Realistic Counterfactual Explanations via Denial Constraints

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

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

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

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

Abstract

In the realm of Explainable AI, classification results are often explained via *counterfactuals* (CFs for short), which are (ideally small) perturbations to an instance that lead to a change of classification label. Such CFs may serve as explanations for the prediction, pinpointing the features that were important. Existing explainability solutions typically aim at *minimizing the distance* of CFs from the original instance so that they are specific to it; and/or *maximizing the diversity* of CFs to cover multiple facets of the reasons underlying the prediction. In this paper, we note that in pursuing these aims, state-of-the-art explainability solutions may (and often do) yield counterfactual explanations that do not correspond to realistic instances. This limits their applicability and usefulness in practice. To remedy this, we combine ideas from Explainable AI with ideas from data management. Specifically, we capture realism of CFs via *logical constraints* that hold with respect to a dataset of examples (e.g. training set); the class of such constraints that we focus on is that of *denial constraints*, extensively studied in the context of relational databases. Algorithmically, we then combine explainable AI solutions to yield CFs, with ideas from data cleaning that we adapt to this unique setting, to transform CFs into realistic ones. Extensive experiments across four datasets validate that our solutions achieve realism with relatively minor compromise in terms of distance and diversity. They further validate that the dedicated optimizations that we have developed to speed up the search for CFs are indeed highly effective.

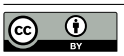
CCS Concepts

• **Information systems** → **Data management systems**; • **Computing methodologies** → **Machine learning algorithms**.

Keywords

Explainable AI, Counterfactuals, Denial Constraints

ACM Reference Format:
Avia Asael, Daniel Deutch, Nave Frost, and Amir Gilad. 2026. Realistic Counterfactual Explanations via Denial Constraints. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '26), August 09–13, 2026, Jeju Island, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3770855.3817712



KDD '26, Jeju Island, Republic of Korea

ACM ISBN 979-8-4007-2259-2/2026/08
https://doi.org/10.1145/3770855.3817712

Resource Availability:
The source code of this paper has been made publicly available at https://doi.org/10.5281/zenodo.20271397.

1 Introduction

Counterfactuals (CFs) explain an ML prediction made by a given model for a given instance, through perturbations that yield a change to the assigned label. Intuitively, these perturbations pinpoint the features that were significant for the original prediction. Indeed, there is a large body of research on CFs, both in terms of modeling (which CFs are more useful for explainability) and in terms of algorithms (how to efficiently find highly useful CFs) [24, 26, 46].

Consider, for example, the NY Housing dataset [20], a snippet of which appears at the top of Figure 1. It includes information on real estate assets, such as their type, number of bedrooms/bathrooms, square footage, sublocality (i.e. neighborhood), etc. A natural classification task is to decide whether an asset's price exceeds 1M USD. If the model outputs "no" for a particular asset, CFs could identify reasons: perhaps the same asset in a different neighborhood would exceed 1M USD? Perhaps a slightly larger square footage would pass the bar? Etc. Among many applications, the availability of such explanations promotes transparency and accountability, and allows identifying bugs and biases in the model. [36, 46, 47]

To be useful, CFs should be *in the proximity* of the original instance: continuing our example, price estimates for very dissimilar assets are uninformative. They should also ideally be *diverse*: as exemplified above, there are many possible reasons for the classification, and diverse CFs could cover much of these grounds. Furthermore, they should be *realistic*: an explanation stating, for a Manhattan condo, that having 10 bedrooms would push the price over 1M USD is not very useful, if there are no 10-bedroom condos in Manhattan.

Several concrete notions for capturing proximity (i.e. distance functions) have been proposed, and the problem of finding CFs that optimize them has been extensively investigated [15, 36, 42, 47]. This in particular includes standard measures such as $L0$ (number of features that have changed) or $L1$ (magnitude of change), where a common variant normalizes per-feature change magnitudes before aggregation. Likewise, the problem of finding diverse CFs has been extensively studied [15, 36].

By contrast, capturing realism is much more subtle, and here there is a wide range of approaches ranging from restricting CFs to be members of an explicitly given dataset (e.g. the training set) to learning a complex model of what is realistic (see Section 6 for further discussion). Recent work [4], in a different context of adversarial Machine Learning, has proposed to capture realism via *database constraints*, and specifically *Denial Constraints* (DCs) [10]. Originally introduced to capture data integrity, these logical constraints were shown in [4] to strike a good balance between allowing for inference and correctly capturing realism. To illustrate, Table 1 shows example DCs for 3 standard ML benchmarks. The first two are *unary* DCs: they capture constraints, in the form of negated conjunctions, over individual tuples/instances, e.g. that no individual younger than 17 holds a Doctorate, or that Manhattan apartments do not have 10 or more bedrooms. The third is an example of a *binary* DC (universally quantified over pairs of tuples, rather than a single tuple), reflecting in this case that tax child exemption laws are consistent for individuals in every state. To capture realism of individual instances (in our case, of CFs), we can think of such binary DCs as a compact representation of many unary ones: given an instance, it will be realistic only if it matches the child exemption status of individuals with the same characteristics in the same state. Using DCs as our basis for capturing realism also has the advantage that their inference/mining from datasets has been extensively studied, and out-of-the-box solutions such as FastADC [48] may be employed.

Our work focuses on *tabular* data, the primary application domain for CF explanations [24, 46] and the setting of all major CF methods. A separate line of work studies CFs for images [23, 25], where realism is captured via generative or natural-image priors; our constraint-based notion is specific to relational data, and extending it to unstructured domains is left to future work.

In this paper, we study for the first time the problem of optimizing distance and diversity of CFs *while constraining them to adhere to DCs*. This combination has not been studied before (see Section 6 for related work overview); in particular, state-of-the-art solutions for diverse CFs such as DiCE [36] are agnostic to constraints and indeed often yield CFs that violate them (as we demonstrate in Section 5).

Algorithmically, finding realistic, diverse, and close CFs requires addressing several layers of complexity. First, even without realism constraints, generating optimal CFs is intractable unless $P = NP$ for common model families such as Neural Networks and Random Forests [17]. Second, we show that projecting a tuple onto the space of realistic tuples (namely those consistent with a given set of DCs) is also intractable unless $P = NP$, even for unary DCs. Third, realism may, and often does, conflict with both the flip-of-label and with the diversity aims.

Our algorithmic solution is then a perturb-and-project framework that repeatedly iterates between *perturbation*, namely constraints agnostic CF generation (we incorporate DiCE [36], a state-of-the-art method for diverse CFs) and *projection* onto the space of DC-consistent tuples, which is essentially an optimization-under-constraints problem. A difficulty is that a straightforward use of optimizers such as SMT solvers [16] for the latter problem is prohibitively inefficient. This is due to two unique features in our setting: (1) we need to invoke the solver many times in a single execution of the CF-search solution. This is both because we look for multiple diverse CFs and because of the above mentioned conflict between realism and flip-of-label, requiring multiple iterations; (2) Binary DCs must be checked against each relevant tuple in the database, resulting in a large number of assertions for the solver to evaluate. We address this through several dedicated optimizations that we have developed: *pre-processing and caching* of solver instances; a dedicated constraint-pruning technique, inspired by works in data cleaning [12, 41], that identifies the relevant subset of database tuples (*suspect set*); and the incorporation of explicit *diversity-based constraints* to avoid redundant projections. The framework is *model-agnostic*: the projection step, our core contribution, treats the classifier as a black-box labeling oracle.

Table 1: Example Denial Constraints

Dataset	Example Denial Constraint
Adult	$\neg\{t_0.\text{age} < 17 \wedge t_0.\text{education} = \text{“Doctorate”}\}$
NY	$\neg\{t_0.\text{sublocality} = \text{“Manhattan”} \wedge t_0.\text{beds} \geq 10\}$
Tax	$\neg\{t_0.\text{State} = t_1.\text{State} \wedge t_0.\text{HasChild} = t_1.\text{HasChild} \wedge t_0.\text{ChildExemp} \neq t_1.\text{ChildExemp}\}$

We have conducted an extensive experimental study over four standard benchmarks (Adult, NY Housing, Tax, and Census). Our results show that 55.9–100% of the CFs produced by DiCE violate Denial Constraints, while our solutions achieve zero violations, namely all CFs that we generate are realistic. This realism comes at only a modest cost: proximity and diversity degrade by under 10% and ∼3% respectively on most datasets, compared to DiCE CFs. In terms of execution time, the constrained problem is naturally hard, and while a direct SMT-solver formulation is the natural exact approach, it does not scale with over 18,000 seconds for a single CF projection on Census. Our optimizations reduce that by up to 63× while preserving the same guarantees.

Summary of Main Contributions. Our main contributions are:

(1) We formalize the problem of generating diverse counterfactual explanations that adhere to Denial Constraints, combining for the first time the goals of proximity, diversity, and DC-based realism (Section 2).
(2) We develop a perturb-and-project framework with multiple solver optimizations, including constraint caching, suspect-set filtering, and diversity-aware projection (Sections 3–4).
(3) We show that the projection subproblem—finding the nearest DC-consistent tuple—is NP-hard (Section 4).
(4) We conduct an extensive experimental evaluation on four benchmarks, demonstrating that our approach eliminates constraint violations while maintaining proximity and diversity comparable to unconstrained baselines (Section 5).

2 Model

We next review basic notions of databases, constraints, and counterfactuals from previous work, leading to our problem statement.

ID	Type	Beds	Bath	Sqft	Subloc.
d_1	Condo	2	2	1400	Manhattan
d_2	Condo	3	2	704	Brooklyn
d_3	Condo	2	4	1568	Staten_Island
d_4	House	5	6	4357	NY

Denial Constraints:

DC1: $\forall t_1, t_2. \neg(t_1.\text{Type} = t_2.\text{Type} \wedge t_1.\text{Beds} > t_2.\text{Beds} \wedge t_1.\text{Bath} > t_2.\text{Bath} \wedge t_1.\text{Sqft} < t_2.\text{Sqft})$

DC2: $\forall t. \neg(t.\text{Subloc.} = \text{'Manhattan'} \wedge t.\text{Beds} > 4)$

DC3: $\forall t. \neg(t.\text{Subloc.} = \text{'Manhattan'} \wedge t.\text{Bath} > 4)$

Figure 1: Sample tuples and denial constraints from the NY Housing Dataset [20]. Locality attribute omitted for brevity.

2.1 Databases and Denial Constraints

A relational schema [2] consists of a *relation name* $\mathcal{R}$ along with a set of *attributes* $\{A_1, \ldots, A_n\}$, where A_j is associated with a domain $dom(A_j)$. A *tuple* t maps each attribute A_j to a value $t.A_j \in dom(A_j)$. A schema in general may include multiple relations, but we focus here on the single relation case.

Integrity Constraints [2] are often imposed to capture data soundness. We focus here on a particularly expressive formalism for describing such constraints, namely Denial Constraints (DCs) [10, 41]. DCs are universally quantified first order logic over tuples over a particular relation, reflecting constraints over tuples that may (co-)occur. Specifically, a *binary DC* over $sch(\mathcal{R})$ is a first-order formula: $\sigma = \forall t_1, t_2 \in \mathcal{R}\ \neg(P_1 \wedge \ldots \wedge P_m)$, where each P_i is a predicate of the form $t_1.A \circ t_2.A$ or $t_1.A \circ a$ such that (1) $A \in sch(\mathcal{R})$, and (2) $a \in dom(A)$, and (3) $\circ \in \{=, \neq, >, <, \geq, \leq\}$. Similarly, an *unary DC* has the form $\sigma = \forall t \in \mathcal{R}\ \neg(P_1 \wedge \ldots \wedge P_m)$ where each P_i has the form $t.A \circ a$ with a and $\circ$ as before. For a given binary DC $dc = \forall t_1, t_2 \in \mathcal{R}\ \neg(P_1 \wedge \ldots \wedge P_m)$ and a pair of tuples t, t' we use $dc[t, t']$ to denote the instantiation of $\neg(P_1 \wedge \ldots \wedge P_m)$ with values from t and t', and we then say $t, t' \models dc$ if $dc[t, t']$ evaluates to *true* and $t, t' \not\models dc$ otherwise. We similarly define $t \models dc$ and $t \not\models dc$ for unary DCs. We overload notation for sets of DCs (all must hold) and for relations (every tuple/pair must satisfy the DCs).

Example 2.1. The NY housing dataset [20] includes information on properties in New York and is commonly used as a benchmark in ML research, where the prediction goal is to decide whether a property is valued at over $\$1M$. Figure 1 shows the dataset schema along with 4 sample tuples. It also shows three example DCs with respect to it: DC1 is a binary DC intuitively stating that for properties of the same type, having more bedrooms *and* more bathrooms implies at least as much square footage. DC2 and DC3 are unary DCs reflecting Manhattan's physical space constraints, namely properties located there do not have more than 4 bedrooms or bathrooms. Indeed, every pair of tuples shown in Figure 1 satisfies DC1, and every individual tuple satisfies DC2 and DC3. We will later (Example 2.4) show examples of cases where DCs are not satisfied.

2.2 Counterfactual Explanations

Given a classifier M and a tuple t with label $M(t)$, a counterfactual (CF) is a tuple cf such that $M(cf) \neq M(t)$ [47]. Intuitively, a CF identifies (preferably small) changes that flip the model's prediction, revealing influential features. Beyond flipping classification, CFs should satisfy proximity (minimal distance), diversity (varied explanations), and realism [15, 36]. We next overview specific measures/notions to capture each desideratum.

Table 2: Counterfactuals for tuple t. Changed values shown with arrows. p_{1-3} are generated by DiCE perturbation; Ours$_{1-3}$ are our realistic results after projection.

ID	Type	Beds	Bath	Sqft	Subloc.	Violation
t	Condo	1	1	679	Manhattan	–
Ours$_1$	Condo	1	→3	→1568	Manhattan	–
Ours$_2$	Condo	→4	1	→2365	Manhattan	–
Ours$_3$	Condo	→4	→2	→3075	Manhattan	–
p_1	Condo	1	1	→1750	Manhattan	DC1
p_2	Condo	→4	1	→2365	Manhattan	–
p_3	Condo	→6	→3	→4103	Manhattan	DC1, DC2

Proximity (Distance Functions). To capture proximity, one needs to define a distance function over tuples. Following common practice [15, 36, 47], we separately quantify distance for categorical and numeric attributes and then discuss their combination. For categorical attributes, the common practice is to count the number of changes, namely to use L_0:

$$dist_{cat}(t, t') = \sum_{A \in attr_{cat}} \mathbb{1}(t.A \neq t'.A) \tag{1}$$

For numeric attributes, by contrast, the magnitude of change should be accounted for. A common way to normalize is based on the Median Absolute Deviation (MAD) [36, 47], which resembles standard deviation but is considered more robust [33]:

$$MAD(A, \mathcal{R}) = \text{median}(\{|t.A - \text{median}(\{t.A : t \in \mathcal{R}\})| \mid t \in \mathcal{R}\}) \tag{2}$$

The numeric distance is then defined as:

$$dist_{num}(t, t') = \sum_{A \in attr_{num}} \frac{|t.A - t'.A|}{MAD(A, \mathcal{R})} \tag{3}$$

Example 2.2. Consider tuple t from Table 2 and tuple d_4 from Figure 1. First, $dist_{cat}(t, d_4) = 2$ since both Type (Condo vs. House) and Sublocality (Manhattan vs. NY) differ. To compute numeric distance, we first compute the MAD values (over the dataset in Figure 1) to be used to normalize. We have MAD(Beds) = 1.0, MAD(Bath) = 1.0, and MAD(Sqft) = 608.5; the latter indicates that square feet varies a lot. We then have:

$$dist_{num}(t, d_4) = \frac{|1-5|}{1.0} + \frac{|1-6|}{1.0} + \frac{|679-4357|}{608.5} = 4 + 5 + 6.04 = 15.04$$

The numeric and categorical distance may then be aggregated in different ways to form an overall distance function over tuples. A common choice is to simply use their (possibly weighted) sum as done e.g. in the implementation of [36]:

$$dist_{agg}(t, t') = dist_{cat}(t, t') + dist_{num}(t, t') \tag{4}$$

We will also need to use a distance between a set of tuples and a given tuple, and for that we define $dist_{agg}(T, t') = \frac{\sum_{t \in T} dist_{agg}(t, t')}{|T|}$, overloading notation.

Our framework is robust to the choice of distance function and we discuss alternatives in Appendix C.5.

Diversity. We aim to find multiple CFs that are diverse, thereby covering different reasons for the classification [47]. A common diversity measure is that of *determinantal point process (DPP)* diversity [36]. Let $C = \{t'_1, \ldots, t'_k\}$ be a set of CFs, and let K be a matrix with entries $K_{i,j} = \frac{1}{1+dist_{agg}(t'_i, t'_j)}$. We define:

$$div(C) = det(K) \tag{5}$$

Intuitively, when CFs are very similar to each other, namely $dist_{agg}(t'_i, t'_j)$ values are typically small for $i \neq j$, the off-diagonal entries $K_{i,j}$ are close to 1 and the matrix has low determinant. Conversely, when they are very diverse (namely large distances), off-diagonal entries approach 0, and K approaches the identity matrix with $det(K) \to 1$ [32].

Example 2.3. Consider two sets of tuples corresponding to the NY-housing schema:

$C_1 = \{$(Condo, 2, 2, 1300, Queens), (Condo, 3, 2, 1200, Queens)$\}$

$C_2 = \{$(Condo, 1, 1, 679, Manhattan), (House, 3, 2, 1824, Brooklyn)$\}$

In C_1, tuples have the same values in the type, bath, and sublocality attributes, differing only slightly in the beds and sqft attributes. Conversely, in C_2, tuples differ across all attributes. Indeed, $div(C_1) = 0.786 < div(C_2) = 0.983$.

Scoring function. Since distance and diversity are often negatively correlated, a common practice [15, 36] is to balance them via a scoring function. Let $w_1, w_2 \geq 0$ with $w_1 + w_2 = 1$ be weight parameters. Given a tuple t and a set C of CFs for it (w.r.t. a given model) we define:

$$score_t(C) = w_1 \cdot div(C) - w_2 \cdot dist_{agg}(C, t) \tag{6}$$

Constraints. We impose two kinds of hard constraints on CFs. The first is that the CFs do not modify particular (pre-defined) attributes called *immutable attributes* [45, 46], thereby focusing the search on desirable attribute modifications. The second type of constraints captures realism: the resulting CF instance should be consistent with a given database and a given set of DCs, namely that its inclusion in the database would not violate any DCs.

Example 2.4. Consider a classifier M predicting whether a property exceeds \$1M, and the tuple t from Table 2. Assume that $M(t) = 0$, namely M assigns label 0 to the corresponding instance. Further assume that p_1, p_2, p_3 in Table 2 capture 3 CFs w.r.t. M and t, namely $M(p_i) = 1 \neq M(t)$. In terms of constraints, we impose the following: (1) sublocality is immutable, reflecting that we wish to examine the effect of other attributes while keeping the neighborhood intact; (2) the resulting CFs should be consistent with DC1 and DC2 with respect to the dataset in Figure 1. The CF p_1, for instance, fails to adhere to these constraints: it is inconsistent with both d_1 and d_2 with respect to $DC1$; intuitively, there is a mismatch between its number of rooms and square footage information. p_3 is invalid in a similar way based on $DC1$, and another reason for its invalidity is captured by $DC2$: it corresponds to a property with 6 bedrooms whereas $DC2$ captures a constraint that no property may have more than 4 bedrooms.

Problem Statement. We next "put it all together" to obtain the problem statement to be studied in the sequel. For ease of presentation, we focus on binary classification but the construction is generalizable to the multi-class setting where one defines CFs with respect to a target label. We then define the problem as follows:

Definition 2.5 (Diverse Realistic Counterfactuals). Given a relation $\mathcal{R}$ over a schema $\mathcal{S} = \{A_1, \ldots, A_m\}$, a classifier M, a set of DCs Σ such that $\mathcal{R} \models \Sigma$, a subset $\mathcal{I} \subseteq \mathcal{S}$ of *immutable attributes*, a natural number k, and a tuple $t \in \mathcal{R}$, we aim to find a set C of size k that is a solution to:

$$\begin{aligned} \max_{C} \quad & score_t(C) \\ \text{s.t.} \quad & \forall cf \in C : \; cf \in dom(A_1) \times \ldots \times dom(A_m), \; M(cf) \neq M(t) \\ & \forall cf \in C : \; \mathcal{R} \cup \{cf\} \models \Sigma, \; \forall A \in \mathcal{I} : \; cf.A = t.A \end{aligned}$$

Intuitively, we aim to find a set of k CFs for a specific tuple t such that each one is within the domain of the schema attributes, it has identical immutable attributes to t, and when added to the database, each CF does not cause DC violations. This set of CFs should maximize the score function across all sets that adhere to these criteria.

Example 2.6. Consider the set $C = \{\text{Ours}_{1-3}\}$ from Table 2. Each CF satisfies the constraints: immutable attributes are preserved (Sublocality remains Manhattan), and $\mathcal{R} \cup \{cf\} \models DC_{1-2}$ for each $cf \in C$. With $dist_{agg}(C, t) = 5.72$, $div(C) = 0.88$, and weights $w_1 = \frac{2}{3}$, $w_2 = \frac{1}{3}$ (a standard choice following [36]), the overall score is $score_t(C) = \frac{2}{3} \cdot 0.88 - \frac{1}{3} \cdot 5.72 = -1.32$.

3 Generating Realistic Counterfactuals

Algorithm 1 presents our framework for finding realistic and diverse CFs. We maintain the collection of CFs found so far (Line 1) and a candidate queue (Line 2). We iterate (Line 3) until sufficiently many CFs are found. In each iteration, we pop a candidate (Line 4) and invoke PERTURB (Line 5) to generate k CFs consistent with immutable attributes, ignoring DCs for now. As such, the resulting CFs may violate DCs. For that, we invoke a PROJECT function (Line 7, details below) whose goal is to find a tuple p' in the proximity of each CF p. The resulting p' may or may not be a CF. If it is (Line 9), we add it to the collection of realistic CFs; and otherwise (Line 11) we add it to the queue for the next iteration. This process may result in more than k CFs and the final step is to greedily choose k CFs out of them (Line 15, details below). Figure 2 includes a schematic illustration of this iterative process. We next detail the implementation of PERTURB, PROJECT, and CHOOSEK-DIVERSE. PERTURB is fairly standard and CHOOSEK-DIVERSE is fairly simple, and we describe both here. By contrast, PROJECT requires new development and is described at a high-level here and in more detail in the next section.

Perturbation Phase (Line 5). The PERTURB function is assigned with outputting k diverse CFs in the proximity of a given tuple, which are further consistent with the immutable attributes (it is agnostic to DCs). This sub-problem has been studied in the literature and shown to be NP-hard for models such as Neural Networks and Random Forests (see e.g. [17]). On the other hand, effective heuristic solutions have been developed, and a particularly notable such

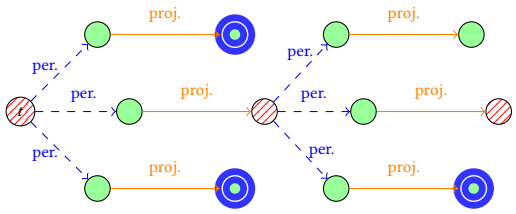


Figure 2: Perturb-and-project for CF generation. Hatched nodes: label 0; solid green: label 1; bordered: final realistic CFs. Dashed blue edges: perturbation; solid orange: projection.

Algorithm 1: Perturb-and-Project for Realistic CFs

input : Original tuple t, number of CFs k, classifier M, relation $\mathcal{R}$, set of DCs Σ, immutable attributes set $\mathcal{I}$
output : Set C of k realistic counterfactuals

1 $C \leftarrow \emptyset$
2 $L \leftarrow [t]$ `// FIFO: pop front, append back`
3 **while** $|C| < k$ *and* $L \neq \emptyset$ **do**
4 $t' \leftarrow L$.popFirst()
5 $P \leftarrow \textsc{Perturb}(t', M, k, \mathcal{I})$
6 **for** *each* $p \in P$ **do**
7 $p' \leftarrow \textsc{Project}(p, \mathcal{R}, \Sigma, \mathcal{I})$
8 **if** $M(p') = 1$ **then**
9 $C \leftarrow C \cup \{p'\}$
10 **else**
11 L.append(p')
12 **end**
13 **end**
14 **end**
15 **return** $\textsc{ChooseK-Diverse}(C, t, k)$

solution is DiCE [36]. DiCE finds CFs for Neural Networks based on backpropagation, jointly optimizing for classification confidence, distance from the input, and diversity. Other solutions are discussed in Section 6.

Example 3.1 (Perturbation Phase). We have trained a Neural Network model M over the NY housing dataset and it classifies the tuple t from Table 2 to 0 (predicted price below $1M). Invoking DiCE over t and asking for $k = 3$ CFs, we obtain p_1, p_2, and p_3 from Table 2. They are CFs for t, namely accepted by the model, yet as mentioned above, two of them violate DCs.

Projection Phase (Line 7). The main novelty of our solution with respect to DiCE lies in the incorporation of the realism requirement, captured through DCs. To this end, the `PROJECT` function is geared towards projecting its input onto the space of realistic tuples, namely, to find a tuple in the proximity of the input tuple, that further satisfies the constraints. We provide details on our implementation of `PROJECT` in Section 4, and for now just illustrate its input and output.

Example 3.2 (Projection Phase). Continuing Example 3.1 and referring to tuples from Table 2, p_1 violates DC1 with respect to d_1 and d_2 from Figure 1, as well as with other tuples in the dataset not shown here for brevity. Intuitively, for a (1,1,Sqft) condo to satisfy DC1, its Sqft must be at most the smallest Sqft among all condos with beds>1 and baths>1. This quantity is 704 and p_1 is projected to obtain $(1, 1, 704)$. This however flips the label back to 0, leading to another iteration of perturb-and-project (see Example 3.3). By contrast, p_2 and p_3 are projected to Ours_2 and Ours_3 respectively, thereby satisfying the DCs while, in this case, maintaining the "accept" label.

Greedy Selection (Line 15). The iterative process may yield more than k valid candidates, since each DiCE invocation produces k new branches (see Algorithm 1 for illustration). The final phase is then to select a subset of size k out of the obtained candidates set. This is done in a greedy fashion: we repeatedly choose the candidate whose inclusion in the set maximizes the overall score (with respect to the score function defined above, aggregating proximity and diversity scores) among all such choices.

Example 3.3 (Complete Pipeline). Continuing Example 3.2, our candidates set $C = \{Ours_2, Ours_3\}$ and the queue $L = \{(1, 1, 704)\}$. This single tuple occupying L is popped and fed as input to DiCE, which yields 3 new candidates. Each of them is again projected onto the space of valid tuples, and two of the resulting projections are accepted by the model, namely Ours_1=(1,3,1568) and a fourth candidate (4,1,2393). With four realistic CFs, greedy selection operates by proximity and diversity, choosing the best 3 which are $\{Ours_1, Ours_2, Ours_3\}$.

4 Projecting over Denial Constraints

The algorithm in Section 3 invokes three functions as subroutines, for perturbation, projection, and greedy selection. Implementations for perturbation and greedy selection were described in the previous section, and here we detail the projection algorithm. Specifically, we show that the problem is NP-hard; present a solution based on SMT solvers [16]; and provide multiple crucial optimizations.

Problem Statement and Intractability. Given a tuple t to project, a set Σ of DCs and a relation $\mathcal{R}$, the projection problem is:

$$\min_{t'} \quad dist_{agg}(t, t') \quad \text{s.t.} \quad \mathcal{R} \cup \{t'\} \models \Sigma, \quad \forall A \in \mathcal{I} : t'.A = t.A \tag{7}$$

We show that, unless P=NP, there is no hope for an exact PTIME solution, since even checking for satisfiability of DCs is already intractable. For that, we define `DEC-PROJ` as the problem of deciding, given Σ and $\mathcal{R}$ as above, whether there exists a tuple t' such that $\mathcal{R} \cup \{t'\} \models \Sigma$. We show (proof in Appendix A):

Proposition 4.1. `DEC-PROJ` *is NP-hard, even if* Σ *contains only unary DCs.*

We consequently focus on solver-based solutions that do not guarantee PTIME convergence but perform well in practice.

"Vanilla" Solver-Based Solution. Algorithm 2 presents a baseline solution for the projection problem based on an SMT solver. We first check for infeasibility (Lines 1–2): if t violates a DC involving only immutable attributes, it cannot be fixed and we return $\perp$ to signal this. Otherwise, the solver is initialized with variables corresponding to the schema (Lines 4–5). Then, for each DC $\sigma \in \Sigma$

Algorithm 2: Vanilla Projection

input : t: tuple to project, $\mathcal{R}$: relation, Σ: DCs, $\mathcal{I}$: immutable attributes
output: t': projected tuple, or $\perp$ if infeasible

// Check for immutable violations
1 **for** *each* $\sigma \in \Sigma$ **do**
2 | **if** $attr(\sigma) \subseteq \mathcal{I}$ ***and*** $\exists t_i \in \mathcal{R} : (t, t_i) \not\models \sigma$ **then return** $\perp$
3 **end**
4 $opt \leftarrow$ CreateOptimizer()
5 $vars \leftarrow$ CreateVariables($sch(\mathcal{R})$)
// Add DC satisfaction constraints
6 **for** *each* $\sigma \in \Sigma$, *each* $t_i \in \mathcal{R}$ **do**
7 | $bound \leftarrow$ InstantiateDC(σ, t_i)
8 | opt.add($\neg \bigwedge_{(A,v,op) \in bound}(vars[A] \circ_{op} v)$)
9 **end**
// Fix immutable attributes
10 **for** *each* $A \in \mathcal{I}$ **do**
11 | opt.add($vars[A] = t.A$)
12 **end**
// Set objective and solve
13 opt.minimize($dist_{agg}(vars, t)$)
14 **if** $opt.solve() = SAT$ **then return** opt.getModel($vars$)
15 **return** $\perp$

and each tuple $t_i \in \mathcal{R}$, we instantiate σ with values from t_i to encode "forbidden regions" that the projected tuple must avoid (Lines 6–8). We then create a boolean expression whose form is the negation of the conjunction of these forbidden regions. We further (Lines 10–11) add boolean constraints capturing that immutable attributes must be fixed to their values in t, set the objective to distance minimization and run the solver, returning the solution it found or $\perp$ if none was found.

Example 4.2. Consider projecting tuple p_1 from Table 2 with immutable attributes $\mathcal{I} = \{\text{Type}, \text{Subloc}\}$. For DC1 and tuple d_1 from Figure 1, instantiation yields the bound $\{(\text{Type}, \text{Condo}, =), (\text{Beds}, 2, <), (\text{Bath}, 2, <), (\text{Sqft}, 1400, >)\}$. Similar bounds are generated for all $|\mathcal{R}| \cdot |\Sigma|$ tuple-DC pairs, and we conjoin over all of them.

4.1 Optimizations and Variations

There are multiple problems slowing down Algorithm 2 and specifically its use in our perturb-and-project framework, as follows:

- The framework involves multiple iterations with many `project` invocations, with the same DCs but with different choices of tuple t to project. This means that solver initialization and DCs instantiations are repeated across these invocations.
- The number of DC instantiations is $O(|\mathcal{R}| \cdot |\Sigma|)$, one for each tuple in $\mathcal{R}$ and DC in Σ, which may be very large.
- Multiple projections of different tuples may yield the same or similar solutions, which is problematic both in terms of effective search space exploration and in terms of diversity of the final result.

Algorithm 3: Pre-processing for Optimizer Construction

input : $\mathcal{R}$: relation, Σ: DCs, $\mathcal{I}$: immutable attributes, *cache*: global optimizer cache
output: $(opt, vars)$: configured optimizer and variables

1 **if** $cache[\mathcal{I}]$ *is empty* **then**
2 | $opt \leftarrow$ CreateOptimizer()
3 | $vars \leftarrow$ CreateVariables($sch(\mathcal{R})$)
4 | **for** *each* $\sigma \in \Sigma$, *each* $t_i \in \mathcal{R}$ **do**
5 | | $bound \leftarrow$ InstantiateDC(σ, t_i)
6 | | opt.add($\neg \bigwedge_{(A,v,op) \in bound}(vars[A] \circ_{op} v)$)
7 | **end**
8 | $cache[\mathcal{I}] \leftarrow (opt, vars)$
9 **end**
10 **return** $cache[\mathcal{I}]$

We next present an optimization or variant accounting for each of these points.

Pre-processing with Optimizer Caching. To account for the repeated initialization step, Algorithm 3 constructs and caches optimizer instances keyed by the immutable attribute set $\mathcal{I}$. The key observation is that DC satisfaction constraints (Lines 6–8) depend only on $\mathcal{R}$ and Σ, not on the specific tuple being projected. Preprocessing thus generates all $O(|\mathcal{R}| \cdot |\Sigma|)$ constraints upfront; once cached, projecting any tuple requires only adding immutable constraints (Lines 10–11) and the distance objective (Line 13).

Suspect Set Filtering. Pre-processing trades upfront cost for universal reuse. But what if we only need CFs for a few tuples, or many tuples share the same immutable attribute values? The next optimization follows the opposite approach: rather than pre-processing all constraints, it builds an optimizer on-demand containing only constraints relevant to the specific tuple's immutable values. This approach is inspired by suspect set techniques used in the different context of constraint-based data cleaning [12].

Definition 4.3. Given tuple t, immutable attributes $\mathcal{I}$, relation $\mathcal{R}$, and DCs set Σ, let $\sigma_{\mathcal{I}}$ denote predicates in σ involving only attributes in $\mathcal{I}$. A tuple $t' \in \mathcal{R}$ is a *suspect* w.r.t. $t, \mathcal{I}, \sigma$ if t, t' satisfy all predicates in $\sigma_{\mathcal{I}}$. Namely:

$$suspect(t, \mathcal{R}, \sigma, \mathcal{I}) = \{t' \in \mathcal{R} \mid \forall_{p \in \sigma_{\mathcal{I}}}(t, t') \models p\}$$

The key insight is that if t and t' disagree on an immutable predicate, no modification to mutable attributes can cause a violation and we thus need no constraints corresponding to t'. To implement this optimization, we modify Lines 6–8 of Algorithm 2: instead of iterating over all tuples in $\mathcal{R}$, we iterate only over suspect tuples, potentially reducing the number of constraints.

Example 4.4. Consider projecting p_1 from Table 2 with $\mathcal{I} = \{\text{Type}, \text{Subloc}\}$. For DC1, which compares properties of the same type, suspects must match p_1.Type = Condo. From Figure 1, only $\{d_1, d_2, d_3\}$ are condos, so we can avoid generating a constraint corresponding to d_4.

In a sense, the two optimizations we have seen thus far are mutually exclusive: the first builds many constraints but does it at pre-processing time, namely before getting t as input; the second

reduces the number of constraints by focusing on instantiations relevant to t, yet this can of course be done only when we have t in hand. We can nevertheless combine these two ideas as follows: suspect-set filtering depends only on immutable attribute values, and we thus cache the optimizer's constraints based on these values, and thereby only compute new instantiations when these values change. In particular, this allows reuse within a tuple's perturb-and-project loop (Algorithm 1) since immutable attributes never change.

Diversity Constraints. A third optimization, which is orthogonal to the first two, relates to the avoidance of generating similar tuples in the perturb-and-project process. To this end, given the set of previously found projections $\mathcal{P} = \{p_1, \ldots, p_{k-1}\}$ and a threshold γ, we add to the optimizer before Line 13:

$$\forall p \in \mathcal{P} : |\{A \notin \mathcal{I} : |t'.A - p.A| > MAD(A, \mathcal{R})\}| \geq \gamma$$

This requires each new projection to differ from every previous one by more than one MAD unit in at least γ mutable attributes.

5 Experiments

We present an experimental evaluation of our solutions. Our main findings, across multiple datasets, are as follows:

- **Realism**: Our solutions achieve realism (zero constraint violations), whereas *55.9–100% of the CFs produced by DiCE violate at least one constraint*, and some violate many constraints (Table 4).
- **Distance and Diversity**: *we are able to achieve comparable distance and diversity of CFs to that achieved by DiCE*, while also achieving realism (under 10% proximity and ∼3% diversity difference on most datasets, see Figure 3 and Table 5). Our diversity optimization significantly improves diversity (Table 5).
- **Execution Time**: *Our optimizations for projection are crucial and outperform the vanilla use of solvers by up to 63×* (Figure 4). Compared to DiCE, our methods incur 1.1–10.5× overhead, as a cost of realism.

Additional experiments (Figure 5) show that our solver-based projection fares well compared to optimal results (when available) and alternatives such as closest-instance retrieval. The appendix also evaluates alternative distance functions (Appendix C.5).

5.1 Experimental Setup

We evaluate on four datasets: Adult-Income [31], a demographic dataset for income prediction; NY-Housing [20], containing New York real estate listings for price classification; Tax [48], a synthetic dataset common in DC research; and Census-Income [30], an extended demographic dataset. Characteristics are summarized in Appendix B. For each dataset, we designate a set of immutable attributes, as we require specifying which attributes cannot be modified; choices per dataset are detailed in Appendix B. Binary DCs are mined offline using FastADC [48]; for Adult and NY, we additionally craft unary constraints based on domain knowledge (see Appendix B.2). We generate $k = 5$ CFs per tuple using weights $w_1 = \frac{2}{3}$ and $w_2 = \frac{1}{3}$ (Equation 6), following DiCE [36]. We use Z3 [16] as the SMT solver. Results are averaged over 10 tuples per dataset across 10 runs. Beyond DiCE [36], we compare against Vanilla (unoptimized SMT projection), Exhaustive (brute-force optimal), and Best-in-Dataset (closest dataset tuple respecting immutable attributes, covering retrieval methods such as NICE [9] and FACE [40]), as well as HoloClean [41] (Appendix C.4) and the plausibility-aware methods of CARLA [38] (Table 3). We omit causal-recourse methods, which require unavailable structural causal models. Implementation and hardware details appear in Appendix B.3.

5.2 Main Results

Realism. Table 4 quantifies constraint violations. We report the average number of violated DCs per CF (Avg. Viol.), unary constraint violations (Unary Viol.), the number of database tuples conflicting with each CF via binary DCs (Tuple Conf.), an inconsistency measure adapted from [34] and the percentage of CFs violating at least one constraint (Unreal. %). *Our solutions yield zero violations, while DiCE produces significant violations*: 100% of CFs outputted by DiCE are unrealistic for Adult and Tax, 97.1% for NY, and 55.9% for Census. These CFs in fact conflict with many tuples: up to 2,300 tuple conflicts per CF.

Plausibility-Aware Baselines. We ask whether methods that already model the data distribution produce DC-consistent CFs. We evaluate CARLA's [38] plausibility-aware methods on Adult: CEM [18], CCHVAE [39], CRUDS [19], and FeatureTweak [44] (GrowingSpheres and CLUE produced no CF). Among domain-valid CFs (with correct one-hot categories and typing), every method, including the VAE-based CCHVAE and CRUDS, violates at least one DC in 100% of cases, with 3.5 to 4.1 violations on average (Table 3).

Table 3: CARLA plausibility-aware methods on Adult. Even generative methods that model the data distribution violate DCs.

Method	Type	CFs Found	Mean Viol.	Viol. %
CEM [18]	gradient	9/10	4.11	100
CCHVAE [39]	VAE	10/10	3.90	100
CRUDS [19]	CSVAE	8/10	3.50	100
FeatureTweak [44]	tree	3/10	3.67	100
Ours	DCs	10/10	0.00	0

Proximity. Figure 3 compares the proximity achieved by DiCE and by our solutions. Our method, Ours, combines perturb-and-project (Algorithm 1) with the projection optimizations of Section 4, including diversity constraints. We also evaluate Ours-Div, a variant without diversity constraints optimization, discussed further in the ablation study. We report two metrics: the combined distance function $dist_{agg}$ (Equation 4), which is the objective both DiCE and our method directly optimize, and L0 distance, counting modified attributes (categorical and numerical). Our methods achieve comparable proximity despite enforcing realism: $dist_{agg}$ increases under 10% on Adult and Tax, up to ∼31% on Census (9% without diversity constraints), with similar L0 trends. On NY, we achieve 36–45% lower $dist_{agg}$ as DiCE overshoots into invalid regions. These gaps to DiCE are statistically significant (paired Wilcoxon, $p < 0.001$) but small, reflecting a consistent, modest cost of realism.

Diversity. We measure diversity using the Determinantal Point Process (DPP, Equation 5). Table 5 shows that Ours achieves DPP scores comparable to DiCE: only 1.7% lower for Adult, 0.6% lower

Table 4: Constraint violations for DiCE. Our solutions yield zero violations.

Dataset	Avg. Viol.	Unary Viol.	Tuple Conf.	Unreal. %
Adult	4.16±0.38	0.82±0.39	778.58±247.03	100.0
NY	0.71±0.49	0.33±0.45	22.11±27.04	97.1
Tax	4.16±1.09	0.00±0.00	2,326.85±442.89	100.0
Census	22.18±15.14	0.00±0.00	722.16±1,101.19	55.9

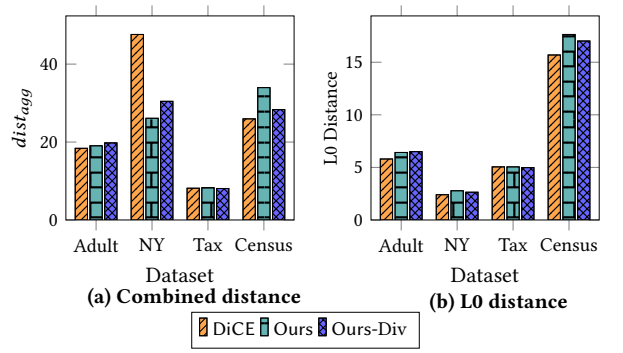


Figure 3: CF proximity metrics. Our methods achieve comparable proximity while guaranteeing realism.

for Tax, and identical for Census. For NY, DiCE achieves higher diversity by 7%, intuitively because the constraints in this dataset limit realistic CF spread. Additional diversity metrics are reported in Appendix C.2. Diversity differences are likewise significant ($p < 0.001$) yet within a few percent.

Table 5: CFs DPP diversity scores [32].

Method	Adult	NY	Tax	Census
DiCE	0.976±0.006	0.979±0.010	0.886±0.015	0.989±0.002
Ours	0.959±0.010	0.913±0.049	0.881±0.014	0.989±0.002
Ours-Div	0.945±0.033	0.714±0.353	0.870±0.027	0.901±0.285

Execution Time. Figure 4(a) shows CF generation times. DiCE is fastest but produces unrealistic CFs. P&P(Vanilla) is our perturb-and-project framework using the vanilla projection of Algorithm 2, rebuilding the solver per projection, incurring prohibitive overhead, e.g., 18,871s on Census. Our optimized variants, Ours(PreProc) and Ours(Suspect), dramatically outperform P&P(Vanilla): up to 63× faster on Census (301s and 334s vs. 18,871s). Compared to DiCE, both variants add moderate overhead as the cost of realism: for Ours(PreProc), 2.9× on NY, 10.5× on Adult, and 6.9× on Census (Ours(Suspect) exhibits similar ratios).

The two optimizations offer complementary tradeoffs. PreProc pre-builds all DC instantiations upfront (Figure 4(b)), with a one-time cost of 1.8s (NY) to 6,168s (Census), after which projections reuse the cached solver. Suspect, by contrast, builds constraints on-demand by filtering to tuples relevant to the specific immutable attribute values (Section 4), incurring no upfront cost. On Census, where the full constraint set contains over 7M instantiations, Suspect reduces this by 96.5% through filtering, achieving comparable runtime (334s vs. 301s) without the heavy preprocessing step. PreProc is preferable when generating CFs for many tuples, amortizing its upfront cost; Suspect is preferable when CFs are needed for few tuples or when preprocessing is impractical. A detailed comparison of bounds-building costs and constraint counts is provided in Appendix C.3. We further study how runtime scales with dataset size and number of constraints in Appendix C.1.

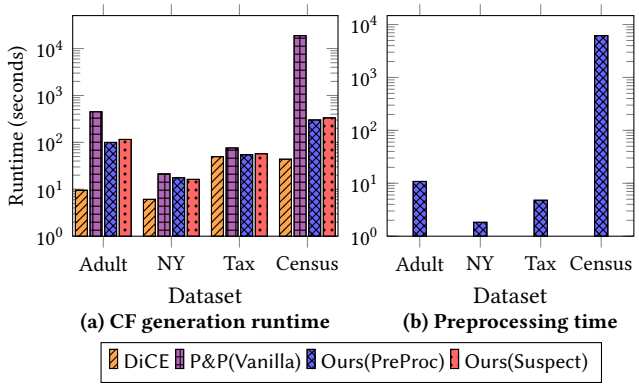


Figure 4: (a) Ours guarantees realistic CFs with up to 63× speedup over Vanilla. (b) PreProc preprocessing runtime cost.

5.3 Ablation

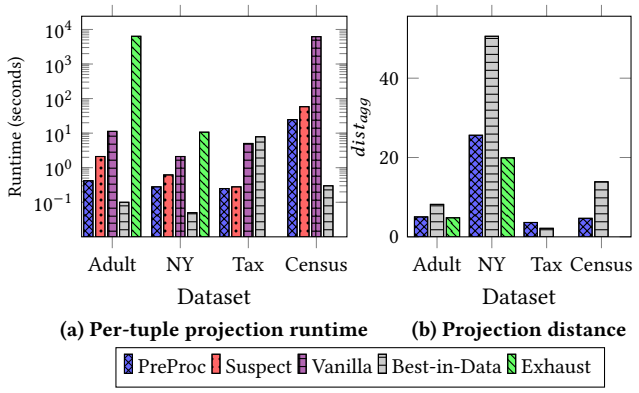


Figure 5: (a) Per-tuple runtime: PreProc is fastest after preprocessing; Exhaustive feasible only for Adult and NY. (b) Projection distance: PreProc achieves near-optimal quality.

Diversity Optimization. Table 5 demonstrates the effect of our diversity constraints, with $\gamma = 2$. Ours consistently improves diversity over Ours-Div, most notably on NY, where the DPP score increases from 0.714 to 0.913. Without diversity constraints, the tight feasible region in NY leads to clustered projections. With diversity constraints, Ours achieves DPP scores comparable to DiCE: within 1.7% on Adult, 0.6% on Tax, and near identical on Census. In terms of proximity, Ours remains comparable to both Ours-Div and DiCE (Figure 3), with less than 4% difference. The diversity optimization incurs negligible runtime overhead (omitted from Figure 5).

Alternative Projection Methods. Figure 5(a) evaluates projection strategies in isolation. Using a solver without our optimizations (Vanilla) is prohibitively slow: 8–27× slower than PreProc on Adult, NY, and Tax, and 250× slower on Census. We also evaluate non-solver alternatives: Exhaustive (enumerating the discretized valid space for optimal projections) and Best-in-Dataset (returning the closest dataset tuple matching immutable attributes). Exhaustive is impractical, exceeding 100 minutes per projection on Adult and 12-hour limits on Tax and Census. Best-in-Dataset has 10–20% failure rates when immutable attributes are present, and among successful projections, 63–197% higher distance on Adult, NY, and Census (Figure 5(b)). On Tax, where no attributes are immutable, Best-in-Dataset achieves lower distance but is significantly slower. We further evaluated HoloClean [41], a probabilistic data cleaning framework; it leaves 89% (Adult) and 100% (NY) of violations unresolved (Appendix C.4), confirming general-purpose cleaning is unsuitable for targeted projection.

6 Related Work

CF generation has been extensively studied in Explainable AI [24, 26, 46, 47]. A related but distinct concept, *algorithmic recourse* [28, 45], specifies *how* to reach a different prediction through actionable interventions, whereas CF explanations identify *what* features would need to differ. Our work focuses on CF explanations, finding realistic alternative instances, rather than prescribing action sequences.

Most lines of research [6, 15] in this context, such as DiCE which we have discussed above, do not address the realism of resulting CFs. Our experiments with DiCE demonstrate that realism is unlikely to be achieved spuriously if one does not optimize for it. Other approaches do aim at realism, there is no single notion of realism, and they capture different, partially overlapping facets of it. Some approaches [9, 27, 40] limit attention to instances in a given dataset, e.g., the training set. As we have demonstrated (see comparisons with best-in-dataset in Section 5), this may hamper quality as the dataset may simply not include sufficiently many, close, or diverse CF candidates. There is also a privacy concern: explaining a prediction made for Bob through the classification of Alice may infringe Alice's right to privacy. A further family of methods captures realism statistically, as closeness to a learned data distribution, e.g. via variational autoencoders or prototypes, as in the methods of the CARLA library [18, 19, 38, 39, 44]. To generalize beyond the existing dataset, causal approaches [28, 29, 35] assume either full structural causal models (SCMs), or a simple causal graph. Both are rarely available in practice and require significant domain knowledge. Crucially, none of these approaches *guarantees* a valid CF: each optimizes closeness to the data, yet a statistically plausible instance may still violate the integrity rules the data obeys, as we confirm experimentally. By contrast, Denial Constraints that we use here (1) may be automatically mined (as we did in our experiments using FastADC [48]) and (2) were shown in [4], in the different context of adversarial training, to capture well the space of valid instances. Indeed adversarial examples somewhat resemble CFs, yet they seek a single example, with no repeated constraint solving (thus no optimization) and no notion of diversity. The only other works, to our knowledge, that incorporate database-style constraints for CF explanations are GeCo [42] and CeC [17]. Both CeC and GeCo support unary but not binary DCs, which significantly hampers expressiveness, and again do not address diversity.

Beyond CFs, constraints [2] are widely used to capture data integrity, and various formalisms have been proposed to capture constraints. These include functional dependencies [13], conditional functional dependencies [8, 21], and others. DCs generalize these and other formalisms [7, 11]. Among them, DCs are attractive on several counts. As a *formalism*, each DC is an explicit logical rule, so whether a tuple satisfies it can be checked exactly, unlike a learned model whose validity notion is implicit. DCs are also the most *expressive* such formalism, and are accordingly adopted by most major data-quality systems [14, 22, 41]. This is what lets us *guarantee* realism: a tuple either satisfies all DCs or it does not, so enforcing them in the projection yields zero violations by construction. The guarantee is also robust to imperfect constraints, as DCs can be mined automatically with perfect recall [7, 11, 48], and an overly specific mined set only narrows the admissible space, never admitting an invalid CF, while a domain expert may add further constraints. Beyond data cleaning, DCs are also used in consistent query answering [3, 5], data integration, and data profiling [1]. Constraints are also extensively used in the context of data repair, where the goal is to find (minimal) modifications to the database so that a set of constraints holds [12, 37, 41, 43]. While our suspect-set optimization is inspired by data cleaning literature [12], a key difference is that we focus on modifying a single tuple rather than the entire database. Indeed, we have shown that using HoloClean [41] for this purpose fails.

7 Conclusions

We have studied in this paper the problem of finding counterfactuals for predictions made by ML models, constraining that the counterfactuals are *realistic* as captured by Denial Constraints with respect to a given database. We optimize for proximity and diversity in this constrained space. We have shown that the state-of-the-art solution often yields non-realistic counterfactuals, whereas we achieve 100% realism and comparable quality with respect to proximity and diversity. We show that a vanilla constraint solver-based solution is not scalable, yet we are able to achieve scalability through dedicated optimizations. For future work, we will explore additional optimizations as well as implementations in additional application domains such as that of adversarial robustness.

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] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. 2015. Profiling relational data: a survey. *VLDB J.* 24, 4 (2015), 557–581.

[2] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of databases*. Addison-Wesley.

[3] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. 1999. Consistent query answers in inconsistent databases. In *Proc. 18th ACM SIGMOD-SIGACT-SIGART Symp. Principles of Database Systems (PODS)*. 68–79.

[4] Matan Ben-Tov, Daniel Deutch, Nave Frost, and Mahmood Sharif. 2024. CaFA: Cost-aware, Feasible Attacks With Database Constraints Against Neural Tabular Classifiers. In *Proc. IEEE Symp. Security and Privacy (S&P)*. 1345–1364.

[5] Leopoldo Bertossi. 2019. Database repairs and consistent query answering: Origins and further developments. In *Proc. 38th ACM SIGMOD-SIGACT-SIGAI Symp. Principles of Database Systems (PODS)*. 48–58.

[6] Tom Bewley, Salim I Amoukou, Saumitra Mishra, Daniele Magazzeni, and Manuela Veloso. 2024. Counterfactual metarules for local and global recourse. *arXiv preprint arXiv:2405.18875* (2024).

[7] Tobias Bleifuß, Sebastian Kruse, and Felix Naumann. 2017. Efficient denial constraint discovery with Hydra. *Proc. VLDB Endow.* 11, 3 (2017), 311–323.

[8] Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2006. Conditional functional dependencies for data cleaning. In *Proc. IEEE 23rd Int. Conf. Data Eng. (ICDE)*. 746–755.

[9] Dieter Brughmans, Pieter Leyman, and David Martens. 2024. Nice: an algorithm for nearest instance counterfactual explanations. *Data Min. Knowl. Discov.* 38, 5 (2024), 2665–2703.

[10] Jan Chomicki and Jerzy Marcinkowski. 2005. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.* 197, 1-2 (2005), 90–121.

[11] Xu Chu, Ihab F Ilyas, and Paolo Papotti. 2013. Discovering denial constraints. *Proc. VLDB Endow.* 6, 13 (2013), 1498–1509.

[12] Xu Chu, Ihab F Ilyas, and Paolo Papotti. 2013. Holistic data cleaning: Putting violations into context. In *Proc. IEEE 29th Int. Conf. Data Eng. (ICDE)*. 458–469.

[13] Edgar F Codd. 1970. A relational model of data for large shared data banks. *Commun. ACM* 13, 6 (1970), 377–387.

[14] Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed Elmagarmid, Ihab F Ilyas, Mourad Ouzzani, and Nan Tang. 2013. NADEEF: a commodity data cleaning system. In *Proc. ACM SIGMOD Int. Conf. Management of Data*. 541–552.

[15] Susanne Dandl, Christoph Molnar, Martin Binder, and Bernd Bischl. 2020. Multi-objective counterfactual explanations. In *Int. Conf. Parallel Problem Solving from Nature (PPSN)*. 448–469.

[16] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. 337–340.

[17] Daniel Deutch and Nave Frost. 2019. Constraints-based explanations of classifications. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 530–541.

[18] Amit Dhurandhar, Pin-Yu Chen, Ronny Luss, Chun-Chen Tu, Paishun Ting, Karthikeyan Shanmugam, and Payel Das. 2018. Explanations based on the missing: Towards contrastive explanations with pertinent negatives. *Advances in Neural Inform. Process. Syst. (NeurIPS)* 31 (2018).

[19] Michael Downs, Jonathan L Chu, Yaniv Yacoby, Finale Doshi-Velez, and Weiwei Pan. 2020. Cruds: Counterfactual recourse using disentangled subspaces. *ICML WHI* 2020 (2020), 1–23.

[20] Nidula Elgiriyewithana. 2024. New York Housing Market. https://www.kaggle.com/dsv/7351086

[21] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional functional dependencies for capturing data inconsistencies. *ACM Trans. Database Syst.* 33, 2 (2008), 1–48.

[22] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. 2013. The LLUNATIC data-cleaning framework. *Proc. VLDB Endow.* 6, 9 (2013), 625–636.

[23] Yash Goyal, Ziyan Wu, Jan Ernst, Dhruv Batra, Devi Parikh, and Stefan Lee. 2019. Counterfactual visual explanations. In *Proc. Int. Conf. Machine Learning (ICML)*. 2376–2384.

[24] Riccardo Guidotti. 2024. Counterfactual explanations and how to find them: literature review and benchmarking. *Data Min. Knowl. Discov.* 38, 5 (2024), 2770–2824.

[25] Guillaume Jeanneret, Loïc Simon, and Frédéric Jurie. 2022. Diffusion models for counterfactual explanations. In *Proc. Asian Conf. Computer Vision (ACCV)*. 858–876.

[26] Junqi Jiang, Francesco Leofante, Antonio Rago, and Francesca Toni. 2024. Robust counterfactual explanations in machine learning: A survey. *arXiv preprint arXiv:2402.01928* (2024).

[27] Junqi Jiang, Luca Marzari, Aaryan Purohit, and Francesco Leofante. 2025. RobustX: Robust Counterfactual Explanations Made Easy. *arXiv preprint arXiv:2502.13751* (2025).

[28] Amir-Hossein Karimi, Bernhard Schölkopf, and Isabel Valera. 2021. Algorithmic recourse: from counterfactual explanations to interventions. In *Proc. ACM Conf. Fairness, Accountability, and Transparency (FAccT)*. 353–362.

[29] Amir-Hossein Karimi, Julius Von Kügelgen, Bernhard Schölkopf, and Isabel Valera. 2020. Algorithmic recourse under imperfect causal knowledge: a probabilistic approach. *Advances in Neural Inform. Process. Syst. (NeurIPS)* 33 (2020), 265–277.

[30] Manish KC. 2020. USA Census Income Data. https://www.kaggle.com/datasets/manishkc06/usa-census-income-data

[31] Ron Kohavi and Barry Becker. 1996. Adult Income Dataset. UCI Machine Learning Repository. https://archive.ics.uci.edu/ml/datasets/adult

[32] Alex Kulesza, Ben Taskar, et al. 2012. Determinantal point processes for machine learning. *Found. Trends Mach. Learn.* 5, 2–3 (2012), 123–286.

[33] Christophe Leys, Christophe Ley, Olivier Klein, Philippe Bernard, and Laurent Licata. 2013. Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *J. Exp. Soc. Psychol.* 49, 4 (2013), 764–766.

[34] Ester Livshits, Rina Kochirgan, Segev Tsur, Ihab F Ilyas, Benny Kimelfeld, and Sudeepa Roy. 2021. Properties of inconsistency measures for databases. In *Proc. Int. Conf. Management of Data (SIGMOD)*. 1182–1194.

[35] Divyat Mahajan, Chenhao Tan, and Amit Sharma. 2019. Preserving causal constraints in counterfactual explanations for machine learning classifiers. *arXiv preprint arXiv:1912.03277* (2019).

[36] Ramaravind K Mothilal, Amit Sharma, and Chenhao Tan. 2020. Explaining machine learning classifiers through diverse counterfactual explanations. In *Proc. Conf. Fairness, Accountability, and Transparency (FAccT)*. 607–617.

[37] Wei Ni, Xiaoye Miao, Xiangyu Zhao, Yangyang Wu, and Jianwei Yin. 2023. Automatic data repair: Are we ready to deploy? *arXiv preprint arXiv:2310.00711* (2023).

[38] Martin Pawelczyk, Sascha Bielawski, Johannes van den Heuvel, Tobias Richter, and Gjergji Kasneci. 2021. Carla: a python library to benchmark algorithmic recourse and counterfactual explanation algorithms. *arXiv preprint arXiv:2108.00783* (2021).

[39] Martin Pawelczyk, Klaus Broelemann, and Gjergji Kasneci. 2020. Learning model-agnostic counterfactual explanations for tabular data. In *Proc. Web Conf.* 3126–3132.

[40] Rafael Poyiadzi, Kacper Sokol, Raul Santos-Rodriguez, Tijl De Bie, and Peter Flach. 2020. FACE: feasible and actionable counterfactual explanations. In *Proc. AAAI/ACM Conf. AI, Ethics, and Society*. 344–350.

[41] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. 2017. Holoclean: Holistic data repairs with probabilistic inference. *arXiv preprint arXiv:1702.00820* (2017).

[42] Maximilian Schleich, Zixuan Geng, Yihong Zhang, and Dan Suciu. 2021. Geco: Quality counterfactual explanations in real time. *arXiv preprint arXiv:2101.01292* (2021).

[43] Shaoxu Song, Han Zhu, and Jianmin Wang. 2016. Constraint-variance tolerant data repairing. In *Proc. Int. Conf. Management of Data (SIGMOD)*. 877–892.

[44] Gabriele Tolomei, Fabrizio Silvestri, Andrew Haines, and Mounia Lalmas. 2017. Interpretable predictions of tree-based ensembles via actionable feature tweaking. In *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.* 465–474.

[45] Berk Ustun, Alexander Spangher, and Yang Liu. 2019. Actionable recourse in linear classification. In *Proc. Conf. Fairness, Accountability, and Transparency (FAccT)*. 10–19.

[46] Sahil Verma, Varich Boonsanong, Minh Hoang, Keegan Hines, John Dickerson, and Chirag Shah. 2024. Counterfactual explanations and algorithmic recourses for machine learning: A review. *ACM Comput. Surv.* 56, 12 (2024), 1–42.

[47] Sandra Wachter, Brent Mittelstadt, and Chris Russell. 2017. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *Harv. JL & Tech.* 31 (2017), 841.

[48] Renjie Xiao, Zijing Tan, Haojin Wang, and Shuai Ma. 2022. Fast approximate denial constraint discovery. *Proc. VLDB Endow.* 16, 2 (2022), 269–281.

A Proofs

Proof of Proposition 4.1. The proof is via reduction from 3-SAT. Given a boolean CNF expression ϕ, we design a relation $\mathcal{R}$ whose attributes correspond to the variables of ϕ. For every clause $C_i = l_1 \vee l_2 \vee l_3$ in ϕ, we design the DC: $\forall t.\ \neg(t.a_1 = b_1 \wedge t.a_2 = b_2 \wedge t.a_3 = b_3)$ where a_i is the variable of literal l_i, and $b_i = 1$ if a_i appears negatively in l_i, otherwise $b_i = 0$.

We have that ϕ is satisfiable if and only if there exists a tuple satisfying all DCs. For one direction, consider a satisfying assignment to ϕ, and design a tuple with 1 in attributes corresponding to variables set to *true*, and 0 elsewhere. Since the assignment satisfies every clause, the tuple satisfies every DC. Conversely, a tuple satisfying all DCs translates into a truth assignment satisfying all clauses. □

B Additional Experimental Details

Table 6: Dataset characteristics: number of tuples, attributes, and DCs.

Dataset	Tuples	Attributes	Attr. Types		Constraints
			Cat	Num	
Adult	30,014	11	8	3	6
NY-Housing	2,996	6	3	3	9
Tax	99,904	9	5	4	8
Census	94,786	39	28	11	200

The Adult-Income dataset [31] contains demographic and educational attributes; the task is to predict whether income exceeds $50,000. The NY-Housing dataset [20] contains New York real estate properties; the task is to classify whether price exceeds $1,000,000. The Tax dataset [48] is synthetic and commonly used in DC discovery research; the task is to predict whether tax rate exceeds 5%. The Census-Income dataset [30] is similar to Adult with more recent data and richer attributes.

B.1 Immutable Attributes

Immutable attributes determine which attributes CFs are allowed to modify, focusing the search on relevant changes. For each dataset, we designated a representative set: demographic attributes such as age, race, and sex for Adult and Census, and structural attributes such as property type and sublocality for NY-Housing. These choices reflect attributes that are naturally fixed in each domain; other reasonable choices are possible and our framework supports any such designation. For Tax, no attributes were designated as immutable.

B.2 Constraint Selection

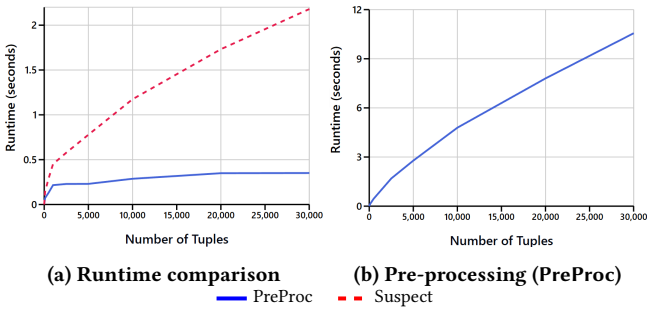
For Adult and NY, we selected meaningful binary constraints from those mined by FastADC and manually crafted unary constraints based on domain knowledge reflected in the data. For Tax, we used all mined constraints. For Census, which produced 2,566 constraints, we randomly sampled 200 to maintain tractability while preserving constraint diversity. Example DCs appear in Table 1.

B.3 Implementation Details

For each dataset, we train a neural network classifier with a single hidden layer (100 units, ReLU activation, softmax output) implemented in PyTorch. All experiments ran on a server with Intel Xeon Gold 6252 CPU (2.10 GHz, 96 cores) and 1TB RAM, running Ubuntu 18.04 LTS.

C Additional Experiments

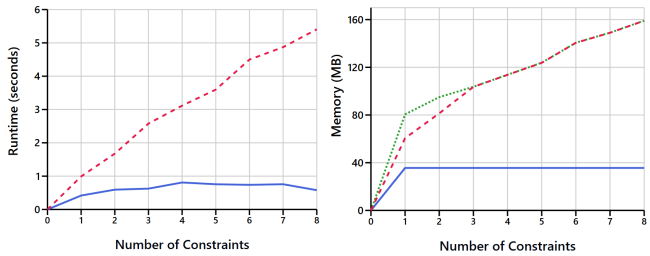
C.1 Projection Scalability



(a) Runtime comparison **(b) Pre-processing (PreProc)**

Figure 6: Scalability w.r.t. dataset size (Adult dataset).

Runtime vs Dataset Size. Runtime scales linearly with dataset size for both variants. Figure 6 shows Adult results: PreProc completes projections in 0.35s at 30K tuples while Suspect requires 2.18s. The linear scaling stems from constraint instantiation: each database tuple potentially participates in DC violations, requiring assertions proportional to relation size. Pre-processing time and memory follow the same linear pattern. Notably, Suspect consumes significantly less memory than PreProc despite similar runtime costs, as it filters constraints at projection time.



(a) Runtime comparison **(b) Memory comparison**

Figure 7: Scalability w.r.t. number of constraints (Tax dataset).

Runtime vs Number of Constraints. Runtime increases linearly with the number of denial constraints, as demonstrated on Tax (Figure 7). PreProc projection time grows from 0.42s with 1 constraint to 0.58s with 8 constraints, while Suspect increases from 0.99s to 5.41s. Memory scales similarly, from 80MB to 159MB for pre-processing. Suspect shows steeper runtime growth because it cannot amortize constraint compilation, rebuilding the optimizer for each projection.

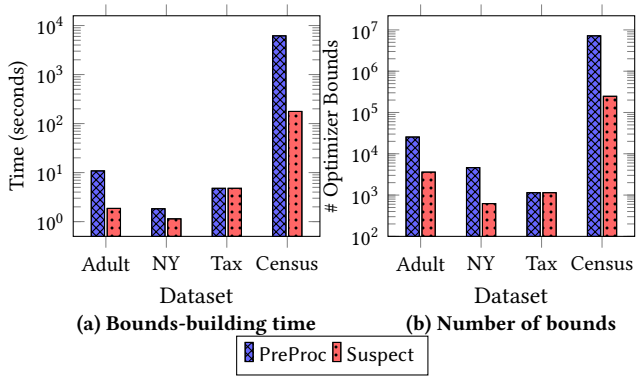


Figure 8: Suspect-set filtering evaluation. (a) Suspect reduces bounds-building time by up to 35×. (b) Suspect reduces the number of DC instantiations by up to 96.5%.

Table 8: HoloClean projection results. Violations remain largely unresolved.

Dataset	Method	Avg. Viol.	Unary Viol.	Tuple Conf.	Unreal. %
Adult	Input	4.18±0.71	0.63±0.48	575.27±538.55	100.0
	HC	3.72±1.05	0.63±0.48	366.09±476.22	89.0
NY	Input	1.40±1.11	1.00±1.00	77.00±229.33	100.0
	HC	1.40±1.11	1.00±1.00	77.00±229.33	100.0

C.2 Additional Diversity Metrics

Table 7 reports additional diversity metrics beyond the DPP scores presented in the main paper. Average pairwise distance (Pair. Div.) measures the mean $dist_{agg}$ between all CF pairs, capturing overall spread. Minimum pairwise distance (Min. Dist. Div.) reports the smallest distance between any two CFs, indicating whether any pair is overly similar [15, 36]. Both are computed using $dist_{agg}$ (Equation 4).

The trends are consistent with the DPP results: Ours achieves pairwise diversity comparable to DiCE on most datasets, with the largest gap on NY due to the tight feasible region imposed by constraints. Our framework currently optimizes for DPP diversity; adapting the diversity constraints in Section 4 to directly optimize alternative diversity measures such as pairwise or minimum distance is a natural direction for future work.

Table 7: Full diversity metrics. DPP: Determinantal Point Process diversity [32]; Pair. Div.: average pairwise distance; Min. Dist. Div.: minimum distance between any CF pair.

Dataset	Method	DPP	Pair. Div.	Min. Dist. Div.
Adult	DiCE	0.976±0.006	21.71±3.49	12.33±2.37
	Ours	0.959±0.010	16.13±2.27	9.00±2.62
	Ours-Div	0.945±0.033	15.95±2.67	8.03±3.34
NY	DiCE	0.979±0.010	38.64±10.53	10.76±5.14
	Ours	0.913±0.049	21.40±8.22	5.05±2.68
	Ours-Div	0.714±0.353	21.34±11.95	3.23±2.35
Tax	DiCE	0.886±0.015	8.23±0.69	5.00±0.95
	Ours	0.881±0.014	7.99±0.56	4.91±1.00
	Ours-Div	0.870±0.027	7.99±0.54	4.18±1.27
Census	DiCE	0.989±0.002	28.76±2.23	20.97±2.87
	Ours	0.989±0.002	29.45±3.09	20.60±2.42
	Ours-Div	0.901±0.285	31.88±3.25	20.33±6.85

C.3 Suspect-Set Optimization Evaluation

Figure 8 evaluates the Suspect-set filtering optimization described in Section 4. Figure 8(a) compares bounds-building time: Suspect reduces this step by 6–35× compared to PreProc, with the largest gain on Census (176.8s vs. 6,168s). For Tax, both methods produce identical bounds since all tuples are suspects. Figure 8(b) shows the number of optimizer bounds (DC instantiations). Suspect reduces instantiations by up to 96.5% on Census (246K vs. 7.19M), explaining the runtime gains. On datasets with fewer tuples or less selective immutable attributes, the reduction is smaller but still substantial (e.g., 85% on Adult and NY).

C.4 Projection via HoloClean

We evaluated HoloClean [41], a probabilistic data cleaning framework, as an alternative for projection. Table 8 shows that HoloClean leaves 89% (Adult) and 100% (NY) violations unresolved, confirming general-purpose cleaning is unsuitable for targeted tuple projection.

C.5 Custom Distance Functions

Our projection algorithm supports arbitrary distance functions encoded in the solver. We evaluate three distance functions on the Adult dataset: L0 (counting changed attributes for both categorical and numerical), $dist_{agg}$ (Equation 4), and a custom function with domain-specific categorical distances. The custom function encodes semantic transition costs; for example, changing occupation from Farming_fishing to Prof_specialty has distance 4 (reflecting a significant career change), while transitioning marital status from Widowed to Never_married is assigned ∞ (impossible).

Table 9: Projection with different distance functions on Adult. Each row optimizes the distance function in the first column; remaining columns show the resulting projection distances under all three metrics.

Optimized	Runtime (s)	$dist_{agg}$	L0	Custom
L0	0.17	8.82	2.55	191.03
$dist_{agg}$	0.35	5.21	2.55	96.54
Custom	1.13	5.21	2.55	5.68

Table 9 shows that each function successfully optimizes its target metric. Runtime scales with distance function complexity: L0 (simple counting) is fastest at 0.17s, $dist_{agg}$ (MAD-normalized) requires 0.35s, and the custom function (pairwise categorical lookups) takes 1.13s. These results demonstrate that our solver-based projection is robust to the choice of distance function, with runtime proportional to encoding complexity.