconsistent
-40_ext	108	149	48,721	3	553.60	1560.17	Inconsistent
-41_ext	108	149	48,721	4	7.59	18.18	Consistent
-42_ext	108	149	48,721	3	1.17	**1.10**	Inconsistent
-43_ext	108	149	48,721	3	1.85	**1.45**	Consistent
-44_ext	108	149	48,721	4	1.03	**0.93**	Consistent
-45_ext	108	149	48,721	4	19.89	44.54	Consistent
-46_ext	108	149	48,721	4	5.86	**5.44**	Consistent
-47_ext	108	149	48,721	4	1.19	**0.74**	Inconsistent
-48_ext	108	149	48,721	4	47.01	84.04	Consistent
-49_ext	108	149	48,721	4	24.38	85.57	Consistent

Source: Authors, (2025).

IV.2.3 On Pret Series and Dubois Series

Tables 3 and 4 show the comparison results of the two algorithms on the Boolean Pret and Dubois series. On these series, $Restart - FC - GHD + NG + DR$ and $FC\ GHD + NG + DR$ solve all instances in short time. On Pret series, the average runtimes of the two algorithms $FC - GHD + NG + DR$ and $Restart - FC - GHD + NG + DR$ are 0.007 and ≈ 0 *seconds* respectively. On Dubois series, their average runtimes are 0.0035 and $\approx$ 0 *seconds* respectively.

Table 3: Comparison between FC-GHD+NG+DR and Restart-FC-GHD+NG+DR on Pret series.

Problems normalized pret	Size			W	$FC - GHD + NG + DR$	$Restart - FC - GHD + NG + DR$	Consistency
	$\|X\|$	$\|C\|$	r		Time	Time	
-60-25_ext	60	40	4	5	0.36	**≃0**	Inconsistent
-60-40_ext	60	40	4	5	0.008	**≃0**	Inconsistent
-60-60_ext	60	40	4	5	0.01	**≃0**	Inconsistent
-60-75_ext	60	40	4	5	0.01	**≃0**	Inconsistent
-150-25_ext	150	100	4	5	0.05	**≃0**	Inconsistent
-150-40_ext	150	100	4	5	0.17	**≃0**	Inconsistent
-150-60_ext	150	100	4	5	0.37	**≃0**	Inconsistent
-150-75_ext	150	100	4	5	0.023	**≃0**	Inconsistent

Source: Authors, (2025).

Table 4: Comparison between FC-GHD+NG+DR and Restart-FC-GHD+NG+DR on Duboi.

Problems normalized Dubois	Size			W	$FC - GHD + NG + DR$	$Restart - FC - GHD + NG + DR$	Consistency
	$\|X\|$	$\|C\|$	r		Time	Time	
-20_ext	60	40	4	2	0.043	**≃0**	Inconsistent
-21_ext	63	42	4	2	0.005	**≃0**	Inconsistent
-22_ext	66	44	4	2	0.004	**≃0**	Inconsistent
-23_ext	69	46	4	2	0.005	**≃0**	Inconsistent
-24_ext	72	48	4	2	0.005	**≃0**	Inconsistent
-25_ext	75	50	4	2	0.005	**≃0**	Inconsistent
-26_ext	78	52	4	2	0.006	**≃0**	Inconsistent
-27_ext	81	54	4	2	0.006	**≃0**	Inconsistent
-28_ext	84	56	4	2	0.006	**≃0**	Inconsistent
-29_ext	87	58	4	2	0.007	**≃0**	Inconsistent
-30_ext	90	60	4	2	0.006	**≃0**	Inconsistent
-50_ext	150	100	4	2	0.011	**≃0**	Inconsistent
100_ext	300	200	4	2	0.049	**≃0**	Inconsistent

Source: Authors, (2025).

IV.2.4 On VarDimacs Series

Finally, Table 5 presents the behavior of the two algorithms on VarDimacs series. $FC\ GHD + NG + DR$ and $Restart - FC - GHD + NG + DR$ succeed to solve four instances. The average runtime of the two algorithms is 4.01 and 130,75 seconds respectively. But we have better results with $Restart - FC - GHD + NG + DR$ except for the instance normalized bf-0432-007_ext.

Table 5: Comparison between FC-GHD+NG+DR and Restart-FC-GHD+NG+DR on Pret series.

Problems normalized	Size			W	$FC - GHD + NG + DR$	$Restart - FC - GHD + NG + DR$	Consistency
	$\|X\|$	$\|C\|$	r		Time	Time	
-bf-0432-007_ext	970	1,943	31	29	35.15	129.87	Consistent
-bf-1355-075_ext	1,818	2,049	5	5	9.74	**0.81**	Consistent
-bf-1355-638_ext	532	339	31	2	0.18	≃0	Consistent
-bf-2670-001_ext	1,244	1,354	31	7	0.31	**0.29**	Inconsistent

Source: Authors, (2025).

V. CONCLUSIONS

In this work, we have presented a new method called Restart-FC-GHD+NG+DR, which combines the FC-GHD+NG+DR algorithm, exploiting GHD, with a restart strategy to solve non-binary CSPs. Our experiments on benchmark of literature have demonstrated the efficiency of the proposed algorithm, particularly on consistent instances. The results show significant improvements over the FC-GHD+NG+DR algorithm, with a 52.62% better performance on modified Renault consistent instances and near-zero execution time for the Normalized Dubois and Normalized Pret series. This confirms the algorithm's potential in enhancing CSP-solving strategies. This approach offers significant contributions, the method advances CSP-solving by addressing limitations of traditional algorithms, introducing a dynamic, restart-based approach that adapts to various problem structures. It opens new research avenues by integrating machine learning for adaptive reordering, encouraging cross-disciplinary applications in fields like artificial intelligence, operations research, and network optimization. However, some limitations remain, such as managing the limit_backtaracks more effectively, as excessive backtracking can still increase execution time. Additionally, enhancing the algorithm's handling of inconsistent problem instances is necessary to avoid exploring all possible orders, which would further improve computational efficiency. For future work, we plan to integrate machine learning and deep learning techniques to dynamically reorder the nodes of the GHD decomposition.

VI. AUTHOR'S CONTRIBUTION

Conceptualization: Fatima Ait Hatrit, Kamal Amroun
Methodology: Fatima Ait Hatrit, Kamal Amroun
Investigation: Fatima Ait Hatrit, Kamal Amroun
Discussion of results: Fatima Ait Hatrit, Kamal Amroun
Writing – Original Draft: Fatima Ait Hatrit, Kamal Amroun
Writing – Review and Editing: Fatima Ait Hatrit, Kamal Amroun **Resources:** Fatima Ait Hatrit, Kamal Amroun
Supervision: Fatima Ait Hatrit, Kamal Amroun
Approval of the final text: Fatima Ait Hatrit, Kamal Amroun

VIII. REFERENCES

[1] S. Choudhury, J. K. Gupta, M. J. Kochenderfer, D. Sadigh, and J. Bohg, « Dynamic multi-robot task allocation under uncertainty and temporal constraints », *Auton Robot*, vol. 46, nº 1, p. 231-247, janv. 2022, doi: 10.1007/s10514-021-10022-9.

[2] A Constraint Programming Approach to Simultaneous Task Allocation and Motion Scheduling for Industrial Dual-Arm Manipulation Tasks ». https://ieeexplore.ieee.org/document/8794022

[3] M. Bodirsky, P. Jonsson, B. Martin, A. Mottet, and Ž. Semanišinová, « Complexity Classification Transfer for CSPs via Algebraic Products », 7 juin 2024, *arXiv*: arXiv:2211.03340. http://arxiv.org/abs/2211.03340

[4] M. Grohe, V. Guruswami, D. Marx, and S. Živný, « The Constraint Satisfaction Problem: Complexity and Approximability (Dagstuhl Seminar 22201) », Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi: 10.4230/DAGREP.12.5.112.

[5] H. Chen, G. Gottlob, M. Lanzinger, and R. Pichler, « Semantic Width and the Fixed-Parameter Tractability of Constraint Satisfaction Problems », 28 juillet 2020, *arXiv*: arXiv:2007.14169. http://arxiv.org/abs/2007.14169

[6] R. Ganian, S. Ordyniak, and S. Szeider, « A Join-Based Hybrid Parameter for Constraint Satisfaction », 29 juillet 2019, *arXiv*: arXiv:1907.12335. Consulté le: 16 novembre 2024. http://arxiv.org/abs/1907.12335

[7] Z. Habbas, K. Amroun, and D. Singer, « A Forward-Checking algorithm based on a Generalised Hypertree Decomposition for solving non-binary constraint satisfaction problems », *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 27, nº 5, p. 649-671, sept. 2015, doi: 10.1080/0952813X.2014.993507.

[8] Z. Younsi, K. Amroun, F. Bouarab-Dahmani, and S. Bennai, « HSJ-Solver: a new method based on GHD for answering conjunctive queries and solving constraint satisfaction problems », *Appl Intell*, vol. 53, nº 13, p. 17226-17239, juill. 2023, doi: 10.1007/s10489-022-04361-y.

[9] G. Gottlob, C. Okulmus, and R. Pichler, « Fast and parallel decomposition of constraint satisfaction problems », *Constraints*, vol. 27, nº 3, p. 284-326, juill. 2022, doi: 10.1007/s10601-022-09332-1.

[10] S. Bennai, K. Amroun, and S. Loudni, « Exploiting Data Mining Techniques for Compressing Table Constraints », in *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, nov. 2019, p. 42-49. doi: 10.1109/ICTAI.2019.00015.

[11] F. Koriche, C. Lecoutre, A. Paparrizou, and H. Wattez, « Best Heuristic Identification for Constraint Satisfaction », in *31st International Joint Conference on Artificial Intelligence (IJCAI'22)*, in Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI-22). Vienne, Austria: International Joint Conferences on Artificial Intelligence Organization, juill. 2022, p. 1859-1865. doi: 10.24963/ijcai.2022/258.

[12] U. Montanari, « Networks of constraints: Fundamental properties and applications to picture processing », *Information Sciences*, vol. 7, p. 95-132, janv. 1974, doi: 10.1016/0020-0255(74)90008-5.

[13] K. Amroun, Z. Habbas, and W. Aggoune-Mtalaa, « A compressed Generalized Hypertree Decomposition-based solving technique for non-binary Constraint Satisfaction Problems », *AIC*, vol. 29, nº 2, p. 371-392, mars 2016, doi: 10.3233/AIC-150694.

[14] E. C. Freuder, « A sufficient condition for backtrack-bounded search », *J. ACM*, vol. 32, nº 4, p. 755-761, oct. 1985, doi: 10.1145/4221.4225.

[15] I. Adler, G. Gottlob, and M. Grohe, « Hypertree width and related hypergraph invariants », *European Journal of Combinatorics*, vol. 28, nº 8, p. 2167-2181, nov. 2007, doi: 10.1016/j.ejc.2007.04.013.

[16] G. Gottlob, N. Leone, and F. Scarcello, « Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width », in *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, in PODS '01. New York, NY, USA: Association for Computing Machinery, mai 2001, p. 195-206. doi: 10.1145/375551.375579.

[17] P. Jégou and C. Terrioux, « Hybrid backtracking bounded by tree-decomposition of constraint networks », *Artificial Intelligence*, vol. 146, nº 1, p. 43-75, mai 2003, doi: 10.1016/S0004-3702(02)00400-9.

[18] P. Jégou, P. Gou, and C. Terrioux, « Combining Restarts, Nogoods and Decompositions for Solving CSPs », in *ECAI 2014*, IOS Press, 2014, p. 465-470. doi: 10.3233/978-1-61499-419-0-465.

[19] A. Dermaku, T. Ganzow, G. Gottlob, B. McMahan, N. Musliu, and M. Samer, « Heuristic Methods for Hypertree Decomposition », in *MICAI 2008: Advances in Artificial Intelligence*, A. Gelbukh et E. F. Morales, Éd., Berlin, Heidelberg: Springer, 2008, p. 1-11. doi: 10.1007/978-3-540-88636-5_1.