# ASPEN: ASP-Based System for Collective Entity Resolution

Zhiliang Xiang, Meghyn Bienvenu, Gianluca Cima, Víctor Gutiérrez-Basulto, Yazmín Ibáñez-García

# ASPEN: ASP-Based System for Collective Entity Resolution

**Zhiliang Xiang**[1] , **Meghyn Bienvenu**[2,3] , **Gianluca Cima**[4] ,
**Víctor Gutiérrez-Basulto**[1] , **Yazmín Ibáñez-García**[1]
[1]Cardiff University, UK
[2]Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, Talence, France
[3]Japanese-French Laboratory for Informatics, CNRS, NII, IRL 2537, Tokyo, Japan
[4]Sapienza University of Rome, Italy
{xiangz6,gutierrezbasultov,ibanezgarciay}@cardiff.ac.uk, meghyn.bienvenu@labri.fr,
cima@diag.uniroma1.it

**Abstract**

In this paper, we present ASPEN, an answer set programming (ASP) implementation of a recently proposed declarative framework for collective entity resolution (ER). While an ASP encoding had been previously suggested, several practical issues had been neglected, most notably, the question of how to efficiently compute the (externally defined) similarity facts that are used in rule bodies. This leads us to propose new variants of the encodings (including Datalog approximations) and show how to employ different functionalities of ASP solvers to compute (maximal) solutions, and (approximations of) the sets of possible and certain merges. A comprehensive experimental evaluation of ASPEN on real-world datasets shows that the approach is promising, achieving high accuracy in real-life ER scenarios. Our experiments also yield useful insights into the relative merits of different types of (approximate) ER solutions, the impact of recursion, and factors influencing performance.

## 1 Introduction

Entity resolution (ER) is a fundamental problem in data quality which aims at identifying different constants (of the same type) that refer to the same real-world entity (Singla and Domingos 2006; Singla and Domingos 2006). Over time, several variants of ER (also known as record linkage or deduplication) have been investigated, including pairwise matching in a single table and (the more general) *collective* ER, which looks at the joint resolution (match, merge) of entity references across multiple tables (Bhattacharya and Getoor 2007). Given the multi-faceted nature of the ER problem, diverse techniques have been already proposed to tackle it (Christophides et al. 2021), including machine learning (ML), and declarative frameworks based upon logical rules and constraints. Most existing approaches to ER focus on single-pass matching of tuples within a single table or between a pair of tables, and ML methods have obtained remarkable results (Li et al. 2020) for such settings. On the other hand, declarative methods are well suited for complex multi-relational settings as they naturally exploit the relational dependencies to perform collective ER. Moreover, some declarative approaches conduct ER in a *recursive* manner (also called *deep* ER (Deng et al. 2022)), instead of examining entity pairs only once.

LACE is a recently proposed declarative framework (Bienvenu, Cima, and Gutiérrez-Basulto 2022) for collective ER, which employs hard and soft rules to define mandatory and possible merges, and denial constraints (Bertossi 2011) to enforce consistency of the resulting database and constrain the allowed combinations of merges. LACE employs a dynamic semantics in which rule bodies are evaluated over the current induced database, taking into account all previously derived merges. This makes it possible to support recursive scenarios, while ensuring that all merges have a (non-circular) derivation. The semantics of LACE is also global since all occurrences of the matched constants are merged, rather than only those constant occurrences used in deriving the match. Additionally, LACE considers a space of maximal (w.r.t. set inclusion) solutions, which emerges from adopting denial constraints to enforce consistency and restricting which merges can be performed together, effectively creating choices. From this, one can define the notions of possible and certain merges, as those merges that belong to some, respectively all, maximal solutions.

While the theoretical foundations of LACE have been already established, a real-life implementation is to-date absent. Bienvenu, Cima, and Gutiérrez-Basulto (2022) showed that LACE solutions can be faithfully captured by answer set programming (ASP) (Lifschitz 2019; Gebser et al. 2012) stable models. Building upon this, in this paper we present **ASPEN**, an ASP-based system for collective ER. ASPEN deals with several practical issues, including the question of how to efficiently compute the (externally defined) similarity facts that are used in rule bodies. It also implements new variants of the encodings (including Datalog approximations) and uses different functionalities of ASP solvers to compute (maximal) solutions, and (approximations of) the sets of possible and certain merges. A comprehensive experimental evaluation of ASPEN on real-world datasets shows that the approach is promising, achieving high accuracy in real-life ER scenarios. Our experiments also yield useful insights into the relative merits of different types of (approximate) ER solutions, the impact of recursion, and factors influencing performance, such as the degree of dirtiness and size of datasets. ASPEN also leverages the xclingo (Cabalar and Muñiz 2023) framework for explaining conclusions of

ASP programs to compute the justification of a merge in a solution, making ASPEN a *justifiable* framework for ER.

**Related Work** We discuss prior work on logic-based approaches to ER; for details on ML and probabilistic methods, see (Christophides et al. 2021). The LACE framework, underlying ASPEN, shares some characteristics with other logic-based ER methods. Similar to approaches based on matching dependencies (MDs) (Bertossi, Kolahi, and Lakshmanan 2013; Deng et al. 2022), ASPEN adopts a dynamic semantics, enabling recursive ER. Like the Datalog-like approaches Dedupalog (Arasu, Ré, and Suciu 2009) and Entity Linking (EL) (Burdick et al. 2016) (and unlike MDs), ASPEN considers a global semantics. Finally, as in the EL framework, ASPEN does not consider only a single solution, but rather a space of maximal solutions, leading to notions of possible and certain merges. While ASP encodings have been proposed for MDs (Bertossi, Kolahi, and Lakshmanan 2013; Bahmani and Bertossi 2017), no implementation nor evaluation are provided. Likewise, there is no implementation of the EL framework. The closest existing implementations of logic-based ER are those of Dedupalog (Arasu, Ré, and Suciu 2009) and MRL (Deng et al. 2022). However, they are not publicly available. The main distinguishing features of ASPEN compared to existing systems are as follows: **i)** Rather than computing a single (possibly non-optimal) solution, ASPEN is not only able to compute, with the guarantee of correctness, a space of maximal solutions but also approximations with different levels of granularity based on different reasoning modes. **ii)** To the best of our knowledge, ASPEN is the first system that is able to explicitly give justifications to merges. We finally note in passing that ASP-based approaches to database repair have been explored (Eiter et al. 2008; Manna, Ricca, and Terracina 2013; Ahmetaj et al. 2022). Additionally, a bespoke ASP system for cleaning healthcare data has also been developed (Terracina, Martello, and Leone 2013).

All programs, code, experiments and data of ASPEN are available at https://github.com/zl-xiang/Aspen.

## 2 Preliminaries

We assume infinite sets of *constants* $\mathbf{C}$ and *variables* $\mathbf{V}$. A *(database) schema* $\mathcal{S}$ consist of a finite set of relation symbols, each having an associated arity $k \in \mathbb{N}$. We write $R/k \in \mathcal{S}$ to indicate that $R$ has arity $k$. A *relational atom* (over schema $\mathcal{S}$) takes the form $R(t_1, \ldots, t_k)$ where $R/k \in \mathcal{S}$ and $t_i \in \mathbf{C} \cup \mathbf{V}$ for $1 \leq i \leq k$, and we call $R(t_1, \ldots, t_k)$ a *fact* if $\{t_1, \ldots, t_k\} \subseteq \mathbf{C}$. We say that $t_i$ *occurs in position* $i$ of an atom $R(t_1, \ldots, t_k)$. A *database* $D$ (over schema $\mathcal{S}$) is a finite set of facts (over $\mathcal{S}$). The set of constants occurring in a database $D$ is denoted by $\mathsf{Dom}(D)$. Sometimes it will prove more natural to employ *attributes* rather than (unnamed) positions, writing $R(A_1, \ldots, A_k)$ to indicate that $(A_1, \ldots, A_k)$ are the attributes of $R$.

A *conjunctive query (CQ)* over a schema $\mathcal{S}$ takes the form $q(\vec{x}) = \exists \vec{y}.\varphi(\vec{x}, \vec{y})$, where $\vec{x}$ and $\vec{y}$ are disjoint tuples of distinguished and quantified variables, and $\varphi(\vec{x}, \vec{y})$ is a conjunction of relational atoms over $\mathcal{S}$, with variables drawn from $\vec{x} \cup \vec{y}$. We use $\mathsf{terms}(q)$ for the set of *terms* of $q$, i.e. the constants and variables appearing in $q$. The set of *answers* to a CQ $q(\vec{x}) = \exists \vec{y}.\varphi(\vec{x}, \vec{y})$ on a database $D$ (over the same schema), denoted $q(D)$, contains those tuples of constants $\vec{c}$ such that there exists a mapping $h : \mathsf{terms}(q) \to \mathsf{Dom}(D)$ such that (i) $h(\vec{x}) = \vec{c}$, (ii) $h(t) = t$ for $t \in \mathsf{terms}(q) \cap \mathbf{C}$, and (iii) for every atom $R(t_1, \ldots, t_k)$ of $q$, $R(h(t_1), \ldots, h(t_k)) \in D$. We will also consider *CQs with inequalities* ($\text{CQ}^{\neq}$), which may additionally include inequality atoms $t_i \neq t_j$, in which case we require that the mapping $h$ further satisfies (iv) $h(t_i) \neq h(t_j)$ whenever $q$ contains $t_i \neq t_j$. When $q$ has only quantified variables, it is called *Boolean*, and we say that a Boolean $\text{CQ}^{\neq}$ $q$ is *satisfied* in $D$ if $q(D) = \{()\}$. A *denial constraint* (DC) takes the form $q \to \bot$, where $q$ is a Boolean $\text{CQ}^{\neq}$. We say that $D$ *satisfies a DC* $q \to \bot$ just in the case that the $\text{CQ}^{\neq}$ $q$ is not satisfied in $D$. Functional dependencies (FDs) and primary key constraints are special cases of DCs.

When we speak of complexity, we will always mean *data complexity*, which is measured only in terms of the size of the input database, with all other inputs (e.g. ER specifications and queries) treated as fixed.

## 3 Entity Resolution Framework

In this section, we recall the syntax and semantics of LACE (Bienvenu, Cima, and Gutiérrez-Basulto 2022). We also introduce new notions that are useful for our system.

### 3.1 LACE Entity Resolution Specifications

Entity resolution can be formulated as the task of discovering pairs of syntactically distinct database constants that refer to the same entity (we will often use the term *merges* for such pairs). We adopt the LACE framework, which focuses on identifying merges of entity-referencing constants (e.g. paper or author identifiers).

LACE employs hard and soft rules to identify mandatory and possible merges. *Hard rules* and *soft rules*, over schema $\mathcal{S}$, take respectively the forms:

$$q(x, y) \Rightarrow \mathsf{Eq}(x, y), \quad q(x, y) \dashrightarrow \mathsf{Eq}(x, y) \qquad (1)$$

where $\mathsf{Eq}$ is a special relation symbol not in $\mathcal{S}$ used to store pairs of merged constants, and $q(x, y)$ is a CQ using relation symbols from $\mathcal{S}$. Intuitively, a hard (resp. soft) rule states that a pair of constants $(c_1, c_2)$ being an answer to $q$ provides sufficient (resp. reasonable) evidence that $c_1$ and $c_2$ refer to the same real-world entity. We use $q(x, y) \to \mathsf{Eq}(x, y) \in \Gamma$ to denote a generic (hard or soft) rule (note the arrow).

It is natural and useful for rule bodies to use *similarity relations*, i.e. binary relations whose extension is fixed and computed using some external function (e.g. by applying a string similarity measure to a pair of constants and keeping those pairs whose score exceeds a given threshold). We shall thus allow the considered schema to contain such similarity relations and will adopt the more intuitive infix notation $x \approx y$ (possibly with indices) for similarity atoms. Note also that in the present section, we will assume that the similarity facts are provided as part of the input database, leaving the question of how to best compute them to later sections.

Band(bid, name, genre, year, founder)

| bid | name | genre | year | founder |
|---|---|---|---|---|
| $b_1$ | Pink Floyd | Psy. rock | 1965 | Barrett |
| $b_2$ | The Pink Floyd | Prog. rock | 1965 | Barrett |

Song(sid, title, lyricist, bid)

| sid | title | lyricist | bid |
|---|---|---|---|
| $s_1$ | Shine On You Crazy Diamond (I-IV) | Waters | $b_1$ |
| $s_2$ | Shine On You Crazy Diamond | Waters | $b_2$ |
| $s_3$ | Shine On You Crazy Diamond (V-IX) | Waters | $b_1$ |

$\sigma_{\mathsf{ex}}$: $\mathit{Song}(x,t,l,b) \wedge \mathit{Song}(y,t',l,b) \wedge t \approx t' \dashrightarrow \mathsf{Eq}(x,y)$
$\rho_{\mathsf{ex}}$: $\mathit{Band}(x,n,g,d,f) \wedge \mathit{Band}(y,n',g',d,f) \wedge n \approx n' \wedge g \approx g' \Rightarrow \mathsf{Eq}(x,y)$
$\delta_{\mathsf{ex}}$: $\mathit{Appear}(s,a,i) \wedge \mathit{Appear}(s,a,j) \wedge i \neq j \rightarrow \bot$

We assume that the extension of the similarity relation $\approx$ (restricted to $\mathsf{Dom}(D_{\mathsf{ex}})$) is the reflexive and symmetric closure of $\{(n_1,n_2),(g_1,g_2),\ (t_1,t_2),\ (t_2,t_3)\}$, where $n_i$, $g_i$, and $t_i$ are the name and genre of band $b_i$ and the title of song $s_i$, respectively.

Appear(sid, album, position)

| sid | album | position |
|---|---|---|
| $s_1$ | Wish You Were Here | 1 |
| $s_2$ | A Delicate Sound of Thunder | 1 |
| $s_3$ | Wish You Were Here | 5 |

Figure 1: A schema $\mathcal{S}_{\mathsf{ex}}$, an $\mathcal{S}_{\mathsf{ex}}$-database $D_{\mathsf{ex}}$, and an ER specification $\Sigma_{\mathsf{ex}} = \langle \Gamma_{\mathsf{ex}}, \Delta_{\mathsf{ex}} \rangle$ for $\mathcal{S}_{\mathsf{ex}}$, where $\Gamma_{\mathsf{ex}} = \{\rho_{\mathsf{ex}}, \sigma_{\mathsf{ex}}\}$ and $\Delta_{\mathsf{ex}} = \{\delta_{\mathsf{ex}}\}$.

**Example 1.** Figure 1 presents our running example inspired by the MUSIC dataset. Relations *Group*, *Song*, *Appear* of the schema $\mathcal{S}_{\mathsf{ex}}$ provide information about music bands, songs, and which songs appear on which albums. The attributes **bid** and **sid** contain identifiers for bands and songs respectively, and the task at hand is to identify which of these identifiers refer to the same band / song. The hard rule $\rho_{\mathsf{ex}}$ says that if two band ids have the same founding year, same founder, similar names, and similar genres, they must refer to the same band. The soft rule $\sigma_{\mathsf{ex}}$ states that two song ids likely refer to the same song if they have the same band and lyricist and similar titles. For simplicity, a single similarity relation $\approx$ is used to say which strings count as similar.

To enforce consistency of the inferred merges and to help block false positives, LACE specifications may also include *denial constraints*. For example, the denial constraint $\delta_{\mathsf{ex}}$ in Figure 1 is an FD for *Appear*, which forbids the occurrence of the same song in different positions within an album.

A LACE *entity resolution (ER) specification* over schema $\mathcal{S}$ takes the form $\Sigma = \langle \Gamma, \Delta \rangle$, where $\Gamma = \Gamma_h \cup \Gamma_s$ is a finite set of hard and soft rules and $\Delta$ is a finite set of denial constraints, all over $\mathcal{S}$. Rulesets are required to satisfy a *sim-safety condition*, whereby the relation positions involved in merges must be distinct from those involved in similarity atoms. The example specification $\Sigma_{\mathsf{ex}}$ is sim-safe as the attributes involved in similarity atoms (**title**, **name**, **genre**) are different from those involved in merges (**bid**, **sid**). Thanks to sim-safety, *we may assume w.l.o.g. that the constants appearing in merge positions do not occur in sim positions.*

## 3.2 Semantics of LACE

The semantics of LACE associates with every database $D$ and ER specification $\Sigma$ a set of solutions, where each solution takes the form of an equivalence relation over $\mathsf{Dom}(D)$, indicating which constants refer to the same entity. Given a set of pairs $P \subseteq S \times S$, we write $\mathsf{EqRel}(P,S)$ to denote the least equivalence relation over $S$ that extends $P$.

In a nutshell, ER solutions in LACE are obtained by applying the hard and soft rules in such a manner that all hard rules and constraints are satisfied, with the inferred Eq-facts determining the equivalence relation. Importantly, the evaluation of LACE rules takes into account previously derived merges, making it possible for merges to trigger further merges. Formally, given a database $D$, and equivalence relation $E$ over $\mathsf{Dom}(D)$, the *database induced by $D$ and $E$*, denoted $D_E$, is obtained from $D$ by replacing each constant $c$ by the (uniformly chosen) representative $\bar{c}$ of its equivalence class. The set $q(D,E)$ of *answers to $q(\vec{x})$ w.r.t. $D$ and $E$* is defined as:

$$q(D,E) = \{(c_1,\ldots,c_n) \mid (\bar{c_1},\ldots,\bar{c_n}) \in q(D_E)\}$$

Rule $q(x,y) \rightarrow \mathsf{Eq}(x,y)$ is satisfied in $(D,E)$ if $q(D,E) \subseteq E$, and DC $\delta$ is satisfied in $(D,E)$ if $\delta$ is satisfied in $D_E$.

With these notions in hand, we can now define solutions. Given an ER specification $\Sigma$ and database $D$, we call $E$ a *candidate solution for* $(D,\Sigma)$ if one of the following holds:

1. $E = \mathsf{EqRel}(\emptyset, \mathsf{Dom}(D))$
2. $E = \mathsf{EqRel}(E' \cup \{\alpha\}, D)$, where $E'$ is a candidate solution for $(D,\Sigma)$ and $\alpha \in q(D,E) \setminus E'$ for some $q(x,y) \rightarrow \mathsf{Eq}(x,y) \in \Gamma$

and it is a *solution* for $(D,\Sigma)$ if additionally (i) $(D,E) \models \Gamma_h$ and (ii) $(D,E) \models \Delta$. We denote by $Sol(D,\Sigma)$ the set of solutions for $(D,\Sigma)$, and let $\mathsf{MaxSol}(D,\Sigma)$ be the set of *maximal solutions*, i.e. solutions $E$ such that there is no solution $E'$ for $(D,\Sigma)$ with $E \subsetneq E'$.

**Example 2.** We determine the maximal solutions for $(D_{\mathsf{ex}}, \Sigma_{\mathsf{ex}})$. The initial trivial equivalence relation $E = \mathsf{EqRel}(\emptyset, \mathsf{Dom}(D_{\mathsf{ex}}))$ is not a solution as the hard rule $\rho_{\mathsf{ex}}$ requires us to merge $b_1$ and $b_2$. Applying $\rho_{\mathsf{ex}}$, we obtain $E' = \mathsf{EqRel}(\{(b_1,b_2)\}, \mathsf{Dom}(D_{\mathsf{ex}}))$, which is a solution for $(D_{\mathsf{ex}}, \Sigma_{\mathsf{ex}})$ but not a maximal one. Indeed, due to the addition of $(b_1,b_2)$, both $(s_1,s_2)$ and $(s_2,s_3)$ can now be obtained by applying the soft rule $\sigma_{\mathsf{ex}}$. Notice, however, that it is not possible to include both of them, as transitivity would force us to include $(s_1,s_3)$, leading to a violation of $\delta_{\mathsf{ex}}$. We thus obtain two maximal solutions for $(D_{\mathsf{ex}}, \Sigma_{\mathsf{ex}})$, namely $E_1 = \mathsf{EqRel}(\{(b_1,b_2),(s_1,s_2)\}, \mathsf{Dom}(D))$ and $E_2 = \mathsf{EqRel}(\{(b_1,b_2),(s_2,s_3)\}, \mathsf{Dom}(D))$.

There can be zero, one, or multiple (maximal) solutions. However, for specifications that do not contain soft rules, there can be at most one solution, and for specifications that do not contain constraints, there is always a single solution.

### 3.3 Summarizing and Explaining Solutions

When there are multiples solutions, it is useful to be able to summarize them by identifying merges that occur in all or some maximal solutions. Formally, we say that a merge $\alpha$ is *certain* if $\alpha \in E$ for every $E \in \mathsf{MaxSol}(D, \Sigma)$, and it is *possible* if $\alpha \in E$ for some $E \in \mathsf{MaxSol}(D, \Sigma)$ (equivalently, some $E \in \mathsf{Sol}(D, \Sigma)$). We use $\mathsf{CM}(D, \Sigma)$ and $\mathsf{PM}(D, \Sigma)$ for the sets of certain and possible merges.

While certain and possible merges provide natural summarizations, they are unfortunately hard to compute:

**Theorem 1.** *(Bienvenu, Cima, and Gutiérrez-Basulto 2022) It is* NP*-complete (resp.* $\Pi_2^p$*-complete) in data complexity to decide if* $\alpha \in \mathsf{PM}(D, \Sigma)$ *(resp.* $\mathsf{CM}(D, \Sigma)$*).*

For this reason, it will prove useful to define efficiently computable approximations. To this end, we consider two ways of simplifying a specification $\Sigma = \langle \Gamma_h \cup \Gamma_s, \Delta \rangle$:

- $\Sigma^{\mathsf{lb}} = \langle \Gamma_h, \emptyset \rangle$, i.e. remove soft rules and constraints
- $\Sigma^{\mathsf{ub}} = \langle \Gamma_h \cup \Gamma_{s \to h}, \emptyset \rangle$, i.e. drop constraints and replace soft rules by the corresponding hard rules ($\Gamma_{s \to h}$)

As $\Sigma^{\mathsf{lb}}$ and $\Sigma^{\mathsf{ub}}$ do not contain any constraints, they will always yield a unique solution. We can therefore define

- $\mathsf{LB}(D, \Sigma)$ as the unique solution to $(D, \Sigma^{\mathsf{lb}})$
- $\mathsf{UB}(D, \Sigma)$ as the unique solution to $(D, \Sigma^{\mathsf{ub}})$

We summarize the properties of the different merge sets:

**Theorem 2.** *The sets* $\mathsf{LB}(D, \Sigma)$ *and* $\mathsf{UB}(D, \Sigma)$ *can be computed in polynomial w.r.t. data complexity. Moreover, if* $Sol(D, \Sigma) \neq \emptyset$*, then*

$$\mathsf{LB}(D, \Sigma) \subseteq \mathsf{CM}(D, \Sigma) \subseteq \mathsf{PM}(D, \Sigma) \subseteq \mathsf{UB}(D, \Sigma)$$

*and* $\mathsf{CM}(D, \Sigma) \subseteq M \subseteq \mathsf{PM}(D, \Sigma)$ *for* $M \in \mathsf{MaxSol}(D, \Sigma)$.

**Example 3.** In our toy example, $\mathsf{LB}(D_{\mathsf{ex}}, \Sigma_{\mathsf{ex}})$ and $\mathsf{CM}(D_{\mathsf{ex}}, \Sigma_{\mathsf{ex}})$ coincide, and the only non-trivial pair they contain is $(b_1, b_2)$. The set $\mathsf{PM}(D_{\mathsf{ex}}, \Sigma_{\mathsf{ex}})$ further contains merges $(s_1, s_2)$ and $(s_2, s_3)$, whilst $\mathsf{UB}(D_{\mathsf{ex}}, \Sigma_{\mathsf{ex}})$ also contains $(s_1, s_3)$ (not present in any solution).

We will employ *proof trees* to explain why a merge appears in a solution (we provide here some intuitions behind proof trees and refer to App. A for more details). Informally, a proof tree for a merge $\alpha$ in a solution $E \in Sol(D, \Sigma)$ is a node-labelled tree such that (a) the root node has label $\alpha$, (b) every leaf node is labelled with a fact from $D$, and (c) every non-leaf node $n$ is labelled with a pair of constants $(d, e)$ corresponding to a single transitive step or a rule application.

To quantify the number of successive rule applications needed to obtain a merge, we let the *rule-depth of a proof tree* $T$ be the maximum number of rule nodes in any leaf-to-root path in $T$. The *level of a merge* $\alpha$ *in a solution* $E$ is 0 if $\alpha = (c, c)$ for some $c \in \mathbf{C}$, and otherwise is the minimum rule-depth of all proof trees of $\alpha$ in $E$.

## 4 ASP Encoding and Algorithms

The ASPEN system implements the LACE framework by encoding ER specifications as ASP programs and making calls to ASP solvers to generate and reason about ER solutions. After recalling some ASP notions, we present the ASP encoding and algorithms employed by ASPEN. We also explore some practical issues that were ignored in the theoretical treatment of LACE, most importantly, the question of how to handle similarity atoms.

We assume familiarity with ASP basics, see (Brewka, Eiter, and Truszczynski 2011; Gebser et al. 2012; Lifschitz 2019) for more details. For our purposes, it is enough to consider *normal rules* and *constraints*. That is, respectively, rules with a single atom head and rules with an empty head. We shall use $\Pi$ to denote an ASP *program* (a set of rules). Given a program $\Pi$, we use $gr(\Pi)$ to denote the set of all ground instantiations rules from $\Pi$ with constants occurring in $\Pi$, and $SM(\Pi)$ to denote the set of all stable models of $\Pi$. Determining whether a program has a stable model is the fundamental decision problem in ASP, which is solved using ASP solvers (Gebser et al. 2018). However, more reasoning modes are needed to cover problems encountered in practice. Most modern ASP solvers are also able to *enumerate* ($n$ elements of) $SM(\Pi)$; *project* answers w.r.t. a given set of atoms, and enumerating ($n$ elements of) those projections; computing the intersection (resp. union) of all stable models of $\Pi$ (*cautious*, resp. *brave* reasoning); and perform *optimisations* by computing some (or enumerating $n$) elements of $SM(\Pi)$ that minimize a given objective function.

### 4.1 ASP Encoding of Solutions

Given a LACE specification $\Sigma$ and a database $D$, we define an ASP program $\Pi_{(D,\Sigma)}$ containing all the facts in $D$, and an ASP rule for each (hard or soft) rule in $\Sigma$. Consider, for example, the specification $\Sigma_{\mathsf{ex}}$ in Figure 1. Rules $\rho_{\mathsf{ex}}$, $\sigma_{\mathsf{ex}}$ and $\delta_{\mathsf{ex}}$ are translated as follows:

$$\mathtt{eq}(X, Y) \leftarrow \mathtt{band}(X, N, G, D, F), \mathtt{band}(Y, N', G', D, F), \mathtt{sim}(G, G', S), S \geq 95, \mathtt{sim}(N, N', S'), S' \geq 95.$$

$$\{\mathtt{eq}(X, Y)\} \leftarrow \mathtt{song}(X, T, L, B), \mathtt{song}(Y, T', L, B'), \mathtt{not\ empty}(L), \mathtt{sim}(T, T', S), S \geq 95, \mathtt{eq}(B, B').$$

$$\bot \leftarrow \mathtt{appear}(S, A, I), \mathtt{appear}(S', A, J), \mathtt{eq}(S, S'), I \neq J.$$

Roughly, each relational atom in the body of a (hard or soft rule) is translated into an atom in the body of an ASP rule. The relation $\mathtt{eq}$ is used to store mandatory merges (and ultimately solutions to $(D, \Sigma)$). Atoms of the form $\mathtt{eq}(X, X')$ in the rule bodies are used to encode that instantiations of $X$ and $X'$ have been determined to denote the same entity. $\Pi_{(D,\Sigma)}$ also includes rules for encoding that $\mathtt{eq}$ is an equivalence relation. Soft rules are encoded using a choice rule encoding the possibility of including $(X, Y)$ in $\mathtt{eq}$. We note that choice rules are special syntactic sugar available in ASP.

Differently from the original ASP encoding in (Bienvenu, Cima, and Gutiérrez-Basulto 2022), we encode similarity relations with a relation $\mathtt{sim}_i(X, Y, S)$, where the first two arguments store the pair of constants to be assessed for similarity, while the third argument corresponds to a similarity score. This offers the flexibility of tuning the threshold for the same similarity measure. Facts of the form $\mathtt{sim}_i(X, Y, S)$ are included in $D$ after a preprocessing stage, as discussed in Section 4.3. Notably, the encoding requires a special treatment of null values. Missing values in

databases might be problematic (Fan and Geerts 2012), in particular, when joins need to be performed to evaluate rule bodies. To represent null values in $\Pi_{(D,\Sigma)}$, we use an atom $\mathtt{empty}(nan)$, where $nan$ is a special constant. To encode the fact that merges are not performed on unknown or missing values, we add an atom of the form $\mathtt{not\ empty}(V)$ in rule bodies, for every joined variable $V$. This encoding prevents merges that otherwise would result from rule bodies being satisfied when considering two nulls equivalent. With the encoding $\Pi_{(D,\Sigma)}$ in place, solutions of $(D, \Sigma)$ are then obtained by projecting stable models of $\Pi_{(D,\Sigma)}$ w.r.t. $\mathtt{eq}$.

**Theorem 3** ((Bienvenu, Cima, and Gutiérrez-Basulto 2022)). *For every database $D$ and ER specification $\Sigma$: $E \in Sol(D, \Sigma)$ iff $E = \{(a, b) \mid \mathtt{eq}(a, b) \in M)\}$ for some stable model $M \in SM(\Pi_{(D,\Sigma)})$. In particular, $Sol(D, \Sigma) \neq \emptyset$ iff $SM(\Pi_{(D,\Sigma)}) \neq \emptyset$*[1].

## 4.2 ASP-based Algorithms

We now explain how to employ the ASP encoding to generate (maximal) solutions and other sets of merges, cf. Sec.3.3.

*Solutions.* Thanks to Theorem 3, we can obtain a single solution from $Sol(D, \Sigma)$ by using the ASP solver to generate a stable model of $\Pi_{(D,\Sigma)}$, then projecting onto its $\mathtt{eq}$ facts. Likewise, we can enumerate all or a fixed number of solutions by requesting an enumeration of $SM(\Pi_{(D,\Sigma)})$.

*Maximal solutions.* The maximal solutions correspond to the stable models of $\Pi_{(D,\Sigma)}$ having a $\subseteq$-maximal set of $\mathtt{eq}$ facts. We rely on `asprin`, a framework for implementing preferences among the stable models of a program (Brewka et al. 2015). In our case, we prefer a model $M'$ over $M$ if its projection to $\mathtt{eq}$ is a proper superset of that of $M$. `asprin` also allows us to compute $n$ optimal stable models of a program.

*Lower and upper bound merge sets.* To compute $\mathsf{LB}(D, \Sigma)$, we first construct the ASP encoding $\Pi_{(D,\Sigma^{\mathsf{lb}})}$ based on $\Sigma^{\mathsf{lb}}$ and then we use the ASP solver to compute the (unique) answer set of $\Pi_{(D,\Sigma^{\mathsf{lb}})}$, which we project onto the $\mathtt{eq}$ relation. We proceed analogously for $\mathsf{UB}(D, \Sigma)$, but using $\Sigma^{\mathsf{ub}}$.

$$\mathtt{eq}(X, Y) \leftarrow \mathtt{song}(X, T, L, B), \mathtt{song}(Y, T', L, B'),$$
$$\mathtt{not\ empty}(L), \mathtt{sim}(T, T', S), S \geq 95, \mathtt{eq}(B, B').$$

*Possible merges.* To generate $\mathsf{PM}(D, \Sigma)$, it suffices to run the ASP solver in brave reasoning mode, and to project onto the $\mathtt{eq}$ relation. To check whether a particular pair $(c, c')$ is a possible merge, we can run the solver on $\Pi_{(D,\Sigma)} \cup \{\bot \leftarrow \mathtt{not}\ \mathtt{eq}(c, c')\}$ observing that $(c, c') \in \mathsf{PM}(D, \Sigma)$ iff this modified program admits a stable model.

*Levels.* To support our analysis of the impact of recursion, we will need a means of retrieving, for a given solution $E \in Sol(D, \Sigma)$, all triples $(c, c', i)$, where $\alpha = (c, c') \in E$ and $i$ is the level of $\alpha$ in $E$. This can be achieved by considering a variant $\Pi^{\mathsf{lvl}}(i)$ of $\Pi_{(D,\Sigma^{\mathsf{ub}})}$, which is an ASP program that takes an integer $i$ as a parameter and, by taking into consideration also the already derived merges inside $E$ (i.e. those with level $< i$), applies single transitive steps or rule applications, attaching integer $i$ as the additional third element of the merges obtained in this way.

## 4.3 Similarity Computation

Our ASP rules contain body atoms $\mathtt{sim}_i(X, X', S)$, but such similarity facts are not present in the data sources and need to be computed via external functions. The question then is how best to compute a sufficient set of $\mathtt{sim}_i$ facts to properly evaluate the program, while avoiding making calls to the external functions for every possible pair of values.

A naïve approach would be to compute and store the set $\mathsf{Sim}_{\mathsf{all}}(D)$, containing all facts $\mathtt{sim}_i(c, d, s)$ such that $c$ and $d$ are data values of a form compatible with $\mathtt{sim}_i$ and $s = f_i(c, d)$, with $f_i$ the function underlying the similarity relation $\mathtt{sim}_i$. Although it requires only a polynomial number of function calls w.r.t. $|D|$, it is nevertheless extremely costly on even moderately-sized databases. A first improvement would be to only consider those pairs of constants $(c, d) \in R[i] \times R'[j]$[2] such that there is a rule $\rho$ which contains an atom $\mathtt{sim}_i(X, X', S)$ in which $X$ and $X'$ appear respectively in the $i$th (resp. $j$th) position of an $R$-atom (resp. $R'$-atom). However, as our experiments will show, this improved approach remains memory-consuming and its time consumption grows as the data size grows.

Another idea would be to exploit the structure of the rules so that we only call the similarity functions on pairs of constants that occur in a similarity atom for which the rest of the rule body is satisfied. For example, for rule $\sigma_{\mathsf{ex}}$, we would remove $\mathtt{sim}(T, T', S)$ and instead store the compared variables $(T, T')$ in a fresh relation $\mathtt{getsim}$ as follows:

$$\mathtt{getsim}(T, T') \leftarrow \mathtt{song}(X, T, L, B), \mathtt{song}(Y, T', L, B'), \mathtt{not\ empty}(L), \mathtt{eq}(B, B'). \quad (2)$$

Note however that the body still contains $\mathtt{eq}(B, B')$, whose extension is initially unknown. We shall therefore employ a further ingredient: an overapproximation of the $\mathtt{eq}$ facts. For this, we could use the program $\Pi_{(D,\Sigma^{\mathsf{ub}})}$, except that we do not know how to evaluate similarity atoms in rule bodies. One option would be to weaken the bodies by dropping all similarity atoms, but this yields a very loose approximation (cf. Appendix B). An alternative is to run the original $\Pi_{(D,\Sigma^{\mathsf{ub}})}$ program, but making *online calls to the similarity functions*[3] by replacing literals $\mathtt{sim}(X, Y, S), S \geq \delta$ with $\mathtt{sim}_{\mathsf{ext}}(X, Y) \geq \delta$. We denote by $\Pi_{(D,\Sigma^{\mathsf{ub}}_{\mathsf{ext}})}$ the modified program. This will not only give us an upper bound on the true set of $\mathtt{eq}$ facts, but it will also compute a portion of the similarity facts. In fact, a further alternative would be to rely entirely on an online computation of similarity atoms over the original program. However, this is less efficient as results cannot be reused to compute different merge sets and may be time-consuming for the computation of $\mathsf{MaxSol}(D, \Sigma)$ and $\mathsf{PM}(D, \Sigma)$ (we refer to Appendix B for details).

Combining these ideas, we obtain the following approach to similarity computation, on input $D, \Sigma$.

[1] We note that the proof in (Bienvenu, Cima, and Gutiérrez-Basulto 2022) does not consider nulls, but it can be easily extended.

[2] $R[i]$ denotes the constant at position $i$ of $R$.

[3] Modern ASP solvers, e.g. `clingo`, support the syntax of external function calls (Kaminski et al. 2023)

**Phase 1** Compute the unique stable model $M_{\mathsf{ub}}$ of $\Pi_{(D,\Sigma^{\mathsf{ub}}_{\mathsf{ext}})}$ using an ASP solver with external function calls enabled. Let $U = \{\mathtt{ubeq}(c,d) \mid \mathtt{eq}(c,d) \in M_{\mathsf{ub}}\}$, and let $L$ contain all $\mathtt{sim}_i$ facts produced during the computation.

**Phase 2** Let $\Pi_{\mathtt{sim}}$ contain, for each rule $\tau$ in $\Pi_{(D,\Sigma)}$ that encodes a (soft or hard) rule in $\Sigma$ with at least one similarity atom in its body, a rule $\tau_{\mathtt{sim}}$ that is obtained from $\tau$ by (i) deleting all similarity atoms and all associated comparison atoms (e.g. $S \geq \delta$), (ii) changing the head atom to $\mathtt{getsim}_i(T,T')$, where $\mathtt{sim}_i(T,T',S)$ occurs in $\tau$, and (iii) renaming $\mathtt{eq}$ as $\mathtt{ubeq}$.

**Phase 3** Let $M_{\mathtt{sim}}$ be the unique stable model of $\Pi_{\mathtt{sim}} \cup D \cup U$. For every pair $(c,d)$ such that $\mathtt{getsim}_i(c,d) \in M_{\mathtt{sim}}$ and there is no $\mathtt{sim}_i(c,d,\_)$ fact in $L$, call the associated similarity function $f_i$ on input $(c,d)$, and create the fact $\mathtt{sim}_i(c,d,f_i(c,d))$. Let $\mathsf{Sim}_{\mathsf{opt}}(D,\Sigma)$ contain $L$ and all of these newly created similarity facts.

Note that in Phase 2, our example rule $\sigma_{\mathsf{ex}}$ would be replaced by (2), but with $\mathtt{ubeq}(B,B')$ in place of $\mathtt{eq}(B,B')$.

The next result shows that this approach is correct, i.e. we get the same $\mathtt{eq}$ facts using $\mathsf{Sim}_{\mathsf{opt}}$ rather than the full $\mathsf{Sim}_{\mathsf{all}}$.

**Theorem 4.** *For every database $D$ and ER specification $\Sigma$,*

$$\{M_{eq} \mid M \in SM(\Pi_{(D_{\mathsf{all}},\Sigma)})\} = \{M_{eq} \mid M \in SM(\Pi_{(D_{\mathsf{opt}},\Sigma)})\}$$

*where $D_{\mathsf{all}} = D \cup \mathsf{Sim}_{\mathsf{all}}(D)$, $D_{\mathsf{opt}} = D \cup \mathsf{Sim}_{\mathsf{opt}}(D,\Sigma)$, and $M_{eq}$ is the set of $\mathtt{eq}$ facts in $M$.*

## 5 The ASPEN System

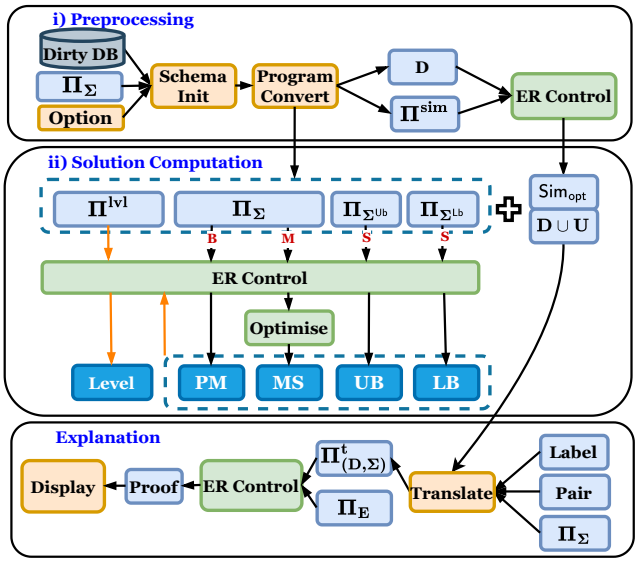



Figure 2: A General Pipeline of ASPEN .

An overview of the pipeline of ASPEN is shown in Figure 2, where boxes coloured light blue, strong blue, orange and green indicate an ASP program, ER solution, `Python` program and `Python`-based ASP solver, respectively. Arrows interconnecting these boxes symbolize the flow of data. The blue dashed boxes highlight a pool of candidates that may be chosen based on the input options.

ASPEN takes as input an ASP encoding of an ER specification (cf. Sec. 4.1), a set of running options (see below) and a dataset in `CSV/TSV` format containing duplicates with corrupted values, such as missing ones. Depending on the input options, ASPEN outputs either the specified merge sets as shown by the deep blue boxes in the second row of Figure 2, and additionally explanations of all/particular merges (cf. Appendix A). ASPEN comprises two primary phases for generating (approximate) ER solutions:

**i)** *Preprocessing Phase:* ASPEN initialises schema information and converts DB tuples into ASP facts, then computes similarity facts.

**ii)** *Solution Computation Phase:* ASPEN takes the DB and similarity facts computed before as input, then it computes various types of LACE -based ER solutions as specified in the input options.

The *explanation phase* (third row, Fig. 2) additionally takes as input a set of rule labels and a merge pair to check, and visually displays proof trees of explanations (see App. A).

*Preprocessing.* Initially, ASPEN uses a `Python` function called the *Schema Initialiser* to load schema information from the input, including database tuples, schema names, relations, attributes, and foreign keys. This information is stored in a schema instance (`Python` object), which is used throughout the system workflow. The schema information is then passed to the *Program Converter (PC)*, which generates variants of $\Pi_\Sigma$ and facts for the computation of different merge sets (cf. Sec. 4.2). During preprocessing, the PC utilises the relation and attributes structure in the schema instance to convert the database tuples into facts, including the reflexive closure of constants on merge positions. Additionally, empty entries in facts are replaced with the special constant $nan$. To precompute the similarity facts, the PC transforms the input ER program into similarity filtering programs $\Pi^{\mathsf{sim}}$. The resulting programs, along with the generated database facts, are passed to the *ER Controller (ERC)*, a `Python` object that encapsulates reasoning facilities of the ASP solver `clingo` (Gebser et al. 2012) and algorithms (cf. Sec. 4.2). The ERC then executes the similarity filtering algorithm. The resulting set of triples (compared pair of constants and score) are stored as ternary `sim`-facts, which are then combined with database facts and the set of `ubeq`s to form the fact base, as indicated by the flow arrow from the first row to the second row in Figure 2.

*Computing solutions.* With the fact base as input, the ERC selects a program variant from the PC based on the selected solution option, as shown in the blue dashed box. Subsequently, the ERC executes grounding and solving with one of the corresponding reasoning modes, as indicated by the red initials in Figure 2. It computes the desired solutions as follows:

1. for the lower and upper bound merge sets (LB, UB) by making a **S**tandard grounding&solving call to `clingo` .
2. for a set of maximal solutions MS-$n$ with an enumeration limit of $n$, with standard grounding calls on $\Pi_\Sigma$ but

employing **M**aximisation solving calls supported by the optimiser `asprin`.

3. for a set of all possible merges PM using the **B**rave reasoning mode of `clingo` .

In addition, if the merge level is required, the derived merge set will be further combined with the level retrieval program $\Pi^{\mathsf{lvl}}$ to assign levels to each of the merges, cf. orange arrows in the second row of Figure 2.

*Explanation.* ASPEN uses several `xclingo` functions (Cabalar, Fandinno, and Muñiz 2020; Cabalar and Muñiz 2023) to provide explanations of merges, including: **i)** the `Translate` function, which takes as input a program $\Pi$ and labels declared upon $\Pi$, and outputs a program $\Pi^t$, including auxiliary rules to track the original rules that have been triggered, and **ii)** the explanation program $\Pi_E$, which constructs explanations by grounding and solving together a stable model $M$ of $\Pi$ and $\Pi^t$.

As illustrated by the third row of Figure 2, ASPEN takes as input a merge pair to be justified and optionally a set of rule labels. A default set of labels is generated automatically capturing rule bodies of the ER program, if no rule labels are provided. Utilising the `Translate` function, ASPEN translates the rules and fact base into a trace program $\Pi^t_{(D,\Sigma)}$ according to the labels. The ERC then combines this with the fact base and the `xclingo` explanation program to derive a proof answer set, which is displayed as a graphical proof tree, explaining the input merge.

# 6 Experiments

We conducted experiments on real-life ER scenarios, including both pairwise and complex multi-relational datasets. We evaluate the following aspects: (1) effectiveness of our approach to similarity computation, (2) accuracy and efficiency of ASPEN, (3) effect of recursion, (4) factors impacting scalability and (5) use of Datalog engines to compute UB and LB approximations. We also exemplify generated justifications of merges in Appendix A.

## 6.1 Experimental Setup

*Datasets.* We consider two pairwise matching datasets from the bibliographic domain: DBLP-ACM (Köpcke, Thor, and Rahm 2010) and CORA, and four multi-relational datasets: **i)** a subset of the IMDB movie dataset (Deng et al. 2022); **ii)** a Music dataset sampled from the Musicbrainz database with synthetic duplicates; **iii)** a dirtier instance of the Music dataset, which contains the same amount of duplicates but a higher percentage of nulls and more syntactical variants on the duplicates; **iv)** a Pokémon dataset, sampled from the Pokémon database with synthetic duplicates and complex inter-table references. For simplicity, we refer to the datasets as *Dblp*, *Cora*, *Imdb*, *Mu*, *MuC* and *Poke*, respectively. Sources and statistics of the datasets can be found in Appendix C, as well as details about duplicates generation, sampling of datasets and the experimental environment.

*Similarity measures and metrics.* We calculate the syntactic similarities of constants based on their data types (Bahmani, Bertossi, and Vasiloglou 2017). We adopt the commonly used metrics (Köpcke, Thor, and Rahm 2010) of *Precision* (**P**), *Recall* (**R**) and *F1-Score* ($\mathbf{F_1}$). See App. C for details.

## 6.2 Similarity Filtering

We denote by $sim_{\mathsf{opt}}$ the similarity algorithm presented in Section 4.3. We implemented and ran $sim_{\mathsf{opt}}$ on all datasets and compared with the baseline procedure $sim_{\mathsf{cs}}$, which computes the similarity of every possible pair of values (Bahmani, Bertossi, and Vasiloglou 2017) occurring in every pair of sim positions appearing in rule bodies.

| Data | #At | #Cat | $\mathbf{t}_{\mathsf{cs}}$ | $\mathbf{t}_{\mathsf{opt}}$ | $\mathbf{M}_{\mathsf{cs}}$ | $\mathbf{M}_{\mathsf{opt}}$ | MRed. |
|---|---|---|---|---|---|---|---|
| *Dblp* | 5 | 10.2M | **96.3** | 531.4 | 512Mb | 256Mb | 50 |
| *Cora* | 17 | 0.8M | **5.49** | 932.9 | 32Mb | 4Mb | 87.5 |
| *Imdb* | 22 | 19.6M | **89.5** | 598.9 | 512Mb | 128Mb | 75 |
| *Mu* | 72 | 143.6M | **664.03** | 772.3 | 4Gb | 256Mb | 93.8 |
| *MuC* | 72 | 147.1M | 867.5 | **446.4** | 4Gb | 256Mb | 93.8 |
| *Poke* | 104 | 769.9M | 9,419 | **4,142** | 32Gb | 128Mb | 99.6 |

Table 1: #-columns denote the number of attributes, the sum of cross-products of constant pairs found in sim positions of a dataset. **MRed.** stands for the reduction rates of memory usage.

*Results.* Table 1 presents the results on execution time ($\mathbf{t}_{\{\mathsf{opt},\mathsf{cs}\}}$) and memory usage ($\mathbf{M}_{\{\mathsf{opt},\mathsf{cs}\}}$). We observe that $sim_{\mathsf{opt}}$ requires substantially less space than $sim_{\mathsf{cs}}$, with memory usage reduction rates of 50%, 87.5%, 75% and 99.6% on *Dblp*, *Cora*, *Imdb* and *Poke*, and 93.8% on *Mu*/*MuC*, respectively. Note that in general the reductions become substantially larger as the number of attributes increases from 5 to 104. This is because a key strength of $sim_{\mathsf{opt}}$ is to leverage joins on attributes as preconditions for two constants to be compared. Consequently, as the number of attributes within a schema increases, a proportional increase occurs in the preconditions that can be employed to restrict unnecessary comparisons.

As for running time, $sim_{\mathsf{cs}}$ tends to be faster than $sim_{\mathsf{opt}}$ in datasets of smaller scale, yielding speed advantages of 5.5, 6.5 and 100+ times on *Dblp*, *Imdb* and *Cora*, respectively. Note that as *#Cat* increases this tendency is inversed: on *Mu* both have similar running times, and then on *MuC* and *Poke* , $sim_{\mathsf{opt}}$ becomes 2 times faster. This shift might be because time spent on similarity computations outweighs the $sim_{\mathsf{opt}}$ program execution time as the number of pairs to be compared grows quadratically for $sim_{\mathsf{cs}}$.

## 6.3 Main Results

We evaluated performances of the lower LB and upper bound UB approximate solutions, a single maximal solution MS-1[4], and all possible merges PM. We consider MS-1 as the default output of ASPEN. Additionally, we compared these solutions with two (pairwise) rule-based ER systems: Magellan (Konda et al. 2016) and JedAI (Papadakis et al. 2020). Note that the two closest approaches, Dedupalog (Arasu, Ré, and Suciu 2009) and MRL (Deng et al.

[4]We do not consider all $n$ enumerated maximal solutions as there were only minimal variations among them.

| Data | Method | $F_1$ | (P / R) | | $t_o$ | $t_g$ | $t_s$ |
|---|---|---|---|---|---|---|---|
| *Dblp* | Magellan | 79.97 | 89.80 | 72.08 | **1.71** | - | - |
| | JedAI | 95.02 | **100** | 90.51 | 49.16 | - | - |
| | LB | 46.09 | 97.10 | 30.21 | 531.63 | **0.23** | **0.0039** |
| | MS-1 | **96.21** | 95.36 | 97.07 | 532.17 | 0.45 | 0.32 |
| | PM | 95.15 | 92.85 | **97.57** | 538.5 | 0.35 | 6.75 |
| | UB | 91.11 | 85.50 | **97.57** | 531.41 | - | - |
| *Cora* | Magellan | 79.70 | 93.09 | 69.68 | 131.13 | - | - |
| | JedAI | 90.53 | 95.53 | 87.72 | **40.92** | - | - |
| | LB | 83.57 | **99.80** | 71.87 | 1,008 | **7.85** | **0.29** |
| | MS-1 | **95.55** | 94.25 | 96.79 | 1,031 | 20.88 | 10.50 |
| | PM | **95.55** | 94.25 | 96.79 | 1,857.6 | 20.19 | 837.58 |
| | UB | 87.50 | 79.66 | **97.05** | 999.87 | - | - |

Table 2: Results on Pairwise Matching. Bold figures indicate the best performing results, and the second-best results are underlined.

2022), are not publicly available. To be comparable, we specified the same preconditions and similarity measures as in ASPEN, and followed the best performing setups for Magellan [5] and JedAI (Efthymiou et al. 2023). Note that since both systems lack native support for multi-relational table inputs and do not recognise inter-references between tables, for multi-relational datasets, we performed directly pairwise matching for each table (see Appendix D for details).

**Accuracy** The main results for the pairwise and multi-relational datasets are presented in Tables 2 and 3, respectively. The default output MS-1 consistently achieves the highest F1-score across all datasets, outperforming both Magellan and JedAI by significant margins. On the pairwise benchmarks *Dblp* and *Cora*, MS-1 surpasses Magellan by 16% and 1.1%, and JedAI by 15% and 5% respectively. Substantial performance differences are observed in the multi-relational datasets. When comparing MS-1 with Magellan and JedAI , improvements of 11% and 1.7%, 7.7% and 26%, 28% and 51%, and 81% and 86% are observed on *Imdb*, *Mu*, *MuC* and *Poke* respectively. This shows ASPEN is promising, particularly the multi-relational setup.

We also observed that different (approximate) ASPEN solutions might lead to different performances:

UB *vs* LB. The comparison between LB and UB reveals extreme results in precision and coverage. LB reached the highest precision (with an average of $> 99\%$) in all but one dataset, but has poor coverage with an average recall of $\approx 43\%$. Given that duplicated tuples often contain different versions of a value, it is not surprising that only few met the strong evidence required by hard rules. UB has the best coverage of the considered (approximate) solutions: on average $\geq 50\%$ higher recall than LB, 0.5%, 0.26%, 0.05%, 0.78% higher than MS-1 on *Dblp* , *Cora* , *Mu* /*MuC* , and slightly higher than PM on *Cora* and *Mu* /*MuC* . However, UB obtains the lowest precision of all solutions in all datasets. This difference can be explained by the presence of additional hard rules (the ones replacing soft rules) for UB, which allow for more merges but also introduce more false positives.

MS-*1 &* PM *vs* UB *&* LB. By contrast, the results for merge sets that make full use of LACE specifications are more balanced. On the one hand, MS-1 and PM outperformed LB in recall by $\approx 44\%$ on average, with slight sacrifices of precision on most datasets, only showing a considerable decrease on *MuC* (11% lower than LB). On the other hand, MS-1 and PM outperformed UB by significant precision margins of 10% and 7.3% on *Dblp* and 14% and 22.4%, on *Cora* and *Poke*, respectively, with only floating-point drops on recall. This highlights the effectiveness of combining soft rules and DCs to encourage the discovery of more merges while enforcing consistency to prevent false merges.

MS-*1 vs* PM. Observe that PM and MS-1 have identical results in both precision and recall in *Cora* , *Imdb* , *Mu* and *Poke*. A possible explanation is that when DCs are complementary to the soft rules in a specification, the combination behaves like hard rules (Bienvenu, Cima, and Gutiérrez-Basulto 2022). With no room for guessing, the specification derives a unique maximal solution, therefore MS-1 and PM include the same set of merges. Additionally, for *Dblp* and *MuC* , we see that precision of MS-1 respectively is 2.5% and 0.52% higher than that of PM. For these datasets, PM respectively obtained a 0.5% and 0.05% higher recall than MS-1. This underscores the characteristic differences between the two type of solutions: MS-1 prioritises precision while maintaining good coverage, whereas PM is more inclusive but may consequently contain more false positives.

**Running Time** The overall time, denoted as $t_o$, consists of preprocessing and ER time. Magellan and ASPEN involve preprocessing steps for blocking or similarity filtering, while JedAI interleaves similarity computation with the ER process, i.e., $t_o$ includes both. For ASPEN, the ER time is further composed of grounding and solving time $t_g$ and $t_s$. Note that $t_o$ may not always reflect the optimal running times of ASPEN. Indeed, when the simplest LB setup is considered, directly running an ASP encoding with online similarity evaluation is much faster (cf. Appendix B). However, to ensure a consistent analysis of the ER times of ASPEN, we present the $t_o$ as the sum of preprocessing and ER time.

As shown by the $t_o$ column of Tables 2 and 3, both Magellan and JedAI significantly outperform MS-1 across all datasets. Magellan achieved speed advantages of 7.8, 8, 13, 16, 150, 311 times on *Cora*, *MuC*, *Mu*, *Poke*, *Imdb* and *Dblp* respectively. Similarly, JedAI is 10, 25, 36, 8.5, 71 and 182 times faster across *Dblp* , *Cora* , *Imdb* , *Mu* , *MuC* and *Poke* respectively. This is largely due to more costly similarity computations in the preprocessing stage, which are necessary to achieve high quality results on the complex multi-relational settings. Indeed, in *Mu* , *MuC* and *Poke* , ASPEN obtains *substantially higher accuracy* than the baselines. We compare $t_o$ of other (approximate) solutions with that of Magellan and JedAI in Appendix D.

When looking at $t_g$ and $t_s$ of the different merge sets, LB is consistently the fastest, with $t_g$ averaging within half

[5] https://tinyurl.com/y6hupmrb

| Data | Method | $F_1$ | (P / R) | $t_o$ | $t_g$ | $t_s$ | Data | Method | $F_1$ | (P / R) | $t_o$ | $t_g$ | $t_s$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *Imdb* | Magellan | 88.09 | 99.80 78.83 | **3.89** | - | - | *Mu* | Magellan | 89.78 | 98.63 82.38 | **64.83** | - | - |
| | JedAI | 97.49 | 99.40 95.67 | 16.65 | - | - | | JedAI | 70.67 | 87.46 59.30 | 100.26 | - | - |
| | LB | 72.73 | **100** 57.15 | 600.65 | **1.73** | **0.027** | | LB | 64.08 | **99.79** 47.19 | 798.22 | **25.85** | **0.071** |
| | MS-1 | **99.27** | 99.39 **99.14** | 609.96 | 10.27 | 0.79 | | MS-1 | **97.52** | 99.25 95.58 | 853.91 | 79.75 | 1.86 |
| | PM | **99.27** | 99.39 **99.14** | 643.7 | 9.94 | 34.87 | | PM | **97.52** | 99.25 95.58 | 1,152.4 | 78.64 | 301.51 |
| | UB | **99.27** | 99.39 **99.14** | 598.9 | - | - | | UB | 97.44 | 99.03 **95.90** | 772.3 | - | - |
| *MuC* | Magellan | 55.54 | 97.51 38.83 | 66.87 | - | - | *Poke* | Magellan | 7.01 | 3.97 29.74 | 260.96 | - | - |
| | JedAI | 32.75 | 73.95 21.02 | **7.88** | - | - | | JedAI | 2.1 | 1.08 46.56 | **23.46** | - | - |
| | LB | 53.95 | **99.79** 36.97 | 474.56 | **28.01** | **0.062** | | LB | 28.00 | **100** 16.27 | 4,144 | **2.29** | **0.018** |
| | MS-1 | **84.10** | 88.11 80.44 | 562.37 | 113.64 | 2.33 | | MS-1 | **88.71** | 92.88 **84.90** | 4,271.8 | 127.67 | 2.17 |
| | PM | 83.87 | 87.59 80.46 | 893.44 | 113.26 | 333.78 | | PM | **88.71** | 92.88 **84.90** | 4,296 | 129.04 | 25.84 |
| | UB | 83.55 | 86.01 **81.24** | 446.4 | - | - | | UB | 77.00 | 70.43 **84.90** | 4,142 | - | - |

Table 3: Results on Complex Multi-relational Datasets

a minute and $t_s$ concluding in fractions of a second. This performance is expected, given that LB straightforwardly derives a single set of merges. Regarding MS-1, $t_g$ took the majority of ER time, while solving is consistently much more efficient, terminating in $\leq 10$ seconds. For PM, $t_g$ behaved as for MS-1, but it required longer solving time (100 times longer in the worst case). This might be explained by the need to perform brave reasoning. We note that the size of a dataset may not always be the main factor impacting the solving time. For instance, despite containing only 1.9k tuples, $t_s$ of MS-1 and PM on the *Cora* dataset are noticeably longer than on much larger datasets like *Mu* and *Poke*. Similarly, despite *Mu* and *MuC* being of the same size, the $t_g$ of UB, MS-1 and PM on these datasets are very different.

## 6.4 Effect of Recursion

We executed the levels algorithm described in Section 4.2 on (approximate) solutions derived in our previous experiments on multi-relational datasets. We report accuracy results of UB and MS-1 on the *Poke* dataset and merge increments for various recursion levels of *MuC* and *Poke* in Figures 3 and 4, respectively. Additional results for other datasets can be found in Appendix E. We can observe that achieving convergence of merges requires more than one recursion level. For example, in Figure 3, *Poke* converged at level 2 (red dashed line). Figure 4 shows noticeable increments of merges obtaining 17.8% and 18.8%, 2.4% and 18.8% increment gains in the MS-1 and UB settings on *MuC* and *Poke*, respectively. This shows the efficacy of recursion in discovering new merges utilising merges derived from previous levels.

We present further results on the interplay of recursion and DCs and dataset characteristics in Appendix E.

## 6.5 Efficiency

*Datalog approximations.* We examined the running times of Datalog programs $\Pi_{\Sigma^{lb}}$ and $\Pi_{\Sigma^{ub}}$ on all datasets using ASPEN and the rule engine VLog4j (Carral et al. 2019; Urbani, Jacobs, and Krötzsch 2016). For LB, ASPEN outperforms

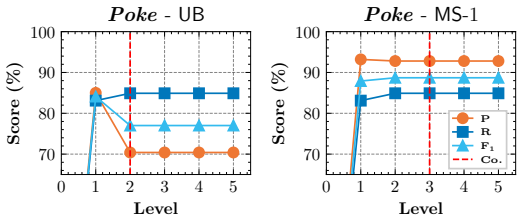



Figure 3: Impact of Recursion on Accuracy

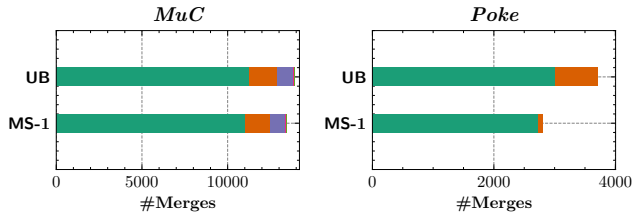



Figure 4: Merge Increments on Levels

VLog4j on most datasets, whereas for UB, VLog4j is faster on all multi-relational datasets except *Poke*. These results suggest that the performance of different reasoning engines is impacted by characteristics of the data. See Appendix F for more details.

*Factors impacting scalability.* Our efficiency analysis shows that both $t_g$ and $t_s$ increase monotonically with the *size of the data* and the *percentage of duplicates*. The impact on $t_s$ is particularly significant for PM, increasing up to 52 times when these factors are raised by fivefold. Similarly, lowering *similarity thresholds* consistently increases $t_g$, while $t_s$ is up to 300 times longer on MS-1 and reaches a time-out ($\geq$24h) on PM. For more details, see Appendix F.

# 7 Discussion

We have introduced ASPEN, an ASP-based implementation of the LACE framework for collective, explainable, and re-

cursive ER. Distinguishing features of ASPEN are the consideration of a space of maximal solutions and the ability to supply explanations of derived merges. It is also one of only a handful of systems to natively support collective ER tasks, involving multiple database tables and entity types. A comprehensive experimental evaluation provided insights into how ASPEN performs, in terms of quality metrics and runtime, depending upon which notion of (approximate) solution is employed and how its performance compares to two baseline rule-based ER systems. The overall takeaway is that ASPEN is promising, as it is able to terminate in a reasonable amount of time (ER is typically an occasional offline task) and is competitive and often outperforms the baseline systems w.r.t. quality metrics, in particular, being able to effectively handle more complex multi-relational scenarios.

Our paper showcases entity resolution as an exciting but challenging application for ASP. Indeed, we believe that ER is as an ideal testbed for ASP techniques, as it is an important practical problem that naturally involves many solver functionalities, such as: brave and cautious reasoning (to identify possible and certain merges), preferred answers sets (to generate or reason over maximal solutions), external function calls (for similarity computation), and explanation facilities (to produce justifications of merges). By making ASPEN's code and data publicly available, we hope to facilitate future research on ASP-based approaches to ER.

While our experiments show that ASPEN can successfully handle some real-world ER scenarios, scalability remains an issue, and we expect that both general purpose and dedicated optimizations will be needed to be able to scale up to larger datasets and support even more complex reasoning over ER solutions. In addition to continuing to improve the similarity computation phase, we plan to explore the potential of employing specialized data structures or custom procedures for handling equivalence relations, as has been considered for Datalog reasoners (Nappa et al. 2019; Sahebolamri et al. 2023). As parallelization has been successfully employed in some rule-based ER systems (Deng et al. 2022), another promising but non-trivial direction would be to see how parallel algorithms can be integrated into ASPEN. For this, we hope to build upon existing work on parallelization of Datalog reasoning (Perri, Ricca, and Sirianni 2013; Ajileye and Motik 2022) and ASP solving (Gebser, Kaufmann, and Schaub 2012).

We also plan to extend ASPEN to handle more expressive ER scenarios, building upon recent extensions to the LACE framework. Our top priority will be support not only global merges of entity-referencing constants, as considered in ASPEN and the original LACE framework, but also local (cell-level) merges of value constants (Bienvenu et al. 2023), so that e.g. some occurrences of 'J. Smith' can be matched to 'Joe Smith' while others are matched to 'Jane Smith'. Another important extension, requiring more significant changes to the ASP encoding, would be to allow for both merges and repair operations, as in the REPLACE framework (Bienvenu, Cima, and Gutiérrez-Basulto 2023), in order to be able to handle constraint violations that cannot be solved solely via merges.

## Acknowledgements

The authors were partially supported by the ANR AI Chair INTENDED (ANR-19-CHIA-0014) and by MUR under the PNRR project FAIR (PE0000013). The authors thank the Potassco community for their support in resolving the issues encountered while using `clingo`.

## References

Ahmetaj, S.; David, R.; Polleres, A.; and Simkus, M. 2022. Repairing SHACL constraint violations using answer set programming. In Sattler, U.; Hogan, A.; Keet, C. M.; Presutti, V.; Almeida, J. P. A.; Takeda, H.; Monnin, P.; Pirrò, G.; and d'Amato, C., eds., *The Semantic Web - ISWC 2022 - 21st International Semantic Web Conference, Virtual Event, October 23-27, 2022, Proceedings*, volume 13489 of *Lecture Notes in Computer Science*, 375–391. Springer.

Ajileye, T., and Motik, B. 2022. Materialisation and data partitioning algorithms for distributed RDF systems. *J. Web Semant.* 73:100711.

Arasu, A.; Ré, C.; and Suciu, D. 2009. Large-scale deduplication with constraints using dedupalog. In Ioannidis, Y. E.; Lee, D. L.; and Ng, R. T., eds., *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, 952–963. IEEE Computer Society.

Bahmani, Z., and Bertossi, L. E. 2017. Enforcing relational matching dependencies with datalog for entity resolution. In Rus, V., and Markov, Z., eds., *Proceedings of the Thirtieth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2017, Marco Island, Florida, USA, May 22-24, 2017*, 718–723. AAAI Press.

Bahmani, Z.; Bertossi, L. E.; and Vasiloglou, N. 2017. Erblox: Combining matching dependencies with machine learning for entity resolution. *Int. J. Approx. Reason.* 83:118–141.

Bertossi, L. E.; Kolahi, S.; and Lakshmanan, L. V. S. 2013. Data cleaning and query answering with matching dependencies and matching functions. *Theory Comput. Syst.* 52(3):441–482.

Bertossi, L. E. 2011. *Database Repairing and Consistent Query Answering*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers.

Bhattacharya, I., and Getoor, L. 2007. Collective entity resolution in relational data. *ACM Trans. Knowl. Discov. Data* 1(1):5.

Bienvenu, M.; Cima, G.; Gutiérrez-Basulto, V.; and Ibáñez-García, Y. 2023. Combining global and local merges in logic-based entity resolution. In Marquis, P.; Son, T. C.; and Kern-Isberner, G., eds., *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning, KR 2023, Rhodes, Greece, September 2-8, 2023*, 742–746.

Bienvenu, M.; Cima, G.; and Gutiérrez-Basulto, V. 2022. LACE: A logical approach to collective entity resolution. In Libkin, L., and Barceló, P., eds., *PODS '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, 379–391. ACM.

Bienvenu, M.; Cima, G.; and Gutiérrez-Basulto, V. 2023. REPLACE: A logical framework for combining collective entity resolution and repairing. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, 3132–3139. ijcai.org.

Brewka, G.; Delgrande, J. P.; Romero, J.; and Schaub, T. 2015. asprin: Customizing answer set preferences without a headache. In Bonet, B., and Koenig, S., eds., *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, 1467–1474. AAAI Press.

Brewka, G.; Eiter, T.; and Truszczynski, M. 2011. Answer set programming at a glance. *Commun. ACM* 54(12):92–103.

Burdick, D.; Fagin, R.; Kolaitis, P. G.; Popa, L.; and Tan, W. 2016. A declarative framework for linking entities. *ACM Trans. Database Syst.* 41(3):17:1–17:38.

Cabalar, P., and Muñiz, B. 2023. Explanation graphs for stable models of labelled logic programs. In Arias, J.; Batsakis, S.; Faber, W.; Gupta, G.; Pacenza, F.; Papadakis, E.; Robaldo, L.; Rückschloß, K.; Salazar, E.; Saribatur, Z. G.; Tachmazidis, I.; Weitkämper, F.; and Wyner, A. Z., eds., *Proceedings of the International Conference on Logic Programming 2023 Workshops co-located with the 39th International Conference on Logic Programming (ICLP 2023), London, United Kingdom, July 9th and 10th, 2023*, volume 3437 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Cabalar, P.; Fandinno, J.; and Muñiz, B. 2020. A system for explainable answer set programming. In Ricca, F.; Russo, A.; Greco, S.; Leone, N.; Artikis, A.; Friedrich, G.; Fodor, P.; Kimmig, A.; Lisi, F. A.; Maratea, M.; Mileo, A.; and Riguzzi, F., eds., *Proceedings 36th International Conference on Logic Programming (Technical Communications), ICLP Technical Communications 2020, (Technical Communications) UNICAL, Rende (CS), Italy, 18-24th September 2020*, volume 325 of *EPTCS*, 124–136.

Calimeri, F.; Faber, W.; Gebser, M.; Ianni, G.; Kaminski, R.; Krennwallner, T.; Leone, N.; Maratea, M.; Ricca, F.; and Schaub, T. 2020. Asp-core-2 input language format. *Theory Pract. Log. Program.* 20(2):294–309.

Carral, D.; Dragoste, I.; González, L.; Jacobs, C. J. H.; Krötzsch, M.; and Urbani, J. 2019. Vlog: A rule engine for knowledge graphs. In Ghidini, C.; Hartig, O.; Maleshkova, M.; Svátek, V.; Cruz, I. F.; Hogan, A.; Song, J.; Lefrançois, M.; and Gandon, F., eds., *The Semantic Web - ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part II*, volume 11779 of *Lecture Notes in Computer Science*, 19–35. Springer.

Christophides, V.; Efthymiou, V.; Palpanas, T.; Papadakis, G.; and Stefanidis, K. 2021. An overview of end-to-end entity resolution for big data. *ACM Comput. Surv.* 53(6):127:1–127:42.

Deng, T.; Fan, W.; Lu, P.; Luo, X.; Zhu, X.; and An, W. 2022. Deep and collective entity resolution in parallel. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*, 2060–2072. IEEE.

Efthymiou, V.; Ioannou, E.; Karvounis, M.; Koubarakis, M.; Maciejewski, J.; Nikoletos, K.; Papadakis, G.; Skoutas, D.; Velegrakis, Y.; and Zeakis, A. 2023. Self-configured entity resolution with pyjedai. In He, J.; Palpanas, T.; Hu, X.; Cuzzocrea, A.; Dou, D.; Slezak, D.; Wang, W.; Gruca, A.; Lin, J. C.; and Agrawal, R., eds., *IEEE International Conference on Big Data, BigData 2023, Sorrento, Italy, December 15-18, 2023*, 339–343. IEEE.

Eiter, T.; Fink, M.; Greco, G.; and Lembo, D. 2008. Repair localization for query answering from inconsistent databases. *ACM Trans. Database Syst.* 33(2):10:1–10:51.

Fan, W., and Geerts, F. 2012. *Foundations of Data Quality Management*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2012. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Gebser, M.; Leone, N.; Maratea, M.; Perri, S.; Ricca, F.; and Schaub, T. 2018. Evaluation techniques and systems for answer set programming: a survey. In Lang, J., ed., *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, 5450–5456. ijcai.org.

Gebser, M.; Kaufmann, B.; and Schaub, T. 2012. Multi-threaded ASP solving with clasp. *Theory Pract. Log. Program.* 12(4-5):525–545.

Kaminski, R.; Romero, J.; Schaub, T.; and Wanko, P. 2023. How to build your own asp-based system?! *Theory Pract. Log. Program.* 23(1):299–361.

Konda, P.; Das, S.; C., P. S. G.; Doan, A.; Ardalan, A.; Ballard, J. R.; Li, H.; Panahi, F.; Zhang, H.; Naughton, J. F.; Prasad, S.; Krishnan, G.; Deep, R.; and Raghavendra, V. 2016. Magellan: Toward building entity matching management systems. *Proc. VLDB Endow.* 9(12):1197–1208.

Köpcke, H.; Thor, A.; and Rahm, E. 2010. Evaluation of entity resolution approaches on real-world match problems. *Proc. VLDB Endow.* 3(1):484–493.

Li, Y.; Li, J.; Suhara, Y.; Doan, A.; and Tan, W. 2020. Deep entity matching with pre-trained language models. *Proc. VLDB Endow.* 14(1):50–60.

Lifschitz, V. 2019. *Answer Set Programming*. Springer.

Manna, M.; Ricca, F.; and Terracina, G. 2013. Consistent query answering via ASP from different perspectives: Theory and practice. *Theory Pract. Log. Program.* 13(2):227–252.

Nappa, P.; Zhao, D.; Subotic, P.; and Scholz, B. 2019. Fast parallel equivalence relations in a datalog compiler. In *28th International Conference on Parallel Architectures and Compilation Techniques, PACT 2019, Seattle, WA, USA, September 23-26, 2019*, 82–96. IEEE.

Papadakis, G.; Mandilaras, G. M.; Gagliardelli, L.; Simonini, G.; Thanos, E.; Giannakopoulos, G.; Bergamaschi, S.; Palpanas, T.; and Koubarakis, M. 2020.

Three-dimensional entity resolution with jedai. *Inf. Syst.* 93:101565.

Perri, S.; Ricca, F.; and Sirianni, M. 2013. Parallel instantiation of ASP programs: techniques and experiments. *Theory Pract. Log. Program.* 13(2):253–278.

Sahebolamri, A.; Barrett, L.; Moore, S.; and Micinski, K. K. 2023. Bring your own data structures to datalog. *Proc. ACM Program. Lang.* 7(OOPSLA2):1198–1223.

Singla, P., and Domingos, P. M. 2006. Entity resolution with markov logic. In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006), 18-22 December 2006, Hong Kong, China*, 572–582. IEEE Computer Society.

Terracina, G.; Martello, A.; and Leone, N. 2013. Logic-based techniques for data cleaning: An application to the italian national healthcare system. In Cabalar, P., and Son, T. C., eds., *Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings*, volume 8148 of *Lecture Notes in Computer Science*, 524–529. Springer.

Tran, K.; Vatsalan, D.; and Christen, P. 2013. Geco: an online personal data generator and corruptor. In He, Q.; Iyengar, A.; Nejdl, W.; Pei, J.; and Rastogi, R., eds., *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*, 2473–2476. ACM.

Urbani, J.; Jacobs, C. J. H.; and Krötzsch, M. 2016. Column-oriented datalog materialization for large knowledge graphs. In Schuurmans, D., and Wellman, M. P., eds., *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, 258–264. AAAI Press.

Wang, T.; Lin, H.; Fu, C.; Han, X.; Sun, L.; Xiong, F.; Chen, H.; Lu, M.; and Zhu, X. 2022. Bridging the gap between reality and ideality of entity matching: A revisting and benchmark re-constrcution. In Raedt, L. D., ed., *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, 3978–3984. ijcai.org.

# A Proof Trees and Explanations

*Proof Trees.* We shall employ proof trees to explain why a merge $\alpha$ appears in a solution $E \in Sol(D, \Sigma)$. Formally, we define a *proof tree for $\alpha$ in $E$* as a node-labelled tree such that (a) the root node has label $\alpha$, (b) every leaf node is labelled with a fact from $D$, and (c) every non-leaf node $n$ is labelled with a pair of constants $(d, e)$ such that one of the following holds:

- node $n$ has exactly two children, which have labels $(d, f)$ and $(f, e)$ for some constant $f$ (*transitive node*)
- there is a rule[6] $P_1(v_1^1, \ldots, v_1^{w_1}) \wedge \ldots \wedge P_m(v_m^1, \ldots, v_m^{w_m}) \rightarrow \mathsf{Eq}(x, y) \in \Gamma$ such that node $n$ has $m$ children labelled with the facts $P_1(c_1^1, \ldots, c_1^{w_1}), \ldots, P_m(c_m^1, \ldots, c_m^{w_m}) \in D$, there exist $v_i^j = x$ and $v_k^\ell = y$ such that $\{c_i^j, c_k^\ell\} = \{d, e\}$, and additionally, whenever $v_i^j = v_k^\ell$ and $c_i^j \neq c_k^\ell$, then $n$ has a child labelled with the pair $(c_i^j, c_k^\ell)$ or $(c_k^\ell, c_i^j)$ (*rule node*)

Observe that each node corresponds to a single transitive step or rule application, while reflexivity and symmetry steps are left implicit. Also note that when database facts used to satisfy the rule body do not 'join', additional children are introduced to ensure that the required merges exist. It is not hard to see that every non-trivial merge $\alpha = (c, d) \in E$ (i.e. with $c \neq d$) has at least one proof tree. A proof tree for the merge $(s_1, s_2)$ in solution $E_1$ is presented in Fig. 5.

We shall also be interested in quantifying the number of successive rule applications needed to obtain a given merge. To this end, we define the *rule-depth of a proof tree* $T$ as the maximum number of rule nodes in any leaf-to-root path in $T$. The *level of a merge $\alpha$ in a solution $E$* is 0 if $\alpha = (c, c)$ for some $c \in \mathbf{C}$, and otherwise is defined as the minimum rule-depth of all proof trees of $\alpha$ in $E$. $(s_1, s_2)$ has level 2 in solution $E_1$, as it possesses a proof tree with rule-depth 2 (and no proof tree with rule-depth 1).

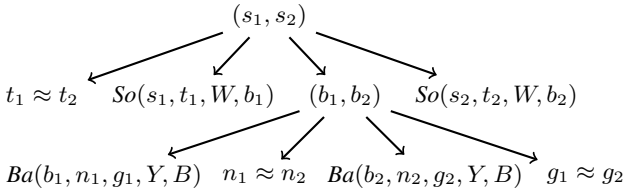



Figure 5: Proof tree for merge $(s_1, s_2)$ in solution $E_1$. In the tree, *So* and *Ba* stand for the relations *Song* and *Band*. $W$, $Y$, and $B$ stand for the constants Waters, 1965, and Barrett, respectively. $n_i$, $g_i$, and $t_i$ are the name and genre of the band $b_i$ and the title of song $s_i$, respectively.

*Example Proof Tree from* ASPEN. An exemplary proof tree generated by ASPEN is shown in Figure 6.

[6]To avoid an overly lengthy definition, we focus on rules without constants, but the definition generalises to rules with constants.

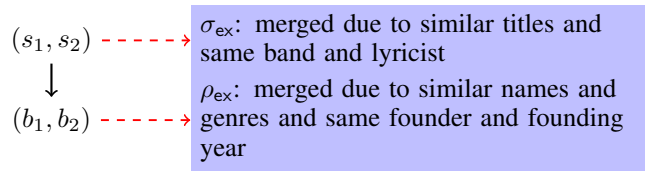



Figure 6: Example of a graphical proof tree provided by ASPEN relative to merge $(s_1, s_2)$ in solution $E_1$ for $(D_{\mathsf{ex}}, \Sigma_{\mathsf{ex}})$.

*A Qualitative Case.* In *Mu* , the specification contains a hard rule $h_1$ and a soft rule $s_3$ declaring merges of album releases. $h_1$: *"two album releases are the same if they have the same barcode"* and $s_3$: *"two album releases are possibly the same if they have similar names and the same list of artists"*. Additionally, it has a hard rule $h_2$ related to release groups: *"two release groups are the same if they have the same release and similar names"*.

Utilising xclingo (Cabalar, Fandinno, and Muñiz 2020), ASPEN generates two proof trees for $(rg_1, rg_2)$. Figure 7 illustrates how the merge $(rg_1, rg_2)$ of release group can be justified in a solution. We observe that the solution generated two proof trees for $(rg_1, rg_2)$ as seen in Figures 7a and 7b.

# B Similarity Computation

*Loose Upper-bound.* Given an upper-bound transformation $\Sigma^{\mathsf{ub}}$ of a LACE specification $\Sigma$, let $\Sigma^{\mathsf{ub}}_{\mathsf{op}}$ be the specification obtained from $\Sigma^{\mathsf{ub}}$ by dropping every similarity atom in rule bodies. From $\Sigma^{\mathsf{ub}}_{\mathsf{op}}$, we obtain a Datalog program $P^{\mathsf{ub}}_{\mathsf{op}}$. If we run $P^{\mathsf{ub}}_{\mathsf{op}}$ over the database $D$, then the extension of `eq` will contain a loose upper-bound of the pairs of constants that can be potentially merged. An instance of such rule transformed from $\sigma_{\mathsf{ex}}$ in Figure 1 will be:

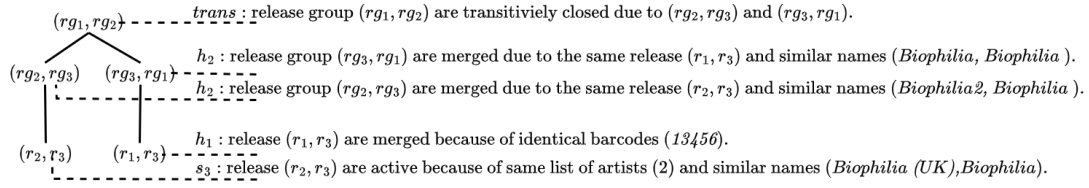
$$\texttt{eq}(X, Y) \leftarrow \texttt{song}(X, T, L, B), \texttt{song}(Y, T', L, B'), \\ \texttt{not empty}(L), \texttt{eq}(B, B').$$

We executed the loose upper-bound program for each specification on the datasets on VLog4j (Urbani, Jacobs, and Krötzsch 2016; Carral et al. 2019). Only on *Dblp* and *Cora* the programs were able to terminate without throwing errors of *memory overflow*. We recorded the running times and number of `eq`-facts, comparing with sum of the size of cross products of each merge position pair.

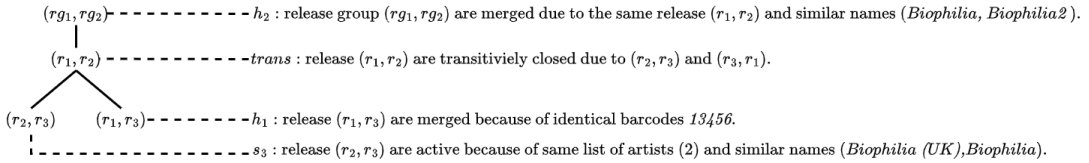
As shown by Table 4, the resulting sets of `eq`-facts for *Dblp* and *Cora* barely differ to *#Cat*.

| **Data** | **#eq** | **#Cat** | **t**(s) |
|---|---|---|---|
| *Dblp* | 6,006k | 6,006k | 35.54 |
| *Cora* | 3,530k | 3,534k | 23.75 |

Table 4: Size of `eq`-facts in loose upper-bound. #-columns denote the number of `eq`-facts, the sum of cross-products of constant pairs found in merge positions of a dataset.




(a) Proof Tree 1 for $(rg_1, rg_2)$




(b) Proof Tree 2 for $(rg_1, rg_2)$

Figure 7: Proof trees of a merge in the *Mu* dataset

*Online vs Preprocessed Similarity.* In principle, one can run directly an ASP program $\Pi_{(D,\Sigma_{\mathsf{ext}})}$ by replacing literals $\mathtt{sim}(T, T', S), S \geq \delta$ with $\mathtt{sim}_{ext}\,(\mathtt{X}, \mathtt{Y}) \geq \delta$ in $\Sigma$ (and its variants) instead of the method in Section 4.3. We conducted a set of experiments comparing the overall running times of ASPEN when using the approach in Section 4.3 (denoted as $\mathsf{Sim}_{\mathsf{opt}}$) and online similarity evaluation (denoted as $\mathsf{Sim}_{\mathsf{ext}}$) on the datasets and reported the running times as Table 5.

We observe that for the simpler solution types LB and UB, using the external similarity call directly consistently leads to faster termination times, especially for LB. Indeed, if only LB or UB are needed, the similarity algorithm is unnecessary. For the more complex settings MS-1 and PM, we found that online similarity calculations provided speed advantages on *Dblp* , *Cora* , and *MuC* , with improvements of 31%/32%, 4%/3.6%, and 41%/25%, respectively. Conversely, our similarity method performed better on most multi-relational datasets, with improvements of 1.4%/4.3%, 27.5%/22%, and 16%/18% on *Imdb* , *Mu* , and *Poke* , respectively. These results suggest that, apart from being able to reuse materialised output ($\mathsf{Sim}_{\mathsf{opt}}$ and the upper-bound merge set $U$) for deriving different solutions, our similarity method can also *speed up* the computation of solutions in complex settings.

**Theorem 4**

*Proof Sketch:.* Let $M^{\mathsf{opt}}_{\mathtt{eq}} := \{M_{\mathtt{eq}} \mid M \in \mathit{SM}(\Pi_{(D_{\mathsf{opt}},\Sigma)})\}$ and $M^{\mathsf{all}}_{\mathtt{eq}} := \{M_{\mathtt{eq}} \mid M \in \mathit{SM}(\Pi_{(D_{\mathsf{all}},\Sigma)})\}$. To show that $M^{\mathsf{opt}}_{\mathtt{eq}} \subseteq M^{\mathsf{all}}_{\mathtt{eq}}$, we use the observation that $D_{\mathsf{opt}} \subseteq D_{\mathsf{all}}$, and that the rules for encoding hard and soft rules from $\Sigma$ in both programs are the same. Let $\mathtt{eq}(e, e') \in M^{\mathsf{opt}}_{\mathtt{eq}}$, then there is a stable model $M$ of $\Pi_{(D_{\mathsf{opt}},\Sigma)}$ containing $\mathtt{eq}(e, e') \in M^{\mathsf{opt}}_{\mathtt{eq}}$. Further, there is a rule $\sigma$ in the grounding of $\Pi_{(D_{\mathsf{opt}}}$, with $\mathtt{eq}(e, e')$ in the head, with all the atoms $S$ in its body contained in $M$. Using the observation, we can argue that there is a stable model $M'$ of $\Pi_{(D_{\mathsf{all}}}$ containing $S$ and therefore $\mathtt{eq}(e, e') \in M'$, which in turn implies $\mathtt{eq}(e, e') \in M^{\mathsf{all}}_{\mathtt{eq}}$.

To prove that $M^{\mathsf{all}}_{\mathtt{eq}} \subseteq M^{\mathsf{opt}}_{\mathtt{eq}}$. Let $\mathtt{eq}(e, e') \in M^{\mathsf{opt}}_{\mathtt{eq}}$, and let $M' \in \mathit{SM}(\Pi_{(D_{\mathsf{all}},\Sigma)})$ a stable model that contains $\mathtt{eq}(e, e')$. Then there is a ground rule $\sigma$ rule in the reduct of $\Pi^{M}_{(D_{\mathsf{all}},\Sigma)}$ and a set of ground atoms $S$ in $M$ that support $\mathtt{eq}(e, e')$. If the body of $\sigma$ does not contain an $\mathtt{eq}$ atom, then we note that all similarity atoms in the body of $\sigma$ are included in $D_{\mathsf{opt}}$ by the computation in Phase 3, and because $D_{\mathsf{opt}}$ occurs in every (stable) model of $\Pi_{(D_{\mathsf{opt}},\Sigma)}$ and both programs contain the same rules encoding $\Sigma$, is easy to see that $\mathtt{eq}(e, e')$ occurs in a stable model of $\Pi_{(D_{\mathsf{opt}},\Sigma)}$. For the case where $\sigma$ contains $\mathtt{eq}$ atoms, we can use an inductive argument, with the previous case being the base of the induction. An important observation to construct the argument is that all the similarity atoms used in the derivation of $\mathtt{eq}(e, e')$ w.r.t $\Pi^{M}_{(D_{\mathsf{all}},\Sigma)}$ are added to $\Pi_{(D_{\mathsf{opt}},\Sigma)}$ either in Phase 1 or in Phase 3 of the similarity computation.

# C Experimental Setup

*Datasets.* Statistics of the datasets used in our experiments are shown in Table 6. Note that regardless of the number of relations and attributes, datasets with a larger number of referential constraints are structurally more complex. Sources of the original databases can be found at: **i)** *Cora* : https://hpi.de/naumann/projects/repeatability/datasets/cora-dataset.html, **ii)** *Mu* : https://musicbrainz.org/doc/MusicBrainz_Database/Schema, **iii)** *Poke* : https://pokemondb.net/about.

*Similarity Measures and Metrics.* We calculate the syntactic similarities of constants based on their data types (Bahmani, Bertossi, and Vasiloglou 2017). **(i)** For numerical constants, we use the Levenshtein distance; **(ii)** for short string constants (length$<$ 25), we compute the score as the editing distance of two character sequences (Jaro-Winkler distance); **(iii)** for long-textual constants (length$\geq$ 25), we use the TF-IDF cosine score as the syntactic similarity measure. We adopt the commonly used metrics (Köpcke, Thor, and Rahm 2010) of *Precision* (**P**), *Recall* (**R**) and *F1-Score* ($\mathbf{F_1}$) to examine the solutions derived from the specifications. Pre-

| Data | Met. | $t_o$ | $t_{oo}$ | $t_{og}$ | $t_{pg}$ | $t_s$ | Data | Met. | $t_o$ | $t_{oo}$ | $t_{og}$ | $t_{pg}$ | $t_s$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *Dblp* $t_p = 531$ | LB | 531.63 | 17.12 | 17.12 | 0.23 | 0.0039 | *Cora* $t_p = 999$ | LB | 1,008.01 | 133.02 | 132.73 | 7.85 | 0.29 |
| | MS-1 | 532.17 | 362.58 | 362.26 | 0.45 | 0.32 | | MS-1 | 1,031.25 | 987.75 | 977.25 | 20.88 | 10.5 |
| | PM | 538.5 | 364.22 | 357.47 | 0.35 | 6.75 | | PM | 1,857.64 | 1,789.35 | 951.77 | 20.19 | 837.58 |
| | UB | 531.4 | 364.06 | 364.05 | 0 | 0.0098 | | UB | 999.87 | 998.27 | 997.77 | 0 | 0.5 |
| *Imdb* $t_p = 598$ | LB | 600.65 | 78.88 | 78.86 | 1.73 | 0.027 | *Mu* $t_p = 772$ | LB | 598.9 | 34.43 | 34.36 | 25.85 | 0.071 |
| | MS-1 | 609.96 | 618.62 | 617.83 | 10.27 | 0.79 | | MS-1 | 853.91 | 1,178.8 | 1,176.94 | 79.75 | 1.86 |
| | PM | 643.71 | 672.92 | 638.05 | 9.94 | 34.87 | | PM | 1,152.45 | 1494.9 | 1,193.39 | 78.64 | 301.51 |
| | UB | 598.94 | 535.06 | 535.012 | 0 | 0.045 | | UB | 772.41 | 698.91 | 698.8 | 0 | 0.11 |
| *MuC* $t_p = 446$ | LB | 474.56 | 29.43 | 29.37 | 28.1 | 0.062 | *Poke* $t_p = 4,142$ | LB | 4,144.31 | 7.28 | 7.26 | 2.29 | 0.018 |
| | MS-1 | 562.37 | 331.68 | 329.35 | 113.64 | 2.33 | | MS-1 | 4,271.86 | 5,093.16 | 5,090.99 | 127.69 | 2.17 |
| | PM | 893.44 | 665.53 | 331.7 | 113.26 | 333.78 | | PM | 4,296.8 | 5,241.05 | 5,215.21 | 129.04 | 25.84 |
| | UB | 446.51 | 306.44 | 306.33 | 0 | 0.11 | | UB | 4,142 | 4,130.03 | 4,130 | 0 | 0.035 |

Table 5: Comparison on solution computation time using online and preprocessed similarity computation. $t_o$ and $t_{oo}$ record the overall running times of $\mathsf{Sim}_{\mathsf{opt}}$ and $\mathsf{Sim}_{\mathsf{ext}}$ respectively. $t_p$ denotes the preprocessing time on a dataset. $t_{og}$ and $t_{pg}$ are the grounding time of online and preprocessed similarity respectively, and $t_s$ is the solving time. Figures in red/green represent worse/better performances between $t_o$ and $t_{oo}$.

| Name | #Rec | #Rel | #At | #Ref | #Dup |
|---|---|---|---|---|---|
| *Dblp* | 5k | 2 | $5^b$ | 0 | 2.2k |
| *Cora* | 1.9k | 1 | 17 | 0 | 64k |
| *Imdb* | 30k | 5 | 22 | 4 | 6k |
| *Mu* | 41k | 11 | 72 | 12 | 15k |
| *MuC* | 41k | 11 | 72 | 12 | 15k |
| *Poke* | 240k | 20 | 104 | 20 | 4k |

Table 6: Dataset Statistics. #-columns represent the number of records, relations, attributes, referential constraints and duplicates, respectively. $\cdot^b$: the DBLP and ACM tables share the 5 attributes.

cision reflects the percentage of true merges in a solution and Recall indicates the coverage of true merges in a solution relative to the ground truth. The quality of a solution is then measured as $\text{F1} = 2 \times \text{Precision} \times \text{Recall}/(\text{Precision} + \text{Recall})$.

*Clean Data Sampling.* We describe the process of data sampling and synthesising duplication for *Mu*/*MuC* and *Poke* datasets. Note that it is important to preserve the relation between entities from different relations when corrupting the instances to retain interdependencies of the data. Thus, we consider the referential dependency graphs of the schema when creating the datasets. Assuming the original schema instances of the *Mu* and *Poke* are clean, we sampled tuples from each table and created clean partitions of the instances. In particular, we started from the relations with zero in-degree and sampled for each step the adjacent referenced entity relations that have all their referencing relations sampled. Since tuples from relations store only foreign keys do not represent entities, they were sampled only after one of their referencing relations were already sampled.

In Figure 8, green nodes and blue nodes respectively represent entity and non-entity relations in the *Mu* schema and edges represent referential constraints from an out-node to an in-node labelled with the corresponding foreign key. In this case, we begin sampling from the entity relations `Track`, `Place`, `Label` since they are not referenced by any other relations. As `Artist_Credit` relation is also referenced by many other relations, in the second step we sample only those that are adjacent to `Track` and with all in-arrows sampled, i.e., the `Medium` and `Recording` relations. Entities of other relations are sampled analogously. Note that since the non-entity relation `Artist_Credit_Name` stores mappings between `Artist_Credit` and `Artist`, it is sampled only after the entities of `Artist_Credit` are picked. We are then able to proceed sampling from `Artist` when `Artist_Credit_Name` is selected. Consequently, clean partitions of the schema instances can be obtained from sampling. The sampling procedure is done by an extra ASP program as a part of data preprocessing step.

*Duplicates Generation.* The original *Mu* and *Poke* datasets are clean, so we synthesised and injected duplicates to create the datasets with duplicates employed in our experiments. To this end, we utilised the Geco corruptor (Tran, Vatsalan, and Christen 2013). The corruptor randomly picked entities from the clean instances and generated per record up to 3 duplicates. Errors of different types, such as keyboard input errors, OCR (characters visually similar) errors, and null values were injected following a predefined distribution across tuple attributes to ensure uniqueness of the duplicates created. Importantly, tuples may contain foreign keys, so it is undesirable that the generated duplicates are not referenced by other tuples at all. Hence, for each tuple with foreign keys, we replaced each foreign key $k$ (which we always consider as merge attributes) with an identifier randomly drawn from the equivalence class of $k$ (including $k$ and its duplicates) as a type of error injection after creat-

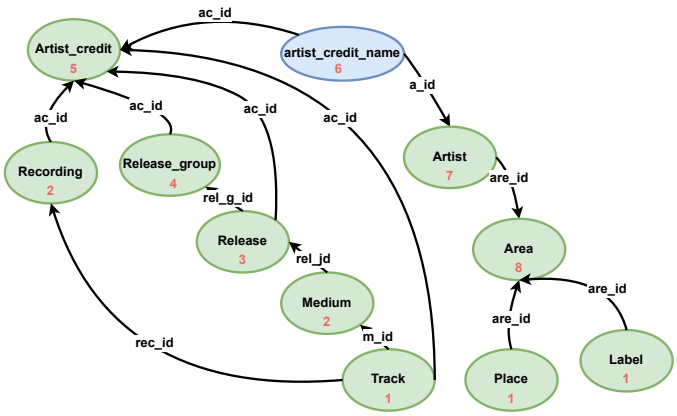



Figure 8: The relational dependencies of *Mu*

ing duplicates. Finally, the ground truth of the datasets is obtained from the identifiers of the original clean instances and their duplicates.

*Environment.* We implemented ASPEN in Python. The ER controller, `asprin` optimiser (Brewka et al. 2015), and explainer `xclingo2` (Cabalar and Muñiz 2023) are all based on `clingo 5.5` [7] Python API. The specification of programs follow the format of the ASPCore2.0 standard (Calimeri et al. 2020). All the experiments were run on a workstation using a single 3.8GHz AMD Ryzen Threadripper 5965WX core and 128 GB of RAM.

## D Main Results

*Multi-relational Input to Baselines.* Note that since baseline systems lack native support for multi-relational table inputs and do not recognise inter-references between tables, for multi-relational datasets. Moreover, these approaches assume that only one merge position is present for each relation (hence consider tuples are entities). Therefore, we performed directly pairwise matching for each table. Specifically, let $\mathcal{S} = \{R_1, ..., R_n\}$ be a multi-relational schema and $D$ be a $\mathcal{S}$-database, we consider pairwise matchers a function $m : D \times D \rightarrow \{0, 1\}$, where 0 and 1 represent not match/match resp. The set of merges w.r.t. $D$ is collected as

$$\{(t_j, t'_j) \mid m(R_i(t_1, ..., t_k), R_i(t'_1, ..., t'_k)) = 1, R_i \in \mathcal{S}\}$$

where $j$ is a merge position.

*Running Time.* The preprocessing time is used as $\mathbf{t}_o$ for UB, as UB can be obtained directly from this step. When comparing baselines with various solutions of ASPEN, it is evident that all ASPEN (approximate) solutions are significantly slower than the baselines. This is primarily due to the costly preprocessing stage. The LB and UB times are comparable, being 310 and 10, 7.6 and 24.6, 154 and 36, 12 and 7.9, 7.1 and 60.2, 15.8 and 176 times slower than Magellan and JedAI on *Dblp*, *Cora*, *Imdb*, *Mu*, *MuC*, and *Poke*, respectively. The worst performance is observed on the most complex setup PM, where ASPEN is up to 340 and 183 times slower than Magellan and JedAI, respectively.

[7] https://potassco.org/clingo/python-api/5.5/

*Dirtiness of Duplicates.* We observe that in *Imdb* and *Mu*, UB, MS-1 and PM achieved nearly perfect F1 scores. Remarkably, results are identical for the three type of solutions in *Imdb*. This uniformity may indicate that values on duplicates of entities exhibit low variance, resulting in a 'cleaner' instance. Indeed, if values of duplicates of an entity are largely identical, the discovery of merges becomes easier as merges derived from soft rules become more certain. Clearly, if merges derived from soft rules are as certain as those raised from hard rules, DCs would not be triggered at all. This observation is confirmed by the differences in accuracy between MS-1 and PM on the dirtier *MuC*. Although the number of duplicates per entity in *MuC* is the same as in *Mu*, the presence of more variants and nulls in *MuC* may have introduced more uncertainties.

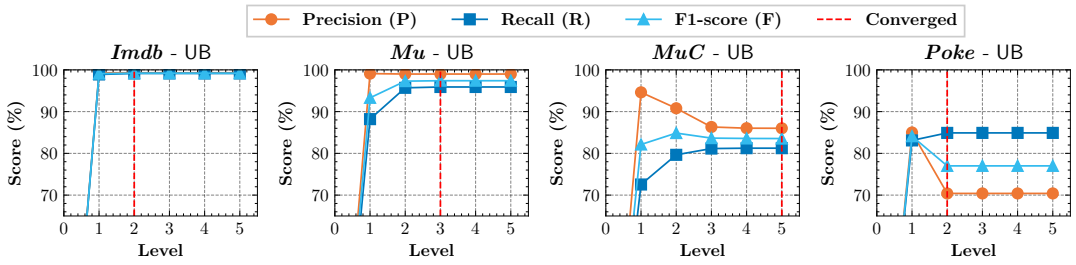
## E Effect of Recursion

We executed the levels algorithm described in Section 4.2 on solutions derived in our previous experiments on multi-relational datasets.

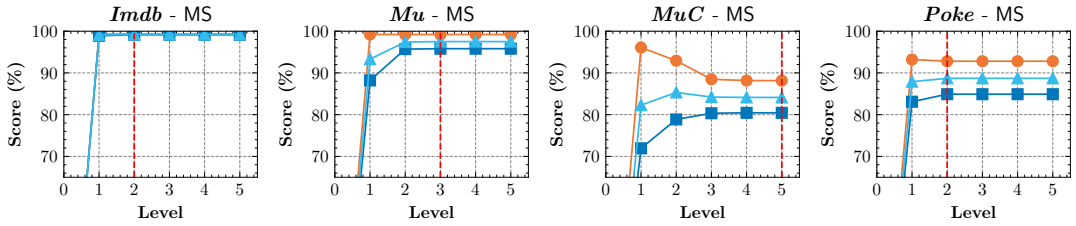
*Recursion is Effective.* In general, achieving convergence of merges requires more than one recursion level: *Imdb* and *Poke* converged at level 2, while *Mu* and *MuC* converged at level 3 and level 5, respectively. As illustrated by Figure 9c, noticeable increments of merges are observed in most of the datasets, obtaining 8% and 8.03%, 17.8% and 18.8%, 2.4% and 18.8% increment gains in MS-1 and UB settings on *Mu*, *MuC* and *Poke* respectively. These patterns confirm the efficacy of recursion in discovering new merges utilising merges derived from previous levels.

*DCs are Important for Recursion.* The impact of DCs on recursion becomes apparent when comparing the performances depicted in Figure 9a and Figure 9b. While recursion generally improves F1 scores across most datasets, it is interesting to observe that in *Poke*, introducing recursion leads to a decline in accuracy of UB. Notably, at the second level of UB, a slight increase in recall comes at the expense of a significant drop in precision, resulting in an 8% decrease in F1 score. Conversely, with the integration of DCs, the precision of MS-1 on *Poke* remains consistent, ultimately leading to an increase in F1 score. This highlights the critical role of DCs in recursive setups, enhancing the consistency of newly included merges across subsequent levels.

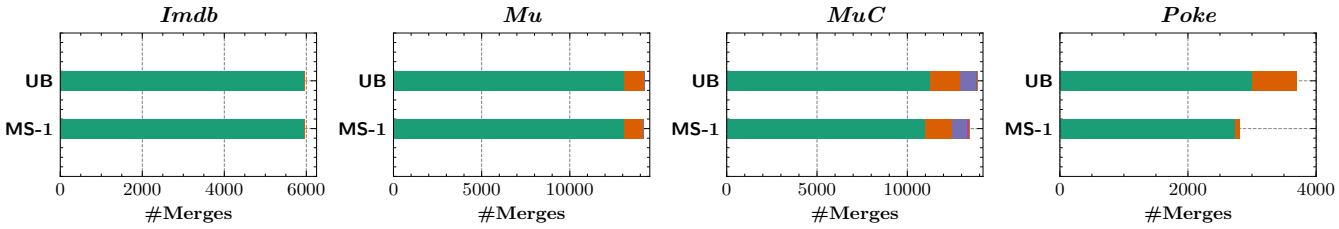
*Recursion and Dataset Characteristics.* One can observe in Figure 9c that *Imdb* shows only a slight increase in the number of merges, whereas *Mu*, *MuC*, and *Poke* (despite a small increment for MS-1 due to the regulation of DCs) exhibit more substantial increments in levels $> 1$. This disparity in merge counts correlates with the increasing number of references within the databases. Indeed, as all merge attributes specified are key attributes, databases with a greater number of inter-table references are better poised to leverage recursion for merge identification. When comparing *Mu* and *MuC*, it is interesting to observe that despite having the same number of referential constraints, *MuC* yields 2,600 more merges in later levels and requires two additional




(a) Multi-level recursion on UB




(b) Multi-level recursion on MS-1




(c) Merge Increments on Levels

Figure 9: Effects of Recursion

iterations to converge. This parallels the findings in Section 6.3, suggesting that recursion may prove more effective in discovering merges when dealing with dirtier duplicates. Indeed, simply comparing the similarity of attributes is less likely to identify merges due to the presence of nulls and value variations. Consequently, identifying duplicates may necessitate to consider the inter-dependencies between entities within the dataset.

# F Efficiency

We conducted two sets of experiments to examine the efficiency of Datalog approximations and impact on efficiency of various factors: **i)** data size; **ii)** the percentage of duplicates in a dataset; **iii)** similarity thresholds on an ER program.

**Datalog Approx.** We assume similarity facts are given by $\mathsf{Sim}_{\mathsf{opt}}$ and ran the Datalog programs $\Pi_{\exists^{\mathsf{lb}}}$ and $\Pi_{\exists^{\mathsf{ub}}}$ on all datasets using ASPEN and VLog4j (Carral et al. 2019; Urbani, Jacobs, and Krötzsch 2016). Table 7 presents the running time results. For LB, ASPEN outperforms VLog4j on most datasets, being 1.03, 1.7, 6 and 9 times faster on *Imdb*, *Cora*, *Poke* and *Dblp*, respectively. Conversely, VLog4j terminated 4 and 11 times faster on *Mu* and *MuC*. For UB, VLog4j outspeeds ASPEN in all but one of the multi-relational datasets, being 4.3, 7 and 46 times faster on *MuC*, *Imdb* and *Mu*, respectively. However, ASPEN surpassed VLog4j by 7 times on *Poke*. This results suggest that the performance of different reasoning engines is impacted by characteristics of the data.

| Prg. | Sys. | $\mathbf{t}_{Dblp}$ | $\mathbf{t}_{Cora}$ | $\mathbf{t}_{Imdb}$ | $\mathbf{t}_{Mu}$ | $\mathbf{t}_{MuC}$ | $\mathbf{t}_{Poke}$ |
|---|---|---|---|---|---|---|---|
| LB | VLog4j | 0.76 | 8.06 | 6.02 | **2.33** | **7.25** | 39.86 |
| | ASPEN | **0.079** | **4.66** | **5.8** | 27.03 | 30.21 | **6.38** |
| UB | VLog4j | 0.83 | 10.8 | **32.19** | **6.66** | **15.47** | 5115.31 |
| | ASPEN | **0.13** | **7.74** | 240.51 | 307.53 | 67.63 | **697.96** |

Table 7: VLog4j vs. ASPEN on LB and UB.

**Varying Size of the Data** We ran ASPEN on variants of *Mu*, where the data size $|D|$ ranged from $\times 1$ to $\times 5$, maintaining a consistent 10% proportion of duplicates (higher than the real-world duplicate distribution of 1% (Wang et al. 2022)). Table 8 illustrates the changes in grounding and solving times on both MS-1 and PM. Overall, both $\mathbf{t_g}$ and $\mathbf{t_s}$ increase monotonically as $|D|$ increases. $\mathbf{t_g}$ follows a similar pattern for both MS-1 and PM, increasing by factors of 5, 22, 42, and 63 as the data scale increases from $\times 2$ to $\times 5$. However, while $\mathbf{t_s}$ increases linearly for MS-1, it increases more drastically for PM, resulting in running times 6, 16, 29, and 52 times longer across the range of sizes $|D|$.

**Varying the Percentage of Duplicates**

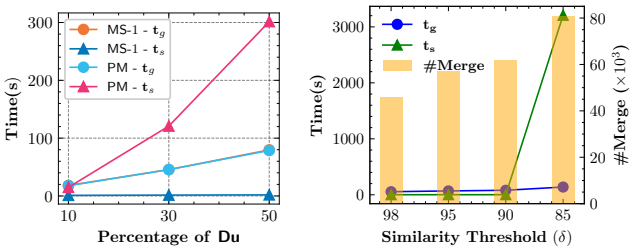
*Setup.* We created variants of *Mu* with duplication ratios of 10%, 30% and 50%, while keeping the same dataset size. We denote the variants as Du$\{10, 30, 50\}$, respectively. To mitigate the potential impact brought by tuple distribution, the sets of duplicates are such that Du10 $\subset$ Du30 $\subset$ Du50.

| $\lvert D\rvert$ | Met. | $\mathbf{t_g}$ | $\mathbf{t_s}$ | Met. | $\mathbf{t_g}$ | $\mathbf{t_s}$ |
|---|---|---|---|---|---|---|
| $\times 1$ | MS-1 | 17.82 | 1.02 | PM | 17.55 | 14.87 |
| $\times 2$ | | 92.4 | 2.21 | | 92.75 | 84.6 |
| $\times 3$ | | 388.2 | 3.4 | | 418.17 | 228.87 |
| $\times 4$ | | 719.2 | 4.8 | | 763.61 | 416.48 |
| $\times 5$ | | 1083.6 | 5.7 | | 1106.7 | 735.43 |

Table 8: Impact of Varying $|D|$

*Result.* The results are presented in Figure 10a. Since LB, UB and MS-1 present a similar behaviour, we only include the results of MS-1 and PM for comparison. Generally, increasing the proportion of duplicates led to monotonic increases in both $\mathbf{t_g}$ and $\mathbf{t_s}$. This explains the previous observation of longer times spent on smaller datasets like *Cora*. Indeed, despite its smaller scale, the ground truth size of *Cora* is much larger than that of other datasets. The increase in solving time $\mathbf{t_s}$ for PM in Du50 was particularly notable, being almost 30 times longer than in Du10. This can be attributed to the larger number of possible merges derived from soft rules due to the increased size of duplicates, leading to a larger space of merge combinations and consequently longer solving times.

**Varying Similarity Thresholds**




(a) Varying Du (b) Varying $\delta$

Figure 10: Impacts on duplicate (%) and similarity threshold

*Setup.* We create variants of the ER program for *Mu* by adjusting similarity thresholds. This results in four versions of the *Mu* specification with similarity thresholds set to 98, 95, 90, and 85. For simplicity, we refer to the similarity threshold as $\delta$.

*Results.* We ran the ER program with different $\delta$s across the solution setups on *Mu*. We observed consistent patterns in $\mathbf{t_g}$ across all setups (LB, UB, MS-1, and PM) with an approximately nine-fold increase in $\mathbf{t_g}$ when reducing $\delta$ from 98 to 85, as exemplified for MS-1 in Figure 10b. This increase can be attributed to the nature of the recursive evaluation algorithm, e.g. semi-naive evaluation (Gebser et al. 2012), adopted by the grounder. As the value of $\delta$ lowers, more new merges may be produced in earlier levels, so that they can be reevaluated in later ones. This is shown in the trend of orange charts w.r.t. merge increments. A more interesting pattern is observed when looking at the solving times on MS-1 and PM. We can observe a dramatic increase of $\mathbf{t_s}$

when reducing $\delta$ from 90 to 85: as shown by the green curve in Figure 10b, $\mathbf{t_s}$ for MS-1 sharply rose from tens of seconds to 3,345 seconds. Moreover, $t_s$ reached a timeout ($> 24$ hours) for PM with $\delta = 85$. This suggests the scalability of ASPEN may potentially be restricted in complex reasoning settings, provided the inherent intractability of computing MS and PM (Bienvenu, Cima, and Gutiérrez-Basulto 2022).