CoDeC: Constraints-Guided Diverse Counterfactuals

Avia Asael
aviaasael@mail.tau.ac.il
Tel Aviv University
Tel Aviv, Israel

Nave Frost
nafrost@ebay.com
eBay Research
Netanya, Israel

Amir Gilad
amirg@cs.huji.ac.il
Hebrew University
Jerusalem, Israel

Daniel Deutch
danielde@post.tau.ac.il
Tel Aviv University
Tel Aviv, Israel

Abstract

In explainable AI, a prominent approach is to explain the predictions of a classifier (such as a Neural Network) using *counterfactuals* (CFs), namely instances resulting from perturbations that lead to a change of the classification label. We propose to demonstrate a prototype system called CoDeC, which computes CF-based explanations for a given classifier and prediction. Uniquely, CoDeC computes a *diverse* set of CFs that adhere to *denial constraints*, which are defined with respect to a database of instances. These constraints may be mined from e.g. a training set, and/or manually crafted. They are then intuitively used to guide the generation of *realistic CFs*. The implementation is based on combining a previously proposed solution for finding diverse CFs (DiCE) with an iterative approach that first looks for CFs while disregarding the constraints, then looks for tuples in the proximity of the obtained CFs that further satisfy the constraints, and repeats the process if necessary. We will demonstrate CoDeC in the context of explanations for neural models trained and deployed with respect to the Adult Income and NY-housing datasets.

Keywords

Explainable AI, Counterfactuals, Machine Learning, Database Constraints

1 Introduction

Counterfactuals for Explainable AI. The widespread use of ML models in high-stakes decisions necessitates explainable AI solutions. A particular kind of explanation is called *counterfactuals* (CFs). Given a classifier M and an instance I let $M(I)$ be the prediction of M for I. Then, a CF is a perturbed instance I' such that $M(I') \neq M(I)$. A typical goal is to find such I' that is in the proximity of I. The perturbations applied to I to obtain I' then highlight root causes of the original label and provide actionable recommendations for individuals seeking label changes.

Example 1.1. Consider a scenario of an income prediction task using a subset of the Adult Income Dataset (https://archive.ics.uci.edu/ml/datasets/adult) as shown in Table 1. In this example, we focus, for brevity, on a specific set of attributes (see Table 1): Education (categorical, reflecting degree), Edu-Num (numerical, ordinal encoding of education), Occupation (categorical) and Hours (numerical, reflecting hours of work per week). The task is to predict whether an individual's annual income is at least \$50K. Further, consider an individual whose profile is shown in the "Input" row of Table 1. They have a bachelor's degree (also encoded as Edu-Num=13), and work in tech support for 27 hours a week. A particular neural network for the task may predict that this individual has a low income (<\$50K). The individual may ask why the model predicted this, and and what they could do to change it. Such questions may be addressed by computing and presenting *counterfactuals* (CFs), namely concrete changes the user can make to their profile (such as pursuing an academic degree, changing occupation, or increasing their working hours) that will likely result in higher income. A CF could be a profile of a person with doctorate education level, working in tech support for 27 hours a week and is thus predicted to earn over \$50K. Additional CFs are presented in Table 2 (ignore for now the "Method" and "Prob.?" columns).

Table 1: Sample dataset tuples with a subset of the attributes from the Adult Income Dataset.

ID	Education	Edu-Num	Occupation	Hours	Label
t_1	Bachelors	13	Exec_managerial	55	$\geq$ \$50K
t_2	Masters	14	Protective_serv	35	$\geq$ \$50K
t_3	Assoc_voc	11	Tech_support	40	$<$ \$50K
t_4	Doctorate	16	Exec_managerial	55	$\geq$ \$50K

Table 2: CFs generated by DiCE [8] and by CoDeC. DiCE generates CFs that violate the DCs, while CoDeC ensures that the generated CFs will satisfy them.

Method	Education	Edu-Num	Occupation	Hours	Prob.?
Input	Bachelors	13	Tech_support	27	–
DiCE	Doctorate	16	Tech_support	27	No
	Doctorate	13	Exec_managerial	38	DC1
	Doctorate	11	Exec_managerial	27	DC1,2
CoDeC	Doctorate	16	Tech_support	27	No
	Masters	14	Exec_managerial	38	No
	Bachelors	13	Protective_serv	46	No

Existing Approaches to Counterfactual Generation. There are a large number of algorithms and system prototypes that compute CFs for predictions made by classifiers [6]. These solutions are guided by different sets of desiderata, including *proximity* to the original instance and the *feasibility* of realizing the CF. For a set of CFs, one typically is interested in its *diversity*, so as to facilitate inherently different recommendations or explanations. Other desiderata appear in the literature as well [6]. A notable approach is the one followed by DiCE [8], aiming to optimize a particular tradeoff between diversity and proximity. As we shall demonstrate, DiCE often falls short with respect to feasibility.

Example 1.2. Rows 2–4 of Table 2 present the CFs generated by DiCE [8] for the input instance appearing in the top row of

EDBT '26, Tampere (Finland)
© 2026 Copyright held by the owner/author(s). Published on OpenProceedings.org under ISBN 978-3-98318-104-9, series ISSN 2367-2005. Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

10.48786/edbt.2026.62

the same table, and a toy neural network that we have trained. DiCE has several parameters that need to be tuned and these results are obtained for a specific configuration (see Section 2 for details), yet we have observed similar phenomena for different configurations and models. The first CF computed by DiCE is valid and involves obtaining a PhD, perturbing both the Education and Edu-Num attributes accordingly. By contrast, the second and third CFs (marked red in the table) exhibit a mismatch between the Education and Edu-Num attributes (PhD education with Edu-Num of 11 or 13 rather than 16). The third CF also exhibits a mismatch between Occupation and Hours (intuitively, the number of work hours is too small for a managerial position).

Denial Constraints for Realistic Counterfactuals. To capture feasibility of CFs, we propose using database Denial Constraints (DCs) [3]. DCs are first-order logic formulas of a restricted form that allow to capture constraints over individual tuples (via unary DCs) or pairs of such tuples (via binary DCs).

Example 1.3. Characteristics of valid instances for the simplified Adult schema, may be formally captured using the following DCs:

- **DC1**: $\forall t_1, t_2.\neg(t_1.Education = t_2.Education \wedge t_1.Edu-num \neq t_2.Edu-num)$
- **DC2**: $\forall t_1.\neg(t_1.Occupation = Exec\_managerial \wedge t_1.Hours \leq 35)$

The first DC is binary, enforcing that education credentials must match their (standard) duration; the second DC is unary and it enforces that executive managers must work more than 35 hours per week. Indeed, the second and third CFs computed by DiCE (see Example 1.2) violate DC1, with the third violating DC2.

Diverse and Realistic Counterfactuals with CoDeC. To address shortcomings of existing approaches for counterfactual-based explanations, we propose to demonstrate a system prototype called CoDeC (COnstraints-guided DiversE Counterfactuals). CoDeC accounts for both CF quality and feasibility. The former is captured by diversity and proximity, and is already attainable by using a solution such as DiCE. For the latter, CoDeC enforces that counterfactuals adhere to DCs, with respect to a given dataset (e.g. training set). In a sense, the database tuples along with DCs serve as yardsticks on what constitutes a valid instance.

Example 1.4. Recall DC1 from Example 1.3 and "instantiate" it with t_2 in DC1 replaced by t_4 from Table 1. The instantiated constraint is then $\forall t_1.\neg(t_1.Education = Doctorate \wedge t_1.Edu-num \neq 16)$, i.e. the ordinal encoding of "Doctorate" is 16. Given a candidate counterfactual, we may now test whether it satisfies this constraint.

DCs may either be mined from an example dataset (see e.g. [11]), or manually tailored by domain experts, and are then fed as input to CoDeC along with the dataset itself. Then, given a model and an instance, CoDeC computes counterfactuals as follows. It first (in a step we refer to as that of "perturbation") uses DiCE to generate initial CF candidates, typically obtaining CFs of high quality in terms of proximity and diversity but ones that are often not feasible. Then (in the "projection" step) CoDeC looks for instances in the proximity of each CF candidate that do satisfy the DCs. This is a classical optimization-under-constraints problem and is solved using an SMT solver. Projection may lead to a change of classification label and the projected instance may thus no longer be a counterfactual, in which case we need to repeat the process. After each perturbation, we get a pool of counterfactuals of increasing size (since DiCE generates multiple perturbed candidates for each instance), and while there is no theoretical guarantee of convergence, we observe that the solution often converges within 10–20 seconds outputting high quality and realistic CFs. We also stress that our solution is generic and other Counterfactual Generation modules may in principle be plugged in instead of DiCE.

Proposed Demonstration. We propose to demonstrate CoDeC in the context of models trained over two datasets and their respective classification task, namely Adult (a simplified version of which is presented in this short paper) and NY housing where the goal is to estimate whether the price of a given house exceeds $\$1M$. We will show the usefulness of CoDeC for both explanations and recommendations, compare its output to the one obtained by DiCE to highlight the usefulness of enforcing constraints, and allow the audience to interact with CoDeC.

Video and Code Repository and Full Paper. A demonstration video is available at https://youtu.be/B_hUA-0d-QM. Source code is available at https://github.com/asaelavia/CoDeC. A full paper presenting in more detail our algorithmic contribution as well as experimental results is currently under review; we do not provide a link here due to anonymity considerations for its review process.

2 Technical Background

We briefly discuss the technical details of CoDeC.

2.1 Model

Datasets and Models. Let $\mathcal{S} = \{A_1, \ldots, A_m\}$ be a single-relation schema, namely a set of attributes where each A_i is associated with a domain $dom(A_i)$. We use $\mathcal{S} = cat \cup cont$ where cat and $cont$ are the sets of *categorical* and *continuous* attributes resp. in $\mathcal{S}$. A tuple t that conforms to $\mathcal{S}$ is an element of $dom(A_1) \times \cdots \times dom(A_m)$. A *classifier* M maps tuples corresponding to the schema $\mathcal{S}$ to $\{0, 1\}$; we refer to $M(t)$ as the *label* that M assigns to a tuple t. A (single-relation) *database* D over $\mathcal{S}$ is a set of tuples conforming to $\mathcal{S}$.

Denial Constraints (DCs). A binary DC σ takes the form:

$$\sigma = \forall t_1, t_2 : \neg(P_1 \wedge P_2 \wedge \ldots \wedge P_l)$$

where each predicate P_i can be either (1) a comparison between attributes of two tuples, i.e., $t_1[A_i]\ \phi\ t_2[A_i]$ or (2) a comparison between an attribute and a constant, i.e., $t_1[A_i]\phi\ v$ or $t_2[A_i]\ \phi\ v$. The comparison operator ϕ is any of $\{=, \neq, >, <, \geq, \leq\}$, and v represents a constant from the attribute's domain. A unary DC only quantifies over t_1 and only includes predicates of type (2). We then say that a database D satisfies a unary DC σ if every tuple in D satisfies the condition specified in σ; similarly for a binary DC and every pair of tuples in D. Further, n-ary formulas are also allowed by the general form of DCs but are less common.

Valid Counterfactuals. Given a schema $\mathcal{S}$, a database D over $\mathcal{S}$, a classifier M, a set of DCs DC, and tuples t, cf corresponding to $\mathcal{S}$ (but may or may not appear in D), we say that cf is a *valid CF* for t if $M(cf) \neq M(t)$, and $D \cup \{cf\}$ satisfies all constraints in DC.

We aim to find a set of k valid CFs $Cfs = \{cf_1, \ldots, cf_k\}$ optimized according to further desiderata introduced in [8] and are next recalled. For a tuple t and an attribute $A_l \in \mathcal{S}$, we use $t[A_l]$

to denote the value of t in the A_l column; for a database D, we use $D[A_l]$ to denote the multiset of values $\{\{t[A_l]\}\}$ where $t \in D$.

Counterfactual Distance. Following [8], we employ different distance metrics for categorical and continuous attributes, then combine them. For categorical attributes, we use $L0$ distance:

$$dist_{cat}(t, t') = \frac{1}{|cat|} \sum_{A_l \in cat} \mathbb{1}(t[A_l] \neq t'[A_l])$$

For continuous attributes, we use a scaled Manhattan distance normalized using the median absolute deviation (MAD):

$$dist_{cont}(t, t') = \frac{1}{|cont|} \sum_{A_l \in cont} \frac{|t[A_l] - t'[A_l]|}{MAD_{A_l,D}}$$

where the MAD for an attribute A_l is the median absolute deviation from the median. Formally, let M_l be the median of $D[A_l]$. We then define $MAD_{A_l,D} = median(\{\{|t_i[A_l] - M_l| \mid t_i \in D\}\})$. Intuitively, we normalize to account for attributes of high and low variances. Finally, we define $dist(t, t') = dist_{cat}(t, t') + dist_{cont}(t, t')$.

Counterfactuals Set Diversity. Let $S = \{cf_1, ..., cf_n\}$ be a set of CFs. As in DiCE [8], to quantify diversity, we employ the Determinantal Point Process (DPP) diversity metric [7]: $dpp\_div(S) = det(K)$, where K is a kernel matrix with entries: $K_{i,j} = \frac{1}{1+dist(cf_i,cf_j)}$. $dpp\_div(S) \in [0, 1]$ and higher values indicate more diversity.

Total Score. Let w_1, w_2 be configurable weights used to balance diversity and distance. For an instance t and a candidate set S of CFs, we define the total score of S as $score(S) = w_1 \cdot dpp\_div(S) + w_2 \cdot \frac{1}{|S|} \cdot \sum_{t_i \in S} dist(t_i, t)$.

Immutable and mutable attributes. We further allow users to specify a subset of the attributes they wish to keep intact ("immutable") in the CFs, i.e. not to perturb them. These may correspond to attributes that may not be changed (such as age) or that a particular user does not wish to change or is not interested in observing their impact on the classification.

2.2 Computing Counterfactuals

Our solution adopts a perturb-and-project approach similar to that proposed in [5], adapted to the generation of multiple CFs. We leverage DiCE [8], an algorithm that finds a diverse set of CFs, to obtain an initial set of k candidate CFs. Since these CFs may violate the DCs, we project them onto the constraints space using a newly designed projection algorithm (see below). If a projected tuple maintains the desired classification, we include it in our candidates set. Otherwise, we continue this iterative process, applying DiCE with each tuple serving as input (we feed the choice of immutable attributes to DiCE), then projecting the result. We end the process when we have obtained a pool containing at least k valid CFs. Naturally, the projection step may have hampered quality (proximity and/or diversity), and we greedily try to improve by selecting a set of k CFs from the pool as follows. We first choose the CF closest to the input tuple; then, at each step, we select the CF whose introduction to the set of currently selected CFs yields an optimal weighted average of distance and diversity scores, among all candidate CFs not yet introduced to the set of selected ones.

Projection Algorithm. Our approach transforms invalid CFs into constraint-satisfying ones using a Z3 SMT solver [4]. The algorithm operates in two phases. During pre-processing, we construct bounds from instantiated DCs—each bound represents a "forbidden region" of attribute values. The solver negates each bound to ensure projected tuples avoid all forbidden regions. During solving, the algorithm fixes immutable attributes to their original values and minimizes distance to find the closest valid tuple. We employ a suspect set optimization [3], significantly reducing the constraint space: rather than considering instantiations involving all dataset tuples, we focus on "suspect tuples", namely those that could potentially violate constraints given the immutable attribute values.

Example 2.1. Continuing our example from the Introduction, given DC1 and tuple $t_4 = (Doctorate, 16, ...)$, the instantiation yields $\neg(Education = Doctorate \wedge Edu\text{–}Num \neq 16)$; similarly for the other database tuples and also for DC2. Now, reconsidering the instance and CFs computed for it by DiCE in Table 2, we note that the first already satisfies all constraints and may be readily outputted. For the second, projection is needed to fix its violation of DC1; one possible such fix may involve changing $Education$ to Masters and $Edu-Num$ to 14. For the third CF computed by DiCE, projection fixes both violations by setting $Education$ to "Bachelors", $Edu-Num$ to 13, $Occupation$ to "Protective_serv", and $Hours$ to 46.

Configuration Choices. Beyond user choices (see Section 3), there are a number of configuration choices to be made in CoDeC. First, in the score function (see paragraph "total score") we set $w_1 = 1$ and $w_2 = -0.5$ and allow exploration of additional choices. Other configuration choices pertain to DiCE, allowing to control its quality/execution time tradeoff. For the demonstration, we set the learning rate of DiCE to be 0.03 for the adult dataset and 0.001 for NY-Housing; its number of gradient iterations is limited to $[250, 500]$ (Adult) and $[500, 1500]$ (NY-Housing). For the examples we show in the demo, this configuration lead to execution times of ~15 seconds for Adult and ~10 seconds for NY-Housing.

Related work and the novelty of CoDeC. DCs are commonly used in the context of data integrity and data repair [10], where the goal is to maintain consistent and/or accurate databases. Although there is extensive work on CF explanations [1, 6, 8, 9], using database constraints for CFs generation remains underexplored. A notable exception is the CeC framework [5], which follows a similar perturb-and-project approach but finds only a single CF (rather than looking for a diverse set of CFs as in our approach) and only supports a restricted class of constraints. Recently, [2] used DCs in a perturb-and-project algorithm for generating adversarial examples; such examples aim at fooling a model rather than explaining a tuple and so desiderata are different, and in particular [2] is not geared towards diversity. *To our knowledge, no prior work has simultaneously targeted both (1) feasibility of generated CFs captured by database constraints and (2) diversity of generated CFs.*

3 Demonstration Details

The demonstration will allow participants to view counterfactuals generated by CoDeC for predictions made by neural networks and to interactively explore the effect of different inputs and configuration choices on the generated counterfactuals.

Datasets and models. We will show counterfactuals for predictions made by two neural networks, trained offline on the Adult Income dataset and the NY-Housing dataset (https://www.kaggle.com/dsv/7351086). Adult Income is an extended version of the dataset shown in this short paper and contains demographic attributes, used to predict whether an individual earns more than $\$50K$ annually. The NY-Housing dataset focuses on New York

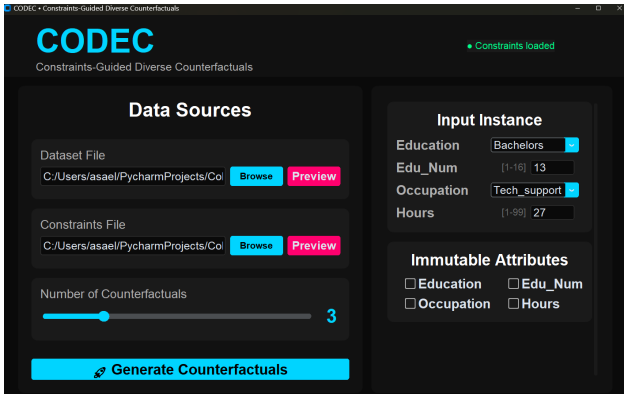

Figure 1: Input Screen

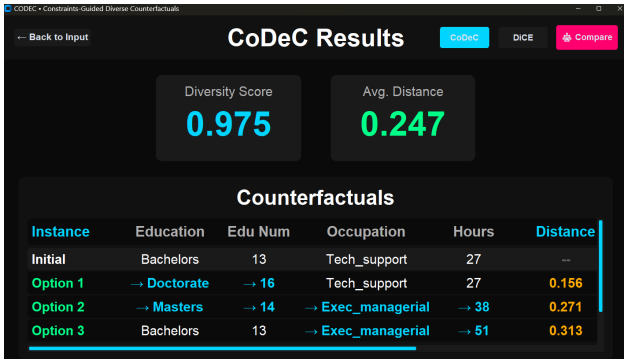

Figure 2: Results Screen

real estate, including attributes of real estate properties such as locality, bedroom and bathroom counts; the task is to predict whether a property's value exceeds $\$1M$. For each dataset we have trained in advance a small Multi-Layer Perceptron (MLP) neural network with ReLU activation functions in hidden layers and softmax activation for binary classification, to be used as the classifiers for which counterfactuals are computed. The demonstration will then consist of the following steps.

Setting the Input. Starting up CoDeC, the input screen depicted in Figure 1 will show. We will start the demonstration with the adult dataset, to which end we will upload from files the (full) adult income dataset as well as corresponding constraints we have prepared in advance and the prepared model we have trained in advance (saved as a file with a name corresponding to the dataset). Using the UI we will choose an example input instance reflecting characteristics of an individual for which the model predicts low income and we wish to understand why that is the case. We will set the number of requested CFs and choose a subset of the attributes to be immutable, and click "Generate Counterfactuals".

Counterfactual Generation. The execution time of CoDeC for the demonstrated configuration is approximately 15 seconds; meanwhile, we provide a high-level explanation of the algorithm.

Exploration of Counterfactuals. When execution terminates, the computed CFs will be shown in the results screen depicted in Figure 2. The screen includes the input instance marked as "Initial" followed by the computed CFs. For each of these, CoDeC highlights in blue the attributes that have been modified compared to the input instance. We will intuitively highlight the explanation to which each such CF corresponds. Statistics on the results, namely the diversity and average distance (see Section 2) of the CFs set are presented. Clicking on the "DiCE" tab (top right of the screen), we will show the audience the counterfactuals computed by DiCE, and explain which are feasible and which are not, as well as the reasons for infeasibility for the latter. Specifically, we will demonstrate which constraints are violated by DiCE (e.g. the one involving Education and Edu-Num discussed here, and others) and how their violations is avoided by CoDeC. We will also compare the obtained distance and diversity, demonstrating that the degradation in these measures that results from enforcing feasibility is acceptable.

Moving to the NY-Housing Dataset. We will then show examples using the NY-Housing dataset, demonstrating the versatility of CoDeC. Here again, we will choose input instances for which the model assigns a negative label (which now stands for a price estimate of less than $\$1M$), and ask CoDeC to explain the label. Constraints involve, for instance, limitations on the number of rooms/bathrooms as well as on the square footage size per room. We will use the "Compare" button (top-right corner of Fig. 2) to compare the CFs computed by DiCE and CoDeC.

Interactive Audience Choices. We will allow the audience to make their own choices for input instances and immutable attributes (right-hand part of Figure 1), and to also view and configure the constraints (clicking the "Browse" button next to the constraints file name in Figure 1 leads to a screen where users can view and edit constraints). We will specifically encourage the audience to explore how modification of constraints affects the results of CoDeC.

Acknowledgments

This work of Daniel Deutch and Avia Asael was partially funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 804302), by the Israeli Science Foundation (Grant 1476/24), by Len Blavatnik and the Blavatnik Family foundation, and by the Deutsch foundation. The work of Amir Gilad was funded by the Israel Science Foundation (ISF) under grant 1702/24, the Scharf-Ullman Endowment, and the Alon Scholarship.

References

[1] Shuai An and Yang Cao. 2024. Counterfactual Explanation at Will, with Zero Privacy Leakage. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–29.
[2] Matan Ben-Tov, Daniel Deutch, Nave Frost, and Mahmood Sharif. 2024. CaFA: Cost-aware, Feasible Attacks With Database Constraints Against Neural Tabular Classifiers. In *SP*. IEEE, 1345–1364.
[3] Xu Chu, Ihab F Ilyas, and Paolo Papotti. 2013. Holistic data cleaning: Putting violations into context. In *ICDE*. IEEE, 458–469.
[4] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *TACAS*. Springer, 337–340.
[5] Daniel Deutch and Nave Frost. 2018. CEC: Constraints based explanation for classifications. In *CIKM*. 1879–1882.
[6] Riccardo Guidotti. 2024. Counterfactual explanations and how to find them: literature review and benchmarking. *Data Mining and Knowledge Discovery* 38, 5 (2024), 2770–2824.
[7] Alex Kulesza, Ben Taskar, et al. 2012. Determinantal point processes for machine learning. *Foundations and Trends® in Machine Learning* 5, 2–3 (2012), 123–286.
[8] Ramaravind K Mothilal, Amit Sharma, and Chenhao Tan. 2020. Explaining machine learning classifiers through diverse counterfactual explanations. In *Proc. of FAccT*. 607–617.
[9] Rafael Poyiadzi, Kacper Sokol, Raul Santos-Rodriguez, Tijl De Bie, and Peter Flach. 2020. FACE: feasible and actionable counterfactual explanations. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*. 344–350.
[10] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. 2017. Holoclean: Holistic data repairs with probabilistic inference. *arXiv preprint arXiv:1702.00820* (2017).
[11] Renjie Xiao, Zijing Tan, Haojin Wang, and Shuai Ma. 2022. Fast approximate denial constraint discovery. *Proceedings of the VLDB Endowment* 16, 2 (2022), 269–281.