The Impact of Solution Diversity on Passive Constraint Acquisition

Vasileios Balafas
v.balafas@uowm.gr
Department of Electrical and Computer Engineering,
University of Western Macedonia
Kozani, Greece

Dimosthenis C. Tsouros
dimos.tsouros@kuleuven.be
Department of Computer Science, KU Leuven
Leuven, Belgium

Nikolaos Ploskas
nploskas@uowm.gr
Department of Electrical and Computer Engineering,
University of Western Macedonia
Kozani, Greece

Kostas Stergiou
kstergiou@uowm.gr
Department of Electrical and Computer Engineering,
University of Western Macedonia
Kozani, Greece

Abstract

Constraint programming provides a powerful framework for modeling and solving combinatorial problems. However, manually defining the required constraints can be a challenging task that requires a high level of expertise. Constraint acquisition (CA) techniques aim to semi-automate this process by learning constraints from examples of solutions and non-solutions. One important factor that can impact the effectiveness of CA is the diversity of the example solutions provided. This paper investigates how solution diversity influences passive learning approaches for CA across three distinct problems and various diversity metrics. Our results demonstrate that solution diversity significantly influences the quality of learned constraints, highlighting the importance of diverse solution sets. In addition, we show how we can predict whether a given set of solutions will enable accurate constraint learning using a machine learning (ML) model. Our experimental evaluation shows that the ML model can accurately predict the recall of the CA system based on the solution set's diversity metrics and the number of solutions.

CCS Concepts

• **Theory of computation → Constraint and logic programming**.

Keywords

Constraint Programming, Constraint Acquisition, Diversity of Solutions, Machine Learning

ACM Reference Format:
Vasileios Balafas, Dimosthenis C. Tsouros, Nikolaos Ploskas, and Kostas Stergiou. 2024. The Impact of Solution Diversity on Passive Constraint Acquisition. In *13th Conference on Artificial Intelligence (SETN 2024), September 11–13, 2024, Piraeus, Greece.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3688671.3688759

SETN 2024, September 11–13, 2024, Piraeus, Greece

ACM ISBN 979-8-4007-0982-1/24/09
https://doi.org/10.1145/3688671.3688759

1 Introduction

Constraint Programming (CP) is a paradigm for modeling and solving Constraint Satisfaction Problems (CSPs) using techniques from AI, computer science, and operations research. CP is applied in various domains, such as scheduling, resource allocation, and assignment problems [27, 34]. To model a real-world problem as a CSP, the user has to define the variables, their domains, and the constraints of the problem. Variables represent the entities of the problem on which some decision needs to be made, domains define possible values for the variables, and constraints describe relationships between variables. By defining these elements, CP facilitates the formulation of complex problems in a way that is both intuitive and close to their original definitions, hence providing a convenient approach to their solution. However, the modeling process has been identified as an important bottleneck for the wider user of CP, as substantial expertise is required for the identification and formulation of the constraints [9, 10]. This has led to an increasing interest in Constraint Acquisition (CA) techniques, which can help to automate the modeling process [6, 7, 31].

CA relies on examples of solutions and non-solutions, and possibly on user feedback, to generate constraints iteratively, potentially saving considerable time and effort compared to the laborious manual modeling of the constraints. There are two primary CA learning paradigms: passive learning (PL) and active learning (AL). AL is a paradigm that exploits explicit user input to acquire constraints. This input is obtained by soliciting feedback from the user, which can be a human or a software tool, on whether a given example (i.e., a set of variable assignments) constitutes a solution. AL is particularly focused on acquiring fixed-arity constraints, with no AL system so far being able to learn global constraints.

In PL, the system tries to acquire constraints by observing and analyzing examples of solutions (and possibly non-solutions) without requiring explicit user input. Most PL approaches rely only on solutions of the problem, as they can be derived from previously solved instances of a problem, historical datasets that record solutions to similar problems, or simulations of the problem scenario. Most PL systems primarily focus on acquiring global constraints, or patterns of constraints, by analyzing the available solutions and identifying common patterns among them. Then, AL can be employed to complete a basic model obtained through PL.

Although the quality of the given set of solutions is of high importance in passive CA, its impact is rather overlooked in the literature. Passive CA systems are commonly evaluated on sets of solutions artificially generated, focusing mainly on the number of solutions used and not on how these solutions are structured. As there are no standard evaluation sets or ways of generating them, the obtained results can be highly affected by the quality of the sets of solutions used. One significant factor regarding the set of solutions used in the acquisition process is their diversity, i.e., how different they are. Like in standard machine learning, where the diversity of solutions has been identified as a very important factor, along with the size of the dataset used [12, 30], diverse solutions in CA provide a broader representation of the problem space, which may enhance the correctness of the acquired set of constraints. By analyzing a wider range of solutions, CA systems can identify and confirm a variety of patterns, leading to the formulation of constraint models that more accurately capture the given problem. Also, incorporating diverse solutions can lead to the detection of rare but critical scenarios that might otherwise be overlooked, thereby improving the system's performance in real-world applications.

In this work, we focus on enhancing the understanding of the impact of solution diversity on PL methods. We evaluate the impact of solution diversity on PL methods in three problems: Sudoku, Greater than Sudoku, and WLP. To this end, we evaluate the performance of a passive CA system, using different sets of solutions, that vary in their diversity. We used well-known diversity metrics to measure solution diversity. Our experiments focus on evaluating how close the learned set of constraints is to the target set of constraints for the given benchmarks. Our experiments demonstrate that solution diversity significantly impacts the effectiveness of passive learning methods in constraint acquisition. Higher diversity in solution sets improves constraint learning accuracy.

However, in real-world applications, it is unlikely that we will have a large pool of (diverse) solutions available. Thus, we cannot easily predict how accurate the CP model extracted using CA will be prior to applying a CA method. To tackle this, we propose to apply diversity metrics on the available set of solutions in order to evaluate its quality. We show how a machine learning (ML) model can be used to predict whether or not a given set of example solutions will result in a good CSP model, based on a feature representation of the given solutions set, regarding its diversity and amount of solutions it contains. The results demonstrate that the ML model can accurately predict the recall (quality) of the constraint model learned by the CA system, based on features capturing the diversity and size of the given solution set. Across all benchmarks the ML model achieved very high R^2 scores, ranging from 97.61% to 99.89%, when using different diversity metrics as features. These high R^2 values indicate that the machine learning model can reliably estimate the expected recall, and consequently the quality of the learned CP model. This predictive capability is valuable when we do not have access to a large, diverse pool of solutions, allowing us to assess if the available solutions are sufficient for accurate CA or if more data is needed.

To summarize, we address the following key questions:

(1) How does solution diversity impact the effectiveness of PL methods in semi-automated CP modeling across different problems?
(2) Can we develop a predictive model to estimate the expected recall of the learned constraints given the diversity measures and the number of solutions provided to the CA process?

The remainder of this paper is structured as follows. Background on CP, CA, diversity of solutions, and supervised ML is given in Section 2. In Section 3, we review the related work. Following this, we outline our methodology employed to evaluate the impact of solution diversity on PL methods in CA systems and present the results obtained from the experimental analysis in Section 4. In Section 5, we present an ML model to estimate the expected recall of the learned constraints. Finally, Section 6 concludes the paper.

2 Background

2.1 Constraint Programming

Constraint Programming (CP) is a powerful paradigm for solving combinatorial problems. Formally, a Constraint Satisfaction Problem (CSP) [33] can be defined as a triple (X, D, C):

- $X = \{x_1, x_2, \ldots, x_n\}$: a set of n variables.
- $D = \{D_{x_1}, D_{x_2}, \ldots, D_{x_n}\}$: a set of n domains, where each domain D_{x_i} is the finite set of possible values that variable x_i can take.
- $C = \{c_1, c_2, \ldots, c_m\}$: A set of m constraints. Each constraint c_i specifies the allowable combinations of values for a subset of variables.

The goal is to find an assignment a for the variables in X such that all constraints in C are satisfied. Such an assignment a is called a solution s of C. The set of all solutions of C is notated as $sol(C)$.

Constraints in CP are divided into two main types: *fixed-arity* and *global* constraints. Fixed-arity constraints involve a predetermined, number of variables. When these constraints apply to exactly two variables, they are known as *binary*. Common fixed-arity constraints include various mathematical relations among a specified number of variables, e.g. $x_1 > x_2$. On the other hand, global constraints are constraints applied to sequences of variables, thus, they do not have a fixed-arity, providing a global view of the patterns present in the problem [26]. Various global constraints exist to capture different patterns in CSPs [2]. In this work, we define the constraints `allDifferent` and `count`, as these are present in the problems used in our experimental analysis. The `allDifferent` constraint applies to a set of variables X' and requires that each pair of variables $x_i, x_j \in X'$ must satisfy $x_i \neq x_j$ for $i \neq j$ [25, 29]. The `count` constraint [2], denoted as $\text{Count}(X', v, c)$, specifies that exactly c variables in the set X' should take on the value v.

2.2 Constraint Acquisition

In Constraint Acquisition, the goal is to learn the set of constraints C of a CSP, given a set of examples. That is, a set of solutions S and possibly a set of non-solutions N. The pair (X, D) is called the *vocabulary* of the problem at hand and is common knowledge shared by the user and the system.

Besides the vocabulary, the learner is also given a *language* Γ with the possible relations of (global or fixed-arity) constraints that may exist in the problem. Using the vocabulary (X, D) and the constraint language Γ, the system builds a *constraint bias* B, which is the set of candidate constraints for the problem (i.e. all

the constraints that may possibly exist in the model). This set of candidate constraints can either include the constraints derived by applying the relations in Γ to all combinations of variables in X or only use specific assumptions on the patterns to search for.

Let C_T, the target constraint network, be an unknown set of constraints such that for every assignment e over X it holds that $e \in sol(C_T)$ iff e is a solution to the problem the user has in mind. The goal of CA is to learn a constraint set $C_L \subseteq B$ that is equivalent to C_T. As in the literature, we assume that the bias B can represent C_T, i.e., there exists a $C \subseteq B$ s.t. $sol(C) = sol(C_T)$. Thus, a set equivalent to C_T can be extracted from B. The acquisition process has *converged* on the learned network $C_L \subseteq B$ iff consistent with the given examples of solutions and non-solutions, i.e., $S \subseteq sol(C_L) \wedge N \cap sol(C_L) = \emptyset$, and for every other consistent constraint network $C \subseteq B$, it holds that $sol(C) = sol(C_L)$.

In passive CA the set of solutions and non-solutions is preexisting, thus, the methods typically focus on learning a constraint network that meets the first property. As shown in [7], proving the second property is coNP-complete, requiring an exponential number of examples. As a result, depending on the method used, the set of constraints learned can be more restrictive than C_T, i.e., $sol(C_L) \subsetneq sol(C_T)$, or more loose than C_T, i.e., $sol(C_T) \subsetneq sol(C_L)$.

2.3 Diversity

As we will demonstrate, the diversity of the examples in the given sets of solutions and non-solutions can significantly impact the quality of the derived constraint models. Mathematically, the diversity of assignments for a CSP can be quantified using various metrics. Commonly used diversity metrics include the Hamming distance, L_1-norm, and L_2-norm. Given two example solutions $s = (s_1, s_2, \ldots, s_n)$ and $s' = (s'_1, s'_2, \ldots, s'_n)$, these metrics are defined as follows:

- **Hamming distance:** measures the number of positions at which the corresponding elements of the examples are different. It is defined as:

$$d_H(s, s') = \sum_{i=1}^{n} \mathbb{I}(s_i \neq s'_i)$$

  where $\mathbb{I}$ is the indicator function, which is 1 if the condition is true and 0 otherwise.
- **L_1-norm (Manhattan distance):** measures the sum of the absolute differences between corresponding elements of the examples. It is defined as:

$$d_{L_1}(s, s') = \sum_{i=1}^{n} |s_i - s'_i|$$

- **L_2-norm (Euclidean distance):** measures the square root of the sum of the squared differences between corresponding elements of the examples. It is defined as:

$$d_{L_2}(s, s') = \sqrt{\sum_{i=1}^{n} (s_i - s'_i)^2}$$

2.4 Supervised Machine Learning

Supervised ML is a task that involves learning a function over a given dataset. The dataset, denoted as $\mathbf{E}$, is a collection of N training examples, $\mathbf{E} = \{(\phi_1, y_1), (\phi_2, y_2), ..., (\phi_N, y_N)\}$. Each training example is a pair (ϕ_i, y_i), where ϕ_i is a feature vector from the input space Φ and y_i is the target value from the output space Y. The feature vector ϕ_i is composed of m features, $\phi_i = (\phi_{i1}, \phi_{i2}, \ldots, \phi_{im})$, with each feature ϕ_{ij} being a quantifiable property or characteristic of training example i. In the case of classification, Y contains discrete values, each representing a class of the problem. In the case of regression, Y contains continuous values. A supervised learning model aims to learn a function $f_\theta : X \rightarrow Y$, parameterized by a set of learnable parameters θ. These parameters are adjusted during the training process to minimize a loss function $L(f_\theta(\phi), y)$, which measures the error between the predicted and actual target values.

The performance of regression models is often evaluated using the coefficient of determination, denoted as R^2. It is defined as:

$$R^2 = 1 - \frac{\sum_{i=1}^{N} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{N} (y_i - \bar{y})^2}$$

where y_i is the actual target value, $\hat{y}_i$ is the predicted value, and $\bar{y}$ is the mean of the actual target values. An R^2 score of 1 indicates perfect prediction, while a score close to 0 indicates that the model cannot predict better than a mean predictor.

3 Related Work

This section reviews the key developments in passive CA algorithms and the generation of diverse solutions.

3.1 Passive CA

Due to the increased interest in semi-automating the modeling process, several approaches for passive CA have been introduced. The various approaches primarily differ on the following: the type of constraints they focus on learning, their ability to handle noisy input, and whether they require only solutions or also non-solutions for the acquisition process. An early approach to passive CA is the algorithm ConAcq.1 [4, 5, 7], which learns constraints from user-provided examples of solutions and non-solutions. A passive learning method based on inductive logic programming was proposed in [15]. This system uses background knowledge on the structure of the problem to learn a representation of the problem, correctly classifying the examples given. An One-Class Constraint Acquisition with Local Search (OCCALS) algorithm was presented in [28], which, given a one-class training set (i.e. only examples of solutions), acquires a non-convex MILP model.

None of these approaches are robust to errors in the labeled data. To this end, SeqAcq and BayesAcq were introduced, being robust to noise in the training set. In SeqAcq, a statistical approach based on sequential analysis is used [22], while in BayesAcq, a naive Bayes classifier is trained to distinguish between solution and non-solution, using the candidate constraints as features. Then, based on the trained classifier, a set of constraints is derived [23]. Additionally, MINEACQ [20] was recently introduced, using unsupervised learning that is capable of learning from both labeled and unlabeled data to get around the data collection bottleneck.

A limitation of the previously mentioned methods is that they operate only with fixed-arity constraints. On the other hand, ModelSeeker [3], a well-known CA system, learns global constraints. It leverages the global constraint catalog [2] to derive constraint models from positive examples, based on patterns that commonly

exist in CP models. Although it cannot learn fixed-arity constraints, it is powerful on learning complex constraint problems.

Another direction was presented with the COUNT-CP method [14]. COUNT-CP is a generate-and-aggregate approach, in contrast to the generate (candidate constraints) and test approach that is commonly used. It learns fixed-arity bounded expressions, aggregating their bounds based on the input set of solutions to the problem. Its main advantage is that it also involves a generalization step, to learn expressive first-order constraints.

In many works, it is assumed that a sufficient number of examples is needed to learn a set of constraints representing the problem, with the evaluation procedure typically using thousands of examples [21, 22]. There are a few systems that can learn from a small number of solutions, presenting an evaluation over sets of solutions of different size [2, 14]. However, to the best of the authors' knowledge, there is no work evaluating the effect of the diversity of the solutions given to CA on its performance, although it is a very important factor.

Most passive CA systems return the most restrictive consistent constraint set. Non-diverse solutions result in a small set of eliminated candidates, leading to many additional constraints. For example, although ModelSeeker has been shown to perform well with a few solutions, its effectiveness can be limited if the solutions are not sufficiently diverse or representative [3]. Non-solution diversity is crucial for validating constraints [22]. To this end, we evaluate the effect of solution diversity on the performance of CA systems, focusing on the extracted set of constraints.

3.2 Diverse Solution Generation

Solution diversity in CP refers to generating multiple solutions to a constraint satisfaction or optimization problem, with each solution being significantly different from the others. The generation of such diverse solutions has received attention in the literature, as it can be useful in several scenarios, e.g. when the user preferences are partially known, due to being difficult to model, or when we want to use different solutions of the same staff scheduling to ensure fairness across the workers.

Hebrard et al. [13] explored the generation of diverse solutions using CP. Their approach leverages heuristic approaches to find solutions that are significantly different from each other, thus providing a broader perspective of the solution space. They introduced several metrics for measuring solution diversity, including Hamming distance and various domain-specific criteria. Their experiments on different benchmarks are very encouraging, and their method has been widely adopted [8, 16]. Petit et al. [18] proposed a framework for generating diverse and high-quality solutions, addressing limitations of methods focusing on either diversity or quality. Petit and Trapp [19] introduced solution engineering to enhance quality and diversity by modifying existing solutions to meet user criteria. Vadlamudi et al. [32] proposed algorithms guaranteeing a set of K diverse solutions, overcoming limitations of heuristic methods.

While the generation of diverse solutions using CP methods has received attention in the literature, being useful in several scenarios, there are no studies that focus on the impact of solution diversity in CA. In this paper, we investigate how solution diversity influences passive learning approaches for CA, and how we can use diversity metrics to predict whether a given set of solutions will enable accurate constraint learning.

4 Evaluating the Impact of Solution Diversity

In this section, we first outline the methodology employed to evaluate the impact of solution diversity on passive learning methods in CA systems. Then we describe our experimental setup, followed by the discussion of the results.

4.1 Methodology

We first briefly describe the CA system used, and then we discuss the different methods that can be used to generate sets of solutions to systematically evaluate passive CA systems. Finally, we show how we assess the diversity impact in the learned constraints set.

Constraint Acquisition System

We utilized a passive CA system that focuses on learning global constraints [1], though any CA system can be used. The CA system employed in this study aims to learn the set of constraints C using a given set of solutions $S \subseteq sol(C)$. Inspired by the well-known ModelSeeker system, it focuses on learning global constraints. As a result, the language Γ includes relations corresponding to global constraints, such as `allDifferent` and `count`. The CA system performs the following steps:

- **Generating the Candidate Global Constraints:** Our CA system uses predefined partitions, like rows, columns, diagonals, and other sequential and local patterns. For each partition of variables that can be created based on the structure of the given problem, candidate global constraints from the language Γ are generated and added to the bias B. For example, variables within a row or column in Sudoku would be matched with the `allDifferent` constraint.
- **Filtering Candidate Constraints:** The system filters the set of candidate constraints B to retain only constraints that are consistent with the set of solutions S. If a constraint is satisfied across all solution sets, it is added to C_L.

$$C_L \leftarrow \{c \mid \bigwedge_{s \in S} s \in sol(c),\ \forall\, c \in B\}$$

The system outputs the learned constraint model $C_L \subseteq B$, that is consistent with the provided solutions S. As this approach is based on eliminating candidates, and not confirming constraints through their violation in non-solutions, it always learns the set of constraints in C_T, but can also learn additional ones, if the set of solutions is not diverse enough.

Generating Diverse Solution Sets

To investigate the impact of solution diversity on the performance of the CA system, we generate a large solution pool for each benchmark problem using a CP solver. Let $S_{pool} = \{s_1, s_2, \ldots, s_N\}$ denote this pool of solutions, where each s_i is a solution to the problem, and $N = |S_{pool}|$ being the size of S_{pool}.

Using the solution pool S_{pool}, we can create a diverse solution set S_d using three different approaches:

(1) **Sequential Collection**: Solutions are collected consecutively from the pool S_{pool}, with fixed gaps between them.

This introduces varying levels of diversity in the resulting solution sets. Formally, for a gap size g, the sequentially collected solution set S_d is defined as:

$$S_d = \{s_i \in S_{pool} \mid i \equiv 0 \pmod{g}\} \quad (1)$$

(2) **Random Selection**: Solutions are randomly selected from the pool S_{pool} and added to S_d.
(3) **Distance-Based Selection**: A solution set S_d is created using a predefined distance for Hamming, L_1, or L_2 metrics.

Evaluating Diversity Impact

To evaluate this impact, we count the number of global constraints in C_L, learned from each generated solution set after the CA process, compared to the set of global constraints in the target set of constraints C_T. When the solutions are not diverse enough, the CA system may identify spurious constraints that hold true for the given solutions but do not generalize to the entire feasible space. This leads to a learned constraint model that is more restrictive than the true model, potentially excluding valid solutions that were not present in the training set. This can be the case for the CA system used in this study when the solutions are not diverse enough. On the other hand, learning fewer constraints suggests that the CA system has not been able to acquire all the target constraints from the given set of solutions. In this case, the learned constraint model is more relaxed than the true model, allowing solutions that violate some of the actual constraints. This happens in CA systems that use non-solutions to confirm candidate constraints.

Ideally, the CA system should learn the exact set of constraints that define the problem. This indicates that the given set of solutions is diverse enough to accurately represent the problem space and allows the CA system to generalize effectively. By comparing the number of learned constraints to the known set of constraints for each benchmark problem, we can evaluate how well the CA system performs under different levels of solution diversity.

4.2 Experimental Setup

Our experiments aim to answer the following questions:

(Q1) How does the diversity of solutions affect the learned constraint models?
(Q2) How does the number of solutions influence the performance of the CA system?

We first describe the benchmark problems used in this study. Next, we outline the solution generation process, highlighting the methods employed to create diverse solution sets with the Hamming, L_1, and L_2 distances. We then explain the evaluation process.

Benchmark Problems

We selected three benchmark problems for this study: Sudoku, Greater Than Sudoku, and the Warehouse Location Problem (WLP).

Sudoku and Greater Than Sudoku. Sudoku is a popular puzzle played on a 9×9 grid where the objective is to fill the grid with numbers from 1 to 9 such that each row, column, and 3×3 subgrid contains distinct numbers. In the context of CP, `allDifferent` constraints are used to model the requirement that all variables within a specified set take different values. Thus, for Sudoku there are `allDifferent` constraints for each of the 9 rows, 9 columns, and 9 subgrids, making a total of 27 `allDifferent` constraints. Greater Than Sudoku extends this by adding inequality (> and <) constraints between given adjacent cells, making it more complex. The Greater than Sudoku boards used in this experimental study has ten inequality constraints.

Warehouse Location Problem (WLP). The WLP involves determining optimal warehouse locations to minimize the total costs of serving customers, including fixed costs for opening warehouses and variable costs for serving customers from these locations. We focused on the satisfaction part of the WLP, which includes constraints such as ensuring each customer is served by exactly one warehouse and limiting the number of customers a warehouse can serve. These constraints are captured using `count` constraints, which ensure that the number of customers assigned to each warehouse does not exceed a specified limit. Specifically, the `count` constraint $\text{Count}(X', i, K_i)$ ensures that the number of variables in the set X' that take the value i is exactly K_i. The WLP problem used in this experimental study had 10 warehouses and 20 customers.

Solution Generation

For each benchmark problem, we generated an extensive pool of solutions utilizing the Choco Solver [24]. Specifically, we produced solution pools including 100,000 solutions for Sudoku, Greater Than Sudoku, and 40,000 solutions for the WLP. These solution pools were used for constructing the datasets with varying diversity. The generation of these solutions required a cumulative computation time of 278 hours, underscoring the computational intensity and time-consuming nature of generating diverse solution sets.

Diversity Metrics and Dataset Creation

To evaluate the impact of solution diversity on the performance of the CA system, we created diverse solution sets for each benchmark problem using various distance metrics and selection methods. For each of the above solution sets, we generated sets of 5, 50, 100, 200, 500, and 1000 solutions in order to study how the number of solutions affect the CA system.

For Sudoku and Greater Than Sudoku, we generated solution sets that met specific distance criteria based on the following metrics:

- **Hamming Distance:** We created solution sets with Hamming distances of 1, 10, 20, 40, and 80.
- L_1 **Distance:** We generated solution sets with L_1 distances of 1, 216, 432, and 658.
- L_2 **Distance:** We generated solution sets with L_2 distances of 1, 24, 48, and 72.

For the WLP, which has fewer decision variables compared to Sudoku and Greater Than Sudoku, we adjusted the distance criteria to account for the smaller problem size:

- **Hamming Distance:** We created solution sets with Hamming distances of 1, 5, 10, and 20.
- L_1 **Distance:** We generated solution sets with L_1 distances of 2, 60, 120, and 180.
- L_2 **Distance:** We constructed solution sets with L_2 distances of 2, 14, 27, and 40.

We also employed the sequential and random selection strategies as follows:

- **Sequential Selection:** We collected solutions consecutively from the solution pool, with fixed gaps between each selected solution. We used gaps of 1, 100, 1,000, and 10,000 solutions to introduce varying levels of diversity in the solution sets.
- **Random Selection:** We randomly selected solutions from the solution pool to create ten different solution sets with various levels of diversity.

Evaluation Process

The generated solution sets were provided to the CA tool. The CA tool was executed using each diverse solution set. During this phase, the tool analyzed the solutions to infer the underlying constraints of the given problem. The number of constraints learned by the CA tool was used to evaluate the impact of solution diversity in CA.

4.3 Results

In this section, we present and discuss the results from evaluating the impact of solution diversity on passive CA systems

Table 1 shows the *AllDifferent* constraints that the CA system identified for diverse sets of Sudoku 9x9 puzzles. Solution diversity significantly affects identifying *AllDifferent* constraints. For Hamming distance, identified constraints decreased as distance increased. For instance, when using a dataset of 1,000 solutions with a Hamming distance of 1, 36 *AllDifferent* constraints were identified by the CA system, whereas a dataset of 1,000 solutions with a Hamming distance of 80 yielded the exact number of 27 constraints. Similar trends were observed for L_1 and L_2 distances, where larger distances correlated with identifying the correct number of constraints. For example, a dataset of 1,000 solutions with an L_1 distance of 658 led to the identification of the exact 27 constraints, compared to 36 constraints when a dataset of 1,000 solutions with a distance of 1 was used. Similarly, sequential collection methods with small gaps led the CA system to acquire more constraints compared to larger gaps. A large gap of 10,000 and 1,000 solutions was not diverse enough, leading to 31 identified constraints, i.e., 4 additional ones. Importantly, randomly selected solution sets led to the identification of the exact number of global constraints in the problem (27 for Sudoku) across almost all dataset sizes.

Table 2 presents the results for the Greater Than Sudoku benchmark. The CA system identified the exact number of *AllDifferent* constraints (27) when using datasets with a Hamming distance of 80, an L_1 distance of 658, or an L_2 distance of 72, even with a relatively small number of solutions (e.g., 500 or 1,000). Randomly selected solution sets also consistently identified the correct number of constraints across almost all dataset sizes. As with Sudoku, smaller Hamming, L_1, and L_2 distances resulted in the identification of more constraints than the actual number present in the problem. In addition, larger datasets helped the CA tool identify fewer constraints, as it was expected.

Table 3 presents the results for the WLP benchmark. Here, we focus on the number of *Count* constraints that the CA system identified. The CA system successfully identified the correct number of *Count* constraints (10) when using datasets with a sufficient number of solutions (500 or more), when any distance metric is employed. However, for smaller dataset sizes, the system tended to identify more constraints than the actual ones, with the exception of randomly selected solution sets, which consistently led to the identification of the correct number of constraints across all dataset sizes. Sequential collection methods with large gaps (e.g., 1,000 or 10,000) also identified the correct number of constraints, even with smaller dataset sizes.

Smaller diversity metrics lead to identifying more constraints than present. This suggests that solution sets with high similarity (i.e., low distance diversity) may not provide sufficient diversity for the CA system to accurately learn the underlying constraints. In contrast, larger distance metrics and randomly selected solution sets tend to yield better results, enabling the CA system to identify the correct number of constraints more consistently.

Interestingly, the effectiveness of different distance metrics varies across the problem types. For Sudoku and Greater Than Sudoku, the Hamming distance appears to be the most informative, with high distances leading to the identification of the exact number of constraints even with relatively small dataset sizes. In contrast, for the WLP, the choice of distance metric has less impact.

Another observation is that the CA system's performance improves as the number of solutions in the dataset increases. This is particularly evident in the WLP results, where datasets with 1,000 or more solutions consistently lead to the identification of the correct number of constraints, irrespective of the distance metric employed. This highlights the importance of having a sufficiently large and diverse set of solutions for effective constraint acquisition.

5 Predicting the Quality of the Acquired Constraint Set

In the previous section, we showed that the diversity of solutions is very important for the quality of the acquired set of constraints, combined with the number of solutions used. However, in real-world applications, it is often challenging to obtain a large and diverse set of solutions for training the CA system. The set of solutions is typically preexisting, and not generated with the purpose of being used for CA. Consequently, it becomes difficult to assess the quality of the learned constraint model. To address this issue, we propose the use of an ML model, for this purpose. As we demonstrate, by leveraging the predictive capabilities of ML we can assess the suitability of the solution set for CA beforehand.

In this section, we first show how a feature representation for the size and diversity of the given set of solutions can be used to estimate whether a given set of solutions will result in a reliable CP model. Then we describe the experimental setup used for evaluating our approach, followed by a discussion of the results.

5.1 Methodology

To use an ML model for this purpose, we build a set of training examples: $\mathbf{E} = \{(\phi_1, y_1), (\phi_2, y_2), ..., (\phi_N, y_N)\}$. In our dataset, each training example (ϕ_i, y_i) represents a CA instance. The features ϕ_i correspond to the characteristics of the set of solutions given, while the target value represents the quality of the CP model that can be extracted using a given CA method. In the dataset, we use instances of different sizes, so that the ML model can learn to predict the

Table 1: Number of *AllDifferent* Constraints Learned for Sudoku 9x9 under Various Diversity Metrics

	Hamming					L_1				L_2				Sequential				Rnd
# Sols	**1**	**10**	**20**	**40**	**80**	**1**	**216**	**432**	**658**	**1**	**24**	**48**	**72**	**1**	**100**	**1,000**	**10,000**	
1,000	36	34	32	28	**27**	36	32	29	**27**	34	31	**27**	**27**	31	31	31	31	**27**
500	36	34	32	29	**27**	37	34	31	**27**	34	31	37	**27**	32	32	31	31	**27**
200	37	36	33	31	**27**	39	37	34	29	36	31	37	28	32	32	31	31	**27**
100	38	37	34	31	28	39	38	34	31	37	31	38	29	32	32	31	31	**27**
50	45	37	34	31	31	41	39	36	31	38	32	41	31	32	32	32	32	**27**
5	46	45	39	37	37	46	46	41	39	46	38	46	36	38	38	40	39	39

Note: 27 is the actual number of *AllDifferent* constraints.

Table 2: Number of *AllDifferent* Constraints Learned for Greater Than Sudoku 9x9 under Various Diversity Metrics

	Hamming					L_1				L_2				Sequential				Rnd
# Sols	**1**	**10**	**20**	**40**	**80**	**1**	**216**	**432**	**658**	**1**	**24**	**48**	**72**	**1**	**100**	**1,000**	**10,000**	
1,000	34	32	37	28	**27**	33	35	31	**27**	38	35	**27**	**27**	32	32	32	32	**27**
500	39	32	37	30	**27**	33	35	33	29	39	36	29	**27**	32	32	32	32	**27**
200	40	34	38	30	**27**	34	36	33	33	40	36	33	30	32	32	32	32	**27**
100	42	35	38	30	**27**	34	36	36	31	40	36	36	30	34	32	32	32	**27**
50	43	34	38	31	**27**	35	36	37	37	41	37	37	31	34	32	32	32	**27**
5	47	36	39	38	**27**	40	38	43	45	45	39	43	38	38	40	38	38	35

Note: 27 is the actual number of *AllDifferent* constraints.

Table 3: Number of *Count* Constraints Learned for WLP with 10 Warehouses and 20 Customers under Various Diversity Metrics

	Hamming				L_1				L_2				Sequential				Rnd
# Sols	**1**	**5**	**10**	**20**	**2**	**60**	**120**	**180**	**2**	**14**	**28**	**40**	**1**	**100**	**1,000**	**10,000**	
1,000	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	12	12	**10**	**10**	**10**
500	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	12	12	**10**	**10**	**10**
200	12	12	11	**10**	13	11	11	**10**	12	12	12	**10**	12	12	**10**	**10**	**10**
100	13	13	13	**10**	15	13	13	12	14	14	14	12	12	12	**10**	**10**	**10**
50	14	14	14	13	17	14	14	13	15	15	15	13	14	12	**10**	**10**	**10**
5	15	15	15	14	18	15	15	14	16	16	16	15	20	12	12	12	**10**

Note: 10 is the actual number of *Count* constraints.

recall of the CA methods in a variety of instances of the problem, so that it can generalize to unseen instances.

In the rest of this section, we first discuss our proposed feature representation, and then we show how the quality of the CP model extracted can be represented using metrics from the CA and ML literature.

Feature representation

Table 4 summarizes the input features ϕ used in the ML model that correspond to characteristics of each set of solutions given.

Our feature representation captures characteristics of the set of solutions used, regarding its size (i.e., the number of solutions it includes), and diversity. As diversity metrics, we used the Hamming, L_1, and L_2 distances, as in Section 4. Because the given dataset will not have a fixed diversity among different solutions, we capture the minimum, maximum, mean, and standard deviation of each metric. As our experiments show, all diversity metrics play a significant role on the quality of the extracted set of constraints.

Table 4: Features Used in the Machine Learning Model

Feature Category	Features
General	Number of Solutions
Hamming Distance	Min Hamming Distance
	Max Hamming Distance
	Mean Hamming Distance
	Standard Deviation of Hamming Distance
L_1 **Distance**	Min L_1 Distance
	Max L_1 Distance
	Mean L_1 Distance
	Standard Deviation of L_1 Distance
L_2 **Distance**	Min L_2 Distance
	Max L_2 Distance
	Mean L_2 Distance
	Standard Deviation of L_2 Distance

Target Value

The goal is to estimate the quality of the extracted CP model when a set of solutions is used with a given CA method. Hence, the target value needs to represent the quality of the extracted CP model.

In CA the goal is not only to discriminate solutions and non-solutions, but also to learn a model that is able to generate new solutions to the given problem. Thus, typically, CA models are evaluated not only on how well they capture a given test set of solutions of the target set of constraints C_T, but also on whether the solutions generated from the learned model are correct. Based on the above, and as stated in the literature [14], *Accuracy*, *Recall* and *Precision* can be used as performance measures of the learned models. With recall, we can measure what percentage of the feasible region of the target set of constraints is captured, i.e., what percentage of $sol(C_L)$ is in $sol(C_T)$. On the other hand, with precision, we can measure what percentage of the feasible region of the learned constraints is actually feasible in the target model, i.e., what percentage of $sol(C_T)$ is in $sol(C_L)$. Accuracy captures the combination.

Depending on the CA method used and its invariants, we can use a different metric to capture the quality of the CP model. Some CA methods, typically the ones using solutions during the acquisition process, guarantee (under certain assumptions) that they learn all constraints of the given problem, but may also acquire constraints that they should not. This happens when the number of available solutions is not large enough, or they are not diverse enough. On the other hand, some CA methods need non-solutions to confirm if a constraint is part of the problem or not, before it is added to C_L, and thus can miss some constraints if the non-solutions do not capture some cases. In this paper, the CA method used belongs to the first category and thus cannot have false negatives. Hence, we use the *Recall* metric as a target value in the dataset, to represent the performance of the CA method using the given set of solutions.

To measure the recall, we employ a test set of solutions for each problem, capturing different parts of its feasible region, generated using the target set of constraints C_T. Let $T = \{t_1, t_2, \ldots, t_M\}$ denote the test set of size $|T| = M$, where each t_i is a valid solution to the problem. The recall is measured by testing what percentage of the solutions in T satisfy the learned set of constraints C_L.

$$y = Recall = \frac{|\{t \in T \mid t \in sol(C_L)\}|}{|T|} \tag{2}$$

To create our set of training examples for the ML model, we use the following procedure: for each training example, we start by selecting a set of solutions S generated by one of the methods described in Section 4. From this set S, we extract the input features ϕ required for the model. Using these solutions, we then construct a CP model M. Subsequently, the CA test cases T are employed to calculate the recall y.

5.2 Experimental Setup

The experiments in this section aim to answer the following question: (Q3) Can an ML model accurately predict the recall of the CA system based on features derived from the solution sets?

We use Random Forest (RF) models for our evaluation. The performance of these models is evaluated using the coefficient of determination (R^2). We consider the three benchmark problems from the previous section: Sudoku, Greater Than Sudoku, and the WLP.

For Sudoku, we use a single problem instance, as the structure and constraints remain consistent across different Sudoku puzzles. However, for Greater Than Sudoku, we generate 20 random instances, each containing between 10 and 30 random greater-than constraints. This allows us to evaluate the ML model's performance across a variety of Greater Than Sudoku problem configurations, ensuring the robustness and generalizability of our approach. Similarly, for the WLP, we generate 20 random instances with varying numbers of warehouses and customers, from instances with 5 warehouses and 10 customers to 50 warehouses and 200 customers.

Evaluation Process

The generated solution sets were provided to the CA tool. For each solution dataset, the following evaluation process was conducted:

(1) **Constraint Learning Phase:** The CA tool was executed using each diverse solution set. During this phase, the tool analyzed the solutions to infer the underlying constraints of the given problem.
(2) **Test Set Generation:** For each benchmark problem, we generate separate test sets to evaluate the performance of the CA tool. For Sudoku, as we use a single problem instance, we generate a single test set consisting of 100 solutions. These solutions are distinct from the ones used in the training datasets to ensure an unbiased evaluation of the CA tool's performance on unseen Sudoku solutions. For Greater Than Sudoku, we generate 20 test sets, each corresponding to one of the 20 random problem instances. Each test set contains 100 solutions specific to its respective problem instance. Similarly, for the WLP, we generate 20 test sets each test set consists of 100 solutions, each associated with one of the 20 random problem instances.
(3) **Recall calculation:** Each solution test set was evaluated with the learned set of constraints. Recall was calculated as the ratio of the number of test cases correctly satisfied by the learned model to the total number of test cases.
(4) **Feature Extraction and Logging to Dataset:** Throughout the evaluation process, we logged the features described in Section 5.1 from each solution set used. The extracted features capture essential characteristics of the solution sets, such as their size and diversity metrics. For each solution dataset, we record the number of solutions it contains, as well as statistical measures of the Hamming, L_1, and L_2 distances between solutions, including the minimum, maximum, mean, and standard deviation. These features served as input features for training the ML model to predict the recall of the CA tool.

Machine Learning Model Training

An RF regressor was trained, implemented using the scikit-learn library [17]. 10-fold cross-validation was used for the evaluation, presenting the average R-squared (R^2) scores.

5.3 Results

We now present the results of the RF model with our approach.

Table 5 presents the results for the RF model when applied to predict the recall of the CA system to various solution sets. We

Table 5: ML model results

Dataset	Feature Group	R^2 (%)
Greater Than Sudoku	Hamming	99.89
Greater Than Sudoku	L_1	99.89
Greater Than Sudoku	L_2	99.89
Greater Than Sudoku	All	99.89
Sudoku	Hamming	98.63
Sudoku	L_1	98.13
Sudoku	L_2	98.38
Sudoku	All	98.32
WLP	Hamming	97.61
WLP	L_1	99.67
WLP	L_2	99.67
WLP	All	99.58

have generated four different models for each benchmark problem, utilizing as features the number of solutions and (i) the metrics related to the Hamming distance, (ii) the metrics related to the L_1 distance, (iii) the metrics related to the L_2 distance, and (iv) all the metrics. For Greater Than Sudoku, the RF model achieved an impressive R^2 score of 99.89% across all feature groups (Hamming, L_1, L_2, and All), indicating a strong ability to predict the recall of the CA system based on different diversity metrics of the solution sets. For Sudoku, the RF model's performance was slightly lower but still highly accurate, with R^2 scores ranging from 98.13% to 98.63% across the feature groups. In the case of the WLP, the RF model demonstrated excellent predictive performance, achieving R^2 scores of 99.67% for both L_1 and L_2 features, and a slightly lower score of 97.61% for Hamming features.

These results demonstrate that the machine learning model can accurately predict the CA system's recall based on the features of solution sets, with very high R^2 scores indicating strong predictive power. The strong predictive performance of the RF model across all problem types and feature groups indicates that the diversity metrics capture meaningful information about the quality of the solution sets for constraint acquisition.

The methods and findings from this study can be extended to other CSPs, such as nurse rostering [11]. For effective model building in such applications, a substantial number of solutions are required to train the ML model adequately. In real-world scenarios, where generating a large set of solutions may not be feasible, it is crucial to explore alternative approaches to build robust ML models with limited data.

Therefore, we further experiment with the predictive ability of the ML model if fewer solutions are present in the solution pool. In the previous experiment, we used 40,000 solutions to generate the diverse solution sets for each instance of the dataset. We now compare the model's performance using also smaller subsets of 2,000 and 1,000 solutions in the solution pool, meaning that the diverse sets of solutions generated will have a limited size and diversity. Each subset was evaluated across the 20 WLP instances.

The results presented in Table 6 demonstrate the impact of the number of solutions on the performance of the ML model for predicting the recall of the CA system in the WLP when trained on varying sizes of solution sets. When using the full dataset of 40,000 solutions, the ML model shows excellent performance, with R^2 scores ranging from 97.61% to 99.67% across all feature groups (Hamming, L_1, L_2, and All). However, as the total number of solutions used for training the ML model decreases, the performance of the model declines. With 2,000 solutions, the R^2 scores drop to a range of 71.75% to 82.31%, indicating a reduction in the model's predictive accuracy. This trend continues when the number of solutions is further reduced to 1,000, with R^2 scores ranging from 65.23% to 76.93%. These results show that while having a large and diverse dataset is ideal for training, highly accurate ML models are still able to be developed with acceptable performance using smaller solution sets. The ML model's ability to maintain R^2 scores close to 77% with just 1000 solutions when all features are used. This demonstrates its potential for application in scenarios where having or generating extensive solution sets is not possible.

Table 6: R^2 scores for WLP under Different Solution Counts

	40,000 sols	2,000 sols	1,000 sols
Feature Group	**R^2 (%)**	**R^2 (%)**	**R^2 (%)**
Hamming	97.61	74.64	67.87
L_1	99.67	71.75	65.23
L_2	99.67	77.92	71.72
All	99.58	82.31	76.93

6 Conclusions

In this study, we investigated the impact of solution diversity on passive CA methods across three problems: Sudoku, Greater Than Sudoku, and WLP. Our findings demonstrate that solution diversity plays a crucial role in the performance of passive learning methods in CA. The results show that higher diversity metrics and random selection improve constraint identification. In contrast, smaller solution diversity metrics tend to result in the identification of more constraints than the actual number present in the problem. This indicates that a lack of diversity in the provided examples may hinder the CA system's ability to distinguish between essential and spurious constraints, leading to overfitting and reduced generalization performance. In addition, the results highlight the importance of the size of the solution set in conjunction with its diversity. As the number of solutions in the dataset increases, the CA system's performance generally improves, particularly when the solutions are diverse. A large, varied set of examples is needed for effective learning.

To address the challenge of assessing the quality of a given solution set for CA in real-world scenarios, where generating a large and diverse set of examples may not always be feasible, we proposed the use of an ML model to predict the recall of the constraint acquisition system based on the diversity of the available solutions. High R^2 scores show diversity metrics effectively indicate solution set suitability. This model helps estimate CA system performance based on available examples.

Acknowledgments

The research work was supported by the Hellenic Foundation for Research and Innovation (HFRI) under the 4th Call for HFRI PhD Fellowships (Fellowship Number: 9446).

References

[1] Vasileios Balafas, Dimosthenis C. Tsouros, Nikolaos Ploskas, and Kostas Stergiou. 2024. Enhancing Constraint Acquisition through Hybrid Learning: An Integration of Passive and Active Learning Strategies. *International Journal on Artificial Intelligence Tools* (2024). In press.

[2] Nicolas Beldiceanu, Mats Carlsson, Sophie Demassey, and Thierry Petit. 2007. Global constraint catalogue: Past, present and future. *Constraints* 12 (2007), 21–62.

[3] Nicolas Beldiceanu and Helmut Simonis. 2016. Modelseeker: Extracting global constraint models from positive examples. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 10101 LNCS, 284715 (2016), 77–95. https://doi.org/10.1007/978-3-319-50137-6_4

[4] Christian Bessiere, Remi Coletta, Eugene C. Freuder, and Barry O'Sullivan. 2004. Leveraging the learning power of examples in automated constraint acquisition. In *Principles and Practice of Constraint Programming – CP 2004*, Mark Wallace (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 123–137.

[5] Christian Bessiere, Remi Coletta, Frédéric Koriche, and Barry O'Sullivan. 2005. A SAT-based version space algorithm for acquiring constraint satisfaction problems. In *Machine Learning: ECML 2005: 16th European Conference on Machine Learning, Porto, Portugal, October 3-7, 2005. Proceedings 16*. Springer, Springer, Berlin, Heidelberg, 23–34.

[6] Christian Bessiere, Abderrazak Daoudi, Emmanuel Hebrard, George Katsirelos, Nadjib Lazaar, Younes Mechqrane, Nina Narodytska, Claude-Guy Quimper, and Toby Walsh. 2016. *New approaches to constraint acquisition*. Springer International Publishing, Cham, 51–76. https://doi.org/10.1007/978-3-319-50137-6_3

[7] Christian Bessiere, Frédéric Koriche, Nadjib Lazaar, and Barry O'Sullivan. 2017. Constraint acquisition. *Artificial Intelligence* 244 (2017), 315–342. https://doi.org/10.1016/j.artint.2015.08.001

[8] Michael Bloem and Nicholas Bambos. 2014. Air traffic control area configuration advisories from near-optimal distinct paths. *Journal of Aerospace Information Systems* 11, 11 (2014), 764–784.

[9] Eugene C Freuder. 2018. Progress towards the holy grail. *Constraints* 23, 2 (2018), 158–171.

[10] Eugene C Freuder and Barry O'Sullivan. 2014. Grand challenges for constraint programming. *Constraints* 19 (2014), 150–162.

[11] Celia A. Glass and Roger A. Knight. 2010. The nurse rostering problem: A critical appraisal of the problem structure. *European Journal of Operational Research* 202, 2 (2010), 379–389. https://doi.org/10.1016/j.ejor.2009.05.046

[12] Zhiqiang Gong, Ping Zhong, and Weidong Hu. 2019. Diversity in machine learning. *IEEE Access* 7 (2019), 64323–64350. https://doi.org/10.1109/ACCESS.2019.2917620

[13] Emmanuel Hebrard, Brahim Hnich, Barry O'Sullivan, and Toby Walsh. 2005. Finding diverse and similar solutions in constraint programming. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence*. AAAI, Pittsburgh, PA, USA, 372–377.

[14] Mohit Kumar, Samuel Kolb, and Tias Guns. 2022. Learning constraint programming models from data using generate-and-aggregate. In *28th International Conference on Principles and Practice of Constraint Programming (CP 2022) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 235)*, Christine Solnon (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 29:1–29:16. https://doi.org/10.4230/LIPIcs.CP.2022.29

[15] Arnaud Lallouet, Matthieu Lopez, Lionel Martin, and Christel Vrain. 2010. On learning constraint problems. In *2010 22nd IEEE International Conference on Tools with Artificial Intelligence*, Vol. 1. IEEE, IEEE, Arras, France, 45–52.

[16] Tuan Anh Nguyen, Minh Do, Alfonso Emilio Gerevini, Ivan Serina, Biplav Srivastava, and Subbarao Kambhampati. 2012. Generating diverse plans to handle unknown and partially known user preferences. *Artificial Intelligence* 190 (2012), 1–31.

[17] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[18] Thierry Petit and Andrew C. Trapp. 2015. Finding diverse solutions of high quality to constraint optimization problems. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*. AAAI Press, Buenos Aires, Argentina, 260–266.

[19] Thierry Petit and Andrew C. Trapp. 2019. Enriching solutions to combinatorial problems via solution engineering. *INFORMS Journal on Computing* 31, 3 (jul 2019), 429–444. https://doi.org/10.1287/ijoc.2018.0855

[20] Steven Prestwich. 2021. Unsupervised constraint acquisition. In *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, Washington, DC, USA, 256–262. https://doi.org/10.1109/ICTAI52525.2021.00042

[21] Steven Prestwich. 2021. Unsupervised constraint acquisition. In *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, Washington, DC, USA, 256–262. https://doi.org/10.1109/ICTAI52525.2021.00042

[22] S. D. Prestwich. 2020. Robust constraint acquisition by sequential analysis. In *24th European Conference on Artificial Intelligence (ECAI 2020)* (29 Aug-08 Sept) *(Frontiers in Artificial Intelligence and Applications, Vol. 325)*. IOS Press, Santiago de Compostela, Spain (online), 355–362. https://doi.org/10.3233/FAIA200113

[23] Steven D Prestwich, Eugene C Freuder, Barry O'Sullivan, and David Browne. 2021. Classifier-based constraint acquisition. *Annals of Mathematics and Artificial Intelligence* 89 (2021), 655–674.

[24] Charles Prud'homme, Fages Jean-Guillaume, and Lorca Xavier. 2016. Choco solver documentation. https://choco-solver.org

[25] Jean-Charles Régin. 1994. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the Twelfth AAAI National Conference on Artificial Intelligence* (Seattle, Washington) *(AAAI'94)*. AAAI Press, Seattle, Washington, USA, 362–367.

[26] Francesca Rossi, Peter van Beek, and Toby Walsh. 2008. Constraint programming. In *Handbook of Knowledge Representation*, Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter (Eds.). Foundations of Artificial Intelligence, Vol. 3. Elsevier, Amsterdam, The Netherlands, 181–211. https://doi.org/10.1016/S1574-6526(07)03004-0

[27] Helmut Simonis. 2001. *Building industrial applications with constraint programming*. Springer Berlin Heidelberg, Berlin, Heidelberg, 271–309. https://doi.org/10.1007/3-540-45406-3_6

[28] Daniel Sroka and Tomasz P. Pawlak. 2018. One-class constraint acquisition with local search. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Kyoto, Japan) *(GECCO '18)*. Association for Computing Machinery, New York, NY, USA, 363–370. https://doi.org/10.1145/3205455.3205480

[29] Kostas Stergiou and Toby Walsh. 1999. The difference all-difference makes. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 1* (Stockholm, Sweden) *(IJCAI'99)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 414–419.

[30] Sug, Hyontai. 2018. Performance of machine learning algorithms and diversity in data. *MATEC Web Conf.* 210 (2018), 04019. https://doi.org/10.1051/matecconf/201821004019

[31] Dimosthenis C. Tsouros, Kostas Stergiou, and Panagiotis G. Sarigiannidis. 2018. Efficient methods for constraint acquisition. In *Principles and Practice of Constraint Programming*, John Hooker (Ed.). Springer International Publishing, Cham, 373–388.

[32] Satya Gautam Vadlamudi and Subbarao Kambhampati. 2016. A combinatorial search perspective on diverse solution generation. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016,*, Dale Schuurmans and Michael P. Wellman (Eds.). AAAI Press, Phoenix, Arizona, USA, 776–783. https://doi.org/10.1609/AAAI.V30I1.10079

[33] Willem-Jan van Hoeve. 2001. The alldifferent Constraint: A Survey. *CoRR* cs.PL/0105015 (05 2001).

[34] Mark Wallace. 1996. Practical applications of constraint programming. *Constraints* 1 (1996), 139–168.