Learning Preferences over Unsatisfiable Subsets

Emilio Gamba
Vrije Universiteit Brussel, Belgium
KU Leuven, Department of Computer Science, Belgium

Marco Foschini
KU Leuven, Department of Computer Science, Belgium

Jayanta Mandi
KU Leuven, Department of Computer Science, Belgium

Bart Bogaerts
Vrije Universiteit Brussel, Belgium

Tias Guns
KU Leuven, Department of Computer Science, Belgium

Abstract

Explanation techniques for constraint satisfaction and optimisation problems often rely on the computation of a minimal unsatisfiable subset (MUS). A MUS, an explanation of an unsatisfiable problem, can also be used to explain why the problem entails a variable's value assignment. Many MUSs typically exist, hence identifying the preferred one for an user is of research interest. Although techniques for computing cost-optimal weighted MUSs can be employed to find the most preferred explanation, it falls upon the user to specify these weights. We investigate whether we can learn preference scoring functions over MUSs from a dataset of pairwise comparison between MUSs using learning-to-rank techniques. We showcase on sudoku that we can learn functions that score well on choosing between two smallest MUSs. Moreover, by using an iterative learning setup, we can directly learn a linear scoring function that can be integrated to generate preferred explanations directly.

2012 ACM Subject Classification Computing methodologies → Discrete space search; Computing methodologies → Supervised learning by classification; Theory of computation → Constraint and logic programming

Keywords and phrases Explainable AI, CSP, MUS

Funding This research received funding by the Flemish Government (AI Research Program), the Research Foundation - Flanders (FWO) project G070521N; the European Research Council (ERC) under the EU Horizon 2020 research and innovation programme (Grant No 101002802, CHAT-Opt).

1 Introduction

The field of Explainable Artificial Intelligence (XAI) aims to build trust by providing intelligent systems with explainable agency. In this paper, we are particularly interested in explainable AI in the context of Constraint Programming (CP) [8]. A prominent problem in *explainable constraint solving* is explaining why a model is unsatisfiable [15, 2, 12, 13, 6]. For instance, consider a nurse rostering problem, which involves creating duty rosters for nursing staff over a specified planning period while adhering to multiple soft and hard constraints, such as the maximum number of working hours. If certain hard constraints cannot be met, it is crucial to provide a clear explanation.

Many methods tackle this problem by providing a deductive explanation in the form of a *minimal unsatisfiable subset* (MUS), which is a subset of the model's constraints that renders the problem unsatisfiable, and thus yielding an inconsistency. Such subsets can also be used to explain why a value is assigned to a specific variable by demonstrating a minimal subset of constraints that entail that value assignment [3]. For example, in the context of the nurse rostering problem, if a nurse is assigned late working hours, the MUS can justify this assignment by identifying the constraints that led to this decision.

Typically, multiple MUSs exist because an unsatisfiable problem can have more than one inconsistency. That is why the identification of the MUS that is most comprehensible to a user has received attention from researchers. [10] proposes the cardinality of the MUS as a potential measure of explanation quality and, as such, compute the cardinality-smallest MUS (SMUS). This concept was later expanded in [3], where a *hand-crafted* linear objective function is introduced to quantify human understandability. Such a function requires identifying features like the number of considered constraints and the number of pre-assigned values. Given this function, they *heuristically* optimize it, aiming to find a MUS that could potentially be more understandable compared to an SMUS. This approach is further extended in [6], by proposing cost-optimal weighted MUS algorithms that find an *optimal* MUS for such weighted linear objective function, for a hand-crafted set of weights.

It can be challenging for human users to explicitly specify weights that consistently lead to good solutions in a specific problem domain [18]. We aim to infer such weights indirectly from user feedback. Specifically, we will use machine learning (ML) techniques for learning such a function. We hence investigate whether we can learn a preference scoring function over MUSs from labelled data, more specifically on a dataset of pairwise comparisons between MUSs in a specific problem domain, where a user chose one of the explanations over the other. The final goal is to embed the scoring function in cost-optimal weighted MUS algorithms, allowing to return explanations in line with the preferred explanations in the dataset, and more generally in line with the implicit preferences of the user.

We will evaluate the learning approach on the pedagogical domain of sudoku puzzles, as the constraint sets of the MUSs are intuitive to visualize over the sudoku grid. This makes it a problem domain in which many users have the required domain knowledge to interpret the explanations, and for which labelling pairs of explanations can be done quickly.

Our **contributions** are as follows:

1. We identify domain-agnostic as well as domain-specific features for MUSs by counting over different subsets of constraints how many are present in the MUS.
2. We compare feature choices and use different point-wise and pair-wise learning-to-rank methods, showing that in the sudoku domain, domain-specific features are essential to accurately discriminate between MUSs, and multiple learning techniques can do this well.
3. We demonstrate that we can not only learn how to *discriminate* between two smallest MUSs, but that a few iterations of iterative learning with linear predictors allows learning a scoring function that can be directly used to *generate* good MUSs.

2 Preliminaries

We now define the mathematical concepts needed for understanding the addressed problem.

2.1 Constraint Satisfaction Problem

▶ **Definition 1.** *A constraint satisfaction problem (CSP) [21] is a triple* $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ *where:* $\mathcal{X}$ *is a set of* variables, $\mathcal{D}$ *is a set of* domains D_x *of allowed* values *for each variable* $x \in \mathcal{X}$, $\mathcal{C}$ *is a set of* constraints *over a subset of the variables.*

A *constraint* is typically described by a formula (e.g., $x + y \neq 1$), and restricts the values that can be assigned to the variables. A *(partial) assignment* $\mathcal{I}$ is a (partial) mapping of variables to values from their domain (also called *facts*). An assignment *satisfies* (or *falsifies*) a constraint if the constraint evaluates to true (or false). A *solution* is an assignment that satisfies all constraints. A set $M \subseteq \mathcal{C}$ is called *satisfiable* if there exists an assignment that satisfies all constraints in M; otherwise it is *unsatisfiable*.

▶ **Example 2** (Sudoku)**.** The 9×9 Sudoku problem is a logic puzzle that requires filling in a set of digits 1 to 9 in a rectangular grid, such that each column, each row and each 3×3 block contains exactly one occurrence of each digit.

The 9×9 Sudoku problem can be modelled as a CSP with a 9×9 array of variables `cells`, each over domain $\{1..9\}$ subject to an `alldifferent` constraint for each row, each column and each 3×3 block in the puzzle.

2.2 Minimal Unsatsfiable Subsets as explanations of unsatisfiability

The most well-known type of explanations in CSPs are *Minimal Unsatisfiable Subsets* (MUS). They explain why a CSP is unsatisfiable by identifying a set of conflicting constraints.

▶ **Definition 3.** *A subset $M \subseteq \mathcal{C}$ is a Minimal Unsatisfiable Subset if M is unsatisfiable and all strict subsets of M are satisfiable.*

Many algorithms exist for finding or enumerating MUSs [15, 17, 13, 14]. We will use [6] which allows computing an *Optimal Unsatisfiable Subset* (OUS) given a linear objective function over the constraints that will be minimized.

▶ **Definition 4.** *Let f be a function assigning a score to every set of constraints $M \subseteq \mathcal{C}$. An* Optimal Unsatisfiable Subset *(OUS) is an unsatisfiable set $M \subseteq \mathcal{C}$ such that all subsets $M' \subseteq \mathcal{C}$ with $f(M') < f(M)$ are satisfiable. The function f is* linear *if it maps M to $\sum_{c \in M} w_c$ for fixed weights $w_c \in \mathbb{R}$.*

For a linear objective function in which all weights are equal the OUS corresponds to the smallest MUS.

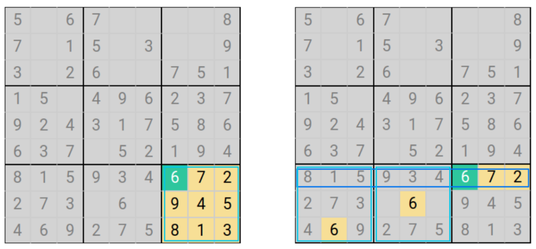
2.2.1 Step-wise explanations using minimal unsatisfiable subsets

While MUSs are used to explain when a problem is unsatisfiable, they can also be used to find a minimal subset of constraints that entails a value assignment, a fact [2]. Two examples of such an explanation step are shown in Fig. 1; they explain the green cell, with used facts highlighted in yellow and used constrains blue boxes.

▶ **Definition 5.** *Let $\mathcal{I}$ be a partial assignment of a CSP $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, an* explanation step *is a triple $(\mathcal{E}, \mathcal{S}, \mathcal{N})$, also denoted $\mathcal{E} \wedge \mathcal{S} \Rightarrow \mathcal{N}$, such that:*

- *$\mathcal{E} \subseteq \mathcal{I}$ is a set of facts $x = v$ where $x \in \mathcal{X}$ and $v \in D_x$;*
- *$\mathcal{S} \subseteq \mathcal{C}$ is a set of constraints;*
- *$\mathcal{N}$ is a set of facts $x = v$, such that x is assigned the value v in all solutions of $\mathcal{S}$ that agree with $\mathcal{E}$.*

Explanation steps are closely related to MUSs: an explanation of $\mathcal{N} = \{x = v\}$ corresponds to the MUS of $\mathcal{E} \wedge \mathcal{S} \wedge \{x \neq v\}$. Hence, MUS-finding algorithms can be used to compute explanation steps [3, 6]. We will use an OUS algorithm, where we denote an explanation step $(\mathcal{E}, \mathcal{S}, \mathcal{N})$ as $\boldsymbol{e}$ and will define a linear scoring function $f(\boldsymbol{e})$ for use with OUS.



5		6	7					8
7		1	5		3			9
3		2	6			7	5	1
1	5		4	9	6	2	3	7
9	2	4	3	1	7	5	8	6
6	3	7		5	2	1	9	4
8	1	5	9	3	4	6	7	2
2	7	3		6		9	4	5
4	6	9	2	7	5	8	1	3

5		6	7					8
7		1	5		3			9
3		2	6			7	5	1
1	5		4	9	6	2	3	7
9	2	4	3	1	7	5	8	6
6	3	7		5	2	1	9	4
8	1	5	9	3	4	6	7	2
2	7	3		6		9	4	5
4	6	9	2	7	5	8	1	3

Figure 1 Two possible explanation steps from the Sudoku app for $\{\texttt{cells}[7,7] = 6\}$ (in green). Used facts are highlighted in yellow, while used constraints are the blue rectangles. Both are MUSs; the one on the left has size 9 (8 facts and 1 constraint to explain the green 6), the one on the right has size 7 (4 facts and 3 constraints).

3 Data-driven Methodology

Our goal is to learn a scoring function for predicting a score for every possible explanation in a certain problem domain. We will use ML methods on user-labelled data, which requires defining a feature $\phi(\boldsymbol{e})$, that maps the explanation $\boldsymbol{e}$ to fixed-size vector of numbers (features). This allows one to learn a function $f_\theta(\phi(\boldsymbol{e}))$ with θ being the parameters of the learning model. A linear function corresponds to $\theta^\top \phi(\boldsymbol{e})$, which is linear in $\boldsymbol{e}$ if $\phi(\boldsymbol{e})$ is.

3.1 Feature Encoding

Technically, the OCUS method [6] allows searching for a OUS given a weight for every constraint. However, since the relevant constraints can differ between problem instances in the same domain, we can not learn a weight for each individual constraint directly.

The feature mapping $\phi(\boldsymbol{e})$ allows us to abstract from that. Each feature should represent a meaningful property of a MUS. A straightforward property is the number of constraints in a MUS, minimizing this leads to SMUS. However, there may be many SMUSs, so additional properties are required to discriminate between them. Additionally, to apply the OCUS algorithm, we have to ensure that the objective is a linear function of the constraints. To do so, we define each feature to be a linear sum over a meaningful subset of the constraints. In other words, if we can *group* the constraints into different groups, the *count* of elements within each group serves as the feature value. We can then use machine learning to learn a weighted combination over these counts.

▶ **Example 6.** Take a simple Boolean satisfiability problem: $a, b, \neg a \vee b, \neg a \vee \neg b$. One SMUS is $a, \neg a \vee b, \neg a \vee \neg b$ and another is $a, b, \neg a \vee \neg b$. Both have size 3 and can not be discriminated based on size, though one can imagine most people finding the second one easier to interpret. If we group constraints into 'unit clauses' and 'other clauses', then the first would have counters (1,2) and the second (2,1) and any function assigning larger weight to the 'other clauses' feature would always generate MUSs of the second kind.

Features for explaining unsatisfiability

We consider the case where a constraint is not restricted to a clause (as in SAT solving), but rather can be any expression over finite-domain variables (as is typical in CP). This means that we can group the constraints based on the **syntactic form** of expression used. For instance, we can group constraints into facts $(x = v)$, pairwise inequalities $(x_1 \neq x_2)$), linear constraints ($(\sum_i (w_i \cdot x_i) \geq t)$, particular types of global constraints (e.g., `alldifferent`(S)), etc. These features are independent of the problem domain and therefore **domain-agnostic**.

We can also identify additional **domain-specific** groupings. For example in the sudoku use case explained below, we will differentiate between alldifferent constraints over rows, over columns and over blocks. Syntactically they are all the same, but visually, users might have preferences of one over the other.

Features for explaining a fact $x = v$

When we wish to explain that $\mathcal{E} \wedge \mathcal{S} \Rightarrow (x = v)$ we compute a MUS of the unsatisfiable $\mathcal{E} \wedge \mathcal{S} \wedge (x \neq v)$. In this case, we can exploit the knowledge that we want to explain $x = v$.

We propose to **measure the distance** from any constraint to the fact $x = v$. More specifically, consider the (bipartite) variable-constraint graph in which variables are connected to the constraints they are involved in. We can group constraints by the *distance* (in this graph) to the variable x. For instance, the constraints at distance one are precisely those that mention x. For the special case of constraints that themselves are simple facts, i.e., $y = z$, we can use knowledge of the target fact $x = v$ in two ways: we can group the facts based on the distance of x to y or we can group them based on whether they assign the same value ($v = z$) or a different value ($v \neq z$). As these features are only dependent on the variable-constraint graph, they are **domain-agnostic**. Also here we can identify additional domain-specific groupings; we will clarify this with sudoku examples in Section 4 below.

3.2 ML Model Training

Our training data consists of a set of triples in the form of $(\boldsymbol{e}_1, \boldsymbol{e}_2, l)$, where $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ are two alternative explanations for the same unsatisfiable problem, and l indicates which one the user preferred. Given that users are expected to rank one explanation higher than another, the learning task can be framed as a Learning-to-Rank (LtR) problem [16]. LtR aims to learn a ranking function that explicitly orders any new set of items. We particularly focus on pointwise classification and pairwise classification, which learn a scoring function. As is common in LtR, we will disregard the triples for which the user has no preference.

Pointwise classification. The LtR problem can be framed as a standard classification task. This is a *pointwise* LtR approach because this approach treats each explanation independently. Given a tuple $(\boldsymbol{e}_1, \boldsymbol{e}_2, l)$ we generate two data points: $(\phi(\boldsymbol{e}_l), +1)$ for the explanation that was chosen by the user, and $(\phi(\boldsymbol{e}_o), -1)$ for the other one. We can then use any off-the-shelf classifier for training, including linear logistic regression [9], decision trees [4], random forests [1], SVM classifiers [22] and more. Out of these, the logistic regression (when dropping the logistic function computation at inference time) and SVM classifier (when used with a linear kernel) produce linear scoring functions.

Pairwise learning-to-rank. In the pairwise LtR formulation, each $(\boldsymbol{e}_1, \boldsymbol{e}_2, l)$ can be used directly as training data. Let $\boldsymbol{e}_l$ be the explanation chosen by the user, and $\boldsymbol{e}_o$ be the other. Then we know that the former is preferred over the latter: $\boldsymbol{e}_l \succ \boldsymbol{e}_o$. In pairwise learning-to-rank, the methods learn a function $f_\theta(\phi(\boldsymbol{e}))$ such that $f_\theta(\phi(\boldsymbol{e}_l)) > f_\theta(\phi(\boldsymbol{e}_o))$ for every labeled query. Examples of pairwise LtR methods are SVMRank [11] and XGBoostRanker [5]. Only SVMRank with a linear kernel will produce a linear scoring function.

4 Use-case: Step-wise Explanations for Sudoku

To evaluate the feasibility of using ML to learn a preference scoring function over unsatisfiable subsets, we will use the concept of explanation steps over sudoku puzzles [6].

An explanation step in sudoku explains why an empty cell needs to take a particular value. This is then repeated for the remaining empty cells, until all cells in the sudoku are explained. All cell values must be explainable by the constraints and the initial assignment, because a sudoku has a unique solution.

For the labelling, we developed a mobile-friendly web app that highlights the facts, rows, columns and squares that are used in an explanation, as shown in Fig. 1. Once familiar with the concept of explanation steps, we observed that users tend to have explicit preferences between them, and that they can label pairs relatively quickly. We store the response as a triple $(\boldsymbol{e}_1, \boldsymbol{e}_2, l)$ where $l = 0$ if no preference is given, $l = 1$ if the first is chosen and $l = 2$ if the second was chosen.

Our generation of explanation steps is as follows: 1) we use QQWing [19] to generate *intermediate* Sudoku puzzles. We opt for this level to avoid limiting participation to only users with expert skills. 2) we use the OCUS tool [6] to enumerate a sequence of explanation steps by computing the smallest MUS. For every step, we then either generate the second-smallest MUS, or use a learned cost-function to generate an OUS depending on the experiment.

Feature encoding

All constraints for sudoku are either facts ($x = v$) or alldifferent constraints. So the two most basic **syntactic features** are: number of facts and number of constraints. Using the concept of **distance to the fact** explained above, we can further group the constraints into *adjacent* constraints (constraints at distance 1 from the fact to explain $x = v$) and other constraints (distance > 1). The facts can also be split into adjacent facts (the variables of the adjacent constraints) and other facts, as well as whether the fact assigns the same value as $x = v$ or a different one. These features are **domain-agnostic**, in our case we selected a subset of 5 linearly independent features (see Appendix for the detailed list).

For the **domain-specific features**, we can further split the above constraint groups into separate subgroups for row, column and block constraints. We also introduce separate counters for adjacent row/column or block facts; it is not an issue that adjacent block facts overlap with row/column facts. The domain-specific features are considered together with the domain-agnostic ones, resulting in 12 linearly independent features (see Appendix for the full list).

5 Experiments and Results

We will evaluate how well machine learning methods are able to capture user preferences over sudoku explanation steps. The research questions are:

RQ1. How important is the feature representation to learn good scoring functions?
RQ2. What learning techniques (pointwise and pairwise) perform best?
RQ3. How well can the method be used for generating new explanations, and does this improve over multiple feedback iterations?

Setup. We use a single core of a 6-core INTEL(R) 2.6 Ghz processor with 16GB of RAM for all experiments. We implement CSP modeling and OUS-based explanation generation using CPMpy 0.9.16 [7], and ML classifiers from Sci-Kit Learn 1.3.0 [20], XGBoost 1.7.6 [5] and PySVMRank using 5-fold cross-validation.

5.1 Learning a discriminator function

We evaluate which type of features and learning methods are suitable to learn to discriminate between two explanations. For around 250 explanation steps, we generate the top-2 SMUSs and let users label these pairs. Table 1 presents the performances of the models, using 5-fold crossvalidation independently for data of 6 different users. We evaluate it according to:

Correct When the scoring function scored the user-chosen explanation highest.
Equal When the scoring function gave the two explanations an equal score.
Wrong When the scoring function scored the user-chosen explanation lowest.

In Table 1 we distinguish between only using the number of constraints and facts as features, using only the domain-agnostic features, and using both domain-agnostic and domain-specific features. As benchmarks, we consider two handcrafted linear functions: the one that is used for computing the SMUS [10] and another from [3] making a distinction between the number of used alldifferent constraints c and the number of used facts f. In terms of ML techniques we considered linear pointwise classifiers (Logistic Regression, SVM with linear kernel) and a non-linear classifier (Gradient Boosted Trees), as well as a linear ranker (SVM rank with linear kernel) and a non-linear one (Gradient Boosted Tree ranker).

Table 1 (top row) shows that only using the size of the MUS leads to a large amount of *Equal* results, which is to be expected since we generate the top-2 SMUSs to choose between. Conversely the manual function from [3] can distinguish more explanations. Curiously, all ML methods perform slightly worse than this function, perhaps due to the amount of data.

We also see that considering only domain-agnostic features still leads to many *Equals*, meaning that such features are largely insufficient to differentiate between the two SMUSs. At the same time all classifiers perform very similar.

Finally, when using all features we rarely have *Equal* cases. We can also see very strong performance by all predictors, linear and non-linear, with an additional benefit for the pairwise ranking methods. This shows that ML methods can learn to discriminate between two SMUSs with very high accuracy given a labelled dataset. This is true across the users.

5.2 Iteratively learning a generator function

Given that SVM Ranker (with a linear kernel) returns a linear scoring function, we proceed to examine whether this learned scoring function is suitable for *generating* the preferred explanation directly. To test this, we generate around 50 new pairs of sudoku explanation steps from a single puzzle. Each pair is an SMUS and one using the OCUS algorithm optimizing the learned scoring function, and we ask the users which one they prefer.

The first row (Iteration 1) in Table 2 shows that this initially performs much worse than generating an SMUS. This is because the predictors are only trained on SMUSs and are hence biased to predict between MUSs that are already small and subset minimal. Since the learned scoring function is not trained to rank all MUSs, this essentially tests its performance on a large out-of-sample distribution and we can see that it is too specific to the training data.

To address the shortcoming of the data, we conduct additional iterations as is common in machine learning with concept drift. In the $i + 1^{\text{th}}$ iteration, the newly labeled explanation pairs are added to the training data. For each iteration we evaluate the predictive accuracy

Features	Classifier	Correct	Equal	Wrong
# cons, # facts	Handcr. SMUS $(c + f)$ [10]	7.1%	90.1%	2.8%
	Handcr. OUS $(20 \times c + f)$ [3]	25.6%	70.6%	3.8%
	Logistic Regression	25.3%	70.6%	4.1%
	SVM Classifier	20.4%	73.0%	6.6%
	SVM Ranker	24.4%	70.7%	4.8%
Domain-agnostic	Logistic Regression	28.8%	64.8%	6.4%
	SVM Classifier	29.9%	64.8%	5.2%
	XGBoost Classifier	28.4%	65.2%	6.4%
	SVM Ranker	29.2%	64.8%	6.0%
	XGBoost Ranker	28.8%	65.7%	5.5%
Domain-specific	Logistic Regression	90.7%	1.2%	8.1%
	SVM Classifier	85.7%	1.4%	12.9%
	XGBoost Classifier	87.3%	1.2%	11.5%
	SVM Ranker	90.8%	1.3%	7.9%
	XGBoost Ranker	91.8%	1.5%	6.7%

Table 1 Correctness of Point wise Classifiers and Pairwise ranker methods. We train one classifier per user and compute the average.

	Test data			New generate & label		
Iteration	Correct	Equal	Wrong	ML wins	No preference	SMUS wins
1	94.0%	0.0%	6.0%	0.0%	0.0%	100.0%
2	75.1%	0.0%	24.9%	7.3%	7.3%	65.4%
3	72.2%	0.0%	27.8%	29.8%	8.8%	61.4%
4	70.5%	0.0%	29.5%	44.6%	39.3%	16.1%
5	66.8%	0.0%	33.2%	62.5%	19.6%	17.9%

Table 2 Iterative learning approach with SVM Rank (linear kernel) over domain-specific features. In each iteration, new data is labelled by the user and added to the training data of the next iteration.

using crossfold validation (3 left columns) and on newly generated pairs that use the learned scoring function (3 right columns).

As the number of iterations increases, the accuracy in predicting the preferred explanation on the training data decreases. This shows that there are fewer 'shortcuts' to take in the data, and the classifier has a harder time balancing the multiple and possibly conflicting preferences in the labelled data set.

At the same time, we can see that iteration after iteration, the scoring function becomes better suited at directly generating explanations. Indeed, the percentage of times where the ML generated explanation is preferred over an SMUS increases. It is hence learning a preference scoring function that better aligns with the user preferences and can be used as to generate explanations directly.

6 Conclusions

In this paper, we consider the task of learning a scoring function for MUSs to capture which ones are more preferred by a user. To do so, we propose a principled feature encoding technique for MUSs. This makes it possible to learn a scoring function from a dataset of pairwise comparison, using learning-to-rank techniques. We demonstrate this approach on the easy-to-interpret task of explaining sudoku cell assignment. In this domain, using domain-specific features was essential to discriminate between preferred smallest MUSs. Using linear features and linear SVMrank, we further showed that an iterative training approach can learn a scoring function that can be used to generate good explanations directly.

Many avenues for future work exist. First, we only applied this on the pedagogical sudoku domain where it was easy and quick to experiment. This approach can now be validated in more realistic problem domains, such as explanations in staff rostering or crew scheduling, where domain experts are harder to find. Our pairwise labeling approach also requires good domain-specific visualisations, for users to quickly see the differences and express a preference. This potential interaction between visualisation quality and perceived preferences invites a Human-Computer Interaction approach to designing and evaluating explanations in domain-specific settings. Our learning techniques can then be used in a subsequent step.

On the technical side, the sufficiency of a linear scoring function and the efficacy of the iterative method invite to investigate online, real-time learning approaches. Online methods could require less data, where data labelling is an important bottleneck to the use as well as the evaluation of the techniques in this work. Online learning would also require sufficiently fast OUS algorithms for interactive use, another key bottleneck.

References

1 Jehad Ali, Rehanullah Khan, Nasir Ahmad, and Imran Maqsood. Random forests and decision trees. *International Journal of Computer Science Issues (IJCSI)*, 9(5):272, 2012.

2 Ignace Bleukx, Jo Devriendt, Emilio Gamba, Bart Bogaerts, and Tias Guns. Simplifying step-wise explanation sequences. In Roland H. C. Yap, editor, *29th International Conference on Principles and Practice of Constraint Programming, CP 2023, August 27-31, 2023, Toronto, Canada*, volume 280 of *LIPIcs*, pages 11:1–11:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

3 Bart Bogaerts, Emilio Gamba, and Tias Guns. A framework for step-wise explaining how to solve constraint satisfaction problems. *Artif. Intell.*, 300:103550, 2021.

4 Leonard A Breslow, David W Aha, et al. Simplifying decision trees: A survey. *Knowledge engineering review*, 12(1):1–40, 1997.

5 Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016.

6 Emilio Gamba, Bart Bogaerts, and Tias Guns. Efficiently explaining csps with unsatisfiable subset optimization. *J. Artif. Intell. Res.*, 78:709–746, 2023.

7 Tias Guns. 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)*, volume 19, 2019.

8 Sharmi Dev Gupta, Begum Genc, and Barry O'Sullivan. Explanation in constraint satisfaction: A survey. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 4400–4407. ijcai.org, 2021.

9 David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression.* John Wiley & Sons, 2013.

10 Alexey Ignatiev, Alessandro Previti, Mark H. Liffiton, and João Marques-Silva. Smallest MUS extraction with minimal hitting set dualization. In Gilles Pesant, editor, *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings*, volume 9255 of *Lecture Notes in Computer Science*, pages 173–182. Springer, 2015.

11 Thorsten Joachims. Training linear svms in linear time. In Tina Eliassi-Rad, Lyle H. Ungar, Mark Craven, and Dimitrios Gunopulos, editors, *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006*, pages 217–226. ACM, 2006.

12 Ulrich Junker. Quickxplain: Conflict detection for arbitrary constraint propagation algorithms. In *IJCAI'01 Workshop on Modelling and Solving problems with constraints*, volume 4. Citeseer, 2001.

13 Niklas Lauffer and Ufuk Topcu. Human-understandable explanations of infeasibility for resource-constrained scheduling problems. In *ICAPS 2019 Workshop XAIP*, 2019.

14 Kevin Leo and Guido Tack. Debugging unsatisfiable constraint models. In Domenico Salvagnin and Michele Lombardi, editors, *Integration of AI and OR Techniques in Constraint Programming - 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings*, volume 10335 of *Lecture Notes in Computer Science*, pages 77–93. Springer, 2017.

15 Mark H. Liffiton, Alessandro Previti, Ammar Malik, and João Marques-Silva. Fast, flexible MUS enumeration. *Constraints An Int. J.*, 21(2):223–250, 2016.

16 Tie-Yan Liu. *Learning to Rank for Information Retrieval.* Springer, 2011.

17 João Marques-Silva. Minimal unsatisfiability: Models, algorithms and applications (invited paper). In *40th IEEE International Symposium on Multiple-Valued Logic, ISMVL 2010, Barcelona, Spain, 26-28 May 2010*, pages 9–14. IEEE Computer Society, 2010.

18 George Mavrotas. Effective implementation of the epsilon-constraint method in multi-objective mathematical programming problems. *Appl. Math. Comput.*, 213(2):455–465, 2009.

19 Stephen Ostermiller. Qqwing–sudoku generator and solver, 2011.

20 Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in python. *CoRR*, abs/1201.0490, 2012.

21 Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006.

22 Bernhard Schölkopf, Alex J Smola, Robert C Williamson, and Peter L Bartlett. New support vector algorithms. *Neural computation*, 12(5):1207–1245, 2000.

A Appendix

Features for Syntactic-only training	Description
number_adjacent_constraints	No. of used adjacent constraints
number_other_constraints	No. of non-adjacent used constraints
number_adjacent_facts_other_value	No. of used and adjacent facts that have a different value from the explained fact
number_other_facts_same_value	No. of used and non-adjacent facts that have the same value of the explained fact
number_other_facts_other_value	No. of used and non-adjacent facts that have a different value from the explained fact

Features for Domain-specific training	Description
number_adjacent_facts_other_value	No. of used and adjacent facts that have a different value from the explained fact
number_other_facts_same_value	No. of used and non-adjacent facts that have the same value of the explained fact
number_other_facts_other_value	No. of used and non-adjacent facts that have a different value from the explained fact
number_adjacent_block_cons	No. of of adjacent used block constraints
number_adjacent_row_cons	No. of of adjacent used row constraints
number_adjacent_col_cons	No. of of adjacent used column constraints
number_other_block_cons	No. of non-adjacent used block constraints
number_other_row_cons	No. of non-adjacent used row constraints
number_other_col_cons	No. of non-adjacent used column constraints
number_adjacent_block_facts	No. of used and adjacent facts from blocks
number_adjacent_row_facts	No. of used and adjacent facts from rows
number_adjacent_col_facts	No. of used and adjacent facts from columns