Addressing Over-fitting in Passive Constraint Acquisition through Active Learning

Vasileios Balafas[1], Dimos Tsouros[2], Nikolaos Ploskas[1], and Kostas Stergiou[1]

[1] University of Western Macedonia, Campus ZEP, Kozani, 50100, Greece
[2] KU Leuven, Celestijnenlaan 200a, Leuven, 3001, Belgium
v.balafas@uowm.gr, dimos.tsouros@kuleuven.be, nploskas@uowm.gr, kstergiou@uowm.gr

Abstract. Constraint Programming (CP) is a powerful approach to solving complex combinatorial problems. However, formulating combinatorial problems as CP models typically demands substantial expertise. *Constraint acquisition* (CA) seeks to assist in model building by deriving constraints from data. In *passive learning*, the system relies on a pre-labeled set of examples (solutions or non-solutions) to infer constraints, whereas *active learning* engages a domain expert or software system through targeted queries that classify newly proposed assignments to the variables of the problem. Hybrid CA frameworks that combine both strategies have also emerged to leverage the strengths of both approaches. However, when training data are scarce or noisy, passive methods may overfit the observed examples—appearing valid on the training set but failing to generalize to other, unseen solutions—and thereby introduce invalid constraints into the model. To address this issue that has been overlooked, we propose a new *query-driven refinement* approach that systematically challenges suspicious acquired constraints, using "violating assignments" designed to refute them while preserving all other constraints. Focusing on the `AllDifferent` constraint, we integrate this refinement into an existing hybrid CA system and experimentally demonstrate that our approach facilitates convergence to a correct final model.

Keywords: Constraint Programming · Active Learning · Constraint Acquisition · Global Constraints · Over-fitting

1 Introduction

Constraint Programming (CP) is a powerful paradigm for solving a variety of combinatorial optimization problems in domains such as scheduling, timetabling, and configuration [20]. As in other approaches to combinatorial optimization (e.g. MIP, SAT), modeling is of primary importance when solving a new problem [10]. It is well known that an efficient model can result in exponentially shorter run times compared to a bad one. A CP model consists of variables with finite domains and constraints that collectively define feasible assignments to these variables. *Global constraints*, such as the `AllDifferent` [5], which states

that the variables in a given set must take different values, are ubiquitous, because they compactly encode patterns across multiple variables and can significantly speed up solving by exploiting specialized propagators.

Despite the effectiveness of CP, constructing reliable and efficient models often requires specialized domain expertise, and this is perhaps the most significant obstacle to the wider adoption of CP technology. To address this shortcoming, (semi)-automated modeling is attracting ever-increasing interest, with *Constraint Acquisition* (CA) being one of the most promising approaches [13, 3, 26]. CA is an area where combinatorial optimization, and CP in particular, meets Machine Learning (ML), as CA methods make use of inductive learning as well as statistical learning techniques [8, 17, 24].

Broadly, CA methods can be categorized into *passive* and *active*. In *passive learning* (PL), the system acquires constraints from a provided dataset of example solutions and non-solutions, without further interaction [8]. Often, passive CA aims at learning constraints that fit specific patterns [13] or global constraints [4]. Conversely, *active learning* (AL) involves an interactive process: the system proposes specific variable assignments (queries) to an oracle (e.g., a human expert or a software system encoding the ground-truth model) to confirm whether those assignments are valid solutions. The feedback received allows the system to refine its constraints more accurately. However, active CA systems cannot deal with global constraints, and are thus limited to learning constraints over small sets of variables (usually binary ones). Recently, *hybrid* CA frameworks have emerged, aiming to harness the complementary strengths of both approaches [2]. In such a framework, an initial set of global constraints is derived passively from available data, and an active learning phase then completes the model by learning fixed-arity constraints, which are often instance-specific.

However, there is a crucial shortcoming that has not received much attention and therefore remains unresolved: when the training data are limited or noisy, *passive learning can introduce invalid constraints*, i.e., constraints that do not appear in the true model, over-fitting on the specific examples provided to the system. For instance, consider a Sudoku puzzle where, based on the few provided solution examples, a passive learner may incorrectly infer an `AllDifferent` constraint on the main diagonal, if by chance the variables in the diagonal take different values in all given training examples. However, this does not hold for all valid Sudoku solutions, meaning that the model will be over-fitted to the observed data and an invalid constraint will be introduced. The problem of overfitting becomes more severe when only a few solutions are available or when they are not diverse [1], as is usually the case with real-world problems.

In this paper, we propose a *query-driven refinement* methodology, specifically designed to address the problem of over-fitting by identifying and discarding from the model global constraints that were incorrectly acquired by passive learning.

For each constraint acquired through a passive learning system, we first compute a probabilistic confidence score using a Random Forest classifier trained on existing CP models. Then, our method generates *violating assignments*, i.e. assignments that violate a specific constraint while satisfying all others, and queries

the oracle. If a violating assignment is labeled valid, it contradicts the constraint, which is therefore removed. Otherwise, the confidence score of this constraint is strengthened through Bayesian reasoning.

Hence, by maintaining and updating confidence scores, the system can make informed decisions about which constraints to retain or discard.

For the purposes of this study, we demonstrate the applicability and effectiveness of our violation-based approach focusing on the `AllDifferent` constraint, which is the most common global constraint. However, the same methodology can be applied to other global constraints (e.g., `Sum`, `Count`), given suitable strategies to generate violating assignments. Although our method does not theoretically guarantee to remove all over-fitted constraints, an experimental evaluation on benchmarks such as Sudoku variants and Exam Timetabling demonstrates that our refinement step manages to do so, reducing the overall number of queries compared to purely active learning methods, while improving model accuracy compared to purely passive or hybrid learning methods.

2 Related Work

Research in CA can be broadly divided into methods that focus on learning *fixed-arity constraints* and those that learn *global constraints or structural patterns*.

Fixed-arity approaches typically rely on active learning to iteratively query an oracle and validate candidate assignments. A recent family of interactive CA methods is based on the QUACQ [6] algorithms, using (partial) membership queries to prune the search space and acquire constraints. Extensions of this paradigm, such as *MQuAcq-2* [25], and *GROWACQ* [23, 24] leverage structural properties or probabilistic cues to optimize query generation. Although effective, these active learning techniques often require a large number of queries.

Passive CA approaches targeting fixed-arity constraints have also been given some attention in the literature. One of the first passive CA approaches is Conacq.1 [7], which is a SAT-based version space algorithm for acquiring CP models from given training examples. Recently, approaches that are robust to noise have been introduced, integrating ML techniques [18, 17] (e.g., classifiers, unsupervised learning) or statistical methods [16].

Methods targeting global constraints or structural patterns (e.g., *ModelSeeker* [4] and *COUNT-CP* [13]) are typically passive learners that rely on pattern matching or frequent pattern mining to capture recurring global properties from data; however, they too may over-fit in scenarios with sparse or unrepresentative examples. Hybrid CA methods combine the strengths of both paradigms by first using passive learning to capture global patterns and then applying active learning to refine and complete the model. For instance, [2] exploits passive learning for initially learning global constraints using structural patterns and then uses active learning to complete the model with missing fixed-arity constraints. Our work builds on this hybrid methodology by introducing a query-driven refinement phase that actively identifies and removes invalid constraints from the over-fitted CP model before the active learning phase finalizes the model.

3 Background

A *Constraint Satisfaction Problem (CSP)* is a triple (X, D, C), where:

- X is a set of variables,
- D is a set of domains corresponding to the variables, and
- C is a set of constraints restricting the allowable assignments.

A *constraint* c is a pair $(rel(c),\ var(c))$, where $var(c) \subseteq X$ is the *scope* of the constraint, and $rel(c)$ is a relation over the domains of the variables in $var(c)$, restricting their allowed value assignments. The *arity* of the constraint, denoted as $|var(c)|$, indicates the number of variables involved. An assignment $A : X \to \bigcup_{x \in X} D_x$ is a *solution* iff it satisfies all constraints in C. Global constraints capture common substructures across multiple variables. For instance, the `AllDifferent` constraint requires that all variables in a set S take distinct values:

$$\texttt{AllDifferent}(S) = \{A : S \to \bigcup_{x \in S} D_x \mid \forall\, x, y \in S,\, x \neq y :\ A(x) \neq A(y)\},$$

and can be *decomposed* into a semantically equivalent set of binary inequalities:

$$\texttt{AllDifferent}(S) \equiv \{x_i \neq x_j \mid x_i, x_j \in S,\ i < j\}.$$

Many CSPs exhibit inherent structural patterns that can be effectively exploited through *matrix modelling* [9]. In matrix models, decision variables are organized into one or more matrices—much like the grid structure of a Sudoku puzzle—that naturally reflect the problem's structure.

In *Constraint Acquisition (CA)*, the goal is to learn a target CSP (X, D, C^*) where the vocabulary X, D is considered known while the target set of constraints C^* is unknown. Besides the vocabulary, typically CA systems are also given a *language* Γ that includes the possible relations that may hold between variables (e.g. `AllDifferent` or $\leq$). CA methods operate over a bias B of (possibly explicitly stored) candidate constraints, which is constructed using the vocabulary (X, D) and the constraint language Γ. During the acquisition process, a set of learned constraints $C_L \subseteq B$ is acquired. The system has correctly converged iff C_L is equivalent to C^*.

In passive CA, the system is also given a set of positive examples a set $\{A_1, A_2, \ldots, A_m\} \subseteq \text{sol}(X, D, C^*)$, which is denoted as E^+. Possibly, a set of negative examples E^- is also given. The goal of passive learning is to acquire a set of constraints C that is consistent with the data. In our study, we focus exclusively on global constraints, denoted by $C_G = \{\, c_g^1, c_g^2, \ldots, c_g^p\}$, which capture common structural patterns, but may over-fit the training data.

In interactive CA, the candidate constraints in B must have a fixed-arity, as global constraints with unrestricted arity require a *bias* of exponential size. An AL system generates an assignment A^* to all or some of the variables and asks the oracle whether A^* is valid or not. This is known as a *Membership Query*. Based on a Yes (valid) or No (invalid) reply to a query, the system removes constraints from B or adds constraints to C_L.

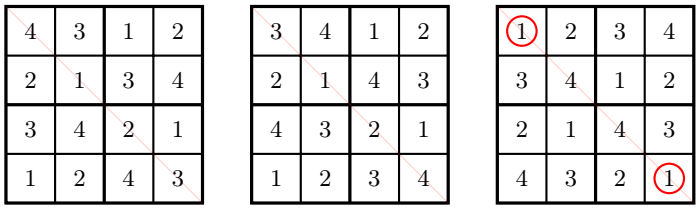

Fig. 1: Example of over-fitting in passive CA.

Example 1. To motivate our work, Figure 1 illustrates an example of over-fitting in passive CA using a 4×4 Sudoku puzzle. In the target model there are 16 variables (one for each cell) and 12 `AllDifferent` constraints (one for each row, column, and 2x2 sub-grid).

Assume that a PL system is used to derive the global constraints of this problem given the language $\Gamma = \{\texttt{AllDifferent}\}$. Such a PL system will create the set of candidate `AllDifferent` constraints in B, based on some predefined patterns, and will use the given solutions to extract the constraints that agree with the given examples E^+. Assume that only the two example solutions from Figure 1 are given. Note that both examples satisfy all 12 `AllDifferent` Sudoku constraints, while their main diagonal (light red shading) also has distinct values ($\{4, 1, 2, 3\}$ in example 1, $\{3, 1, 2, 4\}$ in example 2). Using these two examples, the PL algorithm may over-fit its acquired model to these available examples and erroneously infer an `AllDifferent` constraint on the diagonal if such a pattern is included in the ones it seeks. In the following, we will show how the invalid constraint can be eliminated using a targeted membership query.

4 Methodology

Our approach integrates passive learning with active learning through a novel intermediate phase—Query-Driven Refinement—to eliminate invalid global constraints that were acquired due to over-fitting in the given examples. For the purposes of this study, we focus on the `AllDifferent` constraint.

Figure 2 summarises the data flow among the three phases. From the positive examples E^+ and an initial bias B, the *Passive Learning* step retains only the constraints consistent with the data and extracts a first set of global constraints C_G. These global constraints feed the *Query-Driven Refinement* block, which utilizes the oracle to eliminate any over-fitted constraint, producing C'_G. The refined global constraints—decomposed into binary inequalities—together with the pruned bias bootstrap the final *Active Learning* phase; Finally, the Active Learning phase completes the model by using additional oracle queries to learn any remaining fixed-arity constraints, populating the set C_L. The union $C'_G \cup C_L$ (green box in the figure) is the final model returned by the framework.

The process proceeds in three sequential stages:

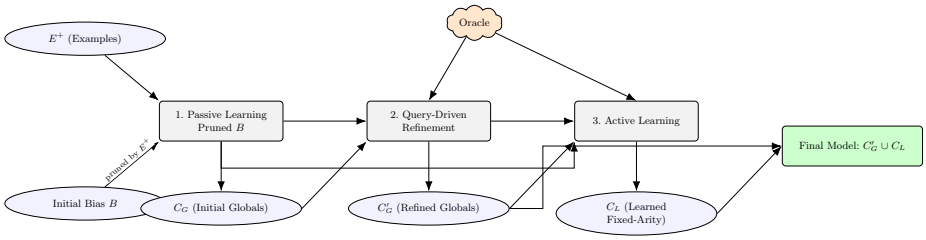

Fig. 2: Overview of the proposed Hybrid Constraint Acquisition framework. E^+: positive examples, C_G: initial global constraints from Passive Learning (PL), B: bias set (pruned during PL), C'_G: refined global constraints after Query-Driven Refinement (QDR), C_L: learned fixed-arity constraints from Active Learning (AL). The Oracle interacts with both QDR and AL phases.

1. **Passive Learning:** Candidate `AllDifferent` constraints are acquired from positive examples E^+ to form the set C_G.
2. **Query-Driven Refinement:** Each constraint in C_G is first assigned a *confidence score*, denoting the probability that the constraint belongs to the target model. This is done using a constraints-level probabilistic classifier, which is trained on existing CP models. Then, each `AllDifferent`(S) in C_G is evaluated via violation queries that assign a common value to a variable pair in S, and it is either refuted or it's confidence probability is updated using Bayesian inference.
3. **Active Learning:** Each `AllDifferent` in the refined set of constraints is decomposed into binary $\neq$ constraints and, together with a bias set B, are passed on to an AL process that finalizes the model.

In the remainder of this section, we first briefly review the Hybrid Passive-Active Learning Framework (stages 1 and 3) and then we detail the novel Query-Driven Refinement phase that we propose in this study.

4.1 Hybrid Passive-Active Learning Framework

In the PL phase, the system is provided with a set E^+ of positive examples. Using E^+, the PL algorithm learns the initial set C_G of `AllDifferent` constraints (see [2] for details). Also, the system constructs a *bias* set B that contains all candidate fixed-arity constraints. For the purposes of this study, B contains all possible binary constraints that can be derived using the basic relations $\{\neq, ==, >, <, \geq, \leq, \}$. This set is refined by removing any candidate constraint that is violated by at least one example in E^+.

In the AL phase, the system refines the constraint network by interactively querying an oracle. First, any `AllDifferent` constraint that remains in the model, after the intermediate refinement step has been applied (as detailed below), is decomposed into a set of binary inequalities. This is necessary for the correct operation of the AL algorithm, which is then initialized with:

- the current learned set C_L, containing the binary inequality constraints derived by the decomposition of the `AllDifferent` constraints, and
- the bias set B of candidate fixed-arity constraints.

The AL component iteratively generates (often partial) assignments and submits them as membership queries to the oracle. If a query is answered *positively* (i.e., the assignment is valid), then any candidate constraints in B that conflict with the query are removed. Conversely, if a query is answered *negatively* (i.e., the assignment is not valid), then the AL algorithm identifies the minimal subset of constraints in B responsible for the violation and adds these to C_L (see [25] for details). Upon termination, the final constraint network is obtained by taking the union of the `AllDifferent` constraints in C_G and the constraints in C_L, excluding the $\neq$ constraints of the `AllDifferent` decompositions.

Although the hybrid framework combines the strengths of both PL and AL, it still carries an important disadvantage: any constraint acquired during the PL phase is by default considered valid, meaning that over-fitted constraints will remain in the final learned model.

4.2 Query-Driven Refinement

To address the over-fitting of passively learned `AllDifferent` constraints, we introduce a Query-Driven Refinement procedure. This method actively tests each learned constraint by generating targeted violation queries and updating our confidence in its validity. Our approach integrates three key components:

1. A probabilistic classification mechanism that assigns prior probabilities to candidate constraints, denoting their likelihood of being over-fitted.
2. A query generation mechanism that constructs violating assignments.
3. A Bayesian update framework that refines our confidence in each constraint.

The overall process, summarized in Algorithm 1, proceeds as follows:

Prior Probability The procedure takes as input the set C_G of `AllDifferent` constraints learned from the passive learning phase of the Hybrid Passive-Active Learning Framework. To assess whether a constraint c actually belongs to the target model or is a result of over-fitting, we assign to it a prior confidence probability $P_{\text{prior}}(c)$ using a Random Forest classifier.

Machine Learning Estimation from Synthetic Data. To train the classifier, synthetic data are generated in a controlled offline process by simulating known CP models using benchmark problems from the CP benchmark library CSPLib [11]. For each model, candidate constraints are extracted from a large set of solution assignments and labeled as follows:

- *Positive examples:* `AllDifferent` constraints that truly belong to the model.
- *Negative examples:* `AllDifferent` constraints that hold only for the generated solutions (i.e., they are over-fitted), created by randomly selecting subsets of variables or perturbing valid groupings.

For each candidate, we extract a feature vector including: *Scope Size* (number of variables); *Is Full Row* and *Is Full Column* (binary indicators for complete rows or columns); *Sliding Window Pattern* (binary indicator if the candidate forms a sliding window grouping over the grid); *Is Diagonal* (binary indicator); and *Average Row* and *Column Positions* (mean indices). These features were chosen based on domain knowledge of common structural patterns in CSPs and were found to effectively capture the properties relevant for distinguishing valid constraints from over-fitted ones.

The classifier outputs a probability $\hat{p}_{\mathrm{ML}}(c)$ that candidate c belongs to the target model, and we set $P_{\mathrm{prior}}(c) = \hat{p}_{\mathrm{ML}}(c)$.

Violation Using Membership Queries The goal of the violation phase is to construct an assignment that deliberately violates a candidate constraint $c = \texttt{AllDifferent}(S)$ while satisfying all other constraints, and to post this as a query to the oracle. For each constraint $c \in C_G$, we enter a loop that is terminated if any of the following occurs:

- The oracle deems that a violating assignment, posted as a query, is valid.
- The confidence probability of c reaches a predefined threshold $\theta_{\max}$.
- It is not possible to generate an assignment that violates c and satisfies all other constraints.

These cases are explained in detail below. For now, note that in the first case, c is removed from the model, whereas in the other two cases it is preserved.

Pairwise Scoring and Query Generation. We first decompose c into binary inequalities over all variable pairs $(x_i, x_j) \in S$. For grid-based problems where spatial positions are known, we rank these pairs using the scoring function:

$$\mathrm{score}(x_i, x_j) = \alpha \cdot |d(x_i, x_j)| - \beta \cdot \big(I(x_i) + I(x_j)\big),$$

- $d(x_i, x_j) = |r(x_i) - r(x_j)| + |c(x_i) - c(x_j)|$ is the Manhattan distance between x_i and x_j, with $r(x)$ and $c(x)$ denoting row and column positions;
- $I(x)$ is the involvement score (i.e., the number of candidate constraints in which x appears);
- α and β are positive weighting parameters that balance the relative contributions of the Manhattan distance and the involvement score.

We then sort the variable pairs in descending order of this score so that pairs with higher scores, indicating greater spatial separation and lower involvement, are prioritized. For each pair, we randomly select a single value from the intersection of the two variables' domains and we assign it to both variables. Obviously, this forces the `AllDifferent` constraint c to be violated, as now two variables in the scope of the constraint have the same value. We then proceed to extend this assignment to all variables in the problem using a CP solver. To do this we remove c from the model and the modified model M' is given to the solver. Crucially, for efficiency, our current implementation selects and tests only one

random value v per pair. This is a heuristic choice trading exhaustiveness for speed; a failure to find a completing assignment A^* with this specific violation (e.g., due to solver timeout or unsatisfiability with v) does not guarantee that the constraint c is valid, only that this specific refutation attempt failed. Testing multiple or all common values is a possible extension but would significantly increase computational cost, potentially impacting the interactivity of the system. The timeout T (line 9) applies *per solver call* when attempting to find a complete assignment A^*. It is necessary to keep the overall runtime practical, as constraint solving is NP-hard. If the solver times out, it is treated as a failure to find a violating assignment for that specific attempt, but it does not prove impossibility.

Generating and Evaluating a Violating Assignment. If the modified model M' is indeed solved and a complete assignment A^* is obtained, this assignment is then submitted as a membership query (ASK(A^*)) to the oracle. Two outcomes are possible:

- If the oracle accepts A^* as a valid solution, this contradicts c; so we immediately remove c from the learned model.
- If the oracle rejects A^*, meaning that it is more likely that c is indeed violated by A^*, we update the probability of c via a Bayesian reasoning, as follows.

Probability Updates and Constraint Removal Let D denote the event that the oracle rejects the violating assignment. Given the candidate's current probability $P_{\text{prior}}(c)$, and assuming:

$$P(D \mid c \in C^*) = 0.95, \quad P(D \mid c \notin C^*) = 0.05,$$

we update the probability using Bayes' rule:

$$P_{\text{post}}(c \in C^* \mid D) = \frac{P(D \mid c \in C^*)\, P_{\text{prior}}(c)}{P(D \mid c \in C^*)\, P_{\text{prior}}(c) + P(D \mid c \notin C^*)\,(1 - P_{\text{prior}}(c))}.$$

If a violation query results in a "Yes" (i.e., A^* is accepted by the oracle), then the candidate c is immediately refuted (i.e., $P(c)$ is set to 0) and removed from C'_G. Otherwise, additional queries are generated for c until the updated probability exceeds the threshold $\theta_{\max}$ (in which case c is retained). To assess the robustness of our approach to these parameter choices, we performed a sensitivity analysis. The analysis shows that varying $P(D \mid c \in C^*)$ within the range $[0.70, 0.98]$ and $P(D \mid c \notin C^*)$ within $[0.02, 0.30]$ has a negligible impact on the final model accuracy and the number of queries. This validates that our parameter settings yield stable results under moderate variations.

Returning to Example 1, recall that the passive learning phase may erroneously infer an `AllDifferent` constraint on the main diagonal. To refute this invalid constraint, our method will generate a violation query, shown in the third grid of Example 1, where we deliberately force a violation by assigning the same value to two cells along the main diagonal. This assignment, which violates the

inferred diagonal `AllDifferent` constraint while satisfying all other constraints, is then submitted to the oracle. The oracle will accept this assignment as valid under the target model, and as a result the invalid constraint will be removed from the model.

Algorithm 1 refines the passively learned set of candidate `AllDifferent` constraints, C_G. First, for each constraint c (line 1), the algorithm collects all variable pairs from the scope of c (line 2) in a set VarPairs, sorts them by a scoring function (line 3), and initializes a boolean flag `violatingFound` to `false` (line 4). Next, it iterates over each pair (x_i, x_j) (line 5) and checks whether the two variables share any common value in their domains (line 6). If so, a random value v is selected (line 7), and a modified model M' is solved (line 8). If a solution (assignment A^*) is found (line 10), `violatingFound` is set to `true` (line 11), and the algorithm breaks out of the inner loop (line 12).

The timeout T is necessary to keep the overall run time manageable. Let us not forget that in order to find an assignment A^* in line 9, we need to solve the CSP that corresponds to M', which is an NP-hard problem. If no solution is found within T, the algorithm proceeds to the next pair of variables in VarPairs. Alternatively, we could try other values that are common to the two variables under examination. If no violating assignment is found after examining all pairs, the algorithm assumes that c is valid and proceeds to the next constraint (line 13). Otherwise, once a violating assignment A^* has been found, it is submitted to the oracle (line 14). If the oracle replies "Yes" (line 15), c is removed from C_G (line 16), since A^* contradicts c. Otherwise, c's prior probability is updated via Bayes' rule (line 19), and if it exceeds the threshold $\theta_{\max}$, the algorithm accepts c without further queries. After all constraints have been processed, the refined set C_G is returned (line 21).

If the inner loop of Algorithm 1 (lines 5-12) completes without finding any violating assignment A^* for constraint c (indicated by 'violatingFound' remaining 'false'), the algorithm conservatively retains c in C_G (line 13's implicit break). This occurs if no pairs had common domain values, or if for all tested pairs and single random values v, the solver timed out or proved unsatisfiability for M'. Importantly, this failure to refute c via our heuristic search does *not* prove that c is valid or redundant with respect to the other constraints in C_G. It merely signifies that this specific, time-limited search strategy could not find evidence to discard c. Removing c in this situation would risk discarding a potentially valid constraint essential to the target model. Determining true redundancy would require a different, potentially more expensive, analysis.

5 Experiments

This section presents our experimental evaluation, including the implementation details, the generation of synthetic data for training the Random Forest classifier, the benchmark problems used, the metrics for comparison with baseline methods, and the experimental results obtained.

Algorithm 1 Query-Driven Refinement

Require: C_G: candidate `AllDifferent` constraints with prior probabilities $P_{\text{prior}}(c)$; confidence threshold $\theta_{\max}$; functions score(x_i, x_j) for pairwise scoring; timeout T for the solver

for all $c \in C_G$ **do**
  Let VarPairs $\leftarrow \{(x_i, x_j) \mid x_i, x_j \in \text{scope}(c),\ i < j\}$.
  Sort VarPairs in descending order of score(x_i, x_j).
  violatingFound $\leftarrow$ false.
  for all $(x_i, x_j) \in$ VarPairs **do**
    if $\text{dom}(x_i) \cap \text{dom}(x_j) \neq \emptyset$ **then**
      Let v be a random value chosen from $\text{dom}(x_i) \cap \text{dom}(x_j)$
      $C'_G \leftarrow \big(C_G \setminus \{c\}\big) \cup \{\, x_i = v,\ x_j = v \,\}$, and let $M' = (X, D, C'_G)$.
      Solve M' with timeout T to obtain assignment A^*
      if A^* is found **then**
        violatingFound $\leftarrow$ true
        break
  if not violatingFound **then continue** ▷ No violation found; accept c
  feedback $\leftarrow$ ASK(A^*);
  if *feedback* = "Yes" **then**
    Remove c from C_G.
    continue ▷ c is refuted by the oracle.
  else
    Update $P_{\text{prior}}(c)$ via Bayes' rule.
    if $P_{\text{prior}}(c) > \theta_{\max}$ **then continue** ▷ c is accepted with high confidence.
return C_G.

The PL component is implemented following the hybrid system described in [2], which uses the Choco CP solver [19] to test if certain sets of variables satisfy the `AllDifferent` constraint in the given examples. In the violation-based AL phase, we use the CPMPy modeling library [12] for constructing and manipulating CP models and Google OR-Tools [15] to generate queries (feasible assignments A^*). The final AL phase is implemented using PyConA [22], a Python library for interactive CA, which includes an implementation of MQuAcq-2. To load our benchmark problems for evaluation, we employ the standardized data format specified by the PTHG21 CA Challenge [21]. This format defines the structure of solutions—typically as matrices for grid-based problems—ensuring that the examples are encoded in a consistent and reproducible manner. For the solving of modified models M' in the violation phase, we set the timeout T to 5 seconds. In our scoring function, we set the parameters to $\alpha = 1.0$ and $\beta = 0.5$. These values were determined through preliminary tuning to balance the contributions of the Manhattan distance and the involvement score.

5.1 Synthetic Data Generation for Classifier Training

To estimate the validity of candidate global constraints, we train a Random Forest classifier on synthetic data generated from several CSPLib benchmarks [11]

that are modeled using `AllDifferent` constraints, including Quasigroup Existence (Latin Square), All-Interval Series, Magic Squares, Langford's Number Problem, N-Queens, Quasigroup Completion, Costas Arrays, n-Queens Completion, and Blocked n-Queens Problem.

For each benchmark, we generate a large set of solution assignments using the corresponding CP model and extract candidate `AllDifferent` constraints. Constraints inherent to the target model are labeled as *positive examples*, while those that hold only for the provided solutions (i.e., over-fitted constraints) are labeled as *negative examples*. A feature vector is constructed for each candidate using the attributes described in subsection 4.2. From the synthetic CSP instances, we extracted approximately 500 candidate constraints in total, with around 200 positive examples (true constraints from the target model) and 300 negative examples (over-fitted constraints). The dataset was split into an 80/20 training/test partition and subjected to 10-fold cross-validation. Using Scikit-learn [14], our Random Forest classifier achieved an average accuracy of 92%. These results indicate that the classifier reliably discriminates between valid and over-fitted candidate constraints on the synthetic dataset.

5.2 Benchmark Problems for Constraint Acquisition

We evaluate our method on puzzle and realistic benchmark problems. Specifically, 9×9 Sudoku (S9x9), Greater-Than Sudoku (GTS), JSudoku (JSud), and two instances of Exam Timetabling (ET1 and ET2). Standard Sudoku features a well-known model with 27 `AllDifferent` constraints—one for each row, column, and sub-grid—while Greater-Than Sudoku and JSudoku incorporate additional binary *greater-than* ($>$) constraints. In the Exam Timetabling problem, `AllDifferent` constraints are critical for ensuring that exams with overlapping student enrollments are scheduled in distinct time slots or rooms. Our method is compared against two baselines:

1. MQuAcq-2, which is a purely active learning method [25].
2. Our earlier hybrid CA framework, which employs PL plus AL, without the query-driven refinement [2].

5.3 Results and Discussion

Table 1 summarizes our experimental results by reporting the following metrics: the number of given solutions (Sols), the number of candidate constraints initially learned by passive learning (StartC), the number of constraints invalidated by our query-driven refinement (InvC), and the number of `AllDifferent` constraints in the target model (C_T). In addition, we give the size of the generated bias (Bias), the number of violation queries for our method's refinement phase (ViolQ), active learning queries for our method's AL phase (MQuQ), and the total queries for our proposed method (TQ = ViolQ + MQuQ). For comparison with baselines, we report the total queries used by the purely Active Learning baseline (MQuAcq-2), denoted as **ALQ**, and the total queries used

by the baseline Passive+Active hybrid approach (which lacks the query-driven refinement phase), denoted as **PAQ**. Timing metrics include durations for our method's violation phase (VT(s)), active learning phase (MQuT(s)), and overall runtime (TT(s)), as well as total runtimes for the purely Active Learning baseline, **ALT(s)**, and the Passive+Active baseline, **PAT(s)**.

Table 1: Experimental results comparing our proposed method against baseline Pure Active Learning and baseline Passive+Active.

Prob.	Sols	StartC	InvC	C_T	Bias	ViolQ	MQuQ	TQ	ALQ	PAQ	VT(s)	MQuT(s)	TT(s)	ALT(s)	PAT(s)
S9x9	2	47	20	**27**	4287	141	255	396	6672	36	50.50	47.19	97.69	634.19	67.83
S9x9	5	35	8	**27**	1814	105	199	304		22	32.81	29.19	61.99		39.18
S9x9	10	33	6	**27**	1400	99	181	280		20	27.83	25.07	52.89		31.37
S9x9	50	30	3	**27**	813	90	107	197		17	25.74	19.02	44.77		23.72
GTS	2	44	17	**27**	5498	132	279	411	6612	129	55.97	137.21	193.17	651.3	32.8
GTS	5	35	8	**27**	3036	105	272	377		263	56.42	127.15	183.58		13.59
GTS	20	32	5	**27**	1642	96	243	339		292	48.85	132.66	181.51		117.75
GTS	200	28	1	**27**	666	84	150	234		250	41.43	53.52	94.94		97.55
JSud	2	68	41	**27**	4952	449	3280	3729	6844	746	30.13	236.29	276.68	653.82	308.74
JSud	20	68	41	**27**	4559	449	3054	3503		938	30.04	227.50	258.44		529.89
JSud	200	59	32	**27**	4514	409	2809	3218		951	31.98	249.58	282.28		356.19
JSud	500	52	25	**27**	2373	156	1218	1374		669	27.56	263.66	291.69		257.32
ET1	2	41	25	**16**	1661	89	921	1110	7114	1025	68.14	842.12	910.42	1844.41	1587.62
ET1	5	36	20	**16**	1021	67	883	950		678	62.38	654.89	717.64		984.56
ET1	10	27	11	**16**	874	48	853	901		457	56.42	159.45	215.32		308.72
ET1	50	21	5	**16**	562	35	451	486		312	51.08	112.24	163.09		287.33
ET2	2	64	44	**20**	2864	147	2592	2739	9483	2121	84.32	974.68	1059.24	2286.09	1723.57
ET2	5	48	28	**20**	2156	92	1968	2060		1487	76.42	714.64	791.15		1461.09
ET2	10	29	9	**20**	1687	67	1452	1519		768	69.98	185.22	255.43		897.44
ET2	50	24	4	**20**	952	49	869	918		687	51.43	167.6	219.25		463.61

In our experiments, we observed that each invalid `AllDifferent` constraint was refuted with a single violation query. For constraints that were retained in the model, the maximum number of violation queries executed was 3. This low query count is not very surprising because: (i) an `AllDifferent` constraint can be refuted by simply forcing any pair of variables within its scope to take the same value, and (ii) the Bayesian update mechanism was designed to aggressively increase the confidence probabilities, thereby avoiding unnecessary queries on valid constraints.

In the standard 9×9 Sudoku experiments, when only 2 training solutions are provided, the passive phase initially infers 47 global constraints. Our query-driven refinement then invalidates 20 of these, yielding the correct final model of 27 `AllDifferent` constraints (C_T). With an increasing number of training solutions (5, 10, and 50), the initial candidate count decreases to 35, 33, and 30 respectively, while the number of invalidated constraints correspondingly decreases to 8, 6, and 3, yet the final model consistently contains the target 27 constraints. Moreover, with more training solutions the bias size is reduced from 4287 to 813 (by eliminating fixed-arity constraints that are inconsistent with the provided examples), and the total number of violation queries declines from 141

to 90. Consequently, the overall query count is reduced from 396 to 197, and the runtime decreases from 97.69 seconds to 44.77 seconds.

When compared to the purely active learning approach (i.e. MQuAcq-2), our hybrid CA method demonstrates significant efficiency gains, both in query burden and cpu time, as MQuAcq-2 requires 6672 queries and 634.19 seconds to learn the model of 9x9 Sudoku. The baseline hybrid CA approach yields low query counts (PAQ ranges from 36 down to 17) and shorter passive phase runtimes (PAT decreases from 67.83 seconds to 23.72 seconds). However, without the refinement mechanism, this baseline fails to eliminate over-fitted constraints, thereby compromising the correctness of the final model.

In the Greater-Than Sudoku (GTS) experiments, with 2 training solutions the passive phase infers 44 candidates and 17 are then invalidated, producing the target model of 27 constraints. With 5, 20, and 200 training solutions, the candidate counts decrease to 35, 32, and 28, with invalidations of 8, 5, and 1, respectively. Correspondingly, the bias size reduces from 5498 to 666, violation queries decline from 132 to 84, overall queries drop from 411 to 234, and the runtime decreases from 193.17 to 94.94 seconds. The purely active learning approach requires 6612 queries and 651.3 seconds, highlighting significant efficiency gains for our method. The baseline hybrid CA uses fewer queries (129 vs. our 411 for 2 training solutions) but fails to eliminate over-fitted constraints.

In JSudoku, with 2 training solutions the passive phase infers 68 candidates and 41 are then invalidated to yield 27 constraints. As training solutions increase to 20, 200, and 500, learned constraint counts become 68, 59, and 52, and invalidations reduce to 41, 32, and 25, respectively. The bias size falls from 4952 to 2373 and overall queries decline from 3729 to 1374, while runtime remains around 277–292 seconds. Although the baseline hybrid CA gives fewer queries and lower runtimes, our query-driven refinement obtains the correct model, with the number of queries ranging from 746 with 2 initial solutions to 669 with 500 solutions. In contrast, the purely active approach requires 6844 queries and 653.82 seconds.

In the Exam Timetabling instances (ET1 and ET2), with only 2 solutions provided, the passive phase infers 41 and 64 candidate constraints for ET1 and ET2, respectively. Our query-driven refinement then invalidates 25 constraints in ET1 and 44 in ET2, yielding the correct final models of 16 and 20 `AllDifferent` constraints. As more training solutions become available, the number of learned `AllDifferent` constraints decreases, bias sizes shrink (from 1661 to 562 for ET1 and from 2864 to 952 for ET2), and runtimes are substantially reduced (from 910.42 to 163.09 seconds for ET1 and from 1059.24 to 219.25 seconds for ET2).

Although the baseline hybrid CA (PAQ) often requires fewer queries, it does not remove over-fitted constraints and thus may produce an incorrect final model. For instance, in ET1 with 2 solutions, PAQ uses 1025 queries—slightly fewer than our 1110 queries—yet fails to eliminate the spurious constraints. A similar pattern arises in ET2, where PAQ again omits the refinement step and consequently runs fewer queries (2121 vs. our 2739 for 2 training solutions, or 687 vs. our 918 for 50 training solutions) but retains invalid constraints. In contrast, a purely active approach requires 7114 (ET1) and 9483 (ET2) queries, with active learn-

ing times of 1844.41 and 2286.09 seconds, respectively, making our query-driven refinement a far more efficient alternative.

Overall, the experimental results demonstrate that through the query-driven violation mechanism, we manage to obtain the correct target `AllDifferent` constraints in the learned model. And importantly, this is achieved while significantly reducing the number of queries and overall runtime compared to the purely active learning method.

6 Conclusion and Future Work

Constraint Acquisition is an area where combinatorial optimization, and in particular CP, meets ML. Despite the numerous recent developments in CA, a shortcoming of existing learning methods that has not been addressed yet, is that of over-fitting the learned constraint model to the available example solutions.

In this paper, we tackle the problem of over-fitting, focusing on learning models with the common `AllDifferent` constraint, by integrating a query-driven refinement mechanism into a hybrid CA framework. Our approach employs a Random Forest classifier to assign prior probabilities to learned constraints and generates targeted violation queries to eliminate constraints that have been wrongly acquired because of over-fitting. Experimental results demonstrate that our method converges to the correct target model while significantly reducing the number of queries and overall runtime compared to purely active learning.

It is important to note a limitation of the current refinement strategy: it focuses on refuting individual candidate constraints $c \in C_G$ by finding an assignment A^* that violates only c while satisfying $C_G \setminus \{c\}$. This approach may not detect more complex over-fitting scenarios, such as when multiple incorrect constraints are symmetrically involved, or where refuting one incorrect constraint requires violating another (potentially also incorrect) constraint simultaneously. Handling such coupled errors, potentially by exploring violations of small subsets of constraints, remains an open challenge and an avenue for future research.

In the future, we plan to extend our query-driven refinement methodology to other global constraint types, such as `Sum` and `Count`. This is a non-trivial extension that will require developing specialized violation generation strategies tailored to the semantics of each constraint. For instance, to refute a candidate $\texttt{Sum}(S, \text{target})$ constraint, one might try to find an assignment where the sum of variables in S slightly deviates from target while satisfying all other model constraints. Similarly, for a `Count` constraint, the strategy would involve finding an assignment that marginally violates the specified count. Furthermore, a promising direction is to not only attempt to violate the constraint on the full set S, but also to explore violations on its critical subsets. Identifying the smallest subset of variables within S whose values cause the violation could provide more precise feedback for refinement or lead to the discovery of related, more accurate constraints. The pairwise scoring heuristic used for `AllDifferent` might also need adaptation or replacement with more general or constraint-specific heuristics (perhaps incorporating subset exploration ideas) to guide the search

for violating assignments effectively for these and other global constraints. We also intend to explore alternative ML techniques for prior probability estimation, potentially incorporating features specific to different constraint types.

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. Balafas, V., Tsouros, D., Ploskas, N., Stergiou, K.: The impact of solution diversity on passive constraint acquisition. In: Proceedings of the 13th Hellenic Conference on Artificial Intelligence. SETN '24, Association for Computing Machinery (2024)
2. Balafas, V., Tsouros, D.C., Ploskas, N., Stergiou, K.: Enhancing constraint acquisition through hybrid learning: An integration of passive and active learning strategies. International Journal on Artificial Intelligence Tools (2024)
3. Barral, H., Gaha, M., Dems, A., Côté, A., Nguewouo, F., Cappart, Q.: Acquiring constraints for a non-linear transmission maintenance scheduling problem. In: CPAIOR 2024. LNCS, vol. 14742, pp. 34–50. Springer (2024)
4. Beldiceanu, N., Simonis, H.: Modelseeker: Extracting global constraint models from positive examples. In: Data Mining and Constraint Programming, Lecture Notes in Computer Science vol. 10101, Springer. pp. 77–95 (2016)
5. Beldiceanu, N., Carlsson, M., Demassey, S., Petit, T.: Global constraint catalogue: Past, present and future. Constraints **12**, 21–62 (2007)
6. Bessiere, C., Coletta, R., Hébrard, E., Katsirelos, G., Lazaar, N., Narodytska, N., Quimper, C.G., Walsh, T.: Constraint Acquisition via Partial Queries. In: IJCAI: International Joint Conference on Artificial Intelligence. pp. 475–481 (Aug 2013)
7. Bessiere, C., Coletta, R., Koriche, F., O'Sullivan, B.: A sat-based version space algorithm for acquiring constraint satisfaction problems. In: ECML 2005. pp. 23–34. Springer, Springer, Berlin, Heidelberg (2005)
8. Bessiere, C., Koriche, F., Lazaar, N., O'Sullivan, B.: Constraint acquisition. Artificial Intelligence **244**, 315–342 (2017)
9. Flener, P., Frisch, A., Hnich, B., Kiziltan, Z., Miguel, I., Walsh, T.: Matrix modelling. In: Proc. of the CP-01 Workshop on Modelling and Problem Formulation. p. 223 (2001)
10. Freuder, E.C., O'Sullivan, B.: Grand challenges for constraint programming. Constraints **19**, 150–162 (2014)
11. Gent, I.P., Walsh, T.: CSPLIB: A Benchmark Library for Constraints (1999)
12. Guns, T.: Increasing modeling language convenience with a universal n-dimensional array, cppy as python-embedded example. In: Proceedings of the 18th workshop on Constraint Modelling and Reformulation at CP (Modref 2019). vol. 19 (2019)
13. Kumar, M., Kolb, S., Guns, T.: Learning constraint programming models from data using generate-and-aggregate. In: 28th International Conference on Principles and Practice of Constraint Programming (CP 2022). LIPIcs, vol. 235, pp. 29:1–29:16
14. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)

15. Perron, L., Didier, F.: Cp-sat, https://developers.google.com/optimization/cp/cp_solver/
16. Prestwich, S.D.: Robust constraint acquisition by sequential analysis. In: 24th European Conference on Artificial Intelligence (ECAI 2020). Frontiers in Artificial Intelligence and Applications, vol. 325, pp. 355–362. IOS Press (2020)
17. Prestwich, S.: Unsupervised constraint acquisition. In: 2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI). pp. 256–262 (2021)
18. Prestwich, S.D., Freuder, E.C., O'Sullivan, B., Browne, D.: Classifier-based constraint acquisition. Annals of Mathematics and Artificial Intelligence **89**, 655–674 (2021)
19. Prud'homme, C., Jean-Guillaume, F., Xavier, L.: Choco solver documentation (2016), https://choco-solver.org
20. Rossi, F., Van Beek, P., Walsh, T.: Handbook of constraint programming. Elsevier (2006)
21. Simonis, H., Freuder, E.: PTHG21 Challenge (aug 2021). https://doi.org/10.5281/ZENODO.5155465
22. Tsouros, D., Guns, T.: A cpmpy-based python library for constraint acquisition - pycona. In: Proc. AAAI 2025 Bridge on Constraint Programming and Machine Learning (CPML) (2025)
23. Tsouros, D.C., Berden, S., Guns, T.: Guided Bottom-Up Interactive Constraint Acquisition. In: 29th International Conference on Principles and Practice of Constraint Programming (CP 2023). vol. 280, pp. 36:1–36:20 (2023)
24. Tsouros, D.C., Berden, S., Guns, T.: Learning to learn in interactive constraint acquisition. In: Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, February 20-27, 2024, Vancouver, Canada. pp. 8154–8162. AAAI Press (2024)
25. Tsouros, D.C., Stergiou, K., Bessiere, C.: Structure-driven multiple constraint acquisition. In: Principles and Practice of Constraint Programming. pp. 709–725. Springer International Publishing (2019)
26. Tsouros, D.C., Stergiou, K., Sarigiannidis, P.G.: Efficient methods for constraint acquisition. In: Principles and Practice of Constraint Programming (CP 2018). pp. 373–388. Springer International Publishing (2018)