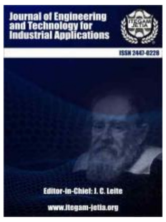
Journal of Engineering and Technology for Industrial Applications

ITEGAM-JETIA

Manaus, v.11 n.51, p. 119-126. January/February., 2025.
DOI: https://doi.org/10.5935/jetia.v11i51.1449

ISSN ONLINE: 2447-0228



RESEARCH ARTICLE　　OPEN ACCESS

TITLE: FROM BACKTRACKING TO DEEP LEARNING: A SURVEY ON METHODS FOR SOLVING CONSTRAINT SATISFACTION PROBLEMS

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), 06000 Bejaia, Algérie.

[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 05, 2024
Revised: January 10, 2025
Accepted: January 15, 2025
Published: January 30, 2025

Keywords:
Constraints Satisfaction Problems,
Solving CSP,
Deep Learning,
CSP resolution method,
Backtracking.

ABSTRACT

Constraint Satisfaction Problems (CSP) are a fundamental mechanism in artificial intelligence, but finding a solution is an NP-complete problem, requiring the exploration of a vast number of combinations to satisfy all constraints. To address this, extensive research has been conducted, leading to the development of effective techniques and algorithms for different types of CSPs, ranging from exhaustive search methods, which explore the entire search space, to modern techniques that use deep learning to learn how to solve CSPs. This paper represents a descriptive and synthetic overview of various CSPs solving methods, organized by approach: systematic search methods, inference and filtering methods, structural decomposition methods, local search-based methods, and deep learning-based methods. By offering this structured classification, it presents a clear view of resolution strategies, from the oldest to the most recent, highlighting current trends and future challenges, there by facilitating the understanding and application of available approaches in the field.

I. INTRODUCTION

Constraint Satisfaction Problems (CSPs) play a crucial role in various fields of computer science, artificial intelligence (AI), and operations research. These problems arise in scenarios where a set of variables must be assigned values that satisfy specific constraints. Applications of CSPs are diverse [1], ranging from activity and scheduling planning [2], to allocation problem [3]. Despite their widespread use, CSPs are inherently complex, often involving a large search space and intricate constraint interactions, making their resolution a challenging task. The formalization of CSPs provides a structured framework to model and solve these problems systematically. Since the foundational work by [4] in 1974, numerous approaches have been developed to tackle CSPs, each aiming to optimize the tree search for a solution, requiring the exploration of a vast number of combinations to satisfy all constraints. To address this challenge, a wide range of methods have been proposed, from traditional systematic search algorithms, such as backtracking (BT) and constraint propagation, to modern techniques that leverage deep learning to learn how to solve CSPs. This study's objective is to provide a comprehensive overview of the current state of art CSP-solving methodologies, highlighting their strengths, limitations, and suitability for different types of CSPs. By examining these approaches, we aim to shed light on the evolution of CSP-solving strategies and propose a structured classification that aids in understanding and selecting appropriate methods for solving CSP problem. Although a number of studies have already proposed classifications. In [5] the author presented a survey on general CSP resolution techniques and classifed them on finite domain techniques and infinite domain techniques. In [6], they classified the resolution methods in two mains groupe, complete resolution methods and incomplete resolution methods. Then we have [7], the authors in their study classified CSP resolution methods based on practical applications like scheduling and planning, They emphasize that constraint satisfaction approaches, especially search and constraint satisfaction algorithms, are favored in AI for addressing complex combinatorial issues.

In this study, we build on these existing classifications to provide a more detailed and up-to-date overview of CSP solving methods, focusing on the latest trends and developments in the field. By presenting a structured classification of CSP-solving techniques, we aim to offer a clear and comprehensive view of the available approaches, from traditional methods to modern deep learning-based techniques. The main contributions of this paper are as follows:

- A comprehensive overview of CSP-solving methods, organized by approach, including systematic search methods, inference and filtering methods, structural decomposition methods, local search-based methods, and deep learning-based methods.
- A detailed analysis of each category, highlighting the main algorithms and techniques used to solve CSPs, their strengths, limitations, and applications.
- A structured classification of CSP-solving methods, providing a clear view of the evolution of resolution strategies, from traditional to modern approaches, and highlighting current trends and future challenges in the field.

The remainder of this paper is organized as follows: Section II presents the preliminary definitions of CSPs, including the formal definition of a CSP, CSP constraints, CSP instantiation, and CSP solution. Section III introduces the classification of CSP-solving methods, categorizing them into five main categories: Systematic Search Methods, Inference and Filtering Methods, Structural Decomposition Methods, Local Search Based Methods, and Deep Learning Based Methods. Sections IV to VIII provide an in-depth analysis of each category, detailing the methods used to solve CSPs, their approaches, and applications. Finally, Section IX concludes the paper, summarizing the main findings and discussing future research directions.

II. PRELIMINARY DEFINITIONS

In this section, we present the fundamental definitions of Constraint Satisfaction Problems (CSPs), including the formal definition of a CSP, CSP constraints, CSP instantiation and CSP solution.

II.1 CONSTRAINT SATISFACTION PROBLEM

A CSP is define as a set of variables, with associated domains, and a set of constraints. Each constraint is defined on a subset of the set of variables and limits the combinations of values that these variables can take.
The formal definition of a CSP was introduced by Montanari [4], a CSP is defined by $< X, D, C >$, where:

- $X = \{X_1, X_2, \ldots, X_n\}$ is a set of n variables,
- $D = \{D_1, D_2, \ldots, D_n\}$ is a set of finite domains, each variable X_i takes its value from its domain D_i,
- $C = \{C_1, C_2, \ldots, C_m\}$ is a set of m constraints. Each constraint C_i is a pair $(Scope(C_i), Rel(C_i))$ where $Scope(C_i) \subseteq X$ is a list of variables, called the scope of C_i and $Rel(C_i) \subseteq \prod_{Xk \in Scope(C_i)} D_k$ (subset of the cartesian product) is the relation of C_i that indicates the valid combinations of values for the variables in $Scope(C_i)$.where each constraint C_i is a relation between a subset of variables.

II.2 CONSTRAINTS

Constraints in the context of CSPs can be expressed in different ways: in extension, by presenting the set of tuples authorised, forbidden, or in intention, by giving mathematical formulae.

The structure of the problem to be solved is difined by the relation between the variables.

The size of $Scope(C_i)$ is called the arity of C_i, and constraints can be classified within its arity into different categories:

- Unary constraints: constraints that involve a single variable, $X_1 \neq Red,$
- Binary constraints: constraints that involve two variables, $X_1 \neq X_2,$
- N-ary constraints: constraints that involve more than two variables, $X_1 + X_2 < X_3$.

II.3 INSTANTIATION AND CONSTANCY

An instantiation I of a subset of variables denoted by X_i is an ordered set of assignments:

$$X_i = \{x_i, \ldots, x_k\} \subseteq X \quad (1)$$
$$I = \{[(x_i = v_i), \cdots, (x_k = v_k)] | v_j \in D(x - j)\} \quad (2)$$

The variables assigned on an instantiation I are denoted $vars(I)$

$$I = [(x_i = v_i), \cdots, (x_k = v_k)] \quad (3)$$
$$vars(I) = \{x_i, \cdots, x_k\} \quad (4)$$

If I instantiates all the variables of the problem, it is called a full instantiation (i.e., $vars(I) = X$).

An instantiation I satisfies a constraint $c_{ij} \in C$ if and only if the variables involved in c_{ij} $(i.e., x_i \ and \ x_j$) are assigned in I. Formally:

- I satisfies c_{ij} iff

$$(x_i = v_i) \in I \wedge (x_j = v_j) \in I \wedge (v_i, v_j) \in c_{ij} \quad (5)$$

An instantiation I is locally consistent iff it satisfies all of the constraints whose scopes have no uninstantiated variables in I. I is also called a partial solution.
Formally, I is locally consistent iff

$$\forall c_{ij} \in C \mid scope(c_{ij}) \sqsubseteq vars(I), I \text{ satisfies } c_{ij} \quad (6)$$

II.4 SOLUTION

A solution to a CSP is a full instantiation that satisfies all the constraints of the problem.

Formally, a solution I is a full instantiation that satisfies all the constraints of the problem, i.e.,
$\forall c_{ij} \in C, I$ satisfies c_{ij} .
Solving a CSP could mean to find existence or nonexistence of a solution, if it existes find :

- One solution, without preference as to which one,
- all solutions,
- an optimal, or at least a good solution.

II.5 EXAMPLE OF CSP

A CSP can be represented by intention, by giving the constraints in a mathematical form, or by extension, by giving the set of tuples authorised or forbidden.
Consider the following CSP instance represented by intention as follows:
$X = \{X_1, X_2, X_3\},\ D = \{D_1, D_2, D_3\}, C = \{C_1, C_2\}$, where:

- X_1, X_2, X_3 are variables,
- $D_1 = \{1, 2, 3\}, D_2 = \{1, 2\}, D_3 = \{1, 2, 3\}$ are the domains of the variables,
- $C_1 = \{(X_1 + X_2) < (X_3 - X_2 + 2)\}, C_2 = \{(X_1 + X_3 < 4)\}$ are the constraints.

The same CSP can be represented by extension as follows:

- X_1, X_2, X_3 are variables,
- $D_1 = \{1, 2, 3\}, D_2 = \{1, 2\}, D_3 = \{1, 2, 3\}$ are the domains of the variables,
- $C_1 = \{(1, 1, 2), (1, 1, 3), (1, 2, 3), (2, 1, 3)\}, C_2 = \{(1, 1), (1, 2), (2, 1)\}$ are the constraints.

The solution to this CSP is the full instantiation: $I = \{(X_1 = 1), (X_2 = 1), (X_3 = 2)\}$, which satisfies all the constraints.

III. CLASSIFICATION PROPOSAL

In this paper, we review some relevant existing literature methods used to solve CSPs and propose a classification that categorizes the cited works into two main levels, where:

• The first level is divided into five main categories: Systematic Search Methods, Inference and Filtering Methods, Structural Decomposition Methods, Local Search Based Methods, and Deep Learning Based Methods.

• The second level is divided into subcategories, which are further divided into specific methods.

This classification offers a more detailed view of the cited methods and facilitate understanding of the different approaches used to solve CSPs. In what follows, following the classification giving in Figure 1, we present and describe in section IV to VII the different categories of methods used to solve CSPs which constitue the first level of the proposed classification.

IV. SYSTEMATIC SEARCH METHODS

Systematic Search Methods for solving CSPs are approaches that explore the solution space in a structured way in order to find value assignments that satisfy all the constraints imposed. These methods generally apply an exhaustive search strategy and may include various optimisations to improve efficiency and avoid unnecessary search paths. In what follows, we present the main algorithms used in systematic search methods to solve CSPs.

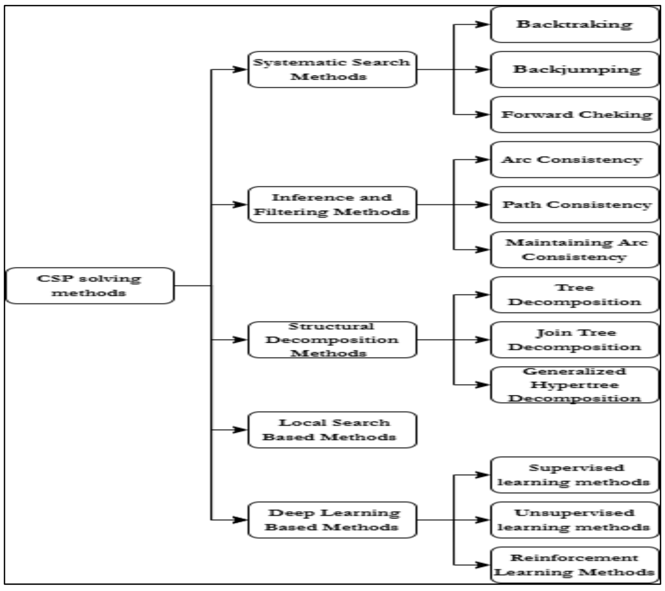

CSP solving methods
Systematic Search Methods
Inference and Filtering Methods
Structural Decomposition Methods
Local Search Based Methods
Deep Learning Based Methods
Backtraking
Backjumping
Forward Cheking
Arc Consistency
Path Consistency
Maintaining Arc Consistency
Tree Decomposition
Join Tree Decomposition
Generalized Hypertree Decomposition
Supervised learning methods
Unsupervised learning methods
Reinforcement Learning Methods

Figure 1: Classification of the CSP solving methods
Source: Authors, (2024).

IV.1 BACKTRAKING

Backtracking (BT) [8] is a systematic search technique which explores all possible combinations of values for variables, thus covering the entire solution space. The principle of the BT algorithm consists of instantiating a new variable at each stage to progressively extend an initially empty partial assignment. With each addition, a consistency test is performed to check that the assignment respects the constraints.

In the event of inconsistency, the assignment is reset, and the algorithm returns chronologically to the last consistent partial instantiation. A new instantiation is then attempted by modifying the value of the last variable. Once all the variables in a constraint have been instantiated, the validity of the constraint is checked. If a partial instantiation violates a constraint, the process returns to the most recently instantiated variable with available alternatives. In this way, each constraint violation eliminates part of the space of possible solutions, reducing the Cartesian product of variable domains.

BT performs a depth-first search of the space of potential solutions to CSPs. This process guarantees the consistency of the solution and optimises the search time by immediately stopping any iteration that does not lead to a valid solution. Although BT is generally performed on a single variable, it can sometimes involve several variables. The advantage of the BT algorithm lies in its exhaustive exploration of the search space, ensuring that if a solution exists, it will be found, or confirming its nonexistence. However, this thorough traversal results in an exponential time complexity, as nearly the entire search space must be examined.

The BT is the foundational approach for solving CSPs, providing the essential framework on which many advanced techniques are built. Each subsequent method adapts and optimizes backtracking principles to improve search efficiency like using variable ordering heuristics to improve BT algorithm [10-12].

According to [9] is one of the primary enhancements, introducing mechanisms to bypass unnecessary steps and adjust variable assignments dynamically for faster resolution.

IV.2 BACKJUMPING

Backjumping algorithm (BJ) [9] is an intelligent variant of the BT algorithm, it is an improvement on the BT algorithm that optimises the search by avoiding unnecessary revisiting of subtrees in the solution space. Unlike traditional BT, which goes back to the last instantiation point in the event of failure, BJ identifies the precise variable at the origin of the conflict and goes directly back to an earlier variable in the tree, closer to the root. This technique is used to avoid re-examining the same sub-tree multiple times. The advantage of the BJ algorithm is that the approach saves and reduce the search time by jumping over irrelevant intermediate instantiations, which is particularly beneficial when the search space is vast and the constraints are complex.

However, the BJ algorithm is not always able to identify the variable at the origin of the conflict, which can lead to a less efficient search and the time complexity is also exponential.

IV.3 FORWARD CHEKING

Forward cheking (FC) described by [13] as a systematic search technique that extends the BT algorithm by adding a consistency check to the partial assignment of variables. It works by reducing the domains of variables by eliminating values that are incompatible with those already instantiated. When a variable is assigned, the FC tests the compatibility of this assignment with subsequent variables and deletes values in domains that would conflict with this new instantiation. This means that each domain available after filtering only contains values compatible with the current instantiations. The advantage of forward checking (FC) is considered to be its ability to anticipate conflicts and thus reduce the search space. It increases search efficiency by avoiding numerous unnecessary backtracks, making it a notable improvement in complex and constrained search environments. Although FC is useful for anticipating and reducing conflicts, it can sometimes be costly and less effective for low-constraint problems or in the absence of appropriate heuristics.

One of the recent application of FC [14] used to simulate the multi-point statistical properties of some synthetic training images, the results show that no anomalies occurred in any of the produced realizations and also show that the presence of hard data does not degrade the quality of the generated realizations.

V. INFERENCE AND FILTERING METHODS

To improve the Systematic Search Methods, several techniques and strategies of constraint propagation have been proposed wich can be classifed as prospective strategies used to choose the variable to be assigned a value and retrospective strategies used to choose the value to be assigned to the variable.
Constraint propagation techniques are used to anticipate the effects of partial assignments on the domains of uninstantiated variables. By filtering out the domains of values that are incompatible with the constraints, they reduce the search space and avoid unnecessary exploration of combinations with no solution. Constraint propagation thus eliminates redundant values and reduces the size of the problem. When a reduction results in an empty domain, this indicates that there is no solution for the given instance. this technique, while beneficial, need to be balanced to avoid excessive computational cost in relation to search performance gains.

These techniques are often combined with Systematic Search Methods to improve resolution time. The most common constraint propagation techniques are described below:

V.1 ARC CONSISTENCY

Arc consistency (AC) defined in [15] for binary constraints then extended to non-binary constraints, is a constraint propagation technique that aims to reduce the search space by eliminating incompatible values in the domains of the variables in a binary constraint. It ensures that for every value in the domain of one variable, there is a corresponding value in the domain of the second variable, thus satisfying the constraint. This process examines each constraint and removes incompatible values from the domains.

By improving domain consistency, AC makes searching more efficient. However, its application can be costly, with exponential complexity in the most difficult cases, as it must evaluate all possible combinations of values.

V.2 PATH CONSISTENCY

Path Consistency (PC) [4] is an enhanced form of constraint consistency that extends the concept of AC. A CSP is path-consistent if any consistent assignment between two variables can be consistently extended to a third. In other words, for every value of a variable, there is a corresponding value in the domains of the other variables satisfying the constraint.
This process improves domain consistency by removing incompatible values, making the search more efficient. However, PC is computationally expensive, with exponential complexity in difficult cases, as it must examine all possible combinations of values.

V.3 MAINTAINING ARC CONSISTENCY

Maintaining Arc Consistency (MAC) [16] is a constraint propagation technique designed to maintain the consistency of variable domains throughout the search. It removes incompatible values at each stage, reducing the search space and avoiding unnecessary exploration of combinations with no solution. In the MAC algorithm, the search space is structured as a binary tree, where each node represents a decision based on the assignment or exclusion of a value for a variable. Ordering heuristics are used to select variables and values, improving search efficiency. Although MAC optimises the search by making domains more consistent, it can be computationally expensive in the most complex cases.

VI. STRUCTURAL DECOMPOSITION METHODS

Structural decomposition methods divide a complex problem into simpler sub-problems, based on the structure of a constraint graph. By grouping variables and constraints into tree-like clusters, they limit interdependencies and simplify computation.

A CSP instance $< X, D, C >$ have constraint hypergraph $\mathcal{H} = (V, E)$, where $V = X$ and $E = C$. The structural decomposition methods are used to decompose the hypergraph $\mathcal{H}$ into simpler sub-problems.

These techniques transform the problems into equivalent but simpler sub-problems, making them more efficient to solve.
The most common structural decomposition methods are described below.

VI.1 TREE DECOMPOSITION

Tree decomposition (TD) [17] is a structural decomposition method that divides a constraint graph into clusters forming a tree structure, where each cluster contains variables and constraints. The width of the decomposition is defined by the size of the largest cluster, simplifying the problem by making it more accessible. This approach is particularly useful for tree-structured CSPs, as it reduces search complexity. Although effective, it can be costly in very complex cases, but it remains widely used for its simplicity and effectiveness on tree graphs.

Formally, a TD [17] of a graphed $G = (V, E)$ is a pair $\langle T, \chi \rangle$ where $T = (N, F)$ is a tree and χ is a labelling function that assigns to each node $t \in N$ a subset of vertices $\chi(t) \subseteq V$ called the bag of t such that:

$$\forall v \in V, \exists t \in N \mid v \in \chi(t), \tag{7}$$
$$\forall e = \{u, v\} \in E, \exists t \in N \mid \{u, v\} \subseteq \chi(t), \tag{8}$$
$$\forall v \in V, \{t \in N \mid v \in \chi(t)\} \tag{9}$$

(9) induces a connected subtree of T.

The $width$ of aTD is equal to $max_{t \in N} (|\chi(t)|) - 1$, treewidth of a graph is the minimum width over all its tree decomposing.

The advantage of TD is that it simplifies the problem by grouping variables and constraints into tree-like clusters. This method is particularly useful for problems with a tree-like structure, as it reduces the complexity of the search. However, TD can also be computationally expensive in the most complex cases.
To exploit this technique for solving CSPs, several algorithms have been proposed in the literature, the most popular being: BT on Tree Decomposition (BTD) [18], that proceeds by an enumerative

search guided by a static pre-established partial order induced by a tree decomposition of the constraint network.

VI.2 JOIN TREE DECOMPOSITION

A Join tree [19], is a structural decomposition method that divides a constraint graph into tree-like clusters called cliques. Each clique contains a set of variables and constraints, forming a hierarchical tree structure. The width of the junction tree decomposition is determined by the size of the largest clique.

A join tree decomposition of a hypergraph $\mathcal{H}$ is a triplet $\langle T, \chi, \lambda \rangle$ where $T = (N, F)$ is a tree, χ is a labelling function that assigns to each node $t \in N$ a subset of vertices $\chi(t) \subseteq V$ called the bag of t, λ is a labelling function that assigns to each edge $e \in F$ a subset of vertices $\lambda(e) \subseteq V$ called the bag of e, such that:

- $\forall v \in V, \exists t \in N \text{ such that } v \in \chi(t),$ (10)
- $\forall e \in F, \exists t \in N \text{ such that } \lambda(e) \subseteq \chi(t),$ (11)
- $\forall v \in V, \{t \in N \mid v \in \chi(t)\}$ (12)

induces a connected subtree of T ,

- $\forall e \in F, \lambda(e) = \bigcap_{t \in N \mid \lambda(e) \subseteq \chi(t)} \chi(t),$ (13)
- $\forall e \in E, \exists t \in N \text{ such that } e \subseteq \chi(t).$ (14)

This technique simplifies complex problems by decomposing them into manageable clusters, making them easier to solve.

It is particularly advantageous for problems with a tree structure, as it reduces the complexity of the search. However, junction tree decomposition can become computationally expensive in the most complex cases. A classic algorithm for solving CSPs using join tree decomposition is the arc-consistency propagation algorithm on join trees, often known as the clique tree propagation algorithm [19]. This algorithm leverages the join tree structure to manage sets of constraints using cliques as computational units.

The main advantage of join tree decomposition is that it exploits redundant relationships and inferences through a simplified tree structure. This reduces the complexity of algorithms by minimising the size of the search space. In particular, it improves the efficiency of solution methods such as BT and optimisation algorithms by providing a better structure for constraint propagation.

However, its limitations include an exponential complexity related to the width of the tree and difficulty in finding an optimal decomposition for complex CSPs. This may restrict its application to large or highly connected problems.

VI.3 GENERALIZED HYPERTREE DECOMPOSITION

The Generalised Hypertree Decomposition (GHD)[20] is a structural decomposition method that segments a constraint graph into clusters organised in the form of hypertrees, each hypertree grouping a set of variables and constraints into a tree structure. The width of the decomposition is defined by the size of the largest hypertree. This method simplifies complex problems by decomposing them, making them easier to solve. The GHD [21] of a hypergraph $\mathcal{H}$ is formally defined as a hypertree $\langle T, \chi, \lambda \rangle$ of $\mathcal{H}$, wich satisfies the following properties:

- For each edge $h \in E$, there exists $p \in vertices(T)$ such

that: $var(h) \subseteq \chi(p)$ (15)

•

- For each vertex $v \in V$, the set

$\{p \in vertices\ (T) | v \in \chi(p)\}$ (16)

induces a connected subtree of T;

- For each vertex
$p \in vertices(T), \chi(p) \subseteq var\big(\lambda(p)\big)$ (17)

- For each vertex
$p \in vertices(T), var\big(\lambda(p)\big) \cap \chi\big(T_p\big) \subseteq \chi(p)$ (18)

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

A hyperedge h of a hypergraph $\mathcal{H} = \langle V, E \rangle$ 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 $\mathcal{H}$ is complete if every hyperedge h of $\mathcal{H}$ is strongly covered in HD.

A hypertree $HD = \langle T, \chi, \lambda \rangle$ is called a Generalized Hypertree Decomposition (GHD), if the conditions (15), (16) and (17) 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 hypertreewidth $(ghw(\mathcal{H}))$ of a hypergraph $\mathcal{H}$ is the minimum width over all its generalized hypertree decompositions.

Several approaches have been developed in order to exploit GHD for solving CSPs, In [22], it was used to evaluate conjunctive queries (CQs) and solve CSP.

GHD is particularly useful for problems with a tree structure. However, in complex cases, constructing the GHD can be computationally expensive, particularly due to the size and flexibility of the hypertrees in multivariate representations. The problem of updating the decomposition of a CSP is resolved in [23] where they propose and implement a framework for effectively update a GHD. Moreover, in [20], authors proposed parallel algorithms to compute GHDs efciently for a wide range of CSPs.

VII. LOCAL SEARCH BASED METHODS

Local search based methods [24] are techniques designed to find an acceptable solution to a CSP by exploring the solution space from an initial (often partial) solution and progressively modifying it through local adjustments to reduce the number of unsatisfied constraints. These algorithms include methods such as Min-Conflicts [25], which adjust an assignment to satisfy a set of constraints by choosing a variable associated with an unsatisfied constraint and assigning it a value that minimizes the number of remaining unsatisfied constraints. By exploring the neighborhood of a solution incrementally, local search methods navigate the space of nearby solutions, making them particularly effective for large and constrained search spaces where exhaustive exploration is impractical.

Recent research in local search methods for CSPs demonstrates the adaptability of these techniques in solving complex and specialized problems. In [26] , local search is used to handle incomplete fuzzy CSPs, allowing for solutions that minimize constraint violations in situations with uncertainty and flexible constraints. This approach is especially effective in cases where constraints are not fully defined or have degrees of satisfaction. Moreover in [27] focuses on optimizing costly industrial processes through a derivative-free local search method, adapted for "black-box" problems with high evaluation costs, applied to refining the start-up optimization of a production plant.

Both studies highlight the adaptability of local search techniques in managing complex, constrained environments and enhancing solution efficiency.

VIII. DEEP LEARNING BASED METHODS

Deep learning methods have been proposed to solve CSPs by leveraging the power of neural networks to learn patterns and make predictions.These methods use deep learning models to predict the values of variables and constraints, optimising the search process and improving the resolution of CSPs.

Deep learning methods have been applied to various types of CSPs, including scheduling, planning, and optimisation problems. They have been used to predict the values of variables and constraints, optimising the search process and improving the resolution of CSPs.

VIII.1 SUPERVISED LEARNING METHODS

Supervised machine learning [28] is a machine learning method where a model is trained on labeled data, meaning examples for which the expected answers are known. This process involves using the labeled data to learn relationships between inputs and outputs, allowing the model to "learn" the relationship between them. Based on known patterns, they generate a model capable of making accurate predictions on new data.

The supervised leaning methods have been applied in the context of CSPs by learning from labelled data and identifying optimal solution configurations, thus guiding research towards more efficient and reliable solutions for solving CSPs.

Among the supevised leaning methods used to solve CSPs, [29] uses a Convolutional Neural Network (CNN) on binary boolean CSPs to predict the satisfiability of CSPs, it includes domain adaptation and data augmentation techniques to handle the sparsity of labelled data. [30] uses a supervised model to learn how to optimise the ordering of variables in a search tree, reducing the depth of searches in CSPs and making resolution more efficient, [31] creates a general framework for selecting the optimal algorithm for each type of CSP, based on supervised learning models that analyse past performance and adjust algorithms accordingly [32] apply the Recurrent Transformer to learn how to solve CSPs. This approach offers an alternative to Graphical Neural Networks (GNNs) and neuro-symbolic models by effectively capturing constraints, especially for visual CSP problems.

VIII.2 UNSUPERVISED LEARNING METHODS

Unsupervised machine learning [33] is a machine learning method where the model is trained on unlabeled data, where only the inputs are available without any expected answers. The goal is to uncover hidden structures or underlying patterns within the data. It detect structures or patterns in data without the use of labels or pre-labelled examples, by learning to group similar data or reduce data dimensionality, the model builds a representation that can reveal useful patterns.

These methods have been applied to various types of CSP, such as planning, scheduling and optimisation problems. Various unsupervised learning methods have been used to solve CSPs, [34] uses a Deep Neural Network (DNN) agnostic model with no prior knowledge of specific constraints, allowing possible solutions to be explored using an agnostic approach that learns from experience about the structures of CSPs, in [35] used GNN and exploits their power to understand and exploit the connections between nodes in a CSP graph, improving the representation of constraints and helping to define efficient global heuristics for solving them.

VIII.3. REINFORCEMENT LEARNING METHODS

Reinforcement Learning (RL) [36] is an approach where an agent learns through direct interactions with an environment, receiving rewards or penalties based on its actions. The objective is to optimize the agent’s strategy to maximize cumulative rewards over time. Unlike supervised methods, there is no immediate correct answer for each situation. The agent explores and adjusts its choices based on the feedback it receives.

In the context of CSPs, RL can be used to improve search heuristics or dynamically adapt solving strategies, such as the choice of variables or values. RL can help prioritise tasks or optimise assignments according to constraints, by learning which actions lead most efficiently to find solution. Some of the techniques used to solve CSPs, as in [37] applies RL algorithm to learn a value function that adapt solving strategies to the specific characteristics of CSP instances, making it easier to solve new similar cases based on accumulated experience, this model adapts search decisions based on the complexity of constraints and optimises realtime search, [38] Integrates a RL model to guide branching decisions in the SeaPearl solver, using the historical characteristics of solutions to guide the process, [39] uses a policy gradient trained GNN approach to learn global heuristics for CSPs without explicit supervision. The model is tuned by feedback on the performance of the heuristic search, enabling various types of constraints to be handled in a single model.

IX. CONCLUSIONS

This paper has provided an overview of CSPs, detailing their formal definition, core components, and the variety of constraints involved. We classified CSP solving methods into five main categories: Systematic Search Methods, Inference and Filtering Methods, Structural Decomposition Methods, Local Search Based Methods, and Deep Learning Based Methods. Each method was analyzed in terms of its approach, efficiency, and application scenarios. Systematic methods like backtracking offer completeness but suffer from high computational cost. Inference methods enhance efficiency by pruning the search space, while structural decomposition simplifies complex problems by leveraging their inherent structure. Local search methods provide flexibility and efficiency in large, dynamic search spaces. Lastly, deep learning techniques, including supervised, unsupervised, and reinforcement learning, represent a growing frontier in CSP solving, offering automated learning and heuristic generation. This classification not only aids in understanding but also in selecting appropriate methods for specific CSP instances. Future work could focus on hybrid approaches that combine the strengths of these methods, particularly integrating machine learning and deep learning techniques with traditional algorithms for adaptive and scalable CSP solving.

X. AUTHOR’S CONTRIBUTION

Conceptualization: Fatima AIT HATRIT1, Kamal AMROUN2
Methodology: Fatima AIT HATRIT1, Kamal AMROUN2
Investigation: Fatima AIT HATRIT1, Kamal AMROUN2
Discussion of results: Fatima AIT HATRIT1, Kamal AMROUN2
Writing – Original Draft: Fatima AIT HATRIT1, Kamal AMROUN2
Writing – Review and Editing: Fatima AIT HATRIT1, Kamal AMROUN2

Resources: Fatima AIT HATRIT1, Kamal AMROUN2
Supervision: Fatima AIT HATRIT1, Kamal AMROUN2
Approval of the final text: Fatima AIT HATRIT1, Kamal AMROUN2

XI. REFERENCES

[1] K. R. Chowdhary, « Constraint Satisfaction Problems », in Fundamentals of Artificial Intelligence, New Delhi: Springer India, 2020, p. 273-302. doi: 10.1007/978-81-322-3972-7_10.

[2] 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.

[3] J. K. Behrens, R. Lange and M. Mansouri, « A Constraint Programming Approach to Simultaneous Task Allocation and Motion Scheduling for Industrial Dual-Arm Manipulation Tasks », 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, p. 8705-8711, 2019, doi: 10.1109/ICRA.2019.8794022.

[4] 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.

[5] M. Dohmen, « A survey of constraint satisfaction techniques for geometric modeling », Computers & Graphics, vol. 19, no 6, p. 831-845, nov. 1995, doi: 10.1016/0097-8493(95)00055-0.

[6] B. Bogaerts, E. Gamba, and J. Claes, « Step-Wise Explanations of Constraint Satisfaction Problems », In : ECAI 2020. IOS Press, 2020. p. 640-647.

[7] S. C. Brailsford, C. N. Potts, and B. M. Smith, « Constraint satisfaction problems: Algorithms and applications », European Journal of Operational Research, vol. 119, no 3, p. 557-581, 1999.

[8] S. W. Golomb and L. D. Baumert, « Backtrack Programming », J. ACM, vol. 12, no 4, p. 516-524, oct. 1965, doi: 10.1145/321296.321300.

[9] J. Gaschig, « Performance Measurement and Analysis of Certain Search Algorithms », Carnegie Mellon University, 1979.

[10] G. Audemard, C. Lecoutre, and C. Prud'homme, « Guiding Backtrack Search by Tracking Variables During Constraint Propagation », LIPIcs, Volume 280, CP 2023, vol. 280, p. 9:1-9:17, 2023, doi: 10.4230/LIPICS.CP.2023.9.

[11] D. Habet and C. Terrioux, « Conflict history based heuristic for constraint satisfaction problem solving », J Heuristics, vol. 27, no 6, p. 951-990, déc. 2021, doi: 10.1007/s10732-021-09475-z.

[12] H. Li, M. Yin, and Z. Li, « Failure Based Variable Ordering Heuristics for Solving CSPs (Short Paper) », LIPIcs, Volume 210, CP 2021, vol. 210, p. 9:1-9:10, 2021, doi: 10.4230/LIPICS.CP.2021.9.

[13] P. Prosser, « HYBRID ALGORITHMS FOR THE CONSTRAINT SATISFACTION PROBLEM », Computational Intelligence, vol. 9, no 3, p. 268-299, août 1993, doi: 10.1111/j.1467-8640.1993.tb00310.x.

[14] M. Shahraeeni, « Enhanced Multiple-Point Statistical Simulation with Backtracking, Forward Checking and Conflict-Directed Backjumping », Math Geosci, vol. 51, no 2, p. 155-186, févr. 2019, doi: 10.1007/s11004-018-9761-y.

[15] A. K. Mackworth, « Consistency in networks of relations », Artificial Intelligence, vol. 8, no 1, p. 99-118, févr. 1977, doi: 10.1016/0004-3702(77)90007-8.

[16] C. Bessière and J.-C. Régin, « MAC and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problems », in Principles and Practice of Constraint Programming — CP96, vol. 1118, E. C. Freuder, Éd., in Lecture Notes in Computer Science, vol. 1118. , Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, p. 61-75. doi: 10.1007/3-540-61551-2_66.

[17] N. Robertson et P. D. Seymour, « Graph minors. II. Algorithmic aspects of tree-width », Journal of Algorithms, vol. 7, no 3, p. 309-322, sept. 1986, doi: 10.1016/0196-6774(86)90023-4.

[18] 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.

[19] S. L. Lauritzen and D. J. Spiegelhalter, « Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems », Journal of the Royal Statistical Society. Series B (Methodological), vol. 50, no 2, p. 157-224, 1988.

[20] 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.

[21] G. Gottlob, N. Leone, and F. Scarcello, « Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width », J. Comput. Syst. Sci., vol. 66, no 4, p. 775-808, juin 2003, doi: 10.1016/S0022-0000(03)00030-8.

[22] 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.

[23] G. Gottlob, M. Lanzinger, D. M. Longo, and C. Okulmus, « Incremental Updates of Generalized Hypertree Decompositions », ACM J. Exp. Algorithmics, vol. 27, p. 1-28, déc. 2022, doi: 10.1145/3578266.

[24] S. J. Russell et P. Norvig, Artificial intelligence: a modern approach, Fourth edition, Global edition. in Prentice Hall series in artificial intelligence. Boston: Pearson, 2022.

[25] A. Kaznatcheev, D. A. Cohen, et P. G. Jeavons, « Representing fitness landscapes by valued constraints to understand the complexity of local search », 12 novembre 2020, arXiv: arXiv:1907.01218.

[26] M. Gelain, M. Silvia Pini, F. Rossi, and K. B. Venable, « A LOCAL SEARCH APPROACH TO SOLVE INCOMPLETE FUZZY CSPs »:, in Proceedings of the 3rd International Conference on Agents and Artificial Intelligence, Rome, Italy: SciTePress - Science and and Technology Publications, 2011, p. 582-585. doi: 10.5220/0003174505820585.

[27] A. Manno, E. Amaldi, F. Casella, et E. Martelli, « A local search method for costly black-box problems and its application to CSP plant start-up optimization refinement », Optim Eng, vol. 21, no 4, p. 1563-1598, déc. 2020, doi: 10.1007/s11081-020-09488-w.

[28] Y. Bengio, I. Goodfellow, and A. Courville, « Deep learning »: The MIT Press, 2016, doi: 10.1007/s10710-017-9314-z .

[29] H. Xu, S. Koenig, and T. K. S. Kumar, « Towards Effective Deep Learning for Constraint Satisfaction Problems », in Principles and Practice of Constraint Programming, vol. 11008, J. Hooker, Éd., in Lecture Notes in Computer Science, vol. 11008. , Cham: Springer International Publishing, 2018, p. 588-597. doi: 10.1007/978-3-319-98334-9_38.

[30] W. Song, Z. Cao, J. Zhang, and A. Lim, « Learning Variable Ordering Heuristics for Solving Constraint Satisfaction Problems », Engineering Applications of Artificial Intelligence, vol. 109, p. 104603, mars 2022, doi: 10.1016/j.engappai.2021.104603.

[31] J. C. Ortiz-Bayliss, I. Amaya, J. M. Cruz-Duarte, A. E. Gutierrez-Rodriguez, S. E. Conant-Pablos, and H. Terashima-Marín, « A General Framework Based on Machine Learning for Algorithm Selection in Constraint Satisfaction Problems », Applied Sciences, vol. 11, no 6, p. 2749, mars 2021, doi: 10.3390/app11062749.

[32] Z. Yang, A. Ishay, and J. Lee, « Learning to Solve Constraint Satisfaction Problems with Recurrent Transformer », 10 juillet 2023, arXiv: arXiv:2307.04895.

[33] T. Hastie, R. Tibshirani, and J. Friedman, The Elements of Statistical Learning. in Springer Series in Statistics. New York, NY: Springer, 2009. doi: 10.1007/978-0-387-84858-7.

[34] A. Galassi, M. Lombardi, P. Mello, and M. Milano, « Model Agnostic Solution of CSPs via Deep Learning: A Preliminary Study », in Integration of Constraint Programming, Artificial Intelligence, and Operations Research, vol. 10848, W.-J. Van Hoeve, Éd., in Lecture Notes in Computer Science, vol. 10848. , Cham: Springer International Publishing, 2018, p. 254-262. doi: 10.1007/978-3-319-93031-2_18.

[35] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, « How Powerful are Graph Neural Networks? », 22 février 2019, arXiv: arXiv:1810.00826.

[36] R. S. Sutton et A. Barto, Reinforcement learning: an introduction, Nachdruck. in Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press, 2014.

[37] Y. Xu, D. Stern, and H. Samulowitz, « Learning Adaptation to Solve Constraint Satisfaction Problems », Proceedings of Learning and Intelligent Optimization (LION), 2009, p. 14.

[38] F. Chalumeau, I. Coulon, Q. Cappart, and L.-M. Rousseau, « SeaPearl: A Constraint Programming Solver guided by Reinforcement Learning », 20 avril 2021, arXiv: arXiv:2102.09193.

[39] J. Tönshoff, B. Kisin, J. Lindner, and M. Grohe, « One Model, Any CSP: Graph Neural Networks as Fast Global Search Heuristics for Constraint Satisfaction », 22 août 2022, arXiv: arXiv:2208.10227.