Universität Hamburg
MIN Faculty
Department of Informatics

Master's Thesis

Evaluation of Tuple Matching Methods on Generated Probabilistic Data

Steffen Friedrich (mat. no. 5944434)
MSc Informatik, 5th semester
7friedri@informatik.uni-hamburg.de

Wolfram Wingerath (mat. no. 5945090)
MSc Informatik, 5th semester
7wingera@informatik.uni-hamburg.de

1st assessor: Prof. Dr.-Ing. Norbert Ritter
2nd assessor: Prof. Dr.-Ing. Wolfgang Menzel

One of the great disadvantages of hurry is that it takes such a long time.
– *Gilbert K. Chesterton*

Abstract

English

Certain data tuple matching, i.e. comparison and decision model in the process of duplicate detection, is object of current research. However, while search space reduction as a part of duplicate detection in probabilistic data has already been investigated, tuple matching in probabilistic data as the subsequent step has not. This thesis has two main objectives: first, the development and evaluation of an approach to adapt existing certain data tuple matching techniques to probabilistic x-tuples and, second, the generation of large labelled probabilistic data sets for evaluation experiments.
The probabilistic ULDB data model is described where every x-tuple consists of one or more mutually exclusive alternatives that represent one and the same real-world entity.
ProbGee, an Eclipse-based framework for the generation of labelled probabilistic data sets from existing certain data, is presented and used to generate the labelled data that is needed for evaluation.
The proposed tuple matching approach uses a certain data decision model to match alternative pairs corresponding to an x-tuple pair and derives the x-tuple pair matching decision from the alternative pair labels. Three reduction strategies to avoid unreasonable tuple pair decisions are discussed: reducing alternatives, reducing alternative pairs and reducing comparison vectors. A special alternative pair reduction strategy is the Best Partner Reduction which does not require any configuration to function. Furthermore, four variants of deriving the x-tuple pair matching status are proposed, namely Expected Similarity, Expected Similarity with Uniformly Distributed Confidences, Best Partner Similarity, Maximal α-Prob. Similarity and Cry with the Wolves.
The experimental results indicate that reducing alternatives or alternative pairs increases recall and F-measure of the result strongly on error-prone data, while it has little effect on rather clean data. None of the tuple pair decision variants clearly outperforms the others. In a real application, applying a combination of reducing alternative pairs with the Best Partner Reduction and deriving tuple pair decisions according to the Expected Similarity with Uniformly Distributed Confidences seems to be reasonable, because it does not require the user to define any parameters.

German

Tuplematching, d.h. Vergleich und Entscheidungsmodell bei der Duplikatenerkennung, ist Gegenstand aktueller Forschung. Doch während die Suchraumreduzierung als Teil der Duplikatenerkennung in probabilistischen Daten bereits untersucht wurde, wurde Tupelmatching in probabilistischen Daten als nachfolgender Schritt noch nicht untersucht. Diese Arbeit hat zwei Ziele: erstens die Entwicklung und Bewertung eines Ansatzes zur Anpassung bestehender Tupelmatching-Methoden an probabilistische x-Tupel und zweitens die Generierung einer großen Anzahl an gelabelten probabilistischen Datensätzen für Evaluationsexperimente.
Das probabilistische ULDB-Datenmodell wird beschrieben, in dem jedes x-Tupel aus einer oder mehreren sich gegenseitig ausschließenden Alternativen besteht, die ein und dieselbe Realweltentität repräsentieren.
ProbGee, ein Eclipse-basiertes Framework zur Generierung gelabelter probabilistischer Daten aus existierenden sicheren Daten (von engl. *certain data*), wird vorgestellt und zur Generierung der gelabelten Daten genutzt, die zur Evaluation benötigt werden.
Bei dem vorgeschlagenen Ansatz zum Tupelmatching wird ein Entscheidungsmodell verwendet, das auf sicheren Daten arbeitet, um den Matchingstatus von Alternativpaaren zu bestimmen, d.h. um zu entscheiden, ob die entsprechenden Alternativen dieselbe Realweltentität darstellen oder nicht. Anhand der Matchingstati der einem x-Tupel-Paar entsprechenden Alternativpaare wird nachfolgend der Matchingstatus des x-Tupel-Paares ermittelt. Drei Reduktionsstrategien zur Vermeidung unsinniger Tupelpaarentscheidungen werden diskutiert: die Reduktion von Alternativen, die Reduktion von Alternativpaaren und die Reduktion von Vergleichsvektoren. Eine besondere Alternativpaarreduktion ist die Best Partner Reduction, die keiner Konfiguration bedarf. Darüber hinaus werden vier Varianten vorgeschlagen, den Matchingstatus eines x-Tupel-Paares abzuleiten: Expected Similarity, Expected Similarity with Uniformly Distributed Confidences, Best Partner Similarity, Maximal α-Prob. Similarity und Cry with the Wolves.
Die Experimentergebnisse deuten darauf hin, dass die Reduktion von Alternativen und Alternativpaaren den Recall und das F-Measure des Ergebnisses auf fehlerbehafteten Daten stark verbessern, während sie kaum einen Effekt bei wenig fehlerbehafteten Daten zeigen. Keine der Varianten zur Ableitung der Tupelpaarentscheidung zeigt sich den anderen deutlich überlegen. In einer tatsächlichen Anwendung scheint der Einsatz einer Kombination aus der Alternativpaarreduktion Best Partner Reduction und dem Ansatz Expected Similarity with Uniformly Distributed Confidences zur Berechnung des x-Tupel-Paar-Matchingstatus sinnvoll, da sie keinerlei Konfiguration durch den Nutzer erfordert.

Contents

1. Introduction

In this thesis, we investigate how existing tuple matching[1] techniques known from duplicate detection in certain data can be applied to uncertain probabilistic data. Furthermore, we implement our approaches and evaluate them on synthetic data generated solely for this purpose.

1.1. Motivation

More and more applications impose requirements on solutions for storing and processing uncertain data that cannot be met by traditional database systems. Uncertain data can for example originate from inaccurate readings in sensor networks such as RFID networks [KBS08, HMMS12], imperfect OCR in automatic number plate recognition [MR10] or from personal information in social networks or crowd-sourced databases like the *Last.fm*[2] music database which typically contain massive amounts of error-prone and contradictory user-generated data.
Modelling uncertainty in a probabilistic database instead of resolving it has already been recognised as a means to reduce the costs of the data integration process [vKdK09]. In some applications, it is even *necessary* to manage and reason with multiple possible interpretations of measured data, so that some kind of probabilistic database system is practically called for. For example, technological breakthroughs in the field of astrophysics place great demands on probabilistic data storage and processing solutions [SCH09] as astrophysical surveys deliver much more data nowadays than they did a few years ago: the integration of today's survey results with each other (and also with data from older surveys and simulations) simply cannot be done with the technology currently available.

1.2. State of Research

The integration of certain data and duplicate detection in particular are actively researched topics and have been for many years [Chr12]. A very good overview over the process of duplicate detection, especially over string metrics commonly used for tuple comparisons and over popular decision models, is given in [EIV07].

[1] By *tuple matching*, we mean the in-depth tuple comparison and the decision model of the duplicate detection process.
[2] Last.fm: `http://www.lastfm.de/`.

Since probabilistic databases are still in their infancy, the integration of and duplicate detection in probabilistic data have received little attention so far. However, some approaches to adapt existing certain data search space reduction (SSR) techniques to probabilistic data have been proposed in [PvKdKR09] and an experimental evaluation of adapted probabilistic search space reduction techniques can be found in [FW10].

In the field of certain data duplicate detection, appropriately large labelled data sets needed for the evaluation of tuple matching techniques are hard to come by, e.g. due to confidentiality. It should be noted, though, that there are a few data generators, so that conducting evaluation experiments for certain data techniques is still possible, even when no real-world data sets are available. To our knowledge, there are virtually no *probabilistic* sets of test data we could use for the evaluation experiments, but in [FW10] a prototypical generator for probabilistic data sets is presented.

1.3. Goals

In our master's thesis, we address two problems: first, how to make existing tuple matching techniques for certain data applicable to probabilistic (relational) data and, second, the generation of large labelled probabilistic data sets for evaluation experiments.
We give an overview over existing techniques for tuple matching on certain data and examine how they can be applied to probabilistic data[3]. The basic idea here is to use the existing certain data techniques on probabilistic x-tuple alternatives and then derive the matching status of two x-tuples.
We implement an approach as baseline that compares all alternative pairs corresponding to a tuple pair and computes the tuple pair matching status from the matching status of all alternative pairs. Besides, we implement variants of this naive approach using particular selection strategies for the alternative comparisons as well as for the derivation of the matching status. Finally, we evaluate the different approaches experimentally with respect to recall, precision and the traditional F-measure.

Most importantly, though, we describe and implement a framework for the generation of probabilistic relational data with labelled duplicates. As we do not have access to appropriately large probabilistic data sets for those experiments, we have to generate them ourselves. To make the final experiments as meaningful as possible, we improve the data generator described in [FW10] by the following features:

- more *control over data quality*: control over the degree of similarity between duplicate x-tuples and between alternatives of an x-tuple.
- more *realistic error patterns*: realistic typos and confusion set errors.
- improved *usability*: a more generic and simpler user interface.

[3]For our implementation, we use the ULDB data model [BSHW06, Wid08].

1.4. Chapter Outline

Chapter 2 contains related work. We introduce the probabilistic ULDB data model and give an overview over tuple matching approaches commonly used in certain data. We also describe a search space reduction technique adapted to probabilistic data and examine literature on the generation of synthetic data.
In Chapter 3, we present our data generation framework *ProbGee*. After a description of the workflow in ProbGee, we go into detail about the error generation architecture we developed to achieve realistic error patterns. In the final section, we briefly characterise some special features like the graphical user interface.
The concept of our approach to the adaption of existing tuple matching methods to probabilistic data is presented in Chapter 4.
The description and analysis of our evaluation experiments are given in Chapter 5.
We conclude our work in Chapter 6 with a short summary and future prospects.

2. Related Work

This chapter provides the theoretical foundations of our work. In the first section, we describe the probabilistic ULDB data model. Section 2 comprises an overview over existing tuple matching techniques on certain data. In the third section, we describe an adaption of the certain data search space reduction technique Sorted Neighbourhood Method to probabilistic data. The final section is devoted to the available literature on the generation of test data needed to evaluate tuple matching techniques.

2.1. The Probabilistic Data Model

There are several probabilistic database prototypes, e.g. MayBMS [Koc08], MYSTIQ [BDM+05] and Trio [Wid08], but there are no commercial products available on the market, yet. The data model used in this thesis is the *Uncertainty Lineage Database* (ULDB) data model used in Trio according to [BSHW06, Wid08] which extends the traditional relational model by possibilities to represent uncertainty regarding a tuple's attributes as well as its very existence. In a ULDB, an entity is not represented by one single combination of attribute values as it is in a certain database, but by a (discrete) probability distribution over mutually exclusive attribute value combinations. Hence, there is not one representation of the modelled part of the world, but a probability distribution over possible representations, called *possible worlds*.

	title	year	studio	conf
t_1a_1	Catwoman	1969	Twentieth Century Fox	0.6
t_2a_1	Batman	1966	Twentieth Century Fox	0.8
t_2a_2		1967	Republic Pictures	0.2

Table 2.1.: A probabilistic relation with two x-tuples incorporating uncertainty on tuple level (t_1) and attribute level (t_2).

A tuple in a ULDB is called an *x-tuple* and consists of one or more mutually exclusive entity representations. Each of those representations is called an *alternative* and is attached with a *confidence* value that represents the probability of this alternative being the correct entity representation. Table 2.1 illustrates how uncertainty on tuple and alternative level is represented.
The confidence of the Catwoman tuple is only 0.6 = 60%, meaning that the correct representation of the domain does *not* contain this entity with a probability of 40%. A tuple

whose existence is not certain is called a *maybe-tuple*. The existence of the Batman tuple is certain, since the sum of all its alternatives' confidences equals 1. The characteristics of the correct representation, however, are not certain: with a probability of 80%, the film was produced by Twentieth Century Fox in 1966 and with a probability of 20% by Republic Pictures in 1967.

2.2. Tuple Matching in Certain Data

Duplicate detection, i.e. the process of identifying pairs of tuples that represent the same real-world entity, is a crucial part of the information integration process. In an error-free relational system with absolutely clean data, duplicate detection could be performed with a simple equality check, since all duplicates were perfect and thus identical. Unfortunately, real-world data are dirty and duplicate detection is far more difficult: duplicates may differ due to different formatting, abbreviations, spelling errors, typos and semantic or other errors.

The first formal mathematical model for duplicate detection was developed by Fellegi und Sunter in the 1960s [FS69] and has been improved by Winkler [Win99, Win00] and others [CH90, Jar95, Gil01, VME03]. Typical duplicate detection systems implement a process similar to the one depicted in Figure 2.1, although the techniques used in the individual steps may vary from system to system [EIV07]. As indicated, our focus lies on the tuple matching by which we subsume the *comparison* and the *decision model*.

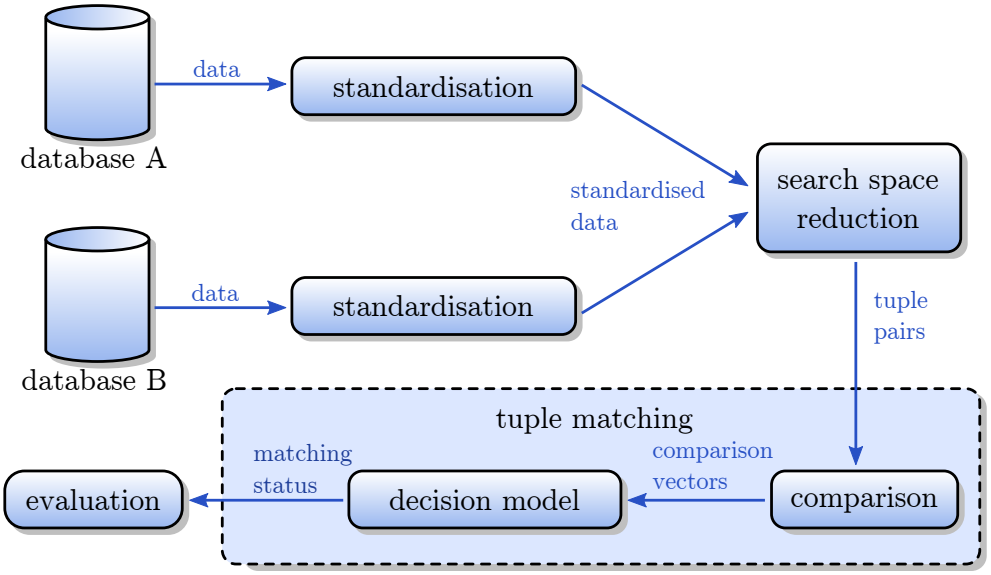


Figure 2.1.: A schematic view on the duplicate detection process.

The first step in duplicate detection is the *standardisation* of the data where minor inconsistencies between data from different sources are removed, for example by converting string values to lowercase, transforming values to another format or removing special char-

acters. As a thorough comparison of every tuple in one data source with every tuple in every other data source is infeasible, obvious non-duplicate tuple pairs are filtered out with simple means in the following step called *search space reduction*. All tuple pairs in the remaining search space then undergo an attribute value *comparison*: the result of such a tuple pair comparison is a *comparison vector* whose entries represent the similarity of the corresponding tuples in each attribute value. Usually, the attribute value comparisons are performed with character-based similarity metrics like the *Levenshtein metric*, token-based similarity metrics like the *Jaccard metric* or hybrid metrics like *Soft TFIDF*.[1] On the basis of the comparison vectors, each tuple pair is then either declared a match or an unmatch by the *decision model*. Finally, an evaluation of the duplicate detection result can be carried out.

The process of duplicate detection as we describe it does not lead to conflict-free tuple pair decisions: for example, if two tuple pairs (t_1, t_2) and (t_2, t_3) are declared matches and (t_1, t_3) is declared an unmatch, a conflict lies in the fact that the two matching labels imply that tuples t_1 and t_3 are matches as well, while the matching label assigned by the decision model indicates otherwise.
One possibility to resolve such conflicts is to compute the transitive closure of the matching relation, so that in the above example, tuple pair (t_1, t_3) would simply be declared a match. More sophisticated strategies to resolve such conflicts are described in Section 6.9 in [Chr12].

In the following subsection, we briefly describe the formal notation of the duplicate detection problem. The remainder of this section deals with different decision model approaches according to [EIV07].

2.2.1. Problem Notation

Given two data sources A and B, the set of all tuple pairs

$$A \times B = \{(a,b) \mid a \in A, b \in B\}$$

is the union of the disjunct sets M and U where $a = b$ for all tuple pairs in M and $a \neq b$ for all tuple pairs in U. So

$$M = \{(a,b) \mid a = b, a \in A, b \in B\}$$

is the set of *matches* and

$$U = \{(a,b) \mid a \neq b, a \in A, b \in B\}$$

is the set of *unmatches*.
When the decision model assigns a tuple pair to either M or U, two kinds of errors can

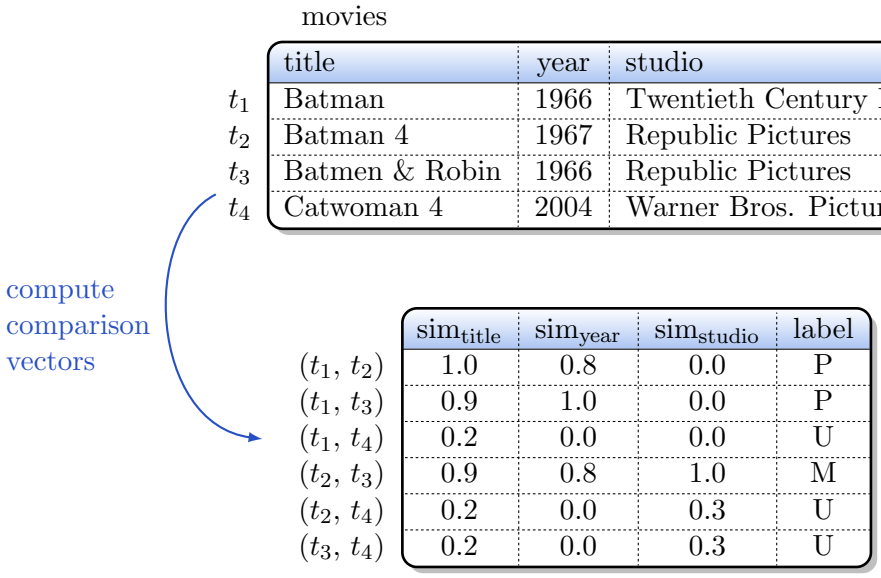
[1] An overview of different string comparison metrics is given in Chapter 5 of [Chr12].

occur: false positives and false negatives. A *false positive* is an actual unmatch that is declared a match, whereas a *false negative* means an actual match that is declared an unmatch. To minimise the number of errors, the set P of *possible matches* can be introduced for tuple pairs whose matching status is too doubtful and should be determined by a domain expert. Since humans are very inefficient and expensive in comparison to an automated decision model, though, as few tuple pairs as possible are assigned to P.

Assuming that n common attributes $a_1,a_2,\ldots,a_n$ of the tuples in sources A and B are chosen for the attribute value comparison, the comparison of a tuple pair (t_i,t_j) results in a *comparison vector*

$$c_{i,j} = [c^1_{i,j},c^2_{i,j},\ldots,c^n_{i,j}]$$

of size n where a vector component $c^k_{i,j}$ is the comparison result of the tuples in attribute a_k with a comparison function C_k. The space of all comparison vectors is called *comparison space* Γ.



	title	year	studio
t_1	Batman	1966	Twentieth Century Fox
t_2	Batman 4	1967	Republic Pictures
t_3	Batmen & Robin	1966	Republic Pictures
t_4	Catwoman 4	2004	Warner Bros. Picture

	$\text{sim}_{\text{title}}$	sim_{year}	$\text{sim}_{\text{studio}}$	label
(t_1, t_2)	1.0	0.8	0.0	P
(t_1, t_3)	0.9	1.0	0.0	P
(t_1, t_4)	0.2	0.0	0.0	U
(t_2, t_3)	0.9	0.8	1.0	M
(t_2, t_4)	0.2	0.0	0.3	U
(t_3, t_4)	0.2	0.0	0.3	U

Figure 2.2.: Four tuples with all corresponding comparison vectors and labels.

An example of a very simple discrete attribute comparison function is

$$C_i(v_1,v_2) = \begin{cases} 1 & \text{if } v_1 = v_2 \\ 0 & \text{else} \end{cases},$$

where the similarity of the compared attribute values is either 1, if the attribute values are equal, or 0, if they differ somehow. Comparison functions can also be much more complex and deliver categorical or continuous values. The comparison functions used in this thesis produce a numerical *matching weight* that is normalised, so that attribute value similarities range from 0 (completely different) to 1 (perfect match).

Figure 2.2 shows four movie tuples with the attributes title, year and studio together

with comparison vectors corresponding to all tuple pairs. The decision model determines the matching status of a tuple pair (t_i,t_j) on the basis of the corresponding comparison vector $c_{i,j}$ and under the assumption that tuples with similar or identical attribute values correspond to the same real-world entity with a high probability, whereas actual unmatches are rather dissimilar. Accordingly, a tuple pair is assigned to M, if its comparison vector holds values near or equal to 1, and is assigned to U in case of low similarity values. In this example, some tuple pairs like (t_2, t_3) and (t_1, t_4) are easily recognisable as match or unmatch respectively, while the decision for other tuple pairs like (t_1, t_3) is not as obvious.

2.2.2. The Probabilistic Decision Model by Fellegi and Sunter

One of the fundamental works on duplicate detection is a paper by Ivan Fellegi and Alan Sunter from the year 1969 [FS69] that is based on a work from the 1950s [NKAJ59]. Fellegi and Sunter present a decision model that minimises the probability for false positives and false negatives and thus is error-optimal.
They define

$$m(c_{i,j}) = P\big(c_{i,j}\big|(t_i,t_j) \in M\big)$$

as the probability that comparison vector $c_{i,j}$ corresponds to a tuple pair (t_i,t_j), given that the tuple pair is an actual match. Correspondingly,

$$u(c_{i,j}) = P\big(c_{i,j}\big|(t_i,t_j) \in U\big)$$

is defined as probability for $c_{i,j}$ resulting from the comparison of t_i and t_j, given that the tuples actually do not match. The authors then compute the *matching weight* $\frac{m(c_{i,j})}{u(c_{i,j})}$ and determine the matching status by thresholding.

The key difficulty in this model is the computation of the matching probabilities and thus determining the matching weights. A very naive approach is labelling a subset of all tuple pairs by hand and approximating the weights on the basis of the resulting labels.
In [Win88], William Winkler proposes a more efficient approach that uses the *expectation-maximisation algorithm* for matching weight estimations. However, this procedure does not seem practicable, since, according to [Win02], it only works under certain limitations which don't seem realistic in general; for example, a proportion of duplicate tuples of 5% or more is required, M and U have to be relatively well-separated and typos should not be too frequent in the data.

Although the model of Fellegi and Sunter is error-optimal, it does not lead to cost-optimal decisions, when false positives and false negatives are not equally expensive. In [VME03], Verykios et al. present an approach to minimise the *expected costs* of a decision. For this purpose, a *cost matrix* is defined whose entries represent the costs of every possible decision (match, unmatch, possible match) given every possible actual matching status (match,

unmatch). The expected costs of every possible decision are computed and the decision with the lowest expected costs is taken. The authors claim that probability distributions over possible costs can also be used instead of fixed values.

In principle, the mentioned probabilistic approaches make a mostly automated duplicate detection possible. However, the computation of matching weights – although in favourable scenarios efficiently possible with the expectation-maximisation algorithm – remains a major obstacle in the general case. A great number of possible similarity values leads to a great number of possible comparison vectors and thus to a great number of matching weights to be computed. So, ideally, binary comparison functions should be used. More complex comparison functions can also be used, but have to be discretised.

2.2.3. Distance-Based Decision Models

A group of decision models that work without training data are based on distance metrics for tuples. A very simple way to compute the distance between two tuples is concatenating the attribute values of each tuple like string values and then computing a similarity value from the two tuple representations as proposed by Monge und Elkan in [ME96]. The basic Levenshtein distance is not appropriate for the comparison of such string representations, because the alignment between the two strings is not modelled well, so that two strings like “Bruce Wayne” and “Wayne, Bruce” may not be recognised as highly similar. Monge and Elkan use the Smith-Waterman edit distance in their paper, as it accounts for such gaps. Other metrics can also be used; in [Coh00] for example, Cohen proposes combining the *TFIDF weighting scheme* and *cosine similarity*. However, these naive approaches do not perform well in comparison to other distance-based approaches.

In [DSD98], the distance between two tuples is computed as the weighted sum of the distances between the attribute values. The decision as to whether the two tuples are matches or unmatches is made according to a similarity threshold.
Another approach is to compute a comparison vector holding the attribute similarities and to declare a match or unmatch depending on its Euclidian or Manhattan distance to the perfect match vector ($[1, 1, \ldots, 1]$) or the perfect unmatch vector ($[0, 0, \ldots, 0]$) respectively.

A more sophisticated distance-based decision model based on *ranked list merging* is introduced by Guha et al. in [GKMS04]. The main idea is illustrated in Figure 2.3: there is one movies relation with tuples t_1, t_2 and t_3 and a single tuple t_4 from the films relation which may be a duplicate of one of the other tuples.
First, the tuples from movies are ranked according to their similarity to t_4 in every attribute. Since the resulting rankings can be contradicting, a *merging function* is used to create an unambiguous ranking; in our example, the tuples are sorted by the sum of all attribute similarities, so that tuple t_3 appears to be the most likely duplicate candidate.
Although the weighted sum works well in this example, Guha et al. claim that it does not perform well in general. They propose a more complex merging function that minimises

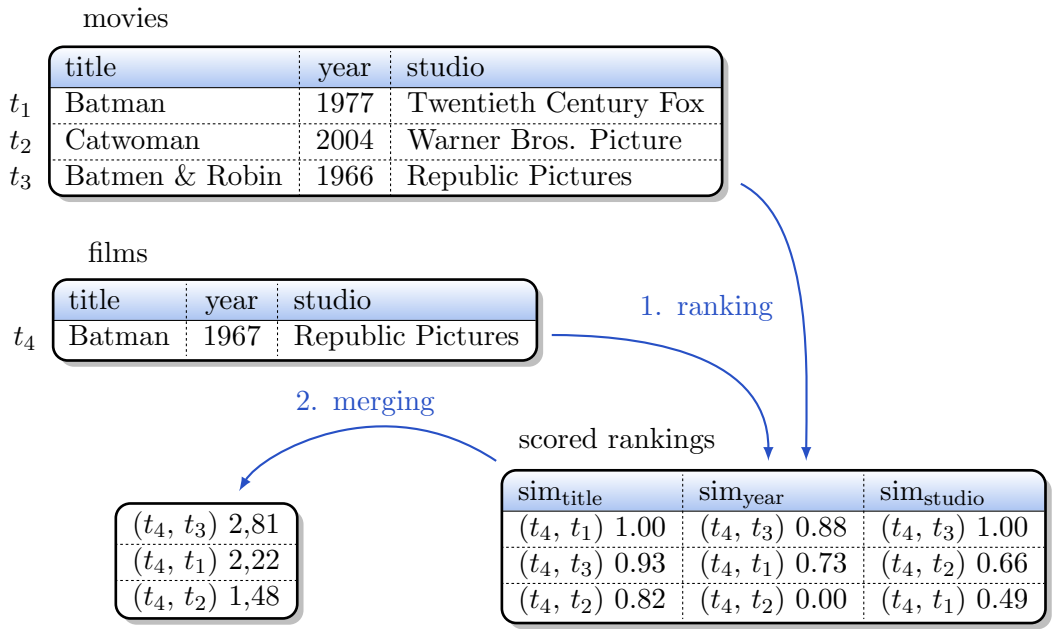


Figure 2.3.: The ranked list merging technique.

the *footrule distance*. The footrule distance over two rankings σ and τ on a relation R with n tuples $t_1, t_2, \ldots, t_n$ is defined as

$$F(\sigma,\tau) = \sum_{i=1}^{n} \mid \sigma(t_i) - \tau(t_i) \mid$$

and can be interpreted as the absolute difference of the σ and τ ranking positions of tuples in relation R.

The distance-based models discussed here treat attribute values as strings and rely on string similarities, but there are also approaches to exploit foreign keys for duplicate detection, see for example [ACG02].

All distance-based models have in common that a similarity threshold has to be defined to distinguish matches and unmatches. While distance-based decision models do not perform well in comparison with other models, they can be used even when there is no training data available which can be a great advantage. Defining similarity thresholds without training data, though, usually is very hard.

2.2.4. Rule-Based Decision Models

In the first work on rule-based duplicate detection we are aware of [WM89], Wang and Madnick examine how a domain expert can define rules that generate an identifying attribute set for a single entity, when there is no global identifier available. But obviously, this approach is not feasible for large data sets, since rules are specified on entity level.

In the late 1990s, Hernández and Stolfo came up with a more promising rule-based approach to duplicate detection in [HS98]: the search space reduction is executed with the Sorted Neighbourhood Method, while the in-depth comparison works according to an equational theory that can be used to specify rules by which the matching status of two tuples can be determined. A simplified example rule for the movie domain known from the previous examples is illustrated in Listing 2.2: the rule states that two tuples represent the same movie, if they have a similar title and were produced by the same studio. Naturally, in an actual rule, the "similar to" would have to be replaced, for example by a similarity threshold and a comparison function.

```
for (a,b) in movies do
  if  a.title is similar to b.title and a.studio = b.studio
  then
    a matches b
  end if
end for
```

Listing 2.1: A simplified example rule that determines the matching status of movie tuples, according to the rule-based equational theory by Hernández and Stolfo [HS98].

A more sophisticated duplicate detection framework was published by Low, Lee and Ling in 2001 [LLL01]. In their framework, duplicate identification rules can be defined that do not simply assign a label to a tuple pair, but can also assign a matching label with a certainty factor. Apart from rules for *duplicate identification*, Low, Lee and Ling define three further rule categories for how to *merge duplicates*, how to *update data* in particular situations and when to *alert the user*.

The literature indicates that rule-based decision models can perform exceptionally well with respect to recall *and* precision. Negative aspects, however, are high costs and domain-dependency: rules typically have to be designed by domain experts for a given domain with the utmost care.

2.2.5. Machine Learning Decision Models

In this section, we describe machine learning decision models. First, we go into detail on supervised machine learning decision models which deliver good results, but cannot work without labelled training data. Then, we describe unsupervised machine learning decision models based on clustering algorithms that can work independently from training data. Finally, we describe hybrid machine learning models that combine the benefits of both machine learning paradigms.

Supervised Machine Learning Decision Models

Supervised learners take a set of pre-labelled training instances as input to train a model that can assign any other instance to one of the classes known from the training set. A

training instance $\langle c, f(c)\rangle$ comprises a pattern c and the correct classification $f(c)$ of this pattern. In the context of duplicate detection, c is a comparison vector and $f(c) \in \{M, U\}$ or $f(c) \in \{M, U, P\}$ respectively is the matching status. The process of duplicate detection with a supervised learning decision model is illustrated in Figure 2.4.
First, a set of training instances is created from the set of all comparison vectors and is used to train a decision model. After that, every comparison vector is assigned with a matching status.

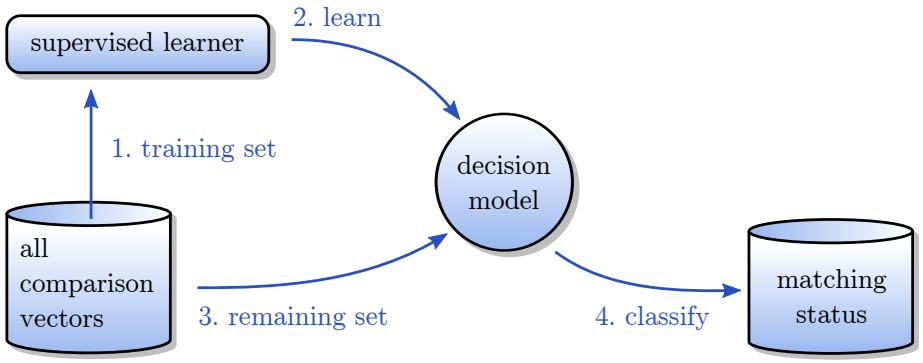


Figure 2.4.: A supervised learning decision model.

A rather prominent supervised classification method is the concept of *decision trees*. Approaches to the automated construction of decision trees were first published by Morgan and Sonquist [SM64] and hence their *CHAID* algorithm is the first decision tree algorithm. Another notable decision tree algorithm is the *CART* algorithm that was published in 1984 by Breiman et al. [Bre84]. The most commonly used decision tree algorithms today, however, probably are *ID3* [Qui83] and its successor *C4.5* [Qui93].

For the application of decision trees as decision models in duplicate detection, Cochinwala et al. [CKLS01] propose a hybrid[2] decision model using a rule-based approach: their basic idea is to select a set of potential matches with a simple rule-based matching technique, first, and then have some of those tuple pairs labelled with M, U and P by a domain expert. In the final step, the comparison vectors of the labelled tuple pairs are used to train a decision tree that can predict the matching status of any unclassified tuple pair, basically by following a set of if-then rules. In their work, the authors propose the *CART* algorithm with a *pruning* strategy to minimise the tree complexity.

Based on the comparison vectors of the previous movie examples, the comparison vectors in Figure 2.5 have been labelled and thus have been transformed into training instances. The corresponding CART tree is also depicted. The basic decision tree algorithm grows the tree by splitting the training set again and again in such a way that a particular *splitting* criterion is optimised. Typically, CART trees are only split binary.
Starting with the root node which contains the entire training set, it is tested for every reasonable combination of attribute and similarity threshold whether the splitting criterion is optimised. For example, there are only three distinct similarity values for the title

[2]Hybrid machine learning decision models are covered in more detail in Section 2.2.6.

	$\text{sim}_{\text{title}}$	sim_{year}	$\text{sim}_{\text{studio}}$	label
(t_1, t_2)	1.0	0.8	0.0	P
(t_1, t_3)	0.9	1.0	0.0	P
(t_1, t_4)	0.2	0.0	0.0	U
(t_2, t_3)	0.9	0.8	1.0	M
(t_2, t_4)	0.2	0.0	0.3	U
(t_3, t_4)	0.2	0.0	0.3	U

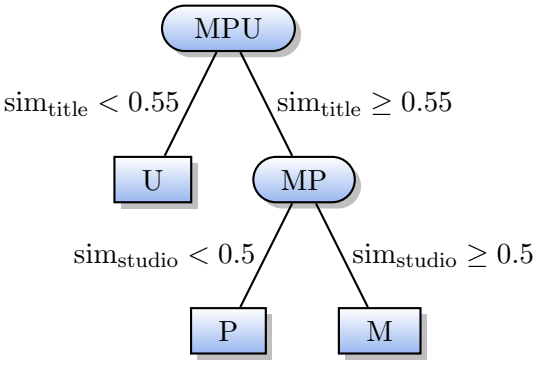


Figure 2.5.: A CART decision tree trained with labelled data from Figure 2.2.

attribute (0.2, 0.9 and 1) and hence reasonable thresholds can only be drawn between 0.2 and 0.9 or between 0.9 and 1, because otherwise the data would not be separated. In the example, the title similarity with a threshold of 0.55 (the mean value between 0.2 and 0.9) is chosen as splitter, so that all training instances with a title similarity below 0.55 are assigned to the left child node and all other instances to the right child node. Since all training instances in the left child node have the same label U, no further split is executed here and the node is called a *terminal node*. The splitting is repeated, until there are no data to separate or no attributes left.
When fully trained, the decision tree from the example assigns the matching status according to the rule in Listing 2.2.

```
if sim_title ≥ 0.55 and sim_studio ≥ 0.5
then
  match
end if
```

Listing 2.2: An example rule that is generated by the CART algorithm.

The *information gain* is used as splitting criterion which is either based on the *Gini coefficient* or the *entropy*.[3] For a binary split into the two classes 1 and 0, the Gini coefficient of a node K is defined as

$$G(K) = 1 - p(K)^2 - (1 - p(K))^2$$

and the entropy is computed by

$$G(K) = -p(K) \cdot \ln(p(K)) - (1 - p(K) \cdot \ln(1 - p(K))$$

[3]The formulas for the computation of information gain, Gini coefficient and entropy are taken from [Ste09].

where $p(K)$ is the relative frequency of class 1 instances in the node K. The information gain resulting from the split of parent node E into left child node L and right child node R is defined as

$$I(E) = G(E) - p(L) \cdot G(L) - p(R) \cdot G(R) \quad .$$

The CART authors prefer the Gini coefficient for computational reasons and because it yields more balanced splits.

In the following, the information gain for the root node split of the example tree is computed. Matching status U corresponds to class 1 and, accordingly, matching status $\overline{U} = M \vee P$ corresponds to class 0. Since matching label U occurs three times, the Gini coefficient of the root node is computed by

$$G(MPU) = 1 - (\frac{3}{6})^2 - (1 - \frac{3}{6})^2 = 0.5 \quad .$$

The three comparison vectors from Figure 2.5 having $sim_{Titel} = 0.2$ are the very comparison vectors that correspond to non-duplicates (matching label U) and are assigned to the left child node, while all the other comparison vectors end up in the right child node. Hence, the resulting Gini coefficient of the left child node is

$$G(U) = 1 - (\frac{3}{3})^2 - (1 - \frac{3}{3})^2 = 0$$

and the Gini coefficient of the right child node is

$$G(MP) = 1 - (\frac{0}{3})^2 - (1 - \frac{0}{3})^2 = 0 \quad .$$

Thus, the information gain of the example split is

$$I(MPU) = 0.5 - \frac{3}{6} \cdot 0 - \frac{3}{6} \cdot 0 = 0.5 \quad .$$

As can be easily checked, a year attribute split with a similarity threshold between 0.0 and 0.8 also results in an information gain of 0.5. With M corresponding to class 1, though, and matching status $\overline{M} = U \vee P$ corresponding to class 0, a studio attribute split between 0.3 and 1 only yields an information gain of about 0.28.

Support vector machines (SVMs) are another group of supervised machine learners. One of the first works on support vector machines in duplicate detection was published by Bilenko et al. in the early 2000's [BM03], although the idea of support vector machines goes back to a work by Vapnik and Chervonenkis from the 1970s [VC74]. A more modern introduction to the topic can be found in the book [SS02].

A simple support vector machine is a binary classifier. In the training process, it takes a set of pre-labelled vectors as input and separates them by a hyperplane in the corresponding vector space according to their labels. Unknown vectors are then assigned to one of the classes, depending on which side of the hyperplane they are. For obvious reasons, this

hyperplane is also called *decision boundary*.
Since a decision model classifies comparison *vectors* in vector space Γ, deploying an SVM decision model with comparison vectors labelled as matches and unmatches is very straightforward.

Figure 2.2.5 illustrates a decision boundary separating matching and unmatching comparison vectors. For better illustration, the vector space is only two-dimensional and hence the decision boundary is even only one-dimensional. The fundamental concepts, however, are the same for higher dimensions.
The decision boundary is chosen to maximise the distance to the closest vectors of each class. Those vectors are called *support vectors* and determine the alignment of the decision boundary. The region between the support vectors and the decision boundary is known as *margin*.

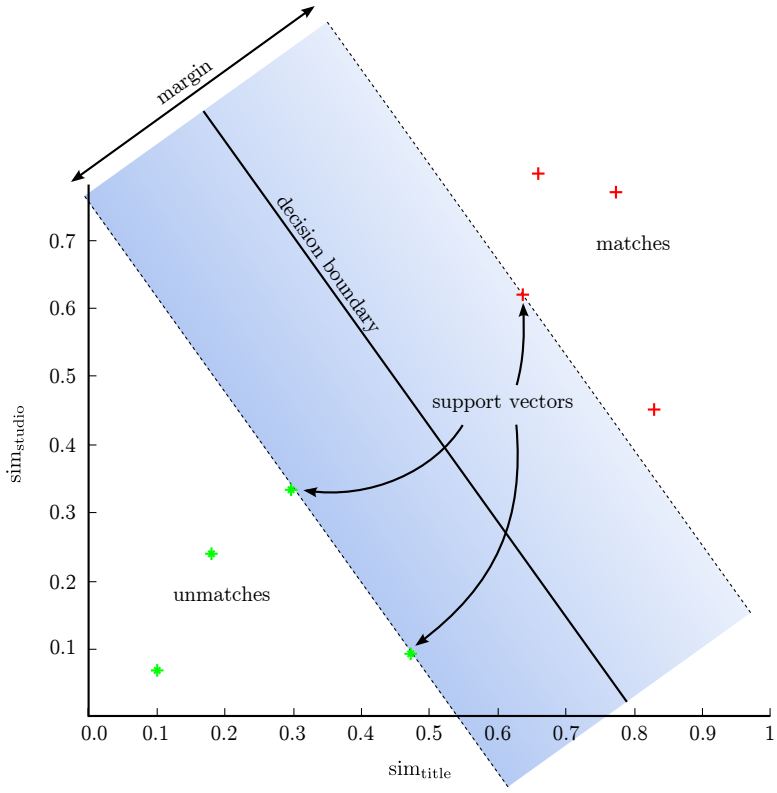


Figure 2.6.: Classification with a support vector machine.

Basically, a support vector machine can only perform a linear separation of the vector space into two classes, but there are some improvements that compensate for those shortcomings: as mentioned in [BM03], a more complex decision boundary can be achieved by mapping the training vectors into a higher dimension, computing a decision boundary for the high-dimensional vector representations and then transforming it back into the lower dimension of the original vector space. The decision boundary is linear in the higher dimension and hence even more complex in the lower dimension. As the functions that map the vectors

into the higher dimension are often referred to as kernels, this approach is called the *kernel trick*.

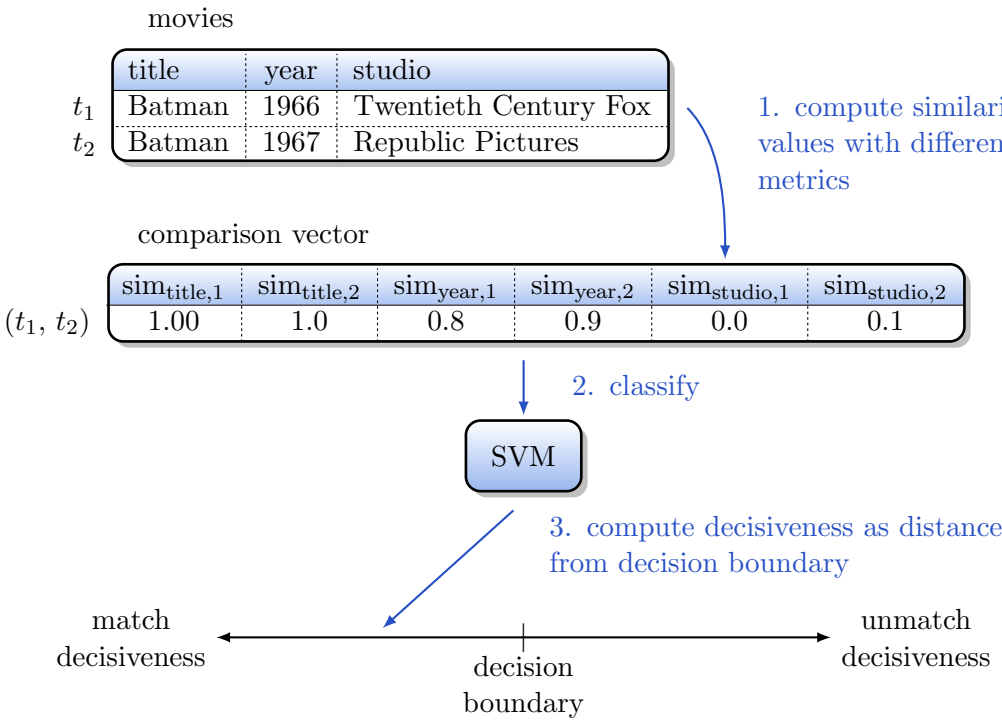


Figure 2.7.: A decision model incorporating a support vector machine according to Bilenko et al. [BM03].

A detailed survey of possibilities to separate more than two classes with support vector machines is given in Section 7.6 of [SS02]. The possibility of ternary separation can be helpful when implementing a decision model for comparison vectors labelled as M, U *and also* P, but is not mandatory. Figure 2.7 illustrates an SVM decision model by Bilenko et al. as described in [BM03] that can assign M, U and P labels, although it basically is just a binary classifier.

There are two movie tuples with title, year and studio attribute. The decision model has to assign a matching status to tuple pair (t_1, t_2). One peculiarity of this approach is the computation of a comparison vector; every attribute value pair is compared by several comparison functions.[4] In this example, there are three attributes and two similarity functions, so that the comparison vector corresponding to a tuple pair comprises six components. The second noteworthy new aspect of this decision model is that it does not only make a binary decision as to whether a tuple pair is a match or an unmatch, but also can declare a tuple pair to be a possible match in doubtful cases. The authors interpret the distance between an unknown comparison vector and the decision boundary as an indication of the *decisiveness* of the corresponding decision. Accordingly, a tuple pair is only declared a match or an unmatch, if the decisiveness of the respective matching status decision exceeds a particular

[4]The authors also propose an SVM-based comparison metric that can learn string similarities and thus adapt to specific domains very well.

threshold; if the decisiveness is too low, the tuple pair is declared a possible match.
In their work, Bilenko et al. show this approach to outperform decision trees in terms of precision and recall, especially with only little training data.

Unsupervised Machine Learning Decision Models

A great disadvantage of supervised learning techniques is that they require training data to function. As opposed to supervised machine learning techniques, *unsupervised machine learning* works without labelled training data.
One of the most important unsupervised machine learning techniques in duplicate detection is *clustering*. Basically, a clustering algorithm divides a given set of objects into clusters, i.e. groups, in such a way that objects in one cluster are rather similar to one another. Based on the assumption that comparison vectors with identical matching status have similar characteristics (and hence are near in comparison space Γ), clustering algorithms can be used to cluster matches and unmatches. However, a clustering algorithm does not provide labels for the clusters.

A very well-known clustering algorithm is the *k-means* algorithm [HW79] which divides a set of vectors into k clusters.
The k-means clustering algorithm works in the following fashion:

1. The algorithm is initialised with k cluster *centroids*.
2. Every comparison vector in the vector space is assigned to the cluster corresponding to the nearest centroid.
3. A new centroid is calculated for every cluster.
4. Step 2 and 3 are repeated, until the matching decisions are stable, i.e. do not change anymore, or until another termination criterion is fulfilled, e.g. until a number of maximal iterations has been performed.

In duplicate detection, the perfect matching and unmatching vectors may be chosen as initial centroids,[5] but the centroids may also be chosen completely at random.

The k-means clustering algorithm is very efficient for small values of k as they are used in the duplicate detection domain where $k = 2$ (M, U) or $k = 3$ (M, P, U). According to [GB06], though, the possible matches often do not form a distinct cluster and hence the 3-means algorithm leads to large possible match clusters in many real-world applications. Based on this observation, the authors of [GB06] propose binary clustering to separate matches and unmatches first, and then defining a *fuzzy region* somewhere between the centroids of the two clusters to classify possible matches.
First, the k-means algorithm is used to divide the comparison space into two clusters which

[5] In this case, duplicate detection could be performed *fully automated*, since the cluster labels are implicitly given by the choice of the initial vector (M for perfect matching vector and U for perfect unmatching vector).

are then labelled M and U by a domain expert. To find a reasonable fuzzy region, the authors use the relative distance

$$\Delta d_c = \frac{\mid d_{c,M} - d_{c,U} \mid}{(d_{c,M} + d_{c,U}) \cdot \frac{1}{2}}$$

of a comparison vector c to the two cluster centroids where $d_{c,M}$ is the distance to the centroid of the matching cluster and $d_{c,U}$ is the distance to the centroid of the unmatching cluster. Hence, a small Δd_i indicates that a final decision can only be made with a small certainty. Whether a comparison vector c is labelled with P or not, depends on whether Δd_c exceeds a threshold T_d. By defining this threshold, a trade-off between high accuracy and a small fuzzy region is made.
In an experimental evaluation, the authors show their approach to perform as well as a decision model purely based on 3-means clustering in terms of precision and recall, but also to achieve a smaller set of possible matches which can be crucial for real-world applications, since human decision-making is very expensive.

2.2.6. Hybrid Machine Learning Decision Models

As the results of purely clustering-based decision models are too imprecise in many cases, Verykios et al. propose a way to automatise supervised learners using clustering techniques [VEH00]. This is illustrated in Figure 2.8.
First, a small subset of all comparison vectors is divided into clusters and labelled by a domain expert. The tuple comparison vectors thus can be used to train a supervised learner which then, finally, can be used to classify all comparison vectors. The authors use a decision tree as supervised learning technique.
In their later work [EVE02], the authors implement this hybrid procedure in the duplicate detection tool box TAILOR. The evaluation of their experiments indicates that the hybrid approach is superior to an approach that is based on clustering alone.

In his work [Chr08a], Christen evaluates different hybrid learning models incorporating combinations of unsupervised and supervised learning methods. Some hybrid models that use clustering or distance-based methods to produce training data and then train a support vector machine are implemented in Christen's duplicate detection framework *Febrl* [Chr08b].

Christen's Iterative Approach

In [Chr08a], Christen describes a hybrid machine learning decision model that iteratively classifies comparison vectors with a support vector machine and thus further minimises human effort. The basic idea is to train the SVM with a small initial training set, first, have it classify the remaining comparison vectors, and then extend the training set by the most obvious matches and unmatches again and again, until a certain termination criterion is fulfilled.

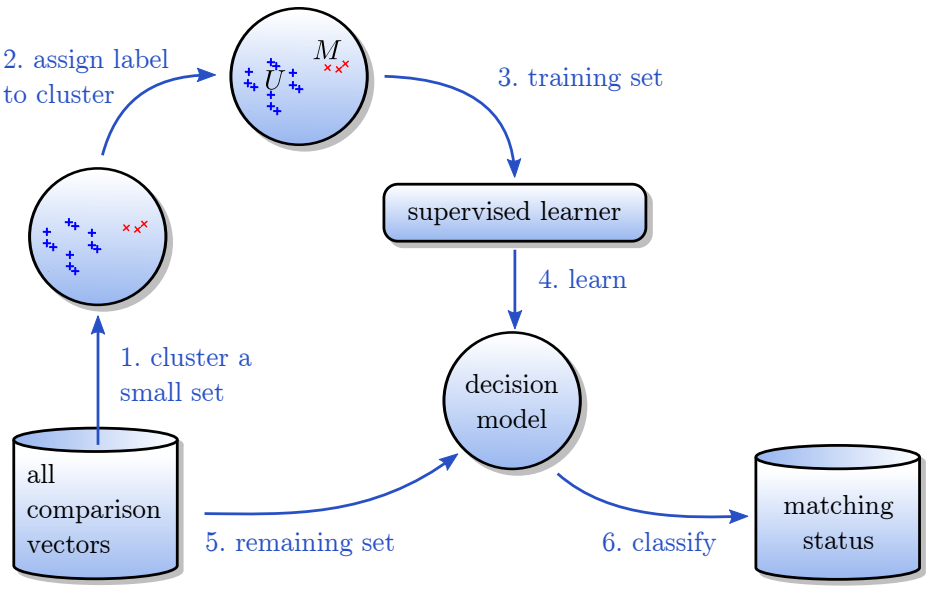


Figure 2.8.: A hybrid decision model combining unsupervised (clustering) and supervised machine learning techniques according to [VEH00].

Figure 2.9 illustrates Christen's iterative hybrid decision model.
First, the initial training set is determined with a distance-based method: the comparison vectors are sorted by their Manhattan distance to the perfect matching vector $[1, 1, \ldots, 1]$ (1.) and a predefined number of vectors from the top of the list (matches) and the bottom of the list (unmatches) are labelled and form the initial training set (2.). Christen calls this selection strategy *nearest-based selection*. The difficulty here is to choose the right number of vectors: choosing too few vectors results in an undertrained decision model, but if too many vectors are chosen, the probability of adding false positives and false negatives to the initial training set increases. The initial training set is used to train an SVM decision model (3.) which then classifies the remaining comparison vectors that have not been classified so far (4., 5.). As before, some of the classified vectors are labelled and added to the training set, but now the decisiveness of the SVM's decision (see Figure 2.7) is used to determine the best candidates. For example, all matches and unmatches with a decisiveness of 0.9 or more could be added (6.). The entire process comes to an end, if a termination criterion is fulfilled, i.e. for example if no comparison vector can be classified with a decisiveness of 0.9 or more.

2.2.7. Summary

All decision models discussed here have their particular benefits and drawbacks.
The probabilistic model by Fellegi and Sunter can, in theory, be used to automatise duplicate detection, but the computation of the required matching weights is often very hard and even infeasible, if the comparison space is too large.
Like the probabilistic model, the distance-based models can be used even when there is no training data available which can be a great advantage, but they require certain thresholds

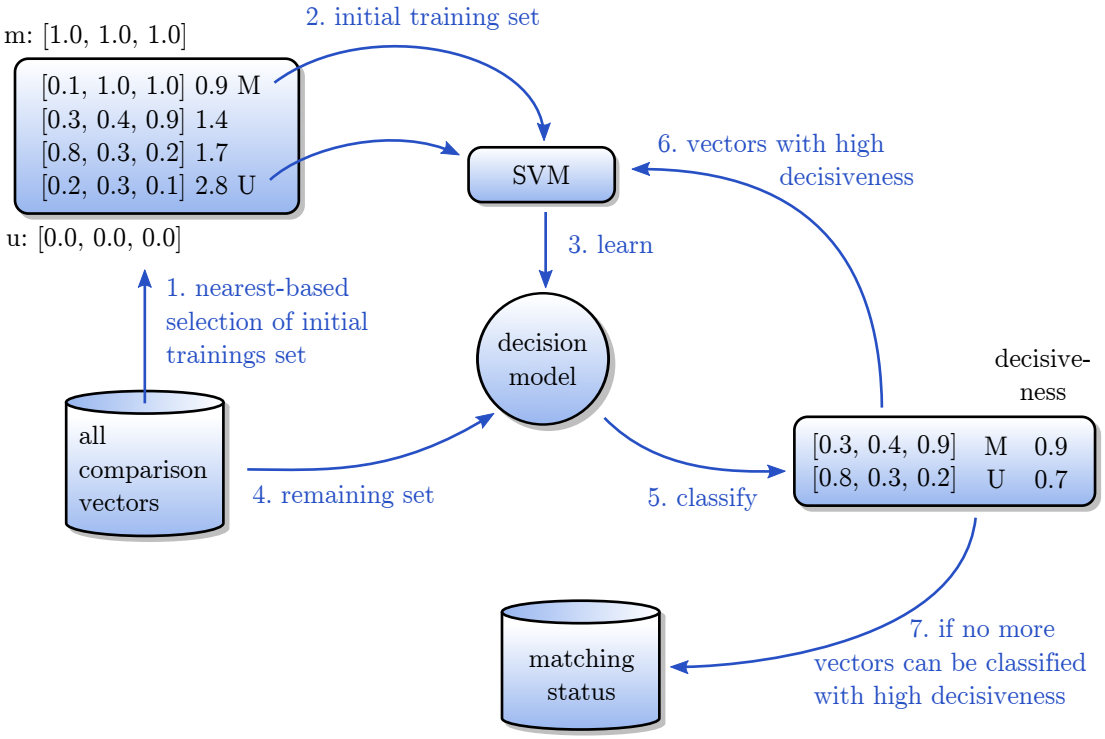


Figure 2.9.: Christen's iterative hybrid decision model according to [Chr08a].

to be defined first. Besides, distance-based decision models usually don't perform very well in terms of precision and recall.
Rule-based systems can deliver the best results, but are usually very complex and thus expensive, since the necessary rules have to be designed by human domain experts.
Supervised machine learning decision models can deliver very good results as well at moderate costs. Using decision trees offers the great advantage that the resulting model comprises a set of intelligible rules, while the model of a support vector machine, although more robust and usually more precise, cannot be comprehended by humans.
Purely clustering-based algorithms deliver not as good results, but, as the distance-based methods, do not require as much human effort.
In the current work on decision models, there appears to be a trend towards hybrid decision models which create an initial training set with clustering or distance-based algorithms to train a supervised learner.

2.3. Search Space Reduction in Probabilistic Data

Search space reduction in general means removing obvious unmatches from the set of all tuple pairs with cheap methods, so that the expensive in-depth tuple comparison in the process of duplicate detection is only executed on a feasible number of tuple pairs. In our bachelor's thesis [FW10], we examine possibilities to adapt the commonly used certain

data search space reduction techniques *Blocking* [EIV07] and the *Sorted Neighbourhood Method (SNM)* [HS95, HS98] to probabilistic data.

The investigated search space reduction techniques compute a key value for each tuple which is then used to decide whether or not a tuple pair remains in the search space. The basic idea is that duplicate tuples agree on their key values, while non-duplicate tuples do not. A key can for example be a particular attribute or a combination of (parts of) attributes.
The standard Blocking technique partitions all tuples in mutually exclusive *blocks* in such a way that only tuples with identical key values are assigned to one block. The reduced search space is then generated by pairing every tuple with every other tuple in the same block. Tuple pairs with tuples from different blocks are not permitted.
The basic Sorted Neighbourhood Method reduces the search space in three steps: first, a key value is computed for every tuple, then, the tuples are sorted by their key values and, finally, all tuples within a certain range in the sorted list are paired with each other (*merged*). The last step is also called *windowing*, because it is often visualised with a window of a fixed size w that is moved sequentially over the sorted list of tuples where all tuples framed by the window at the same time are paired with each other. When the window is moved down, the topmost tuple slides out of the window and a new tuple slides into the window, so that the set of all tuples is practically divided into overlapping subsets. The window size w can be increased to improve the recall slightly, but this also tends to reduce the precision dramatically [HS98].

For both search space reduction techniques, a good key design is crucial to achieve usable results.
A key with too much discriminating power results in a good precision, but also reduces the recall drastically: especially for Blocking, this is a problem, since duplicate tuples may be assigned to individual blocks and thus may not be paired, if their key values are not exactly identical. On the other hand, a key value that is not only identical for duplicates, but for many non-duplicates as well leads to a poor precision. In general, preprocessing the keys, for example by using a phonetic encoding such as the Soundex code instead of the actual attribute value, can make search space reduction more robust against small errors and thus improve recall without having a negative effect on the precision.

Depending on the data, there can be more than one usable key design. The authors of [HS98] claim that a multi-pass approach, i.e. running SSR techniques in several independent runs with different keys and combining the results, can increase the recall without decreasing the precision much, if this is the case.

In [FW10], several adapted variants of Blocking and the Sorted Neighbourhood Method are compared with respect to their performance in recall and precision on generated probabilistic data. Even though SNM is more expensive than Blocking due to the sorting step, it usually outperforms Blocking in terms of recall while maintaining an acceptable precision. Figure 2.10 illustrates one of the adapted Sorted Neighbourhood Method variants with

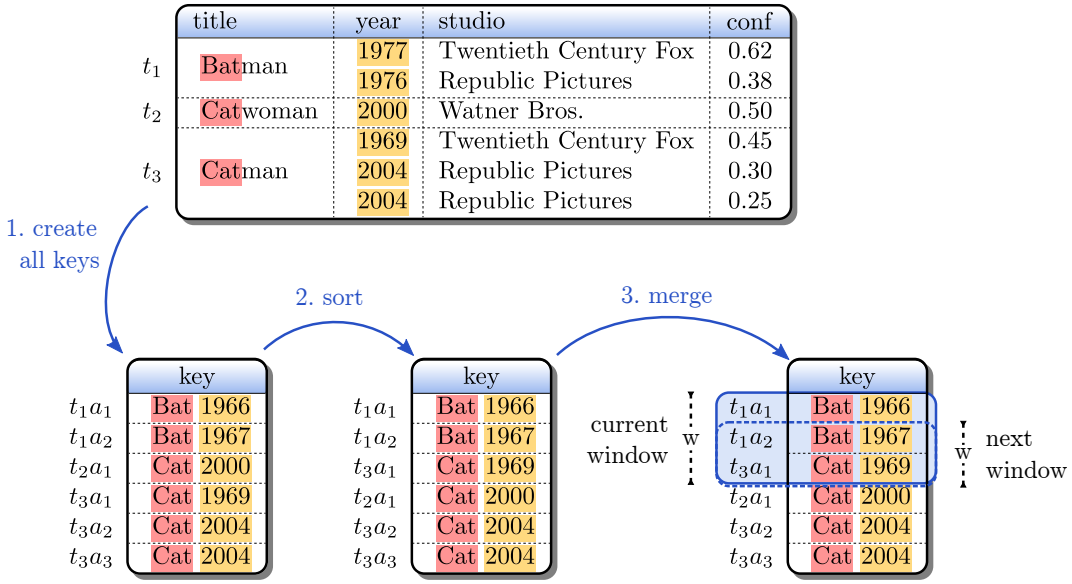


Figure 2.10.: This variant of the Sorted Neighbourhood Method is adapted to probabilistic data. A key value is created for every x-tuple alternative. The window size is increased dynamically, so that always the same number of tuples are paired.

window size $w = 2$ which delivers a comparatively good recall at an acceptable precision according to the experiment results.

In the example, a key value comprises the first three characters of the movie title and the entire year attribute. Obviously, the key attributes are not certain for all tuples, so that more than one key can be computed for some of the tuples. According to our experiments, computing keys for all alternatives as illustrated in the figure leads to the best recall, but there are also other approaches, for example generating only one single key per tuple from the most probable alternative.

In the sorted list, all neighbouring keys within a certain distance in the list are paired with each other where the corresponding tuple pairs form the reduced search space. To guarantee that the same number of tuple pairs is generated after every time the window is moved, the window size is dynamically increased: the first two alternatives in the sorted list correspond to the same tuple, so not even one tuple pair can be generated, unless the window is increased by one. By increasing the window, an alternative representing another tuple slides into the window and a tuple pair can be generated.

2.4. Generating Certain Data

A great challenge in the research of duplicate detection is the lack of publicly available real-world data sets to evaluate decision models.[6] This lack is partly based on the fact that for most real-world data sets the actual duplicates are not known, but privacy and legal

[6] Although this section is about generating *certain* data, this statement also, and even more so, applies to duplicate detection in probabilistic data.

issues also prevent data sets such as customer or patient databases from being published in many countries. Although such data sets can be used by the companies administrating them, they cannot be used by others and thus never lead to comparable results.
A small collection of data sets can be found in the *RIDDLE repository*[7], but it appears rather poor in comparison to what is available in the machine learning community.[8]

Under these premises, the *generation* of test data sets for evaluation seems attractive, since some parameters such as size and quality of the data are configurable and the resulting data sets can be published. However, generating synthetic data with realistic error patterns that meet particular requirements such as given frequency distributions for some attributes or dependencies between attributes is an extremely hard task.

In this section, we survey the state of research on data generation. First, we describe two different approaches to building a generator, namely generating completely synthetic data and generating data on the basis of real-world data sets. After that, we go into detail about literature on realistic error patterns and present existing data generators.

2.4.1. Two Types of Data Generators

In the following, we describe the basic two types of data generators according to [Chr05, CP09] and Section 7.5, 7.6 and 7.7 from the book [Chr12].

Completely Synthetic Data

Since data sets containing personal information usually cannot be published due to privacy issues, such data sets are often generated. For example, a fictitious customer database can be created from lists of forenames, surnames and addresses, so that the data do not refer to real people and the database can be published. Some attributes can also be synthesised without such attribute lists, for example telephone numbers can be generated according to rules. Apart from not giving rise to any legal issues, the great advantage of completely synthetic data is that the user has more control over certain characteristics of the data. Depending on the complexity of the generator, parameters like the desired number of records, attribute lists and sets of rules to use can be provided by the user, so that the data are generated according to the user's will.
A major obstacle in terms of realism, though, is modelling dependencies or correlations between attribute values: certain forenames, for example, cannot be used for males and females, and since the surname of a person often has something to do with the person's cultural background, it may be correlated with his or her address. The complexity of the generator and the number of parameters to set grows with the number and complexity of the modeled dependencies. If the configuration of the generator is too complex, some parameters may not be configured correctly or not be configured at all, because their effect

[7] RIDDLE data set repository: `http://www.cs.utexas.edu/users/ml/riddle/data.html`.
[8] UCI Machine Learning Repository: `http://archive.ics.uci.edu/ml/`.

on the generated data is not clear to the user.
Another difficulty in generating test data for duplicate detection is modelling realistic error patterns. For example, duplicates in a real customer database may come to existence because of simple typos during the input process, so realistic typos have to be applied to the artificial data in order to generate realistic duplicates. It is crucial that the generated data and the real-world data which the duplicate detection system is created for have similar characteristics, for example similar attributes with similar frequency distributions and similar error patterns. If this is not the case, the result of the evaluation on the synthetic data cannot be used to make predictions on how well the duplicate detection will work in a real application.

Using Real-World Data Sets

Another approach to obtain test data is to use an existing – preferably error-free – data set and to modify it, e.g. by adding duplicates. As with generating completely synthetic data, implementing realistic error patterns is crucial here as well.
A great advantage of this approach is that attribute dependencies and correlations don't have to be taken care of explicitly, since they are part of the original data. Accordingly, though, privacy is still an issue, since the original data are real-world data.

2.4.2. Error Modelling

In this section, we describe the introduction of realistic error patterns into the data according to [CP09]. As illustrated in Figure 2.11, data in a database can originate from different sources and be entered using different channels. Each source and input channel is prone to errors with particular characteristics, and there are characteristic errors for certain combinations as well.
For example, the confusion of synonyms or other words that are in some way related to one another can, for example, result from imprecise memories, while *phonetic errors* like the input value "bad man" instead of "Batman" are more likely when information is dictated, for example to a typist or to a speech recognition software. When the data are retrieved from printed or handwritten documents with optical character recognition (OCR) software, there most certainly are some words where characters have been confused with similar looking ones, e.g. "1" and "I". *Typographical errors* like the confusion of neighbouring characters on the keyboard or the transposition of subsequent characters in a word are typical for typed data.

Since the 1960s, text corpora have been analysed in the field of automated error detection and spelling correction to yield typographical error characteristics such as frequency distributions of particular errors.

The most typographical errors encompass simple omissions, substitutions and insertions of single or transposition of two neighbouring characters. In an early study [Dam64],

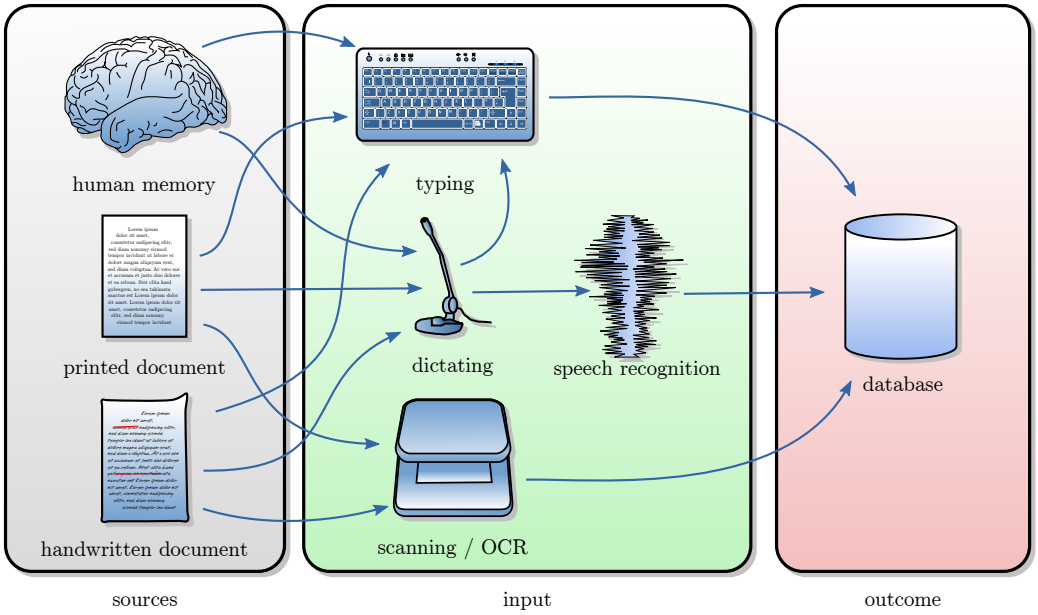


Figure 2.11.: Different data sources and input channels according to [CP09].

Damerau reports that 80% of all errors are such simple errors. Pollock and Zamora report an even higher share of 90% to 95% [PZ84]. The exact percentages of the particular simple errors, though, seem to differ from text to text. Table 2.2 illustrates the percentage of the mentioned simple error types reported in different works.
Apart from the frequency distribution, it can also be regarded that most of the errors occur in the middle part [PZ84, BS12] or near the end of a word [CCM+10].

Typographical errors can be implemented using *confusion matrices*. In [CG91], for example, Church and Gale provide confusion matrices for all four error types. For substitution errors, their confusion matrix represents the probability[9] of one particular character a being substituted by another particular character b for every character combination. The transposition of two characters is represented similarly. Omission and insertion errors are somewhat differently represented: the probability associated with the combination of two characters a and b represents the probability of character b being omitted or inserted, respectively, after the occurrence of character a.

error type	[BS12]	[CCM+10]	[Dam64]	[Pet86]	
omissions	45%	29%	16%	31.6%	34.4%
substitutions	15%	21%	59%	40.0%	26.9%
insertions	10%	1%	10%	18.7%	20.3%
transpositions	30%	25%	2%	2.6%	13.1%

Table 2.2.: Different reported percentages of the simple error types and the respective reference.

[9] In many cases, scores are used instead of probabilities.

The remaining 5% to 20% of errors are more complex errors, e.g. multiple typographical, phonetic or semantic errors. Such errors can be modelled with lists of *confusion sets*. An entry in such a list consists of one word, the so-called *head*, that can be confused with all of the subsequent words, the actual confusion set.[10] Optionally, the probability[9] for every single word to be confused with the head can be defined. By default, those probabilities are assumed to be uniformly distributed.
Some lists of general language confusion sets are publicly available, for example Pedler and Mitton published a list of almost 6,000 English confusion sets[11] whose creation is described in [CCM+10]. Another list of a few hundred homophones[12] has been published under the GNU General Public License as a part of the language checker *After the Deadline (ATD)*.

Confusion sets for phonetic errors can also be generated rather easily for a given set of strings by grouping them according to their phonetic encoding. To compute phonetic encodings, algorithms like *Soundex* [Rus18, Rus22] or *NYSIIS*[Taf70] can be used.
In [CP09] a list of phonetic rules is used to generate phonetic errors. Such a rule can for example state that the suffix "le" can be replaced by "ile", if the preceding character is a consonant. So the word "rumble" can be replaced by "rumbile" according to this rule. A great advantage of using phonetic rules is that they can be applied to any domain without preprocessing.

2.4.3. Existing Generators

The *UIS Database Generator*[13] [HS95] developed by Hernández and Stolfo in 1995 is the first software to generate data for duplicate detection in certain data. This generator creates a completely artificial address database consisting of the attributes social security number, first name, initial, last name, address, apartment, city, state and ZIP Code where some basic parameters like the number of tuples in the database and the number of duplicates can be controlled by the user.
Duplicates and the corresponding original tuples can differ by just small typographical errors, but they can also have completely different name or address values and even missing values. While the frequencies of the different error types are modelled according to statistics gathered from spelling correction studies, the attribute frequency distributions are not modelled realistically: for example, the names are chosen randomly from a list of real names and thus are equally distributed.

In 2003, Bertolazzi et al. presented an improved generator that gives the user more control over the generated errors [BSS03].

A generator that incorporates many improvements in comparison is part of the duplicate detection framework *Febrl* [Chr08b, Chr05]. There are two versions of the Febrl gener-

[10] In some works, there is no head element in a confusion set, because every word in the confusion set can be confused with every other word.
[11] Confusion sets by Pedler and Mitton: `http://www.dcs.bbk.ac.uk/~jenny/resources.html`.
[12] Homophone list from ATD: `http://static.afterthedeadline.com/download/homophonedb.txt`.
[13] UIS Database Generator: `http://www.cs.utexas.edu/users/ml/riddle/data.html`.

ator which are both enclosed in the Febrl system: Christen's original and an enhanced version [CP09]. In both versions, the adjustable parameters are documented comprehensively to give the user more control over the generation process. Attribute values are generated using external look-up tables that can even contain the frequencies of the attribute values. For every attribute, parameters like the probability for phonetic, typographical or OCR errors are configurable. While the model for typographical errors is hardcoded, phonetic and OCR errors are realised using external input files. For example, one line in the OCR error input file is "All, 2, Z" and thus specifies that the character "2" can be substituted by the character "Z" in all words.
All in all, the generator offers great control over the predefined error types and allows the user to integrate his or her own look-up tables and even to extend error generation rule sets and lists of confusion sets. Apart from the mentioned error types, there are also other error types like omission or permutation of attribute values. The new version of the data generator can also model dependencies between attributes, for example between gender and forename, and it supports the generation of groups of tuples that correspond to families.

In 2009, Talburt et al. presented a generator for duplicate detection test data that produces realistic *name and address histories* [TZS09]. Using real data such as publicly known addresses, the generator produces sequences of personal address data with attached timestamps that reflect a person's or a couple's change of residence or surname after marriage over time.
The authors plan to add more functionality to the generator, for example control over data quality or the possibility to output different file formats.

3. ProbGee: A Generator for Probabilistic Data Sets

In this chapter, we describe the data generator we developed to provide data for our evaluation experiments. We start with a high-level description of our generator and the generation process and then go into detail about the architecture, describing the most important parameters and their influence on the generated data. Finally, we briefly present some special features of the generator.

3.1. Basic Functionality

For the execution and evaluation of our adapted decision models, probabilistic data incorporating duplicates is mandatory. Since no such data exist to our knowledge, the first step in preparing the experiments is generating appropriate data. For this purpose, we propose *ProbGee*, an easy-to-use generator for probabilistic data sets with graphical user interface.

Currently, ProbGee does not generate completely synthetic data, but uses an existing data source with ideally no errors to generate probabilistic tuples with duplicates. However, we plan to implement the generation of completely synthetic data from attribute lists and production rules in the near future.
We deliberately use a few basic functionalities from the data generator for probabilistic data sets we implemented for our bachelor's thesis [FW10] such as loading or storing data, so that the output format is the same as in earlier experiments. This is an important feature, because it makes it possible to run the search space reduction techniques from our bachelor's thesis on the data produced with ProbGee.[1] Like its predecessor, ProbGee uses the very fast and easy-to-use relational database system *HSQLDB*[2] to store all certain and probabilistic databases.

As illustrated in Figure 3.1, the basic workflow in ProbGee is divided into three steps: the extraction of data from an external database into CSV files (1.), the import of CSV files into an internal database system (2.) and the actual generation process (3.).
The first step is to *extract* data from an external database and store them to CSV files. To connect to an external database, the user has to provide the corresponding JDBC driver.

[1] As described in Chapter 5, we use our own search space reduction techniques to obtain more realistic tuple matching results.
[2] HyperSQL DataBase: http://hsqldb.org.

The result set of any SQL statement given by the user can be stored to a CSV file. The extraction step is optional, since the raw data can also be provided in form of CSV files that were not generated using the graphical user interface.
The CSV files are then *imported* into the internal database system of the generator to gain more control over the data and to make it accessible via SQL. The user has to provide a list of SQL statements to create the relations the data are imported to; thus, it is possible to import not only single relations, but also more complex data structures, especially relations with foreign keys.
In the final step, the probabilistic database is generated from the imported raw data. First, the imported raw data are copied into a new database. Subsequently, new attributes for the tuple ID and alternative ID are introduced to every relation as new primary keys. To preserve the foreign key relationships from the original data, new foreign key attributes are added to all relations with foreign keys. Finally, x-tuple alternatives and duplicate tuples are generated according to the parameters specified by the user via the graphical user interface.

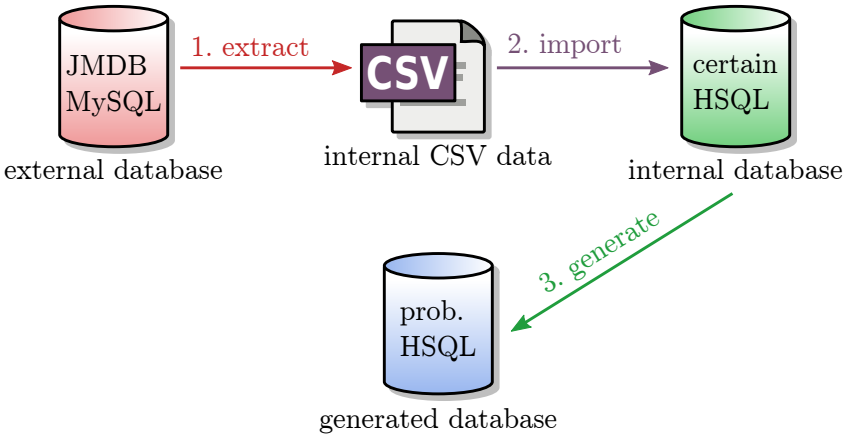


Figure 3.1.: The basic data generation workflow with ProbGee.

We decided for CSV files as input format during the import step, because they can be generated easily and because data such as attribute lists often are available on the internet in this format.
Besides, compatibility issues between our generator and an unknown (user-provided) database system can be avoided, because there is no necessity for our generator to communicate with any database system due to the possibility of importing simple CSV files. Many database systems are capable of exporting relations or query result sets into CSV files, so that importing data *indirectly* from a database via CSV files is usually no problem.

For our experiments, we use movie data from the *Internet Movie Database (IMDb)*[3]. The data are extracted from a MySQL database holding information on millions of movies, for example title, producer, director and production year. How an identical database can be

[3]Internet Movie Database: `http://www.imdb.com/`.

created from freely available text files is described on the official homepage of the project *JMDB*[4].

3.2. The Architecture

As indicated in Figure 3.2, the program modules responsible for the generation of probabilistic data are structured in a hierarchy of generators. Every generator implements a particular interface to make the exchange of single generators possible.
In this section, we describe our implementations of the single generators in detail.

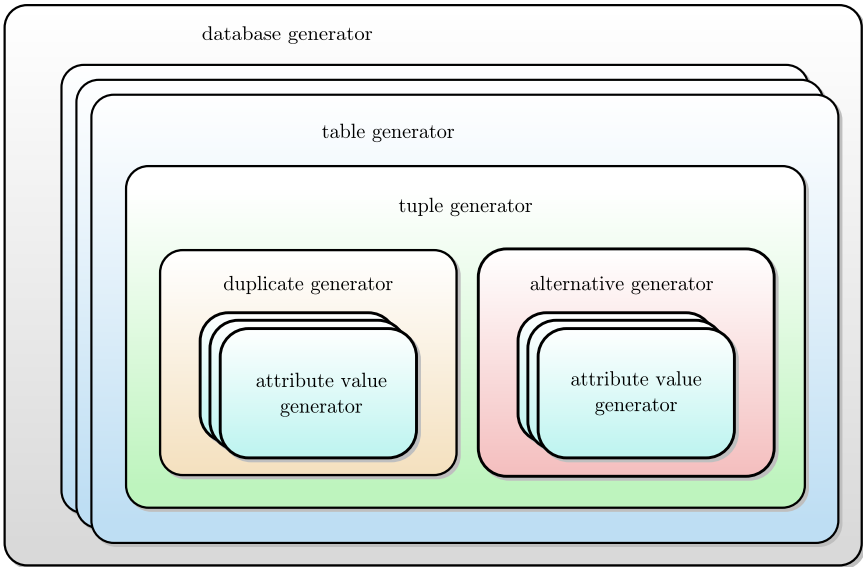


Figure 3.2.: The generator hierarchy in ProbGee.

3.2.1. The Database Generator

The database generator initialises the entire generation process. Since the generation of duplicates and other errors is highly domain-dependent, the database generator assigns every relation in the database an appropriate table generator.

Besides, the database generator controls in which order the relations are processed. This is an important issue when generating duplicates: if for example relation R_1 references relation R_2 and errors are introduced into R_1, before the referenced relation R_2 is processed, no duplicate in R_2 can be referenced by any tuple in R_1.
Since our movie database consists of one single relation without any foreign keys, this feature is not used, yet. However, a simplistic error generator that produces foreign key errors has been implemented by us already (see Subsection 3.3.3).

[4]Java Movie Database: http://www.jmdb.de.

3.2.2. The Table Generator

Before a probabilistic database can be generated, some parameters have to be specified, most importantly:

- *distribution of the duplicate cluster size*: this is not a number, but a complex parameter that defines how many probabilistic tuples are generated from a certain data tuple and thus represent the same real-world entity. Tuples representing the same real-world entity form a *duplicate cluster*.
  For example, it can be specified that 1,000 entities shall be represented by one tuple each, 500 entities shall be represented by two tuples each and 100 entities shall be represented by three tuples each. By specifying the distribution of the duplicate cluster size, the number of probabilistic tuples in the destination database is automatically given; a database with the just mentioned distribution will have $1{,}000 \cdot 1 + 500 \cdot 2 + 100 \cdot 3 = 2{,}300$ tuples.
- *distribution of tuple size*: similarly to the duplicate cluster size distribution, this parameter determines how many alternatives form one tuple.
  A user can, for example, determine that 800 tuples shall have only one alternative, 600 tuples shall have two alternatives and 400 tuples shall have three alternatives. The implied number of probabilistic tuples for these values is $800+600+400 = 1{,}800$.
- *number of maybe-tuples*: the number of tuples with a confidence smaller than 1.

The two exemplary distribution parameter settings just mentioned are contradictory: according to the cluster size distribution, the destination database will contain 2,300 tuples, while the tuple size distribution indicates 1,800 tuples. To make the generation process more robust against those kinds of contradictions, all surplus tuples receive a single alternative by default.

The table generator does not only resolve contradictory settings, but it also initiates the generation of every single duplicate cluster by the tuple generator, making sure in the process that all the above parameters are put into practice *exactly*. In principle, the duplicate cluster size and the size of a tuple are selected with a probability that corresponds to the respective user-defined distribution, but are also constrained by it. In the example setting, the first duplicate cluster that is generated is formed of one, two or three tuples, each with a respective probability of $\frac{1{,}000}{1{,}000+500+100} = \frac{1{,}000}{1{,}600}$, $\frac{500}{1{,}600}$ and $\frac{100}{1{,}600}$. If the cluster size is chosen to be three, the probabilities for the cluster sizes of one, two and three in the next turn are $\frac{1{,}000}{1{,}000+500+99} = \frac{1{,}000}{1599}$, $\frac{500}{1{,}599}$ and $\frac{99}{1{,}599}$ respectively, and so on.

When the generation process has finished, the table generator iterates over all tuples, reordering the confidence values in such a way that the alternatives with the greatest cleanness value receive the greatest confidence.

Finally, the table generator creates a LaTeX file and compiles it to a PDF file illustrating the most important parameters described in this chapter with some tables and plots.

3.2.3. The Tuple Generator

The most complex of our generators is the tuple generator that creates a duplicate cluster from a certain data tuple. As described in the introduction already, one requirement for our new generator is to give the user control over the similarity between duplicate tuples *and* the similarity of alternatives of the same x-tuple.
To make the similarity between two x-tuple alternatives easily configurable, it is measured as the weighted sum of all attribute similarities, i.e. a real number in the range of [0,1]. Obviously, for every attribute, a weight and a similarity metric have to be defined first, though; in our implementation, every attribute value pair is compared with the Jaro-Winkler metric by default.[5]

Error Provenance Trees

One problem with generating probabilistic data is the complexity of the process and of the resulting data. To make the generation process more comprehensible and to help the user understand whether it led to the desired result or not, we generate duplicate clusters from certain data alternatives using a data structure called *error provenance tree*. Figure 3.3 shows a schematic view of an error provenance tree and illustrates its most important defining properties. Every node in the tree corresponds to an x-tuple alternative and every child node is, basically, just a copy of its respective father node that has some additional errors. In simple words, the tree nodes (alternatives) grow dirtier from the root to the leaf nodes.
The root of the tree holds the uncorrupted certain data and thus is the correct version of every alternative in the entire duplicate cluster. The child nodes of the root node are prototypes for every x-tuple in the cluster. These prototypes are used for the generation of x-tuple alternatives, but are not part of the actual tuples themselves. In the database, they are stored with the alternative ID 0 and hence are called *zero-alternatives*.[6] Nodes in the tree are either zero-alternatives, alternatives of an x-tuple (opaque nodes) or neither (transparent nodes). Nodes that have no descendant that is part of a tuple and that are not part of a tuple themselves are of no importance whatsoever for the generation process and hence can be removed.
The point of organising generated alternatives in a tree structure is that alternatives of the same tuple have similar errors because they all – directly or indirectly – inherit errors from the same (zero-)alternatives. Apart from that, the correct representation and the development of every alternative are easily perceptible.

The characteristics of an error provenance tree as illustrated in Figure 3.3 are also key parameters for the generation process. The following parameters for the generation of an error provenance tree are specified by an *expectancy value* and a *standard deviation* and hence are normally distributed:

[5]The Jaro-Winkler metric is not the only available metric.
[6]In our notion, the alternative IDs of an x-tuple range from 1 to the number of alternatives in the tuple.

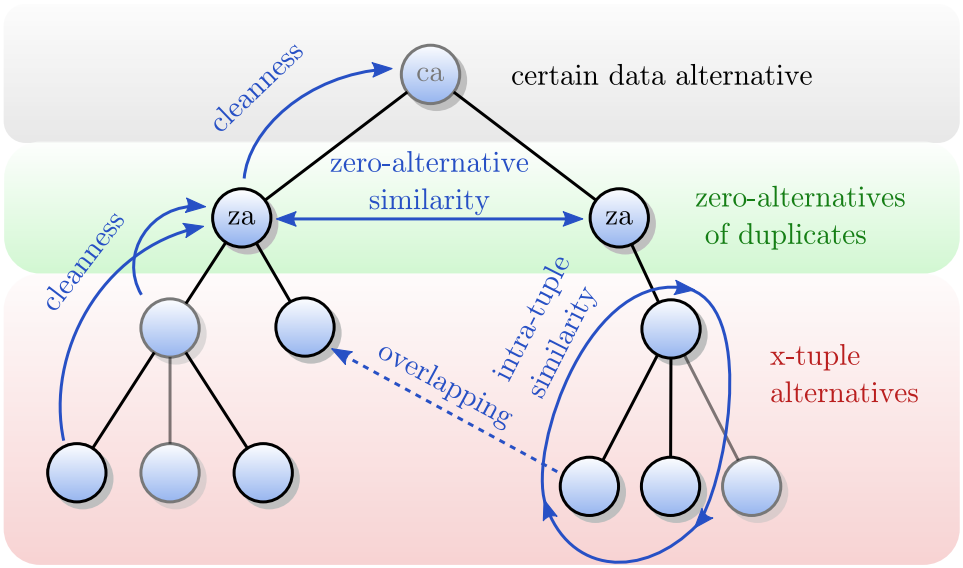


Figure 3.3.: A schematic view of an error provenance tree.

- *zero-alternative similarity*: the average similarity of one zero-alternative to all other zero-alternatives. The similarity between duplicate tuples can be configured indirectly with this parameter.
  Actually, the *duplicate similarity*, i.e. the average similarity between the alternatives of different duplicates, would be a better parameter, but in a duplicate cluster, there are practically always much more alternative pairs between duplicate tuples than zero-alternative pairs, so that the computation of the duplicate similarity is much more expensive than the computation of the zero-alternative similarity. Furthermore, the zero-alternative similarity and the duplicate similarity are usually very similarly distributed anyway (see Figure 5.6 in Section 5.2.1), so that this trade-off between computation time and correctness seems reasonable.
- *intra-tuple similarity*: the average similarity of one x-tuple alternative to all other x-tuple alternatives of the same tuple. Alternatives that are not part of the tuple are not taken into account.
- *cleanness*: the cleanness of an x-tuple alternative is its similarity to its corresponding zero-alternative. Correspondingly, the cleanness of a zero-alternative is its similarity to the root node and the tuple cleanness is the expected cleanness of its alternatives.
- *overlapping*: a number in the range of [0,1] representing the share of alternatives in a tuple that are identical to alternatives of another tuple in the same duplicate cluster. The actual duplicate similarity can be influenced by this parameter, since a high share of identical alternatives among duplicate tuples raises the duplicate similarity without affecting the zero-alternative similarity.
  Overlapping alternatives are copied into the destination subtree as direct descendant of the corresponding zero-alternative to indicate to which duplicate they belong. To illustrate their origin, they are also marked with a dashed arrow from the original to

the copy.

For example, this is how a duplicate cluster consisting of three tuples with ten alternatives each is created from an error provenance tree with an overlapping value of 0.3: since there is no other tuple to copy alternatives from, yet, the alternatives of the first tuple are entirely selected from the tree, regardless of the overlapping value. For the second tuple, about[7] three alternatives are copied from the first tuple and about seven alternatives are chosen from the error provenance tree. For the third tuple, also about three alternatives are copied from one of its duplicates, before the remaining alternatives are chosen from the tree.

Generating a Duplicate Cluster

The basic course of action when generating a duplicate cluster is to grow an error provenance tree first, and then select alternatives greedily from the tree to build x-tuples in such a way that the user-specified parameters are realised as precisely as possible. In the following, this procedure is explained in more detail. After that, some more parameters are described by which the user can interact with the tuple generator.

Initially, a list of zero-alternatives is generated with the duplicate generator (see Subsection 3.2.5), so that there is one zero-alternative as prototype for each tuple in the duplicate cluster. After that, alternative nodes are generated for every tuple where rather clean alternatives are chosen to grow a new child node with a higher probability. Since the alternatives near the root node are cleaner, the resulting tree is rather wide and does not grow deep very quickly. The tree growth is terminated after the generation of the first alternative whose cleanness drops below a certain threshold that lies way below the user-defined average cleanness, so that there are many alternatives available for varying degrees of dirtiness, ranging from clean to very dirty. This dynamic termination criterion makes sure that there are always alternatives in the tree that are much cleaner and alternatives that are much dirtier than the desired average, while at the same time the tree is never unnecessarily large[8]. To ensure the termination of the tree growth, it is also stopped, when the absolute number of alternative growth spurts exceeds a very large multifold of the desired tuple size.

The selection of alternatives that become part of the final tuple is executed with a greedy algorithm: first, some alternatives are copied from other tuples in the duplicate cluster, depending on the respective overlapping value and, of course, depending on whether there are other tuples in the cluster or not. The remaining alternatives of a tuple are chosen from the set of descendants of the corresponding zero-alternative in such a way that the tuple's intra-tuple similarity is as close as possible to the desired value. If the first alternative is not copied from another tuple, the alternative is chosen whose cleanness is closest to the

[7] The exact overlapping value varies from tuple to tuple, depending on the standard deviation specified by the user.

[8] A high cleanness value given by the user leads to a rather early termination, because only few dirty alternatives have to be generated. When the user specifies a low average cleanness, the tree generation takes longer, because more dirty alternatives are needed and thus a deeper tree has to be grown.

specified value. Since the other alternatives are chosen to be similar to the first alternative, their cleanness is usually similar as well.

The generated tree is *cleaned*, i.e. all branches are removed, if they do not contain any alternatives that are part of one of the tuples. Finally, a LaTeX file is generated from every cleaned error provenance tree and then compiled into a PDF file, so that the user has an overview over every duplicate cluster. Figure 3.4 shows an error provenance tree exactly as it was generated during the generation process. By default, the generated error provenance trees grow from left to right, but they can also be configured to grow vertically as depicted. In addition to the cleaned trees, the uncleaned trees can also be compiled. However, to save memory, the entire LaTeX feature can be turned off as well.

Figure 3.4.: An error provenance tree as it is displayed in the ProbGee user interface after the generation process.

There are many more parameters by which the generation process can be influenced, most importantly:

- *poolsize factor*: the minimum number of alternatives to generate for a tuple subtree in an error provenance tree, depending on the desired tuple size. For example, if the poolsize factor is 5 and the tuple in question will be formed of four alternatives, the respective zero-alternative in the error provenance tree will have at least $4 \cdot 5 = 20$ descendants.
- *depth factor*: whether the tree grows more into the width or into the depth can be influenced by this parameter. A positive value increases the probability of a node on the lower levels to be chosen for tree growth, while a negative value further increases the probability of nodes in the proximity of the root node to be chosen and thus favours growth into the width.
- *probability to choose the best alternative*: this parameter represents the probability by which the greedy selection algorithm described above chooses the best candidate, i.e. the alternative that leads to the intra-tuple similarity closest to the specified value, when forming a tuple. Values in the range from 0 (always random) to 1 (always best) define the selection strategy. For example, a value of 0.8 means that the best alternative is chosen with a probability of 80% and a random alternative is chosen with a probability of 20%.

3.2.4. The Alternative Generator

The alternative generator creates a copy of a given x-tuple alternative and introduces an error to one of its attributes by applying an attribute value generator (see Subsection 3.2.6).
After a copy of the given x-tuple alternative has been created, one of its attributes is chosen to be corrupted. If there are attributes that are uncorrupted in comparison to the corresponding zero-alternative, only those attributes can be chosen. The probability for one of those attributes to be chosen is anti-proportional to the user-defined matching weights described in Subsection 3.2.3. For example, if the attributes title (weight: 50), studio (weight: 10) and producer (weight: 30) are error-free, the probability for the producer attribute to be chosen is $\frac{50}{50+10+30} = \frac{5}{9}$. If there is no uncorrupted attribute, all attributes can be chosen.

Having determined an attribute, an error type is chosen. A simple typographical error is applied with a probability of 80% and a complex error is applied with a probability of 20%. Figure 3.5 Illustrates that the exact error type is determined according to scores.
When a simple typo is in order, one of four different types of typographical errors can be applied: omission of a single character, substitution of a single character by another single character, insertion of a character or transposition of two subsequent characters. The default values for the probabilities for the different error types are also depicted and

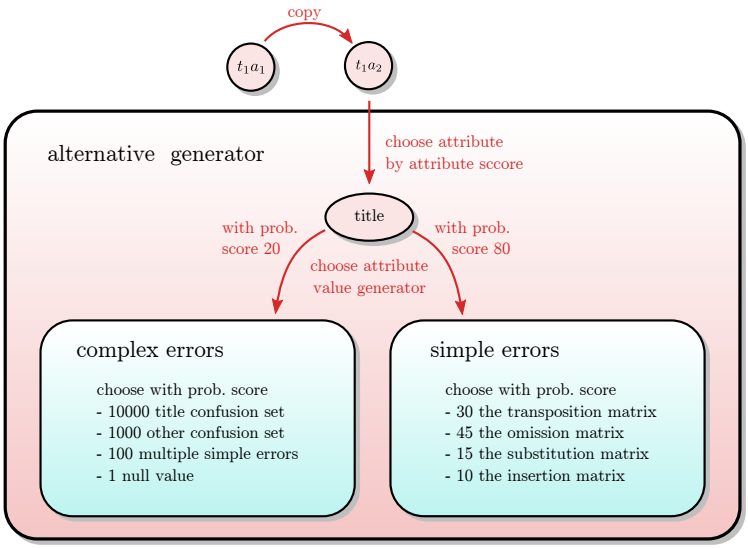


Figure 3.5.: The alternative generator chooses an attribute, first, and then introduces either a simple or a complex error.

are chosen according to the references given in Section 2.4.2.
While simple typos can be applied in most cases, some complex errors can only be applied to particular values: so, for example, when a title confusion set error is chosen, it is well possible that it cannot be applied, because the available list of confusion sets does not contain any substitutions for the given title. In this case, another generator is chosen the same way as before, but, of course, without the possibility of choosing the generator that just failed. For this reason, a very high score for a particular generator does not necessarily mean that this generator is applied very often. Apart from the title confusion set list, ProbGee provides other lists of confusion sets, for example confusion set lists for names or general words like “the” and “a”. Multiple typographical errors receive a rather low score in our default setting, because they can be used almost always and hence are used as a fallback strategy anyway, so that they are applied more often than it is implied by the low score. Replacing an attribute value by null is practically the worst error that can be applied, so it has the minimal score and is only tried, if nothing else works.
Depending on the tuple generator parameters, the influence of the scores defined for the alternative generator on the final data can be minimal. For example, if the tuple generator is configured to select alternatives with a very high cleanness, confusion set errors are unlikely to appear in the final data.

The configuration of all generators is stored to an XML file that can be edited using ProbGee’s graphical user interface or a common text editor. All scores are freely configurable and single generators can be added or removed. More details on the possible attribute value generator configurations are given in Section 3.2.6.

3.2.5. The Duplicate Generator

In principle, the duplicate generator works like the tuple generator and hence can be configured with almost identical parameters: it first generates a set of candidate alternatives from a certain data alternative using an alternative generator and then selects a predefined number of candidates, the first according to the user-defined zero-alternative cleanness and the rest in such a way that the specified zero-alternative similarity is approached as closely as possible.
The main difference to the tuple generator is that the used alternative generator is by default configured in a completely different way: the probability of simple and complex errors for zero-alternatives are inverse to the corresponding probabilities in the tuple generator's alternative generator, i.e. simple errors occur with a probability of only 20%, while complex errors occur in 80% of the corrupted attributes. The idea behind this is that duplicates should have more fundamental differences than alternatives of the same x-tuple. Two tuples can have a certain similarity because of small errors like single typos in *several* attributes or because of a single complex error in *one* of the attributes. Since zero-alternatives are the prototypes of the duplicates in a duplicate cluster, it makes more sense for them to differ in more fundamental aspects, for example to have semantically different movie titles, than merely differ in the exact spelling of some attributes.

3.2.6. The Attribute Value Generator

An attribute value generator takes an attribute value of a particular type as input and generates a corrupted version of this value. Since all attribute values in our movie database are string values, all our implementations are applicable to strings only; the attribute value generator interface, though, is defined for arbitrary values.

Simple typos are modelled with *confusion matrices*[9] that contain probability scores for particular errors in certain character combinations. In the default configuration, the confusion matrices from [CG91] are used, although arbitrary matrices can be incorporated by simply manipulating the XML configuration file. Since the matrices we use are based on empirically determined relative frequencies, all confusion matrices are *smoothed* by default, i.e. all scores are first multiplied with 1,000 and then all scores smaller than 1 are raised to 1. This ensures that every character combination is at least remotely possible, even if it has not been observed in the study that yielded the respective matrix. The smoothing option can also be turned off.
When a typographical error is to be applied, it is determined which of all the character combinations modelled in the matrix occur in given attribute value. Whether or not this is done case-sensitively is determined by a parameter. Every occurrence of such a character combination represents a possibility to introduce an error. One of those occurrences is chosen with a probability proportional to its score.

[9] Details on confusion sets and confusion matrices can be found in Subsection 2.4.2.

As mentioned above, four kinds of typographical errors are modelled explicitly: the omission or insertion of a character after the occurrence of a particular other character, the substitution of one character by another character or the reversal of two neighbouring characters.

Semantic errors are modelled as lists of *confusion sets*[9] holding synonyms or different spellings of a word or phrase. A list of confusion sets can be used to confuse the entire attribute value or only single words in the attribute value, the latter being especially reasonable for commonly used words like articles or adjectives. A list of confusion sets can be applied to an attribute value, if the value equals the head of one confusion set contained in the list. Just like with the confusion matrices, the user can determine whether or not the character case is taken into consideration for this containment check.
Apart from the lists of confusion sets mentioned in Subsection 2.4.2, our generator uses confusion sets for movie titles and names that we generated from synonym tables in the JMDB database (see Section 3.1). The lists of confusion sets are stored as text files. Which list of confusion sets is used by a generator is specified in the XML configuration file.

A *generator group* can be understood as a kind of meta generator, since it only picks one generator from a set of generators to use and does not generate errors itself. It contains several attribute value generators where each one has an associated weight that corresponds to the probability by which it is chosen.
Generator groups are used to group several generators into simple and complex errors in the alternative generator (see Subsection 3.2.4). By using generator groups, a set of generators can be used as if it was a single generator, making the two-level choice between simple and complex errors (level one) and the actual error generators (level two) as illustrated in Figure 3.5 possible in the first place.

The year attribute in the database holds string values in various formats, for example “1976”, “1974–1981”, “1999–2002,2005–????”, “1971–1982,1998–2007” or “2008 – (working title)”. To build more natural errors, we implemented a highly domain-dependent *year attribute generator* that introduces small numerical errors or shifts time intervals, while at the same time making sure that the errors don’t get too absurd; for example, the first year in an interval can never be greater than the last year, so that values like “2001–1987” can never occur.

In the real world, some attributes can be confused with one another, for example the producer and the director of a movie. For such cases, we implemented an *attribute confusion generator* that simply swaps the values of two different attributes. This is the only generator that does not comply with the general interface, since it needs more than just one single attribute value as input. In the default setting, only the producer and director attributes can be swapped, but the user is free to make any other combination of attributes confusable via XML.

The *null value generator* is the simplest of our generators; it just returns a null value, no matter what the input is.

3.3. Special Features

In this section, we describe some features that we consider unique characteristics of our data generator. We present our graphical user interface and then explain how configuration files are persisted. Finally, we briefly describe our example implementation for the generation of probabilistic databases with foreign keys.

3.3.1. The RCP GUI

The user interface of ProbGee is built on top of the *Rich Client Platform (RCP)*, so that it does not just resemble the Eclipse GUI, but even uses some of the Eclipse modules, for example the XML editor. Being able to use already existing program modules is a great advantage of using an RCP GUI: to begin with, this saves a lot of work that otherwise would have to be spent on the development of modules, but apart from that, it makes the use of our program more intuitive, since parts of the program are already known to the common Eclipse user.

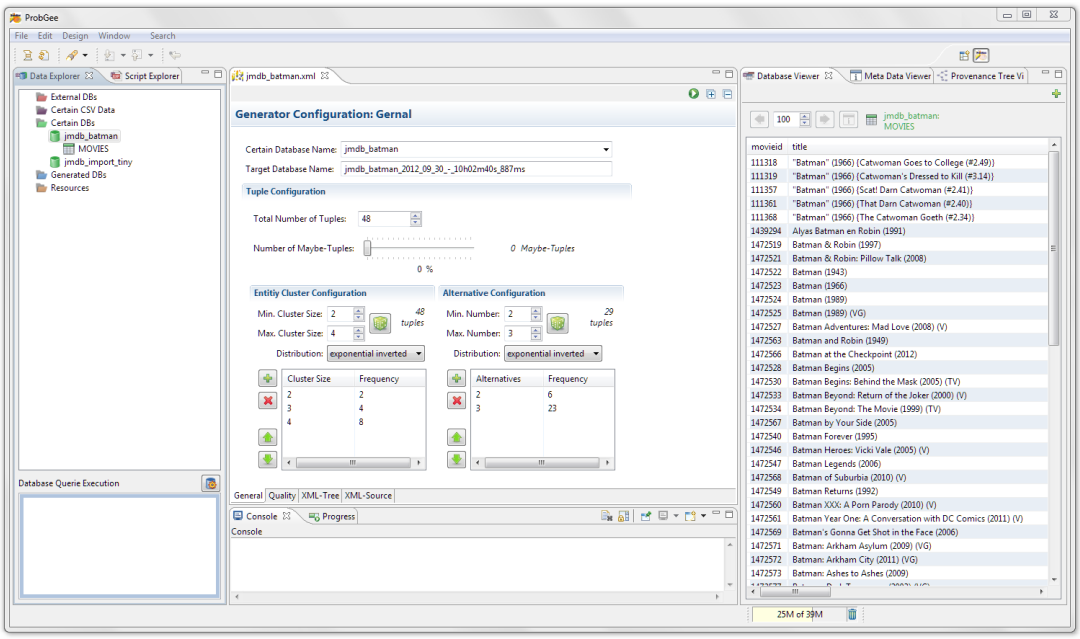


Figure 3.6.: ProbGee's graphical user interface.

Figure 3.6 shows a screenshot of our graphical user interface. There are several views that behave like Eclipse views; they can be positioned, scaled, closed and reopened.
On the left of the picture, the *Data Explorer* can be seen that lists external databases, certain data databases, probabilistic databases and CSV files. A double-click on a relation or a CSV file opens the *Database Viewer* on the right which shows the first 100 rows or lines of the respective relation or CSV file. Depending on what has been double-clicked, the user can initiate different processes: a double-click on an external database initiates the export of data into a CSV file, double-clicking a CSV file initiates the data import process and

double-clicking a relation in an internal certain data database triggers the generation of a probabilistic database. The necessary configuration for each process is done in a view that opens up on double-click in the centre of the application window. Every process is started by clicking on the run button known from Eclipse in the upper right of the configuration view; all parameters are automatically saved to an XML file, so that the process can be repeated later. All console printouts are displayed at the bottom, just like in Eclipse.
The *Script Explorer* is hidden behind the Data Explorer and only visible through the tab label in the given screenshot. It works similarly to the Data Explorer, but is used to administer script files. One way of repeating a process that already has been executed is to double-click on the corresponding script file in the Script Explorer: the appropriate editor opens in the centre of the application, so that changes can be made, and then the execution can be started by clicking on the run button in the top right corner of the view. Beneath the Data Explorer, the user can enter an SQL query that can be executed in the currently opened relational database. The result set produced by the query is then displayed in the Database Viewer.
Double-clicking a tuple in a probabilistic database that has been generated with ProbGee opens the *Provenance Tree Viewer* that displays the complete[10] error provenance tree corresponding to the clicked alternative. Clicking the information button at the top of the Database Viewer opens the *Meta Data Viewer* displaying statistics on the generated database.

All parameters for the generation of a probabilistic database can be specified with the generic XML editor known from Eclipse which is integrated into our GUI. Many parameters have reasonable presets and don't have to be touched by the user, though, while some parameters have to be defined by the user. Those parameters can also be configured with a specialised, more intuitive interface.
The configuration view for the generation of a probabilistic database comprises four tabs. The first tab covers general parameters like the number of tuples, the duplicate cluster size distribution and the distribution of the number of alternatives per tuple. The second tab offers an intuitive interface for the most important data quality settings, especially for similarity values and probabilities. The third and the fourth tab contain the tree view and the source view of the eclipse XML editor for the configuration of each and every single parameter in the configuration file. Since a generic XML editor is used, arbitrary parameters can be configured with the graphical user interface, so that it does not have to be adapted, when new parameters are added to the program logic.

3.3.2. XML Binding

In ProbGee, not only all parameters for the generation of a probabilistic database including the parameters for all generators are persisted, but also the import and export parameters.

[10] Note that not only all tuple alternatives, but also all alternatives from the entire duplicate cluster are illustrated in the tree.

The *Java Architecture for XML Binding (JAXB)* is used to create a binding between Java objects and XML documents.
The generation of an XML document from a Java object is called marshalling and, correspondingly, the creation of an object from an XML document is called unmarshalling. Implementing these two procedures requires some work at the beginning and the compliance to some conventions such as providing an empty constructor as well as getter and setter methods for the persisted attributes, but it is relatively easy. The JAXB implementation *MOXy* from EclipseLink Project[11] allows to create the data binding by simple annotations in the Java source code. Listing 3.1 shows the declaration of a `double` attribute including the corresponding annotation that defines the XML Path of this attribute in the configuration file structure.

```
@XmlPath("quality/tuples/intra_tuple_similarity/text()")
private double intraTupleSimilarity;
```

Listing 3.1: Creating an XML binding with annotations in the source code.

The relevant part of the corresponding configuration file is shown in Listing 3.2.

```
<?xml version="1.0" encoding="UTF-8"?>
<ProbGee_Generator_Configuration>
   ⋮
   <quality>
      ⋮
      <tuples>
         ⋮
         <intra_tuple_similarity>0.95</intra_tuple_similarity>
         ⋮
      </tuples>
      ⋮
   </quality>
   ⋮
</ProbGee_Generator_Configuration>
```

Listing 3.2: A simplified representation of an XML configuration file used by our generator.

3.3.3. The Easy-to-Understand Dummy Implementation

Besides the generation architecture discussed in this chapter, we also implemented a simplistic *dummy generator* for every interface to make the basic workflow of every generator better comprehensible. As a matter of fact, the complex generator implementations presented above originate from the dummy implementations. Other developers can implement their own generators in our framework by copying the dummy package and changing or

[11] EclipseLink Project: `http://www.eclipse.org/eclipselink/`.

extending the existing implementations, just as we did.
Since a single probabilistic table without foreign keys suffices for the evaluation of our tuple matching experiments, there is no complex version of the *foreign key generator dummy* we also implemented. It introduces errors to foreign key attribute pairs (tuple ID and alternative ID) and thus makes the generation of more complex databases possible.

4. Adapting Certain Data Tuple Matching Techniques to Probabilistic Data

In this chapter, we examine how existing tuple matching[1] methods for certain data can be applied to probabilistic data. First, we give a high-level description of our approach to tuple matching on probabilistic x-tuples and then go into detail about different variants of this approach.

The fundamental idea is to generate alternative pairs from each probabilistic tuple pair (t_i,t_j) and to process them with methods known from certain data duplicate detection: a comparison vector is computed for each alternative pair first, which is then used to classify the alternative pair as a match or an unmatch. Since an alternative pair does not differ much from a certain data tuple pair, the computation of a comparison vector from an alternative pair can be done with traditional methods.
The obvious difficulty in this approach is that the decision model is trained with alternative pairs and produces labels for alternative pairs, while the goal of the entire process is to label *tuple* pairs.

[1] As described in Section 2.2, the process of duplicate detection can be divided into the five steps standardisation, search space reduction, in-depth comparison, decision model and evaluation where we subsume the in-depth comparison and the decision model under the term *tuple matching*.

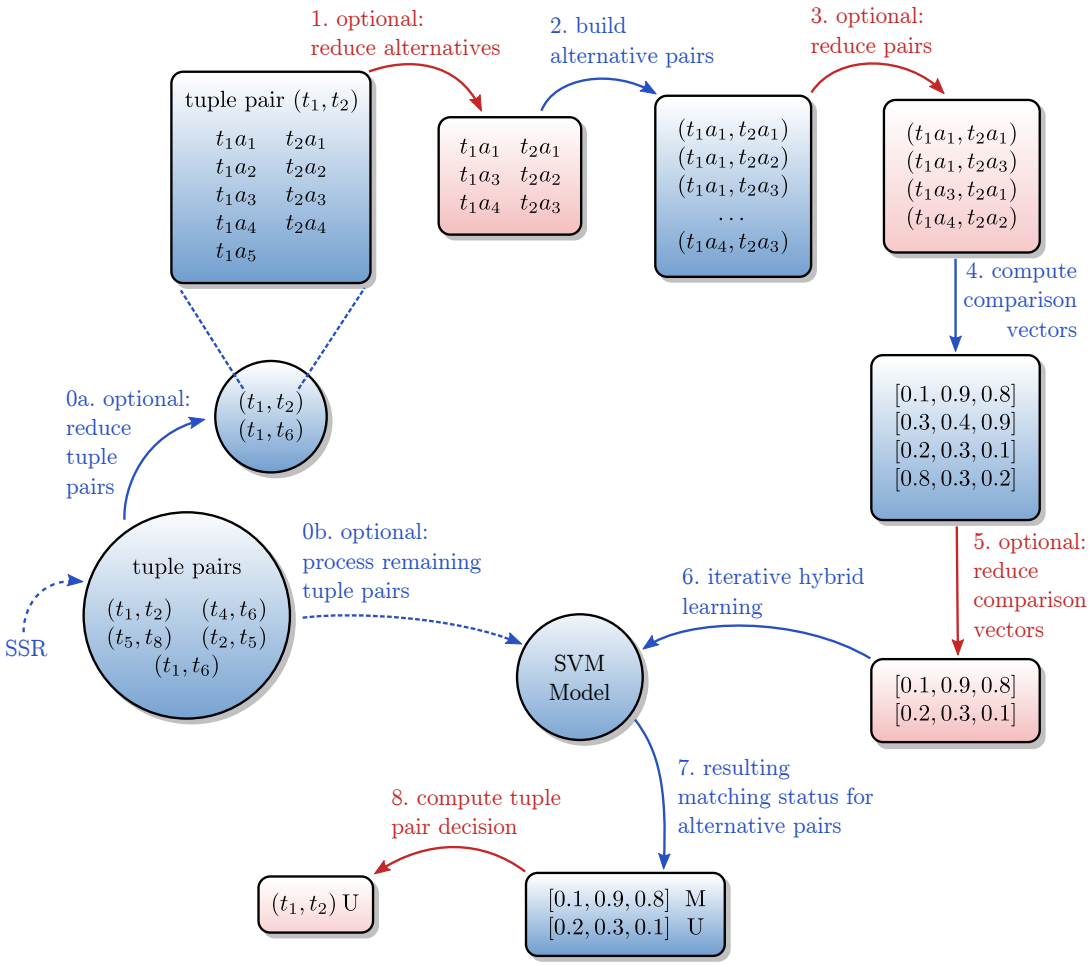


Figure 4.1.: The different steps of our probabilistic tuple matching approach. The steps highlighted in red are discussed at length in this chapter.

Figure 4.1 illustrates our approach to tuple matching on probabilistic data. The variants of our approach differ in the steps highlighted in red which are detailed in the following sections. The individual steps of our approach are:

0. A subset of all tuple pairs that remain after the search space reduction is selected to train the decision model (a). The rest of the tuple pairs is processed with the trained model (b).
1. Optional: before generating alternative pairs, the number of alternatives in the single tuples can be reduced with different strategies.
2. Alternative pairs are generated from the (remaining) alternatives of the two tuples of each tuple pair.
3. Optional: the number of the resulting alternative pairs is reduced.
4. A comparison vector is computed for each alternative pair with conventional certain data techniques.
5. Optional: the number of comparison vectors is reduced.

6. The model is trained: we use the iterative hybrid machine learning technique by Christen (see Figure 2.9).
7. The decision model delivers a matching status for every alternative pair.
8. A matching status for every tuple pair is derived from the matching status of the corresponding alternative pairs.

Given a set of possibly matching tuple pairs resulting from the search space reduction, a subset of tuple pairs to train a decision model is chosen in Step 0. In our implementation, we use the iterative hybrid machine learning decision model by Christen which augments the training set, until it contains the entire set of alternative pairs to classify. Splitting off a subset of all tuple pairs for training is the right thing to do, though, when another decision model is used or the entire tuple set results in a huge number of alternative pairs, so that training the iterative model with all available data is too expensive. Steps 1 to 5, of course, have to be performed for the tuple pairs that are not used for training in the same fashion as they are performed for the training tuple pairs.

4.1. Step 1: Reducing the Alternatives

When pre-labelled x-tuples are used for training, the tuple pair labels cannot be learned directly, but the corresponding alternative pairs have to be labelled first, so that they can be used for training. As illustrated in Figure 4.2, though, simply applying the tuple pair label to all alternative pairs may lead to unreasonable labels for some or even most of the alternative pairs.
The upper table shows a probabilistic x-tuple pair (t_1, t_2) that has been labelled as a match, the lower table shows the corresponding alternative pair comparison vectors and their respective confidence values. The by far most probable alternative pair (t_1a_1, t_2a_1) gives a comparison vector that intuitively seems a legit match, whereas the other alternative pairs don't look like matches at all. In this example, three out of four matching vectors practically seem mislabelled. It is well possible that using those alternative pairs for training leads to an absurd decision model, since the comparison vectors of many matching alternative pairs strongly resemble the matching vectors of typical unmatches.

Two duplicate tuples represent the same real-world entity and since the most probable tuple alternatives usually are good entity representations, the alternatives of two actually matching tuples that form the most probable alternative pair are often very similar. A good representation may not be very similar to a bad representation and two bad representations may not have much in common, at all. To prevent bad entity representations from being used, some x-tuple alternatives may be discarded, before the alternative pairs are built. Assuming that the confidence indicates how well an alternative represents the corresponding entity, one idea is to further process only a predefined number of alternatives of the tuple, for example the n most probable alternatives. Another idea is to ignore all alternatives with too low confidence values; if no alternative exceeds a particular threshold,

matching tuple pair (t_1,t_2)

	title	year	conf
t_1a_1	Batman & Robin	1997	0.9
t_1a_2	Batman: Mask of the Phantasm	2011	0.1
t_2a_1	Batman and Robin	1997	0.9
t_2a_2	Batman	2000	0.1

compute all comparison vectors

	$\text{sim}_{\text{title}}$	sim_{year}	conf
(t_1a_1, t_2a_1)	0.8	1.0	0.81
(t_1a_1, t_2a_2)	0.42	0.07	0.09
(t_1a_2, t_2a_1)	0.40	0.15	0.09
(t_1a_2, t_2a_2)	0.45	0.68	0.01

Figure 4.2.: A tuple pair with the corresponding alternative pair comparison vectors. The alternatives with low confidence values are bad representations of their corresponding real-world entities and, correspondingly, alternative pairs with low similarity values don't necessarily correspond well to their actual matching status.

at least one alternative per tuple has to be chosen, though, e.g. the alternative with the highest confidence value. If the confidence value of an alternative cannot be used to decide whether it is a good or a bad representation of the corresponding entity, it is hard to tell which alternatives should be discarded and which should not.

The iterative hybrid decision model used by us is trained without a pre-labelled initial training set, so no alternative pairs are labelled unreasonably, since there are no tuple pair labels to derive unreasonable alternative pair labels from. But even though discarding bad entity representations does not seem necessary, it still might improve the model.

4.1.1. Using a Single Alternative per Tuple

A special case of the approach to reduce the number of alternatives per tuple is choosing the most probable alternative, so that only one alternative pair is generated from a tuple pair and, hopefully, no alternative pairs with unreasonable labels are generated. In this case, determining the matching status of a tuple pair (Step 8) becomes trivial, since there is only one alternative pair to derive the matching status from.

Instead of selecting one of the existing alternatives, the computation of a representative[2] as illustrated in Figure 4.3 is another option. The probabilistic x-tuple in the left table comprises four alternatives with three different titles and four different years. A representative is computed by selecting the most probable value of every attribute. In Subsection 3.1.2 of our bachelor's thesis [FW10], some possibilities to compute a tuple representative

[2]Naturally, several representatives can be used as well, but we don't cover this case.

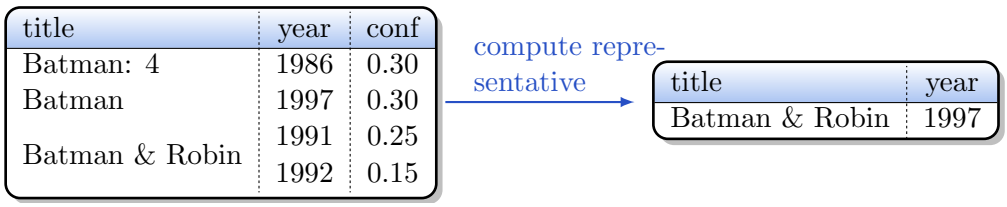


Figure 4.3.: A representative is computed for the given x-tuple: the most probable values of each attribute are chosen.

using conflict resolution strategies known from data fusion are described.
Using the *Cry with the Wolves (CWTW)* strategy means to use the most frequent value, but since confidence values are available, the choice of the most probable value as in the example may be preferable. Although the alternatives with the title "Batman & and Robin" have the lowest confidences, the confidence of their title value is $0.25 + 0.15 = 0.40$ and thus greater than the confidence of any other title value. Since there are two most probable years, the choice between those two years can be made with *Roll the Dice (RTD)*, i.e. at random. Another reasonable fallback strategy here would be *Meet in the Middle (MITM)* according to which some value in the middle is chosen, for example the average or the median. So, instead of either 1986 or 1997, $\frac{1986+1997}{2} \approx 1992$ could be picked. MITM could also be chosen as primary strategy for the year, so that the example representative would receive the average $\frac{1986+1997+1991+1992}{4} \approx 1992$ or the expected year $1986 \cdot 0.30 + 1997 \cdot 0.30 + 1991 \cdot 0.25 + 1992 \cdot 0.15 \approx 1991$.
Although not illustrated in the example, null values can also occur with a high confidence. The selection strategy *Take the Information (TTI)* dictates that any value is preferred to the null value.

4.2. Step 3: Reducing Alternative Pairs

Based on the remaining alternatives of every tuple, alternative pairs are generated from the tuple pairs in Step 2. In Step 3, alternative *pairs* that should better not be used for training are filtered out.
Using the confidence values as a decision criterion is, again, an obvious approach. For example, alternative pairs that do not exceed a certain confidence threshold can be discarded or only the n most probable alternative pairs can be kept. The main difference between reducing the number of alternatives and reducing the number of alternative pairs is that an alternative that has been discarded in Step 1 cannot be used to build an alternative pair; so discarding an alternative corresponds to discarding all alternative pairs containing this alternative. In contrast, when sorting out low confidence *pairs*, an alternative with low confidence can still be used for training if paired with a high confidence alternative.
As described in the following, the similarity between alternatives can also be used to decide whether or not they should be used.

4.2.1. Best Partner Reduction

Figure 4.4 shows two identical actually matching tuples in the upper left table and the corresponding attribute similarity values of all alternative pairs in the upper right table. Although the tuples have the exact same values, two out of four alternative pairs with an overall confidence of $0.24 + 0.24 = 0.48$ appear like clear unmatches. In the example, the mean value of the attribute similarities is used compute a similarity value for each alternative pair. Thus, the expected alternative pair similarity can be computed as $1 \cdot 0.36 + 0.235 \cdot 0.24 + 0.235 \cdot 0.24 + 1 \cdot 0.16 \approx 0.63$. Obviously, removing the alternative pair with the lowest confidence value is not the optimal strategy here; if alternative pair (t_1a_2, t_2a_2) is discarded, two out of three alternative pairs used for training are mislabelled and the expected similarity computed from the remaining alternative pairs drops to $1 \cdot 0.36 + 0.235 \cdot 0.24 + 0.235 \cdot 0.24 \approx 0.47$.

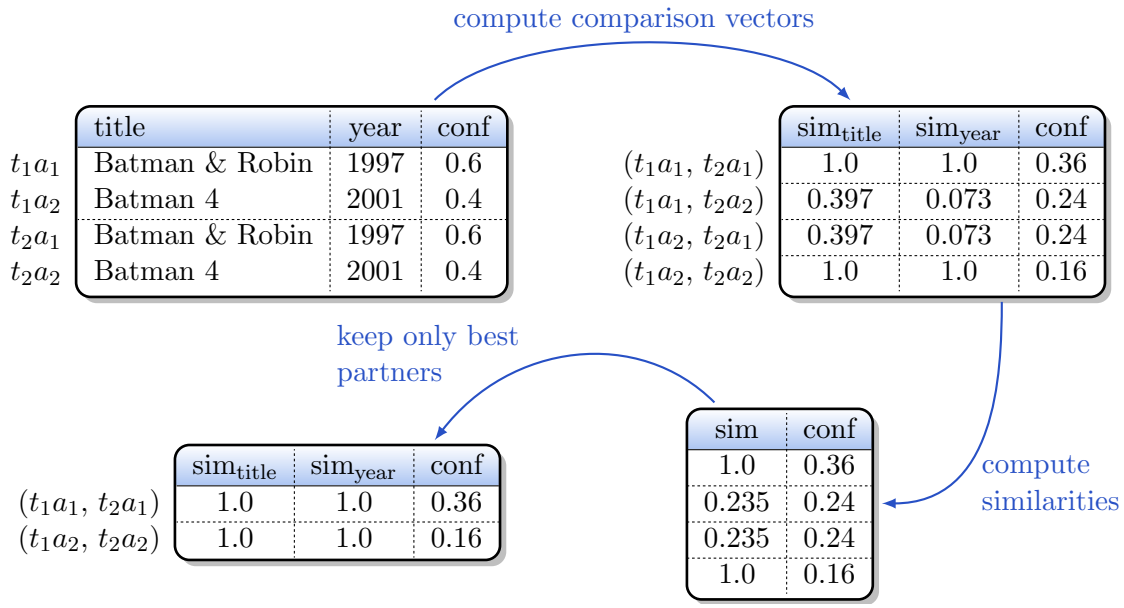


Figure 4.4.: An example of the *Best Partner Reduction* where every tuple alternative is paired with its best match in the other tuple.

One approach that avoids this kind of problem is the *Best Partner Reduction*. As indicated in the example, the alternative pairs are chosen in such a way that every alternative a_i in tuple t_1 is paired with the most similar alternative a_j from tuple t_2 and vice versa. To determine the similarity of two alternatives, the weighted sum of all attribute similarities can be used as described in Subsection 3.2.3.

In simple words, when computing the tuple pair similarity, every alternative is only compared to its best match, so that the alternative pairs are well-selected in case of matching tuple pairs. Based on the assumption that unmatching tuple pairs usually do not have highly similar or identical alternatives, this approach does not lead to an unreasonable selection for unmatching alternative pairs, either.

4.3. Step 5: Reducing Comparison Vectors

After the comparison vectors of the alternative pairs have been computed in Step 4, they can also be reduced in Step 5. Since every comparison vector corresponds to exactly one alternative pair, they can be reduced with the same confidence-based strategies (see Step 3 in Section 4.2) yielding exactly the same results.

In our opinion, a reasonable way to reduce the number of comparison vectors might be to compute a *representative* comparison vector for all other comparison vectors of a tuple pair, for example one holding the expected or the best similarity values for the respective attributes. The computation of more than one representative is also possible; for instance, one representative holding the expected and one representative holding the best similarity value could be created and could *both* be used further.

4.4. Step 8: Computing the Tuple Pair Decision

Steps 1 to 5 are followed by either Step 6, i.e. another iteration to train the decision model, or by Step 7 where the fully trained decision model assigns a matching status to every alternative pair. In Step 8, finally, a matching decision for each tuple pair is computed from the corresponding alternative pairs' matching labels.

	$\text{sim}_{\text{title}}$	sim_{year}	conf	$P(m)$	$P(u)$	label
(t_1a_1, t_2a_1)	1.0	1.0	0.36	1.0	0.0	M
(t_1a_1, t_2a_2)	0.397	0.073	0.24	0.2	0.8	U
(t_1a_2, t_2a_1)	0.397	0.073	0.24	0.2	0.8	U
(t_1a_2, t_2a_2)	1.0	1.0	0.16	1.0	0.0	M

Table 4.1.: The comparison vectors from Figure 4.4 and the learned matching labels with the corresponding decisiveness values.

When all alternative pairs have the same matching status or when there is only one alternative pair (or comparison vector) corresponding to a tuple pair, the tuple pair's matching status can simply be adopted. Otherwise, i.e. when the alternative pairs have ambiguous matching labels, a more complex decision-making strategy has to be used, for example the conflict resolution strategies described in Subsection 4.1.1.

Apart from confidence value and matching label, the *decisiveness* that some decision models output in addition to the actual matching label can be used. Table 4.1 shows the comparison vectors of tuple pair (t_1,t_2) from Figure 4.4, their matching labels and the corresponding matching and unmatching probabilities as they are output by the SVM decision model we use: the decision model delivers $P(M)$ and $P(U) = 1 - P(M)$ which can be interpreted as the likeliness of the alternative pair being a match or an unmatch, respectively. For an alternative pair (a_k,a_l) with $a_k \in t_i$ and $a_l \in t_j$, $P(M)$ is a real value

between 0 (certain unmatch) and 1 (certain match) and thus can also be interpreted as the similarity $sim(a_k,a_l)$ between a_k and a_l.

In this section, we describe techniques to compute a matching decision for a given tuple pair (t_i, t_j) in two steps: first, a similarity value for the given tuple pair is computed and then the matching status is determined with a threshold. An experimental evaluation of those techniques can be found in Section 5.3.

4.4.1. Expected Similarity

When $P(M)$ and the confidence value $conf_{k,l} = conf(a_k) \cdot conf(a_l)$ are given for all alternative pairs (a_k,a_l), the function

$$sim(t_i,t_j) = \frac{\sum\limits_{(a_k,a_l)} sim(a_k, a_l) \cdot conf_{k,l}}{\sum\limits_{(a_k,a_l)} conf_{k,l}}$$

offers an intuitive approach to computing the *expected tuple pair similarity*: all alternative pair similarity values are summed up weighted by their confidence values and then normalised to the confidence value. For the sake of simplicity, we only cover tuple pairs with an overall confidence of 1 in our examples, so that the denominator in the above formula is 1. When the sum of all alternative pair confidence values is smaller than 1, though, the similarity has to be normalised or otherwise it highly depends on the confidence: for example, the unnormalised expected similarity of two identical tuples with a confidence of 0.5 and only one alternative each cannot exceed $1 \cdot 0.5 \cdot 0.5 = 0.25$, whereas it equals 1, if both tuples have a confidence of 1.

The expected tuple pair similarity can be used to assign a matching status to tuple pair (t_i, t_j); for a binary classification (match and unmatch), only one threshold t_M is needed. For example, (t_i, t_j) can be declared a match, if $sim(t_i,t_j) > t_M$ and an unmatch else. Another possibility is to declare an unmatch, only if $sim(t_i,t_j) < t_M$, and a possible match, if $sim(t_i,t_j) = t_M$. A ternary classification with a larger margin for possible matches can be realised with two thresholds t_U and t_M where (t_i, t_j) is classified as match, only if the similarity value exceeds t_M, as unmatch, only if the similarity value is below t_U, and as possible match, if the similarity value is between t_U and t_M.

Computing the expected similarity of the tuple pair depicted in Figure 4.4 yields

$$sim(t_1,t_2) = 1 \cdot 0.36 + 0.2 \cdot 0.24 + 0.2 \cdot 0.24 + 1 \cdot 0.16 = 0.616 \quad .$$

4.4.2. Expected Similarity with Uniformly Distributed Confidences

When no confidence values are given, *uniformly distributed confidence* values can be assumed. The matching label confidences in our example actually are almost uniformly distributed ($0.36 + 0.16 = 0.52$ for M and $0.24 + 0.24 = 0.48$ for U), so that the result is almost the same as well:

$$sim(t_1,t_2) = \frac{1 + 0.2 + 0.2 + 1}{4} = 0.6 \quad .$$

The problem here is basically the same as the problem discussed in Section 4.2.1: even highly similar tuple pairs have dissimilar alternative pairs and hence their expected similarity is sometimes rather low. It can even get worse, when there are more tuple alternatives and the confidence of the similar alternative pairs decreases.

4.4.3. Best Partner Similarity

To diminish this problem, the approach proposed in Section 4.2.1 can be applied, so that the *expected similarity among the best partner matches* is computed. For every alternative, the single alternative pair is chosen that has the greatest $P(M)$ value. The obvious best partner pairs in the example are (t_1a_1, t_2a_1) and (t_1a_2, t_2a_2), so that the Best Partner Similarity is

$$sim(t_1,t_2) = \frac{1 \cdot 0.36 + 1 \cdot 0.16}{0.36 + 0.16} = 1 \quad .$$

4.4.4. Maximal α-Prob. Similarity

Another approach to computing a tuple pair similarity value that is high for matches and low for unmatches is to take only the most similar alternative pairs. For example, the best-matching 60% of the alternative pairs could be used ($\alpha = 0.6$), so that 40% ($1 - \alpha$) of the alternative pairs are disregarded.
First, the alternative pair with the greatest $P(M)$ is chosen. If the confidence of this alternative is not equal to or greater than α, the alternative pair with the greatest $P(M)$ among the remaining alternative pairs is chosen. This process is repeated, until the confidence value sum of all chosen alternative pairs is at least α. Then, the *expected similarity* of this subset of all alternative pairs and the tuple pair matching status are computed as described in 4.4.1. The result is the *maximal tuple pair similarity that can be achieved with an α share* of all alternative pairs.
It is important to note that the expected similarity has to be normalised, i.e. divided by the sum of all the chosen alternative pairs' confidence values, so that a reasonable tuple pair matching decision can be made by thresholding.

With $\alpha = 0.6$ and a threshold of 0.5, the two alternative pairs with perfect similarity and one of the other alternative pairs are chosen, so that the tuple pair similarity is

$$sim(t_1,t_2) = \frac{1 \cdot 0.36 + 1 \cdot 0.16 + 0.2 \cdot 0.24}{0.36 + 0.16 + 0.24} \approx 0.75 \quad .$$

With $\alpha = 0.5$ and a threshold of 0.5 as before, only the two alternative pairs with perfect similarity are chosen and the similarity results in

$$sim(t_1,t_2) = \frac{1 \cdot 0.36 + 1 \cdot 0.16}{0.36 + 0.16} = 1 \quad .$$

Obviously, a lower α value tends to result in a higher similarity, since only a few alternative pairs with maximal similarity are chosen. A lower α value might lead to an increased number of true *and* false positives, which in turn might increase the recall and decrease the precision. In some cases, even a very low α value may be reasonable and deliver good results, for example if tuples with very many alternatives are compared.

4.4.5. Cry with the Wolves

If $P(M)$ is not available, the tuple pair decision has to be made based on matching labels and confidence values alone. A very simple way to cope with ambiguous matching labels is to assign the *most probable matching label* and declare a possible match in case of equal confidence values.
The confidence values of all matching alternative pairs can be interpreted as the matching probability and thus as the tuple pair similarity. In the example, the tuple pair similarity can be computed as

$$sim(t_1,t_2) = 0.36 + 0.16 = 0.52 \quad .$$

As mentioned before, the confidence values can be assumed to be uniformly distributed, if no confidence values are given. Since half of the alternative pairs are labelled as matches, the tuple pair similarity in our example is then exactly

$$sim(t_1,t_2) = \frac{2}{4} = 0.5 \quad ,$$

so that it is completely unclear whether the tuple pair should be declared a match, an unmatch or a possible match.

5. Experimental Evaluation

In this chapter, we compare the tuple matching variants discussed in Chapter 4 with respect to recall, precision and the traditional F-measure on the basis of experimental results. In the first section, we describe our implementations of the different tuple matching variants. The subsequent section covers the experimental setup in general and provides detailed information on the probabilistic data sets the experiments are executed on, the applied search space reduction technique and the configuration of our baseline experiment. The third section covers the experimental evaluation of different strategies to reduce alternatives, alternative pairs and comparison vectors in combination with the different tuple pair decision variants.

5.1. Implementation of the Tuple Matching Approach

We measure the similarity between two attribute values using Soft TFIDF combined with the Jaro-Winkler metric, because this particular matching scheme performs best in an experimental comparison of several string distance metrics for the task of entity name matching conducted by Cohen et al. [CRF03]. In our implementation, we use *SecondString*[1], a Java-based library of string matching techniques that is also being developed by Cohen and others.
Steps 1 to 5 of our tuple matching approach, namely all reduction strategies and the computation of the tuple comparison vectors, are implemented in Java, so that SecondString can easily be used.

The subsequent steps, i.e. the iterative hybrid decision model described in Subsection 2.2.6 (see Figure 2.9) and the computation of the tuple pair decision (Step 8), are implemented with the *Konstanz Information Miner (KNIME)*[2]. KNIME is a freely available Eclipse-based software for interactive data visualisation, processing and analysis. KNIME provides a great number of modules to execute machine learning tasks and enables the user to implement workflows completely via the graphical user interface.
The high degree of usability and interactivity comes at the price of performance: KNIME implementations are well comprehensible and allow the user to inspect the workflows at run time very easily, but are not necessarily efficient. For this reason, we do not compare running times in this work.

[1] SecondString: `http://secondstring.sourceforge.net/`.
[2] Konstanz Information Miner: `http://www.knime.org/`.

5.2. Experimental Setup

Prior to the actual tuple matching experiments, the search space has to be reduced, since in a real application, the tuple matching is performed on the reduced search space as well. Apart from that, the entire set of tuple pairs is prohibitively large, so that applying our tuple matching approach without reducing the search space is impossible anyway.
Our tuple matching approach can be varied in several steps. Since comparing all possible combinations goes beyond the scale of this thesis, we define a *baseline experiment* and evaluate the outcome of the tuple matching process after varying individual steps. We conduct all experiments on two databases, one being rather clean and one being rather dirty, using a computer with an Intel i7-2700K at 3.50Ghz with 16GB RAM under Windows 7 x64.

In the first subsection, we present the probabilistic databases generated with ProbGee on which the experiments are conducted. The following subsection provides detailed information on the applied search space reduction technique. Finally, the configuration of our baseline experiment is described in detail.

5.2.1. Generating the Databases

For our experiments, we consider several probabilistic movie databases that might, for example, be the result of the integration of several certain movie databases. In each of the databases, there is only the movies relation where title, director, producer, studio and release year of the movies are stored.

As mentioned in Section 3.1, ProbGee uses IMDb movie data extracted from a JMDB database to generate probabilistic x-tuples. Unfortunately, the JMDB movie table does not only contain movies, but also video games, series and series episodes and the individual title attribute values do not only contain the respective movie, game or episode titles, but also meta data.
For many series, there are dozens or hundreds of episodes listed in the movies table: for example, the title value “"Batman Beyond" (1999) {Babel (#2.12)}” represents the 12^{th} episode of the second season of the series “Batman Beyond” entitled “Babel” from the year 1999. Likewise, some movie titles like “Batman Beyond: Return of the Joker (2000) (V)” or video game titles like “Batman: Arkham Asylum (2009) (VG)” contain the release year and the media type.

The movie table is cleaned, before the probabilistic database is generated with ProbGee. Since only series episode tuples contain sharp signs (“#”), they can be removed via SQL.[3] Video games, marked by a “(VG)” in the title, are also removed, because they are not part of the movie domain in our opinion. The title values of the remaining tuples are

[3]Note that, although the series *episodes* are removed, there is still one tuple for every series: for instance, the tuple with title “"Batman Beyond" (1999)” remains in the database and is cleaned from meta data later.

cropped in such a way that they only consist of the actual title and contain no additional information.

	database information			
general	tuples	54,386		
	maybe-tuples	5,439		
	alternatives	100,000		
	entities	cluster size	frequency	
		1	31,078	
		2	7,769	
		3	2,590	
	alternatives	tuple size	frequency	
		1	28,070	
		2	14,035	
		3	7,018	
		4	3,509	
		5	1,754	
quality	comparison	attribute	metric	score
		title	JaroWinkler	50
		year	JaroWinkler	30
		director	JaroWinkler	30
		producer	JaroWinkler	30
		studio	JaroWinkler	30

Table 5.1.: Some general parameters used for the generation of all databases.

Table 5.1 shows some basic parameters that are the same for all generated databases. Every database holds 100,000 alternatives distributed among 54,386 tuples including 10% maybe-tuples. A database of this size can still be handled by all components of our tuple matching workflow in an acceptable period of time with an acceptable amount of memory; increasing the number of alternatives significantly raises some problems, for example training the iterative hybrid decision model with 200,000 alternatives or more takes inconveniently much time.

There are $31{,}078 + 7{,}769 + 2{,}590 = 41{,}437$ entities represented in every database: 31,078 entities have no duplicates, 7,769 entities are presented twice and 2,590 entities are represented three times. With one actual match per duplicate cluster of size two and three actual matches in every duplicate cluster of size three, there are $7{,}769 + 3 \cdot 2{,}590 = 15{,}539$ actual matches. Every tuple consists of at least one and at most five alternatives where larger tuples are less frequent.

As described in Subsection 3.2.3, the similarity between two tuples is computed as the weighted sum of the attribute similarities. As is clear from the table, all attribute similarities are computed using the Jaro-Winkler metric. When computing the similarity between two tuples, the title has the most influence on the tuple similarity.

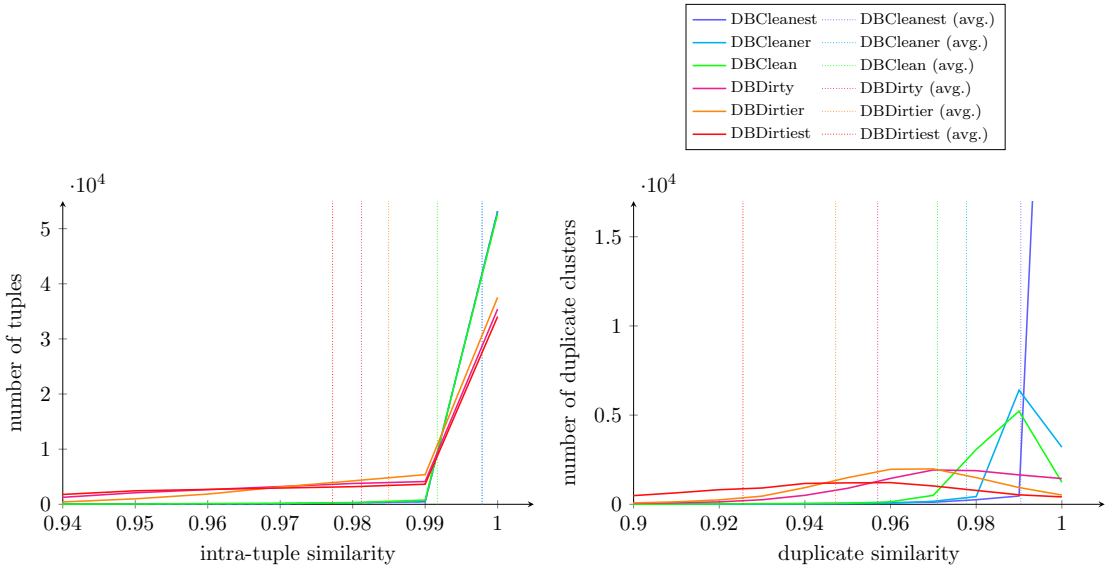


Figure 5.1.: The intra-tuple and duplicate similarity of all candidate databases in a direct comparison.

In order to find out in what way the degree of dirtiness influences the performance of the tuple matching variants, we execute our experiments on two different databases. All our candidate databases are generated without any overlapping (see Subsection 3.2.3). The intra-tuple and duplicate similarity of six candidate databases are compared in Figure 5.1. The database names reflect their respective data quality: DBCleanest holds x-tuples with only minor typos and very similar duplicates, while there is hardly any alternative in DBDirtiest without several complex errors.
The databases can be partitioned into *clean* databases (DBCleanest, DBCleaner, DBClean) and *dirty* databases (DBDirty, DBDirtier, DBDirtiest). In the clean databases, the intra-tuple similarity of almost all tuples is above 0.99, whereas there are also tuples with a much lower intra-tuple similarity in the dirty databases. Practically all duplicates in the clean databases have a similarity value of 0.97 or above, whereas the duplicate similarity in the dirty databases varies from about 0.90 to 1.
It is noteworthy that a great variety of data quality is covered by the similarity values above 0.90, because we use a similarity metric that is very robust, especially against word permutations or gaps. If another metric is used, the range of the similarity values might be completely different.

To cover a great variety of databases, we choose one clean and one dirty database for further experiments. DBCleanest and DBDirtiest are rather uninteresting, because they are too extreme in their respective error frequencies. In order to make the tuple matching not too easy, we choose the dirtiest database from each group, namely DBClean and DBDirtier.
Detailed information on the distribution of the intra-tuple similarity, the duplicate similarity, the tuple cleanness and the confidence values among the maybe-tuples of every

candidate database are given in Appendix A. Furthermore, there are two exemplary error provenance trees illustrating typical duplicate clusters in our final test databases: the tree extracted from DBClean is on page 86 and the tree extracted from DBDirtier is on page 89.

5.2.2. Search Space Reduction

Since the input to the in-depth comparison is the already reduced search space, some kind of search space reduction has to be performed on the experimental data, before the actual tuple matching experiments can be executed. Problematic, though, is that the performance of the SSR techniques has a great influence on the performance of the tuple matching methods and thus distorts the results: either the *combined performance* of SSR and tuple matching are evaluated or the performance of the tuple matching is evaluated relatively to the preceding SSR performance. If the SSR delivers a poor recall, the relative performance of the tuple matching may seem great, although it actually is not. Besides, tuple matching results on different data sets might become practically incomparable, when recall and precision of the applied search space reduction techniques differ greatly.

To generate meaningful, comparable results, an SSR technique that always yields a perfect recall and a constant precision seems optimal.
Since we run our experiments on labelled data, such an optimal search space reduction can be simulated by selecting a subset of all tuple pairs which contains all actual matches (perfect recall) and a fixed number of actual unmatches (constant precision). The great advantage of this approach is that it offers perfect control over recall and precision. However, a selection strategy for the unmatches has to be used that leads to realistic unmatches: selecting unmatches randomly might lead to an unrealistic result, since a real SSR technique chooses duplicate candidates according to some criterion, e.g. only tuple pairs with equal key values.
Another option to generate a reduced search space is to perform an actual SSR technique. While the unmatches here certainly are realistic, neither the recall nor the precision are constant or perfect.

For our experiments, we use a combination of simulated and real search space reduction: to guarantee realistic unmatches, we use a variant of the Sorted Neighbourhood Method that is adapted to the use in probabilistic data (see Section 2.3) and augment the generated set of tuple pairs by the missing actual matches, so that both a perfect recall *and* a realistic, although not constant, precision are achieved. The used window size is $w = 3$ and the sorting key consists of the Soundex codes of the attributes title, producer, studio, director concatenated with the year. Preliminary experiments show that this key leads to a good recall.

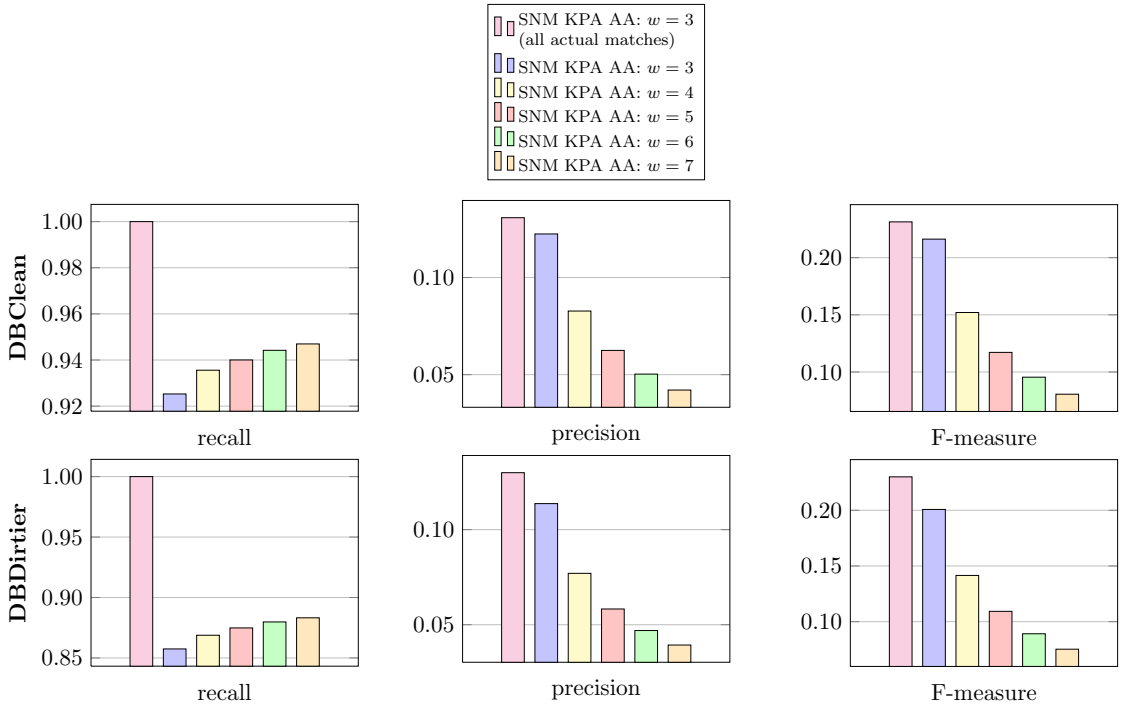


Figure 5.2.: A comparison of our SNM approach on DBClean and DBDirtier with varying window sizes.

Figure 5.2 shows recall, precision and F-measure yielded by our SNM approach on the database DBClean with window size $w = 3$ to $w = 7$. The leftmost bar in each of the diagrams represents the performance of our artificially improved approach with $w = 3$ that is actually applied.

Using a window size of $w = 3$ already leads to a recall of over 0.9 and a precision of about 0.12. As one would expect, the recall can be improved by increasing the window size, but this also strongly reduces the precision. The best F-measure is achieved with window size $w = 3$. Improving the result by raising the recall to 1 slightly improves the precision to 0.13 and the F-measure to 0.23.

5.2.3. The Baseline Experiment

The tuple matching approach described in Chapter 4 can be varied in Step 1 (reducing alternatives), Step 3 (reducing alternative pairs), Step 5 (reducing comparison vectors) and Step 8 (tuple pair decision). All reduction strategies are optional, while a tuple pair decision always has to be made.
In the following, we present several candidate baseline configurations and describe the single configuration that is used for all further experiments.

configuration	Config1	Config2	Config3	Config4	Config5
database	DBClean				
reducing alternatives	-				
reducing alternative pairs	-				
reducing comparison vectors	-				
similarity metric	SoftTFIDF – JaroWinkler				
trainings vector selection	nearest-based				
initial match vectors	100	500	1000	100	100
initial unmatch vectors	780	3898	7795	780	780
SVM kernel	linear			radial basis	polynomial

Table 5.2.: An overview over the different candidates for our baseline configuration.

Table 5.2 shows some possible configurations for the baseline experiment. All configurations are tested on the database DBClean without any of the optional reduction strategies, but with all tuple pair decision variants. The performance of the individual tuple pair decision variants are illustrated in Figure 5.3. For all attribute value comparisons, Soft TFIDF combined with the Jaro-Winkler metric is used.
Configuration 1 to 3 use a linear support vector machine kernel and only differ in the number of initial vectors that are used for training: they use 100, 500 and 1,000 initial match vectors, respectively. The number of unmatch vectors is estimated on the basis of the given number of match vectors and the number of alternatives and comparison vectors as described in [Chr08a]. The remaining two configurations use 100 initial matcher vectors, but non-linear kernels: Config4 uses a radial basis and Config5 a polynomial kernel.

All tuple pair decision variants require a threshold to be defined according to which the tuple pair decision is made. In addition to this threshold, an α value has to be chosen for the Maximal α-Prob. Similarity. We use a value of 0.5 for both thresholds, because it is our intuitive first choice and does not seem an unreasonable value.

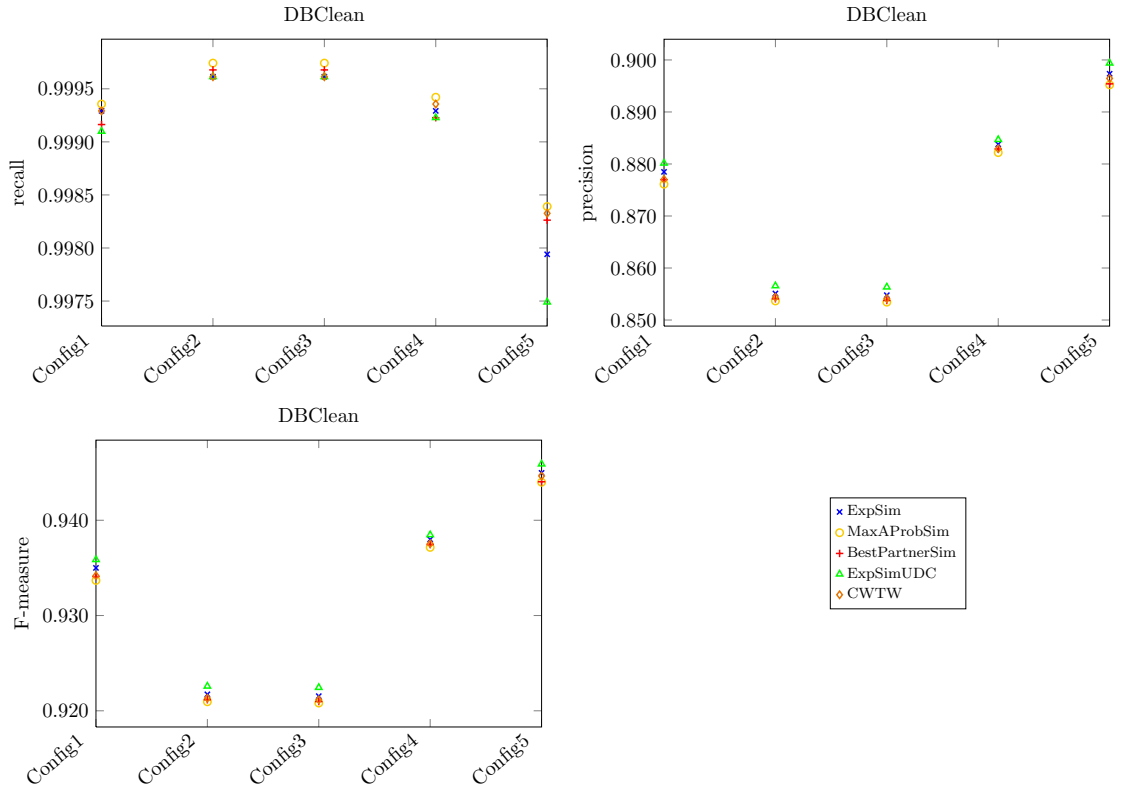


Figure 5.3.: The performance of all tuple pair decision variants in all candidate baseline configurations with respect to recall, precision and F-measure in comparison.

Figure 5.3 shows the performance of the tuple pair decision variants Expected Similarity (ExpSim), Maximal α-Prob. Similarity (MaxAProbSim), Best Partner Similarity (BestPartnerSim), Expected Similarity with Uniformly Distributed Confidences (ExpSimUDC) and Cry with the Wolves (CWTW) in all candidate baseline configurations in terms of recall, precision and F-measure.

None of the tuple pair decision variants performs exceptionally well or badly under the individual configurations, but MaxAProbSim always delivers a slightly better recall than the other variants at a slightly worse precision, while it is the other way around with ExpSimUDC.

With Config1, an already almost perfect recall of over 0.999 and precision around 0.88 are achieved. Increasing the number of match vectors from 100 (Config1) to 500 (Config2) causes a hardly noticeable increase in recall, but a drop in precision from 0.88 to 0.855. Increasing the number of match vectors further to 1,000 (Config3) brings no change. Using one of the non-linear kernels with 100 matching vectors decreases the recall slightly, but also increases the precision. Since the F-measure is best with the polynomial kernel, we declare Config5 our baseline experiment configuration.

An additional experiment reveals that decreasing the number of match vectors to 20 (Config0 in Appendix B.1) delivers an extremely bad result with a recall of 0.0005 at best (8

out of 15,539 matches) and a precision of 0.5 or less. More information on the performance of the tuple pair decision variants in the different experiment configurations on DBClean and in the baseline configuration on DBDirtier are given in Appendix B.

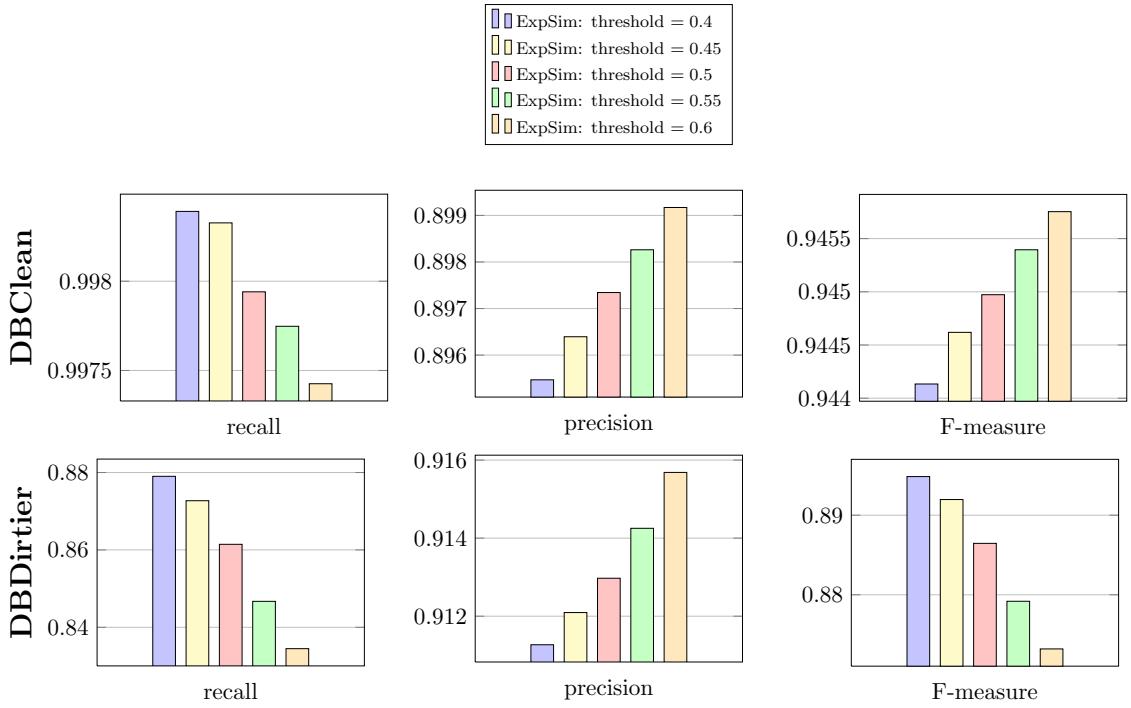


Figure 5.4.: Different thresholds for the Expected Similarity tuple pair decision.

The diagrams in Figure 5.4 illustrate the performance of the Expected Similarity tuple pair decision variant in the baseline experiment with varying threshold values between 0.4 and 0.6 on DBClean and DBDirtier. The exact measurements can be found in Appendix B.3.

As one would expect, using the lowest similarity threshold leads to the best recall and the worst precision. Increasing the similarity threshold decreases the recall and increases the precision in both databases, since more tuple pairs are declared matches. However, the different thresholds make hardly any difference to the absolute recall in the cleaner of the two databases, because it is almost perfect even in the worst case, so that the F-measure is (slightly) better with a greater threshold, just like the precision: the precision here is between 0.895 ($t = 0.4$) and 0.9 ($t = 0.6$). In contrast to that, the deterioration of the recall more than outweighs the decline of the precision in the dirty database, so that the F-measure is in favour of the smaller threshold here: the recall in DBDirtier varies from about 0.835 ($t = 0.6$) to almost 0.879 ($t = 0.4$), while the precision is in the range of 0.911 to almost 0.915.

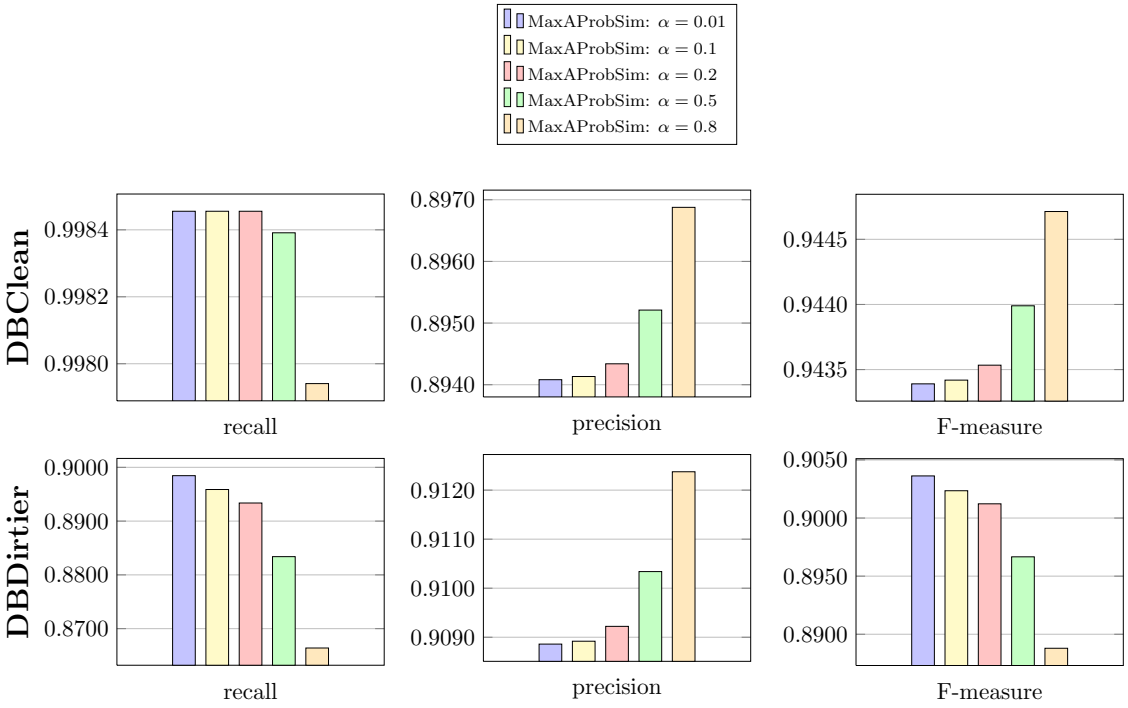


Figure 5.5.: Different α values for the Maximal α-Prob. Similarity tuple pair decision.

In a similar fashion as above, Figure 5.5 shows a comparison of recall, precision and F-measure of tuple matching with the Maximal α-Prob. Similarity strategy. The baseline experiment is executed on DBClean and DBDirtier with α values between 0.01 and 0.8. In both databases, smaller α values lead to a higher recall and also to a worse precision. But as above, the recall in DBClean is almost perfect in any case, so that the F-measure corresponds to the precision and slightly grows with the value of α. Ranging from above 0.894 to below 0.897, the precision can practically be called constant. While the precision in the dirtier database is affected as little by the value of α as the precision in the cleaner database, the recall ranges from roughly 0.866 to 0.9.
For the exact recall, precision and F-measure values, see Appendix B.4.

A similarity threshold and an α value of 0.5 seem usable, although lower values result in better recall in both databases and a better F-measure in the dirty database. The performance on the cleaner database is hardly affected by varying the thresholds: the recall is practically perfect for all covered values and the variations in precision and F-measure are minimal.

To demonstrate that our generation process is stable, i.e. that the same parameters yield highly similar results, and to show how the overlapping parameters influences the data quality, we generate a database DBDirtierOverlapping that is identical to DBDirtier in all parameters except the overlapping which has an expected value of 0.2 with a standard deviation of 0.1. More details on DBDirtierOverlapping can also be found in Appendix A.

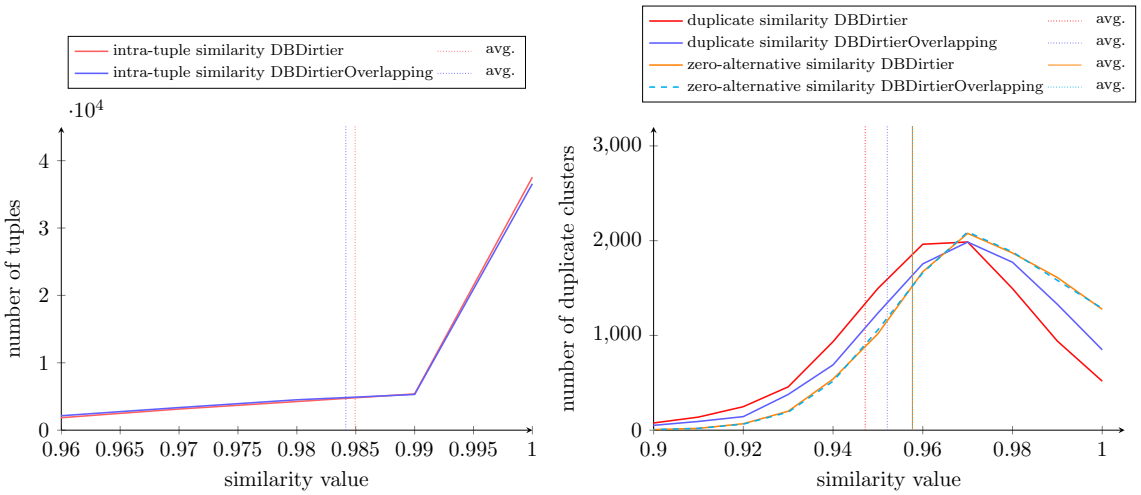


Figure 5.6.: Intra-tuple similarity, duplicate similarity and zero-alternative similarity on DBDirtier and DBDirtierOverlapping.

Figure 5.6 illustrates the effect of overlapping on intra-tuple, zero-alternative and duplicate similarity by the example of the two databases DBDirtier and DBDirtierOverlapping.
As is clear from the left diagram, overlapping barely affects the distribution of the intra-tuple similarity and slightly lowers the average value. The right diagram shows that the zero-alternative similarity is not affected by overlapping at all, whereas the duplicate similarity is increased significantly: the entire duplicate similarity distribution and the average value are shifted to the right.

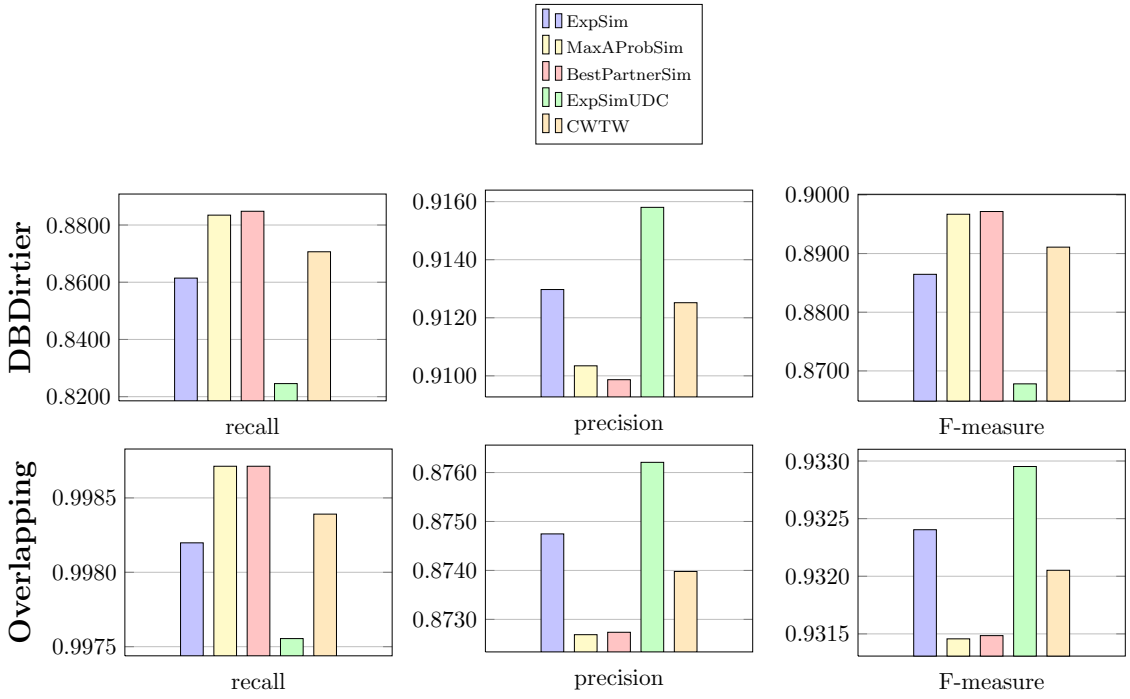


Figure 5.7.: Recall, precision and F-measure of the baseline experiment on DBDirtier and DBDirtierOverlapping.

Figure 5.7 provides a performance comparison of the baseline experiment on the databases DBDirtier and DBDirtierOverlapping.
The best tuple pair decision variant on DBClean (ExpSimUDC) delivers the by far worst recall of 0.82, while all other variants perform above 0.86; BestPartnerSim and CWTW even achieve 0.88 and thus yield the best results. Precision values only vary from about 0.91 to 0.916, so that the F-measure follows the recall.
The performance on DBDirtierOverlapping highly resembles the performance on DBClean: the recall is almost perfect and the precision is almost constant, ranging from 0.873 to 0.876.

When an alternative is generated via overlapping, it is copied from another tuple in the duplicate cluster. As expected, the zero-alternative similarity is completely unaffected by overlapping, as it is only computed from zero-alternatives. However, since the copied alternative can have arbitrary errors, it obviously tends to decrease the intra-tuple similarity and, furthermore, increases the average similarity between the two overlapping tuples. Most importantly, though, the actual recall is raised to almost 1 in our experiment. And while this makes sense, because the overlapping increases the number of identical alternatives between duplicate tuples, it also shows that our similarity measures do not account for overlapping.

5.3. Reduction Strategy Experiments

In this section, we evaluate different variants of the reduction strategies discussed in Chapter 4. For this reason, we examine whether the tuple matching results of our baseline experiment configuration can be improved by adding the individual strategies. All experiments are executed on the two databases DBClean and DBDirtier. We do not perform an additional post-processing step to resolve conflicting tuple pair decisions after the tuple matching, although it might improve the evaluation results, because it might also distort the comparison results.
In the first three subsections, we evaluate strategies to reduce alternatives, alternative pairs and comparison vectors. In the final subsection, we give an overview over the results.

5.3.1. Step 1: Reducing the Alternatives

In this subsection, we investigate how the reduction of tuple alternatives can improve the performance of our baseline configuration.

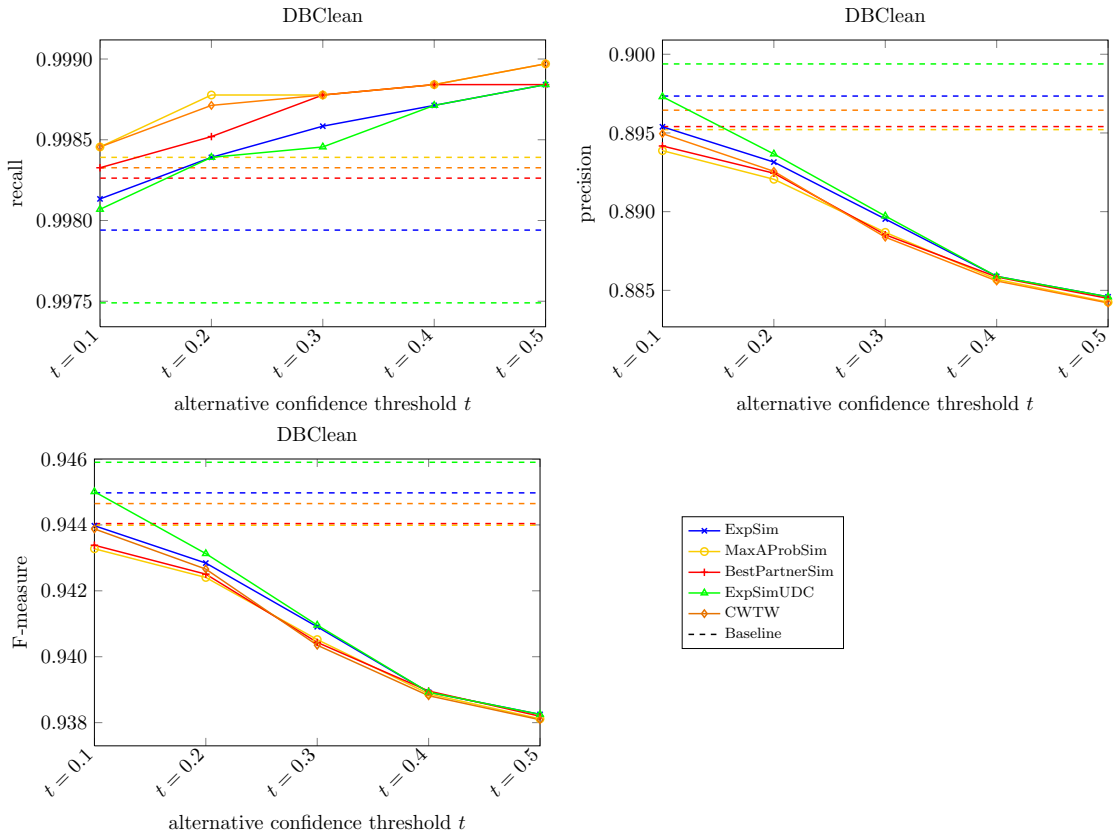


Figure 5.8.: Reducing alternatives with different threshold values on DBClean.

The three diagrams in Figure 5.8 show the influence of different confidence thresholds used to reduce the number of alternatives in comparison to the respective baseline performances

(dashed lines) in DBClean. More detailed information is available in Appendix C.1.
In comparison to the baseline experiments, applying a confidence threshold improves the recall, if only marginally, since the recall is already almost 1. All tuple pair decision variants achieve a baseline precision of about 0.9 which is decreased by roughly 0.05, when a threshold $t = 0.1$ is applied, and which further decreases when this threshold is raised. With a threshold of $t = 0.5$, all tuple pair decision variants have practically the same precision of 0.885. Similar to the F-measure in the baseline experiment, the F-measure here evolves exactly like the precision: the best F-measure is between 0.943 and 0.945 (baseline) and the worst F measure is about 0.938 ($t = 0.5$) for all tuple pair decision variants.

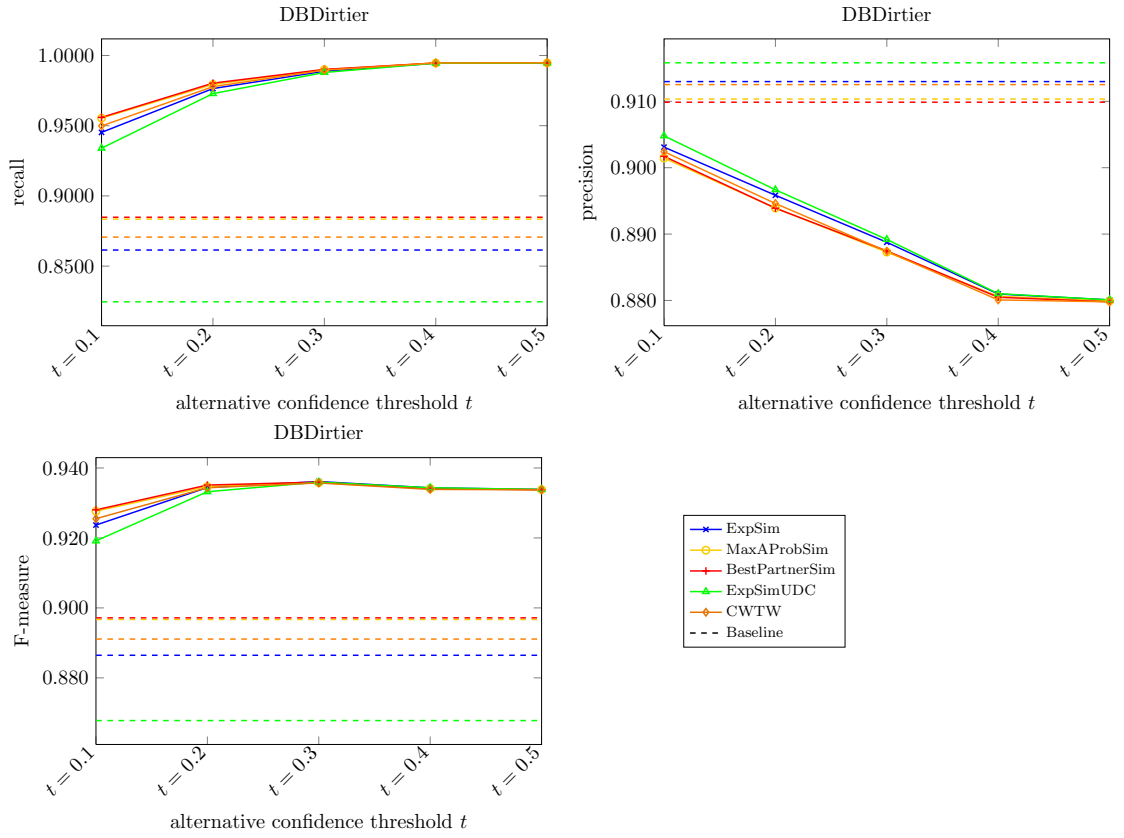


Figure 5.9.: Reducing alternatives with different threshold values on DBDirtier.

Figure 5.9 illustrates the same experiment on DBDirtier which is also detailed in Appendix C.1.
In contrast to DBClean, the DBDirtier recall baselines are far from perfect and range from just above 0.86 (ExpSimUDC) to just below 0.9 (BestPartnerSim). With a threshold of $t = 0.1$, the recall values are boosted to values between 0.934 (ExpSimUDC) and 0.956 (BestPartnerSim). A threshold of $t = 0.4$ improves the recall to over 0.99 with any of the tuple pair decision variants. The baseline precision values of over 0.91 are diminished by something between 0.01 ($t = 0.1$) and roughly 0.03 ($t = 0.4$). Further increasing the

threshold to $t = 0.5$ does not change recall or precision much. Due to the great improvement of the recall, the F-measure is also clearly better when a threshold is used: while all baselines are clearly below 0.9, the F-measure values are raised to roughly 0.92 with a threshold of $t = 0.1$ and climax at more than 0.93 with a threshold of $t = 0.4$.

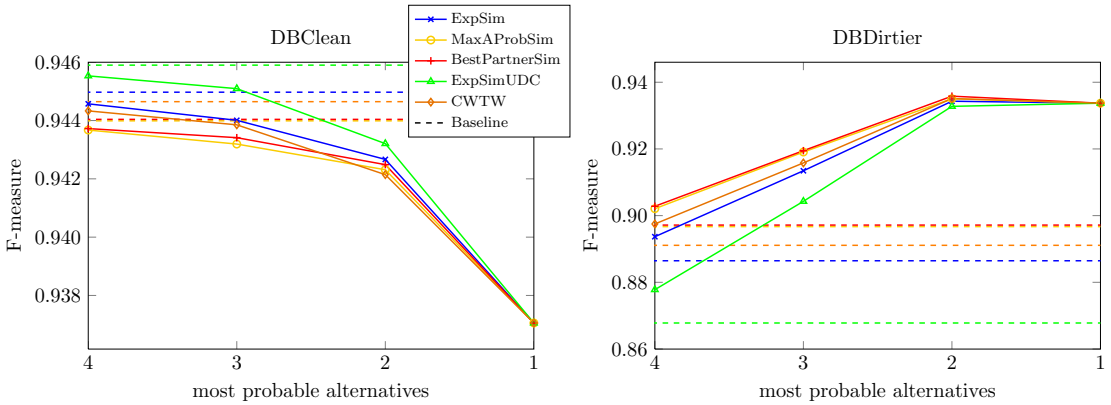


Figure 5.10.: Using only the n most probable alternatives on DBClean and DBDirtier.

The results of our experiments on reducing alternatives by only the n most probable ones on DBClean and DBDirtier are similar to the result of the the experiments on the threshold-based alternative reduction. For this reason, we only compare the F-measures of the different tuple pair decision variants in Figure 5.10. The exact measurements of the individual experiments can be found in Appendix C.2. There are also diagrams illustrating the recall and precision on page 108 (DBClean) and on page 110 (DBDirtier).
As above, the recall of the cleaner database is almost constantly 1, while the precision, depending on the tuple pair decision variant, drops from somewhere between 0.895 and 0.9 to 0.882 and thus also reduces the F-measure slightly from about 0.945 to 0.938. In DBDirtier, it is the other way around: the recall is improved dramatically from under 0.9 (baseline) to almost 1, while the precision drops by 0.03 from around 0.91 to 0.88, so that the F-measure is increased from under 0.9 to over 0.93.

5.3.2. Step 3: Reducing Alternative Pairs

As the results of the alternative pair reduction experiments are very similar to the results produced by the corresponding alternative reduction experiments, we do not go into detail here.
Extensive information on the individual experiments can be found in Appendix C.5. For the precision and recall diagrams of the confidence-based approach, see pages 116 (DBClean) and 119 (DBDirtier). The corresponding diagrams of only using the n most probable alternative pairs can be found on pages 122 (DBClean) and 125 (DBDirtier).

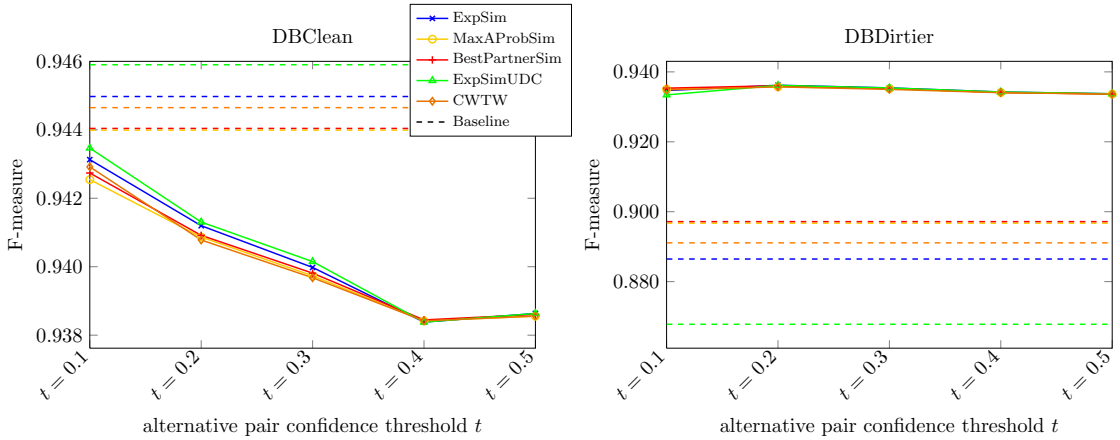


Figure 5.11.: Reducing alternative pairs with a confidence threshold.

The two diagrams in Figure 5.11 shown the F-measure of our alternative pair reduction experiments with varying thresholds on DBClean and DBDirtier. Figure 5.12 covers the corresponding experiments using only the n most probable alternative pairs.
The recall in the clean database is extremely high, no matter whether alternative pairs are reduced or not, so that the F-measure is again dominated by the precision. Like in the experiments on the reduction of alternatives, the F-measure only drops to about 0.938 in both experiments here.
On DBDirtier, recall, precision and hence the F-measure eventually reach almost the same values as in the experiments before. It is interesting, though, that a threshold of $t = 0.1$ directly leads to a recall of about 0.98 and an F-measure of almost 0.94 which are then only slightly improved by increasing the threshold.

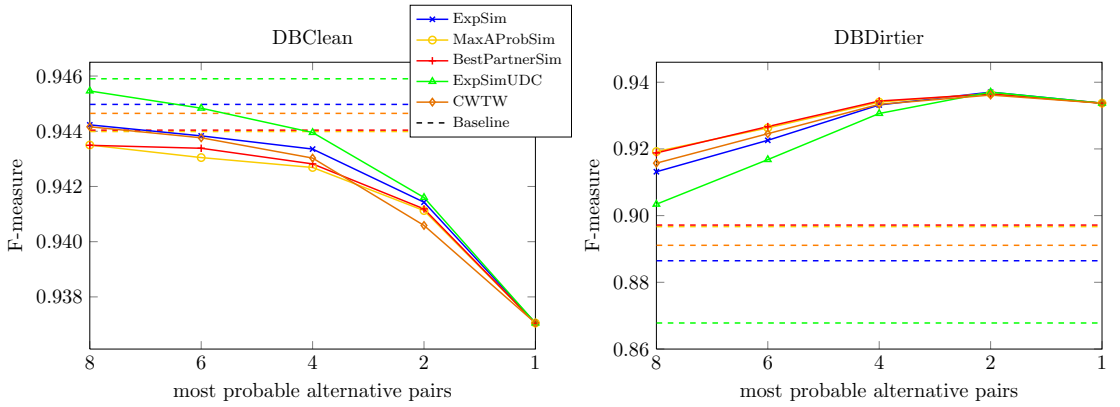


Figure 5.12.: Using only the n most probable alternative pairs.

5.3.3. Step 5: Reducing Comparison Vectors

It turns out that computing the tuple pair decision on the basis of a representative comparison vector does not work well. While a precision of over 0.97 in DBClean and 0.9 in DBDirtier can be achieved, the recall is devastating, as it is only 0.43 in the clean and even less than 0.05 in the dirty database. For obvious reasons, we do not go into detail here, either. For detailed results, see Appendix C.7.

5.3.4. Final Evaluation

For the final evaluation, we compare the reduction strategies on DBClean and DBDirtier that deliver the best F-measure. We only show diagrams on the F-measure here, but the corresponding diagrams illustrating recall and precision are given in Appendix C.10 on pages 142 (DBClean) and 143 (DBDirtier).

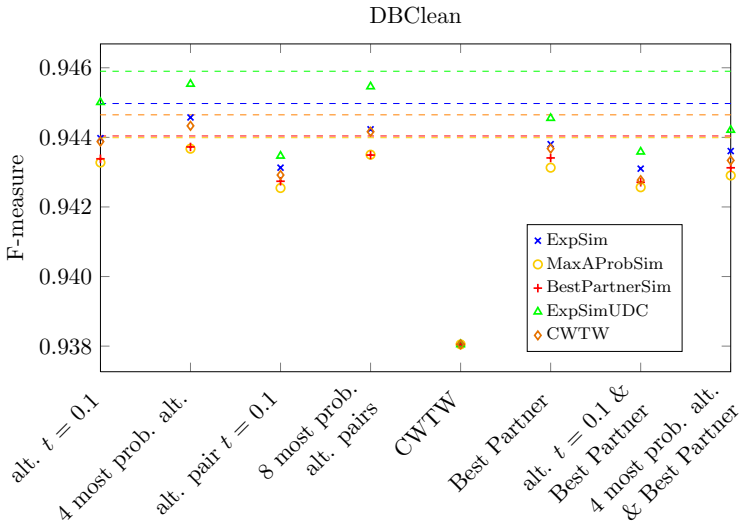


Figure 5.13.: An F-measure comparison of the best reduction strategies on DBClean.

An overview over the reduction strategies with the best F-measure on DBClean is given in Figure 5.13.
The most obvious observation is that none of the reduction strategies can improve the baseline performance: applying a reduction strategy basically improves the recall slightly, but also diminishes the precision a little bit stronger, so that the F-measure is decreased. The by far worst reduction strategy is Cry with the Wolves with an F-measure of 0.938. Apart from that, all reduction strategies achieve an F-measure between 0.942 and 0.946 where using only the 4 most probable alternatives and the 8 most probable alternative pairs delivers the best F-measure. ExpSimUDC always performs better than all the other tuple pair decision variants.
Combining reduction strategies does not increase the F-measure: the combined reduction strategies perform even worse than the individual strategies.

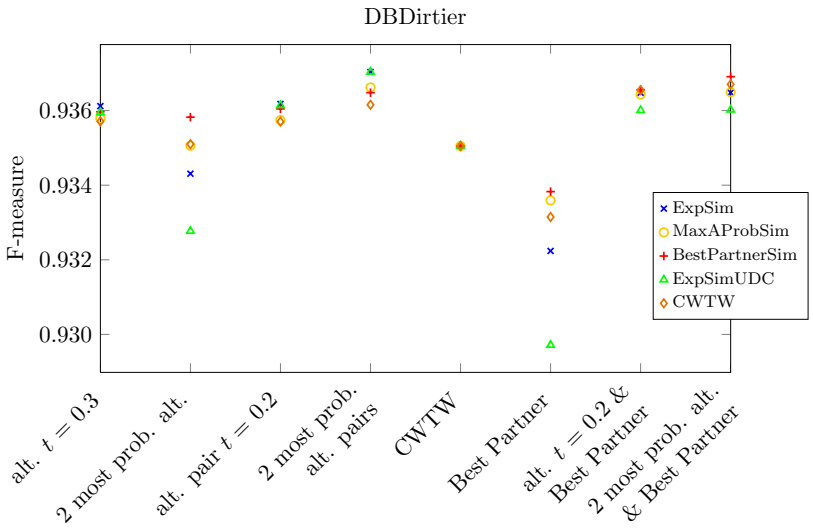


Figure 5.14.: An F-measure comparison of the best reduction strategies on DBDirtier.

The best reduction strategies and two combinations are compared in Figure 5.14 by their respective F-measure on DBDirtier.
On DBDirtier, all reduction strategies improve the F-measure strongly: although the precision is 0.01 to 0.03 below the baseline of 0.91, the recall is also improved from significantly below 0.9 to 0.96 and above. There is no single best-performing tuple pair decision variant, but ExpSim and especially ExpSimUDC perform noticeably worse than all other tuple pair decision variants, when only the 2 most probable alternatives are used or the Best Partner Reduction is applied. The Best Partner Reduction in combination with the reduction of alternatives with a threshold of $t = 0.2$ or in combination with using only the 2 most probable alternatives performs better than the individual reduction strategies. However, the best F-measure is achieved, when only the 2 most probable alternative pairs are used.

The results of all tuple pair decision variants are highly similar, but some differences can be observed: ExpSimUDC works without confidence values and dominates the F-measure in DBClean, while it performs slightly worse than the other tuple pair decision variants in DBDirtier. Applying an alternative or alternative pair reduction strategy does not affect our tuple matching approach on the clean database much, but substantially improves the recall and hence the F-measure on the dirtier database, so that applying some kind of reduction seems reasonable. The aim of all reduction strategies is to reduce the number of low-quality alternative pairs. However, the best-performing strategies do this according to confidence values and require the user to configure some parameters: a confidence threshold or the number of alternatives or alternative pairs to use.
In a real application, though, it may be unknown in what way the confidence values correspond to the quality of the alternatives. Considering this, the combination of the Best Partner Reduction and ExpSimUDC seems to be the most reasonable choice in the

general case, since no confidence values are required and no parameters have to be defined by the user.

6. Summary and Future Prospects

The objectives of this work are to investigate how existing tuple matching techniques known from duplicate detection in certain data can be applied to uncertain probabilistic data, to evaluate the developed approaches experimentally and to generate appropriate synthetic test data to make the evaluation possible in the first place.
To this end, we present *ProbGee*, an Eclipse-based generator for labelled probabilistic data sets from existing certain data. We introduce *error provenance trees* to illustrate the generation process and parameters such as the duplicate cluster size, intra-tuple or zero-alternative similarity, cleanness and overlapping. Typographical errors are modelled with confusion matrices, but other errors such as phonetic or semantic errors can be introduced with confusion sets. Due to ProbGee's highly modular architecture, the existing implementation can easily be extended or adapted.
The basic idea behind our tuple matching approach is to match alternative pairs corresponding to a tuple pair with an iterative hybrid decision model known from certain data duplicate detection and to derive the tuple pair matching decision from the corresponding alternative pair labels. We describe how using all alternative pairs can impair the performance of the decision model and propose three counter-strategies: reducing alternatives, reducing alternative pairs and reducing comparison vectors. Apart from strategies using confidence values for the reduction of alternatives and alternative pairs, the Best Partner Reduction is discussed which reduces alternative pairs on the basis of similarity computations alone. We also propose four variants of deriving the tuple pair decision, namely Expected Similarity, Expected Similarity with Uniformly Distributed Confidences, Best Partner Similarity, Maximal α-Prob. Similarity and Cry with the Wolves.
Our tuple matching approach is evaluated on two probabilistic movie databases generated with ProbGee. The experimental results indicate that reducing alternatives or alternative pairs increases recall and F-measure of the result strongly on the dirtier of the two databases, while it has little effect on the result on the cleaner database. The different tuple pair decision variants perform similarly on both databases. In a real application, using a combination of reducing alternative pairs with the Best Partner Reduction and deriving tuple pair decisions according to the Expected Similarity with Uniformly Distributed Confidences seems to be reasonable, because it functions without confidence values and does not require the user to define any parameters.

The best-performing reduction strategies work under the assumption that the confidence value of an alternative is a good indicator for how well it represents the corresponding entity. Future research could investigate how well those strategies perform in databases where this

assumption is invalid and whether the combination of the Best Partner Reduction and the Expected Similarity with Uniformly Distributed Confidences performs better in this case. A comparison of the different reduction techniques with respect to run time can also be conducted in the future.
Furthermore, it remains an open issue whether the resolution of conflicting tuple pair decisions can improve the tuple matching result.

Concerning ProbGee, the list of open issues is long. Although the generator offers a generic approach towards error generation by using lists of confusion sets, it can only be used in the movie domain at the moment, because there are hardly any other lists of confusion sets available. We are not aware of any approaches towards the generation of large synonym confusion sets for the English language, but we think that WordNet[1] could be used for the task. The generation of phonetic errors according to rules is another goal for future work. Most importantly, though, we are planning to extend ProbGee by the feature of generating certain data in the near future, so that it can also be used in the field of certain data duplicate detection.

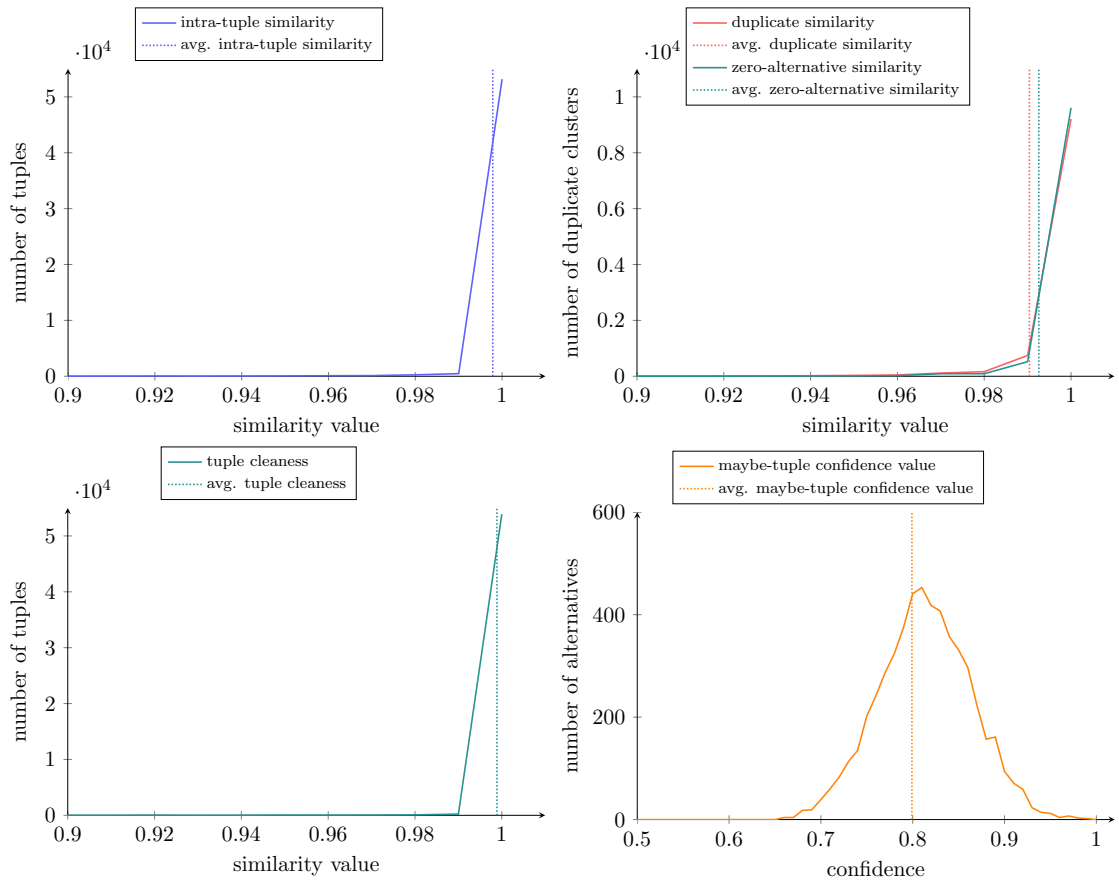
[1] WordNet: `http://wordnet.princeton.edu/`.

Appendix

A. Generated Databases

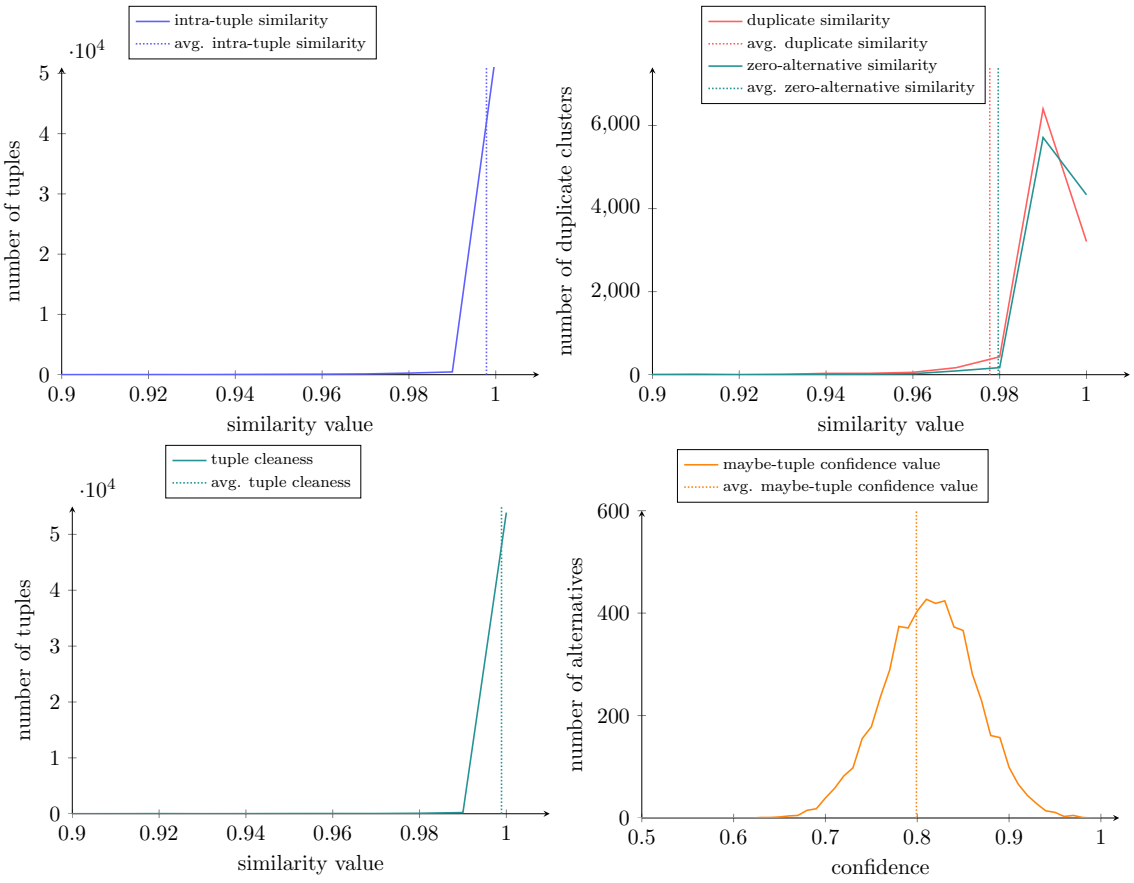
DBCleanest

database quality		
avg. intra-tuple sim.	0.998	
avg. duplicate sim.	0.99	
avg. zero-alternative sim.	0.993	
avg. tuple cleanness	0.999	
overlapping	expected	0.0
	standard deviation	0.0



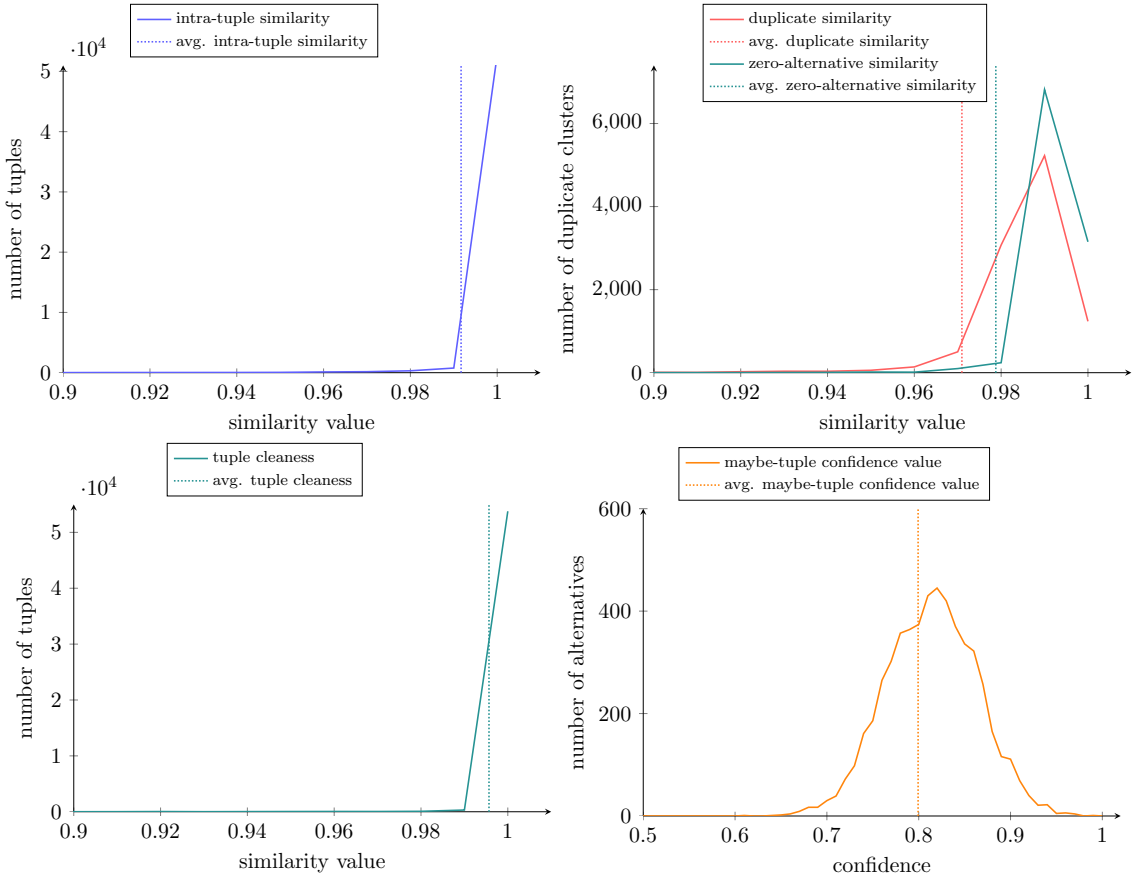
DBCleaner

database quality		
avg. intra-tuple sim.	0.998	
avg. duplicate sim.	0.978	
avg. zero-alternative sim.	0.98	
avg. tuple cleanness	0.999	
overlapping	expected	0.0
	standard deviation	0.0

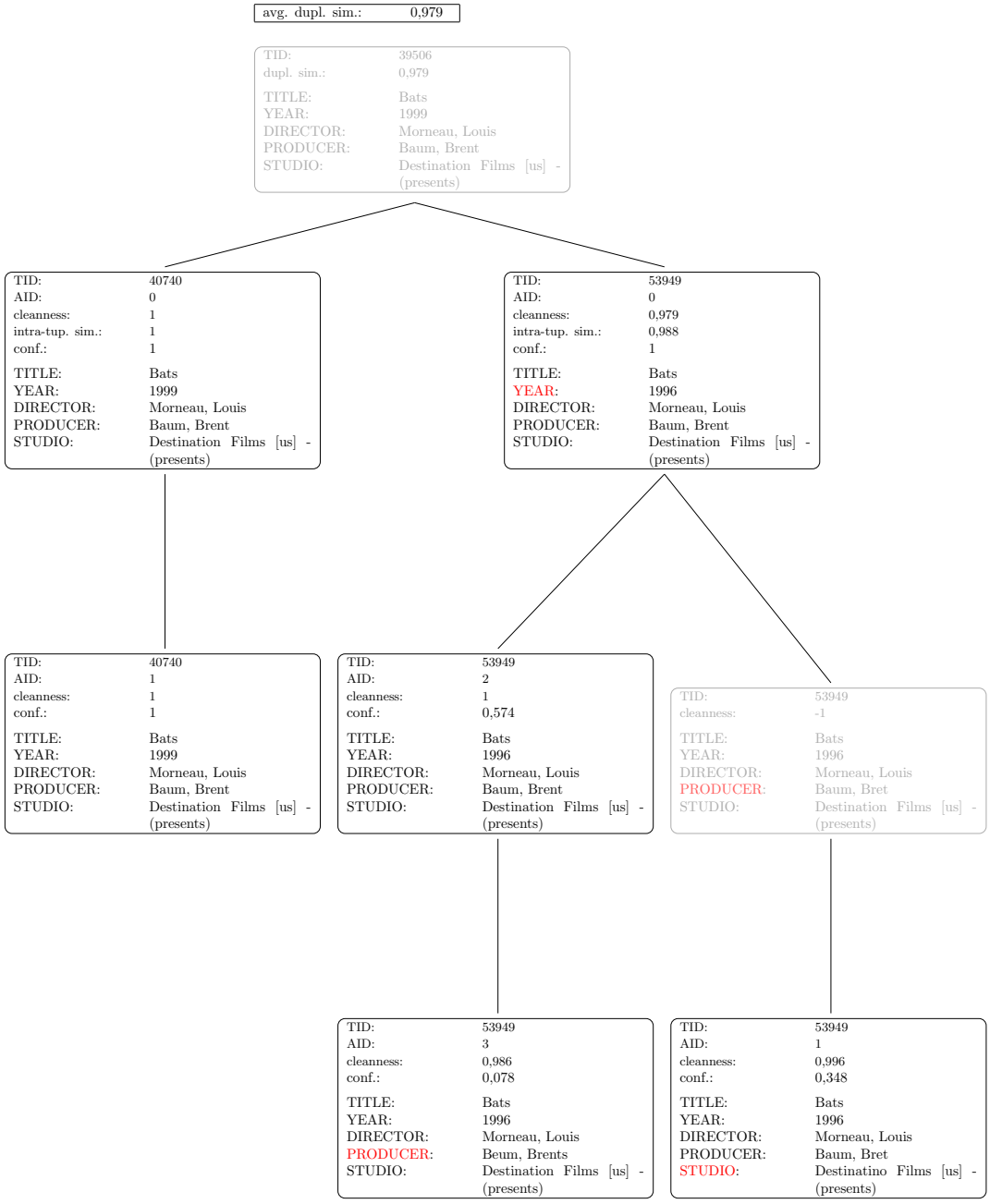


DBClean

database quality		
avg. intra-tuple sim.	0.992	
avg. duplicate sim.	0.971	
avg. zero-alternative sim.	0.979	
avg. tuple cleanness	0.996	
overlapping	expected	0.0
	standard deviation	0.0

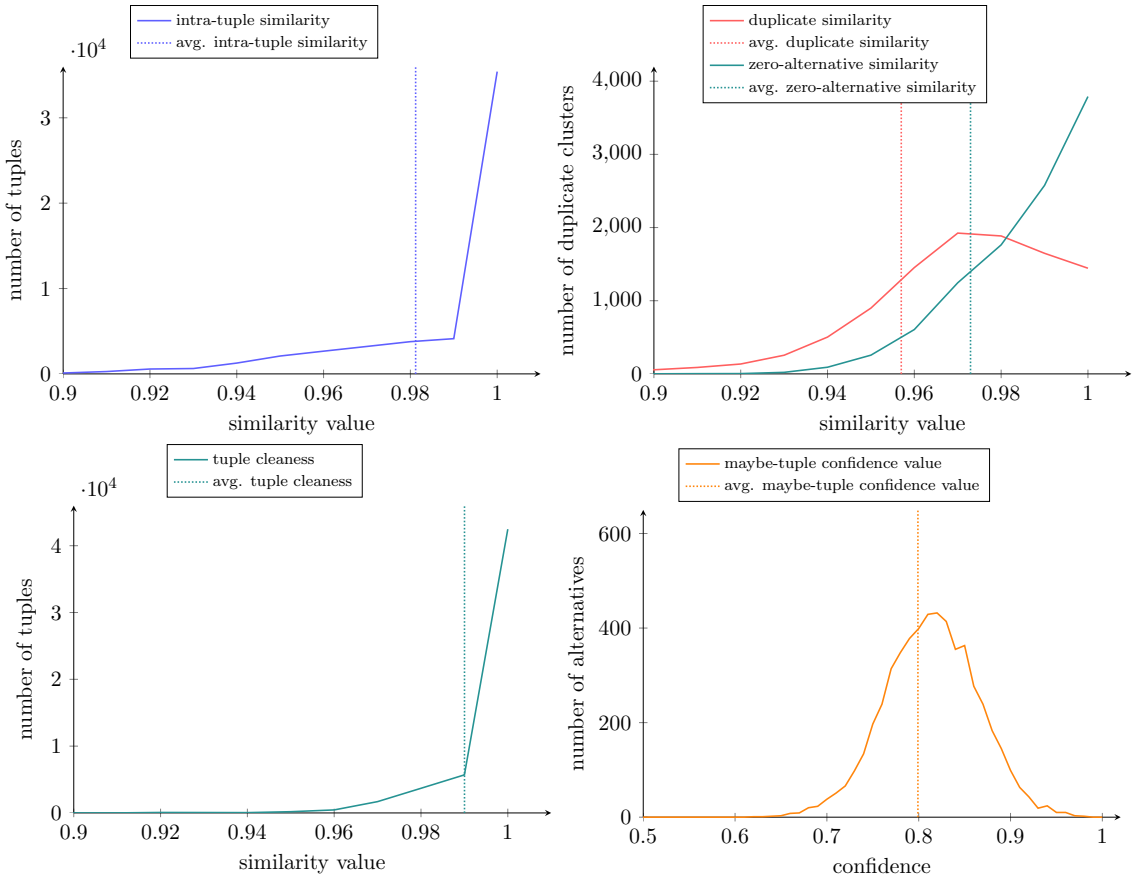


Example Error Provenance Tree (DBClean)



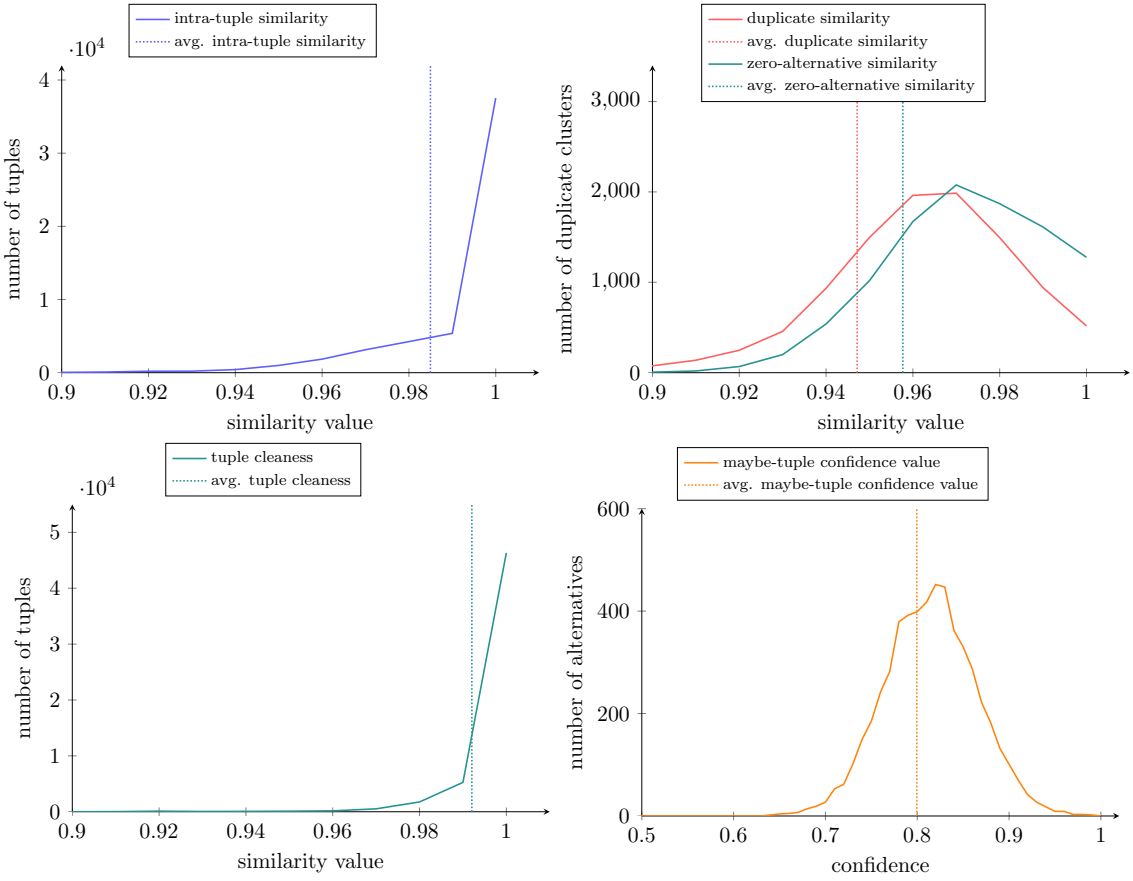
DBDirty

database quality		
avg. intra-tuple sim.	0.981	
avg. duplicate sim.	0.957	
avg. zero-alternative sim.	0.973	
avg. tuple cleanness	0.99	
overlapping	expected	0.0
	standard deviation	0.0

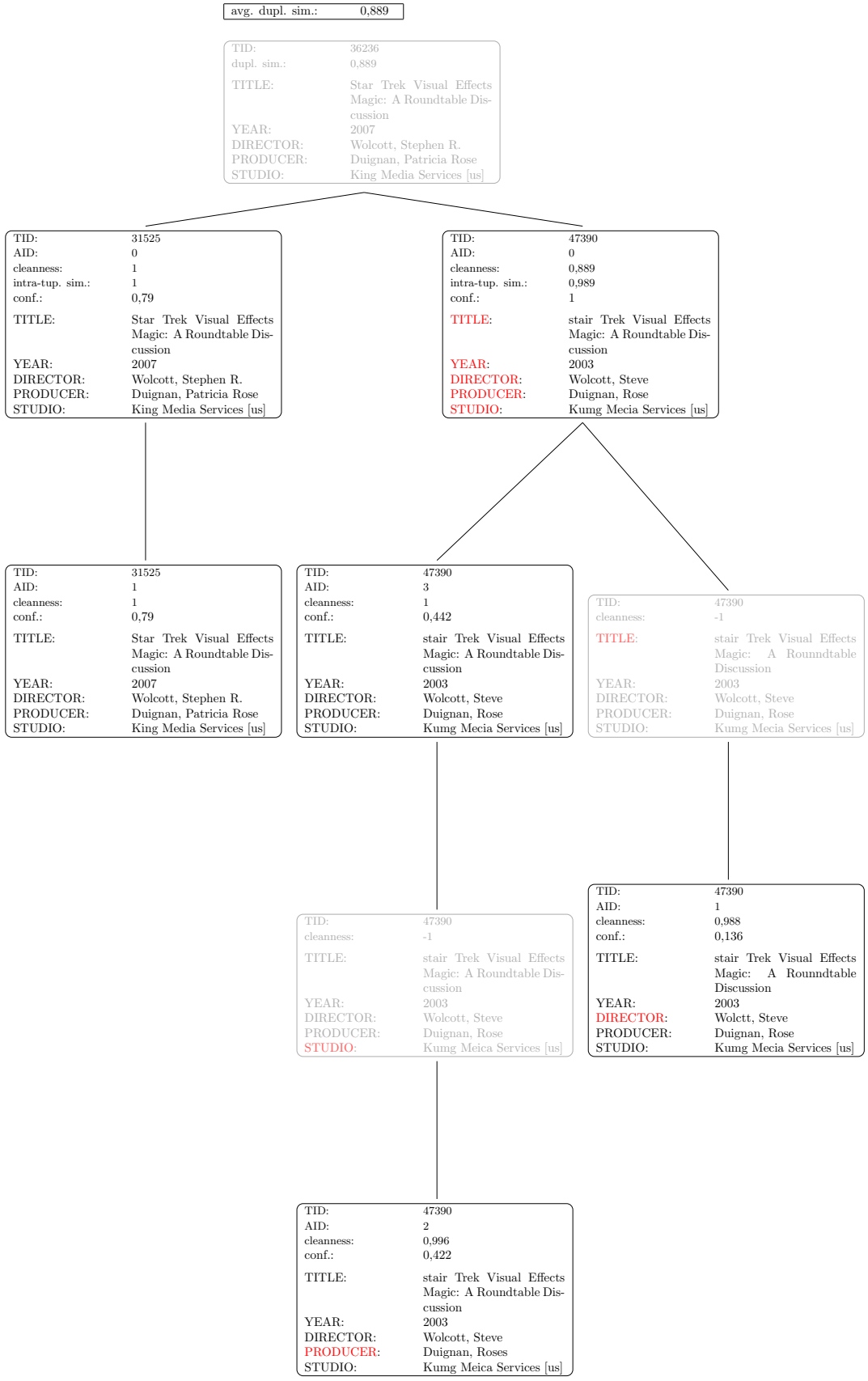


DBDirtier

database quality		
avg. intra-tuple sim.	0.985	
avg. duplicate sim.	0.947	
avg. zero-alternative sim.	0.958	
avg. tuple cleanness	0.992	
overlapping	expected	0.0
	standard deviation	0.0

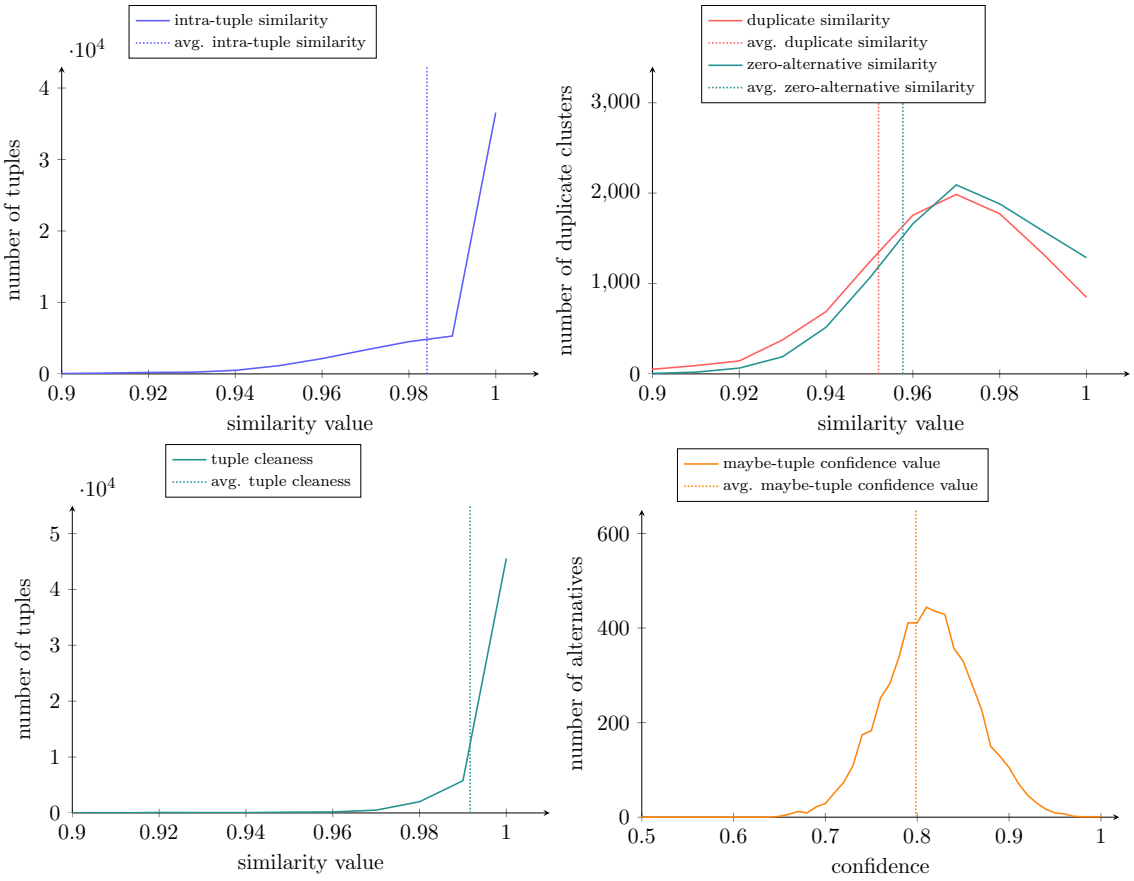


Example Error Provenance Tree (DBDirtier)



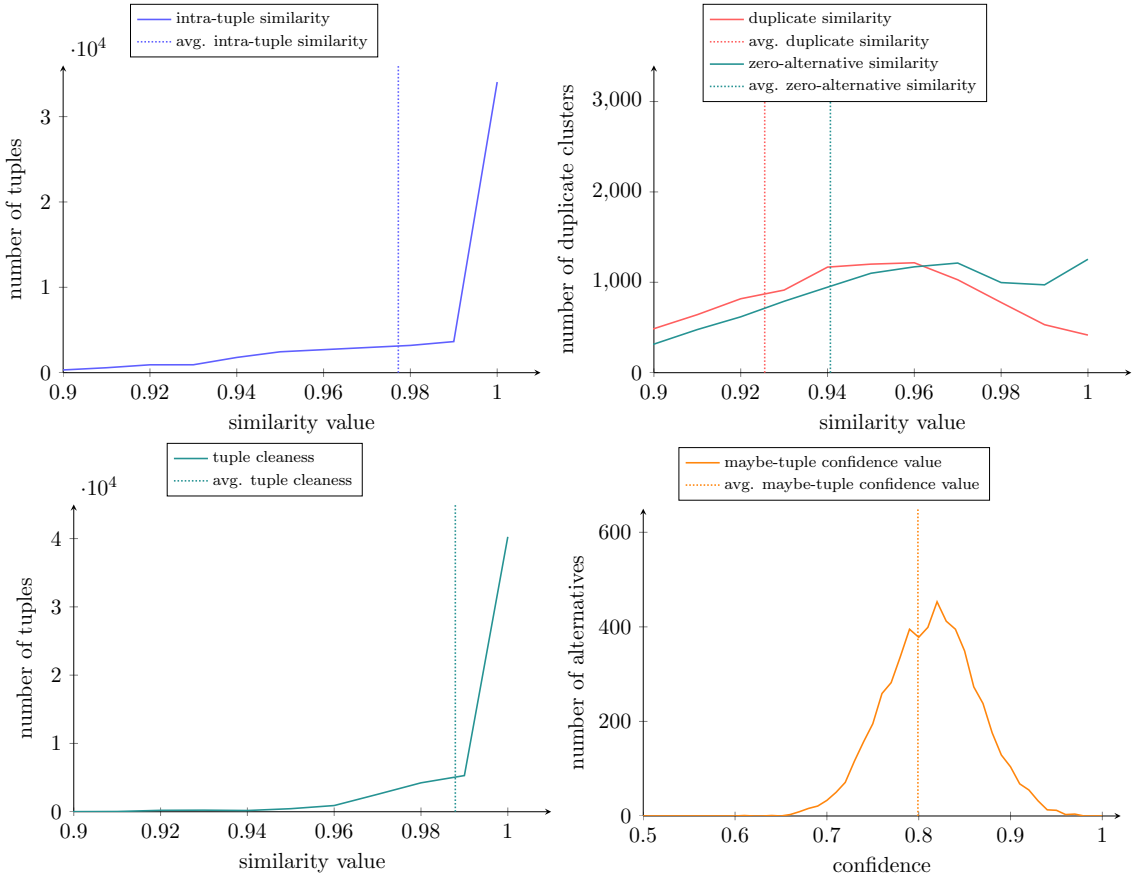
DBDirtierOverlapping

database quality		
avg. intra-tuple sim.	0.984	
avg. duplicate sim.	0.952	
avg. zero-alternative sim.	0.958	
avg. tuple cleanness	0.992	
overlapping	expected	0.2
	standard deviation	0.1



DBDirtiest

database quality		
avg. intra-tuple sim.	0.977	
avg. duplicate sim.	0.926	
avg. zero-alternative sim.	0.941	
avg. tuple cleanness	0.988	
overlapping	expected	0.0
	standard deviation	0.0



B. Baseline Configuration Experiments

B.1. Nearest-Based Initial Match Vector Selection

20 Initial Match Vectors (Config0)

configuration	
database	DBClean
reducing alternatives	-
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	20
initial unmatch vectors	156
SVM kernel	linear

Measure	*ExpSim*	*MaxAProbSim*	*BestPartnerSim*	*ExpSimUDC*	*CWTW*
TruePositives	5	7	8	3	5
FalsePositives	6	8	8	5	6
TrueNegatives	103266	103264	103264	103267	103266
FalseNegatives	15534	15532	15531	15536	15534
Recall	0.000322	0.00045	0.000515	0.000193	0.000322
Precision	0.454545	0.466667	0.5	0.375	0.454545
F-measure	0.000643	0.0009	0.001029	0.000386	0.000643

100 Initial Match Vectors (Config1)

configuration	
database	DBClean
reducing alternatives	-
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	780
SVM kernel	linear

Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15528	15529	15526	15525	15528
FalsePositives	2148	2196	2178	2114	2177
TrueNegatives	101124	101076	101094	101158	101095
FalseNegatives	11	10	13	14	11
Recall	0.999292	0.999356	0.999163	0.999099	0.999292
Precision	0.878479	0.876107	0.876977	0.880152	0.87704
F-measure	0.934999	0.933682	0.934091	0.935861	0.934184

500 Initial Match Vectors (Config2)

configuration	
database	DBClean
reducing alternatives	-
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	500
initial unmatch vectors	3898
SVM kernel	linear

Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15533	15535	15534	15533	15533
FalsePositives	2632	2663	2654	2601	2647
TrueNegatives	100640	100609	100618	100671	100625
FalseNegatives	6	4	5	6	6
Recall	0.999614	0.999743	0.999678	0.999614	0.999614
Precision	0.855106	0.853665	0.85408	0.856568	0.8544
F-measure	0.92173	0.920947	0.921161	0.922579	0.92132

1,000 Initial Match Vectors (Config3)

configuration	
database	DBClean
reducing alternatives	-
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	1000
initial unmatch vectors	7795
SVM kernel	linear

Measure	*ExpSim*	*MaxAProbSim*	*BestPartnerSim*	*ExpSimUDC*	*CWTW*
TruePositives	15533	15535	15534	15533	15533
FalsePositives	2639	2668	2661	2606	2653
TrueNegatives	100633	100604	100611	100666	100619
FalseNegatives	6	4	5	6	6
Recall	0.999614	0.999743	0.999678	0.999614	0.999614
Precision	0.854777	0.853431	0.853751	0.856332	0.854119
F-measure	0.921539	0.920811	0.92097	0.922442	0.921156

B.2. Different SVM Kernels

Radial Basis SVM Kernel (Config04)

configuration	
database	DBClean
reducing alternatives	-
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	780
SVM kernel	radial basis function

Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15528	15530	15527	15527	15529
FalsePositives	2042	2074	2060	2024	2059
TrueNegatives	101230	101198	101212	101248	101213
FalseNegatives	11	9	12	12	10
Recall	0.999292	0.999421	0.999228	0.999228	0.999356
Precision	0.883779	0.882186	0.882868	0.884679	0.882932
F-measure	0.937993	0.937151	0.937451	0.938471	0.937543

Baseline (Config5, DBClean)

configuration	
database	DBClean
reducing alternatives	-
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	780
SVM kernel	polynomial

Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15507	15514	15512	15500	15513
FalsePositives	1774	1816	1812	1734	1792
TrueNegatives	101498	101456	101460	101538	101480
FalseNegatives	32	25	27	39	26
Recall	0.997941	0.998391	0.998262	0.99749	0.998327
Precision	0.897344	0.895211	0.895405	0.899385	0.896446
F-measure	0.944973	0.94399	0.94404	0.945901	0.944647

Baseline (Config5, DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	-
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	791
SVM kernel	polynomial

Measure	*ExpSim*	*MaxAProbSim*	*BestPartnerSim*	*ExpSimUDC*	*CWTW*
TruePositives	13386	13728	13749	12813	13529
FalsePositives	1276	1352	1362	1178	1297
TrueNegatives	102767	102691	102681	102865	102746
FalseNegatives	2153	1811	1790	2726	2010
Recall	0.861445	0.883455	0.884806	0.82457	0.870648
Precision	0.912972	0.910345	0.909867	0.915803	0.912519
F-measure	0.886461	0.896698	0.897162	0.867795	0.891092

B.3. Expected Similarity Threshold

DBClean

configuration	
database	DBClean
reducing alternatives	-
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	780
SVM kernel	polynomial

Measure	ExpSim 0.4	ExpSim 0.45	ExpSim 0.5	ExpSim 0.55	ExpSim 0.6
TruePositives	15514	15513	15507	15504	15499
FalsePositives	1811	1793	1774	1756	1738
TrueNegatives	101461	101479	101498	101516	101534
FalseNegatives	25	26	32	35	40
Recall	0.998391	0.998327	0.997941	0.997748	0.997426
Precision	0.895469	0.896394	0.897344	0.898262	0.89917
F-measure	0.944133	0.944619	0.944973	0.945395	0.945753

DBDirtier

configuration	
database	DBDirtier
reducing alternatives	-
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	791
SVM kernel	polynomial

Measure	ExpSim 0.4	ExpSim 0.45	ExpSim 0.5	ExpSim 0.55	ExpSim 0.6
TruePositives	13659	13561	13386	13157	12967
FalsePositives	1330	1307	1276	1234	1194
TrueNegatives	102713	102736	102767	102809	102849
FalseNegatives	1880	1978	2153	2382	2572
Recall	0.879014	0.872707	0.861445	0.846708	0.834481
Precision	0.911268	0.912093	0.912972	0.914252	0.915684
F-measure	0.894851	0.891966	0.886461	0.879185	0.873199

B.4. Maximal α-Prob. Similarity Threshold

DBClean

configuration	
database	DBClean
reducing alternatives	-
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	780
SVM kernel	polynomial

Measure	*MaxAProbSim 0.01*	*MaxAProbSim 0.1*	*MaxAProbSim 0.2*	*MaxAProbSim 0.5*	*MaxAProbSim 0.8*
TruePositives	15515	15515	15515	15514	15507
FalsePositives	1838	1837	1833	1816	1783
TrueNegatives	101434	101435	101439	101456	101489
FalseNegatives	24	24	24	25	32
Recall	0.998455	0.998455	0.998455	0.998391	0.997941
Precision	0.894082	0.894133	0.894339	0.895211	0.896877
F-measure	0.94339	0.943419	0.943534	0.94399	0.944714

DBDirtier

configuration	
database	DBDirtier
reducing alternatives	-
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	791
SVM kernel	polynomial

Measure	MaxAProbSim 0.01	MaxAProbSim 0.1	MaxAProbSim 0.2	MaxAProbSim 0.5	MaxAProbSim 0.8
TruePositives	13961	13921	13882	13727	13463
FalsePositives	1400	1395	1386	1352	1293
TrueNegatives	102643	102648	102657	102691	102750
FalseNegatives	1578	1618	1657	1812	2076
Recall	0.898449	0.895875	0.893365	0.88339	0.866401
Precision	0.90886	0.908919	0.909222	0.910339	0.912375
F-measure	0.903625	0.90235	0.901224	0.896662	0.888794

C. Reduction Experiments

C.1. Reducing Alternatives with a Confidence Threshold

Confidence Threshold 0.1 (DBClean)

configuration	
database	DBClean
reducing alternatives	confidence threshold 0.1
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	652
SVM kernel	polynomial

Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15510	15515	15513	15509	15515
FalsePositives	1812	1842	1836	1775	1821
TrueNegatives	101460	101430	101436	101497	101451
FalseNegatives	29	24	26	30	24
Recall	0.998134	0.998455	0.998327	0.998069	0.998455
Precision	0.895393	0.893876	0.894173	0.897304	0.894958
F-measure	0.943976	0.943276	0.943384	0.945008	0.943878

Confidence Threshold 0.2 (DBClean)

configuration	
database	DBClean
reducing alternatives	confidence threshold 0.2
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	488
SVM kernel	polynomial

Measure	*ExpSim*	*MaxAProbSim*	*BestPartnerSim*	*ExpSimUDC*	*CWTW*
TruePositives	15514	15520	15516	15514	15519
FalsePositives	1856	1878	1870	1846	1868
TrueNegatives	101416	101394	101402	101426	101404
FalseNegatives	25	19	23	25	20
Recall	0.998391	0.998777	0.99852	0.998391	0.998713
Precision	0.893149	0.892057	0.892442	0.893664	0.892563
F-measure	0.942842	0.942405	0.942506	0.943129	0.942659

Confidence Threshold 0.3 (DBClean)

configuration	
database	DBClean
reducing alternatives	confidence threshold 0.3
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	320
SVM kernel	polynomial

Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15517	15520	15520	15515	15520
FalsePositives	1927	1944	1947	1923	1950
TrueNegatives	101345	101328	101325	101349	101322
FalseNegatives	22	19	19	24	19
Recall	0.998584	0.998777	0.998777	0.998455	0.998777
Precision	0.889532	0.888685	0.888533	0.889724	0.88838
F-measure	0.940909	0.940521	0.940435	0.940959	0.94035

Confidence Threshold 0.4 (DBClean)

configuration	
database	DBClean
reducing alternatives	confidence threshold 0.4
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	171
SVM kernel	polynomial

Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15519	15521	15521	15519	15521
FalsePositives	1999	2003	2000	1999	2005
TrueNegatives	101273	101269	101272	101273	101267
FalseNegatives	20	18	18	20	18
Recall	0.998713	0.998842	0.998842	0.998713	0.998842
Precision	0.885889	0.8857	0.885851	0.885889	0.885599
F-measure	0.938924	0.938874	0.938959	0.938924	0.938817

Confidence Threshold 0.5 (DBClean)

configuration	
database	DBClean
reducing alternatives	confidence threshold 0.5
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	158
SVM kernel	polynomial

Measure	*ExpSim*	*MaxAProbSim*	*BestPartnerSim*	*ExpSimUDC*	*CWTW*
TruePositives	15521	15523	15521	15521	15523
FalsePositives	2025	2032	2027	2025	2033
TrueNegatives	101247	101240	101245	101247	101239
FalseNegatives	18	16	18	18	16
Recall	0.998842	0.99897	0.998842	0.998842	0.99897
Precision	0.884589	0.88425	0.884488	0.884589	0.884199
F-measure	0.93825	0.938116	0.938193	0.93825	0.938087

Confidence Threshold 0.1 (DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	confidence threshold 0.1
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	658
SVM kernel	polynomial

Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	14688	14846	14854	14513	14759
FalsePositives	1576	1623	1619	1527	1596
TrueNegatives	102467	102420	102424	102516	102447
FalseNegatives	851	693	685	1026	780
Recall	0.945235	0.955403	0.955917	0.933973	0.949804
Precision	0.903099	0.901451	0.901718	0.9048	0.902415
F-measure	0.923686	0.927643	0.928027	0.919155	0.925503

Confidence Threshold 0.2 (DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	confidence threshold 0.2
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	491
SVM kernel	polynomial

Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15173	15219	15233	15117	15194
FalsePositives	1764	1806	1808	1742	1790
TrueNegatives	102279	102237	102235	102301	102253
FalseNegatives	366	320	306	422	345
Recall	0.976446	0.979407	0.980308	0.972843	0.977798
Precision	0.895849	0.893921	0.893903	0.896672	0.894607
F-measure	0.934413	0.934713	0.935114	0.933206	0.934354

Confidence Threshold 0.3 (DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	confidence threshold 0.3
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	323
SVM kernel	polynomial

Measure	*ExpSim*	*MaxAProbSim*	*BestPartnerSim*	*ExpSimUDC*	*CWTW*
TruePositives	15365	15382	15385	15351	15377
FalsePositives	1923	1954	1951	1913	1951
TrueNegatives	102120	102089	102092	102130	102092
FalseNegatives	174	157	154	188	162
Recall	0.988802	0.989896	0.990089	0.987901	0.989575
Precision	0.888767	0.887287	0.88746	0.889191	0.887408
F-measure	0.93612	0.935787	0.93597	0.935951	0.935711

Confidence Threshold 0.4 (DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	confidence threshold 0.4
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	172
SVM kernel	polynomial

Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15454	15455	15456	15454	15456
FalsePositives	2088	2095	2098	2087	2106
TrueNegatives	101955	101948	101945	101956	101937
FalseNegatives	85	84	83	85	83
Recall	0.99453	0.994594	0.994659	0.99453	0.994659
Precision	0.880971	0.880627	0.880483	0.881022	0.880082
F-measure	0.934313	0.934147	0.934095	0.934341	0.933869

Confidence Threshold 0.5 (DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	confidence threshold 0.5
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	160
SVM kernel	polynomial

Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15456	15457	15458	15456	15458
FalsePositives	2106	2109	2112	2106	2112
TrueNegatives	101937	101934	101931	101937	101931
FalseNegatives	83	82	81	83	81
Recall	0.994659	0.994723	0.994787	0.994659	0.994787
Precision	0.880082	0.879939	0.879795	0.880082	0.879795
F-measure	0.933869	0.933817	0.933764	0.933869	0.933764

C.2. Using the n Most Probable Alternatives

4 Most Probable Alternatives (DBClean)

configuration	
database	DBClean
reducing alternatives	4 most probable alternatives
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	746
SVM kernel	polynomial

Measure	*ExpSim*	*MaxAProbSim*	*BestPartnerSim*	*ExpSimUDC*	*CWTW*
TruePositives	15508	15514	15512	15502	15513
FalsePositives	1789	1827	1823	1749	1803
TrueNegatives	101483	101445	101449	101523	101469
FalseNegatives	31	25	27	37	26
Recall	0.998005	0.998391	0.998262	0.997619	0.998327
Precision	0.896572	0.894643	0.894837	0.898615	0.895877
F-measure	0.944573	0.943674	0.943725	0.945532	0.944331

3 Most Probable Alternatives (DBClean)

configuration	
database	DBClean
reducing alternatives	3 most probable alternatives
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	649
SVM kernel	polynomial

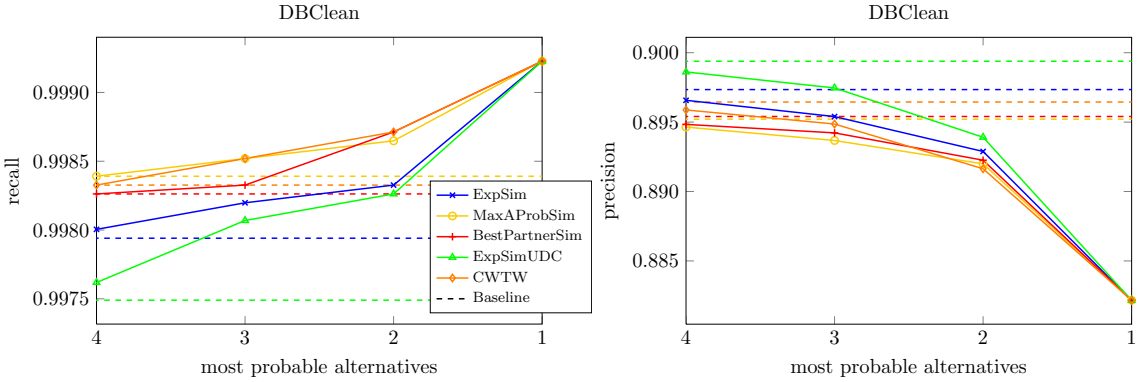
Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15511	15516	15513	15509	15516
FalsePositives	1812	1846	1835	1772	1823
TrueNegatives	101460	101426	101437	101500	101449
FalseNegatives	28	23	26	30	23
Recall	0.998198	0.99852	0.998327	0.998069	0.99852
Precision	0.895399	0.893676	0.894224	0.89746	0.894861
F-measure	0.944008	0.943193	0.943412	0.945094	0.943853

2 Most Probable Alternatives (DBClean)

configuration	
database	DBClean
reducing alternatives	2 most probable alternatives
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	452
SVM kernel	polynomial

Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15513	15518	15519	15512	15519
FalsePositives	1861	1879	1874	1841	1886
TrueNegatives	101411	101393	101398	101431	101386
FalseNegatives	26	21	20	27	20
Recall	0.998327	0.998649	0.998713	0.998262	0.998713
Precision	0.892886	0.891993	0.892256	0.893909	0.89164
F-measure	0.942667	0.942312	0.942488	0.943208	0.942144

Using the n Most Probable Alternatives – Comparison (DBClean)



4 Most Probable Alternatives (DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	4 most probable alternatives
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	756
SVM kernel	polynomial

Measure	*ExpSim*	*MaxAProbSim*	*BestPartnerSim*	*ExpSimUDC*	*CWTW*
TruePositives	13625	13910	13942	13126	13742
FalsePositives	1328	1391	1405	1242	1342
TrueNegatives	102715	102652	102638	102801	102701
FalseNegatives	1914	1629	1597	2413	1797
Recall	0.876826	0.895167	0.897226	0.844713	0.884355
Precision	0.911188	0.909091	0.908451	0.913558	0.911032
F-measure	0.893677	0.902075	0.902804	0.877788	0.897495

3 Most Probable Alternatives (DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	3 most probable alternatives
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	656
SVM kernel	polynomial

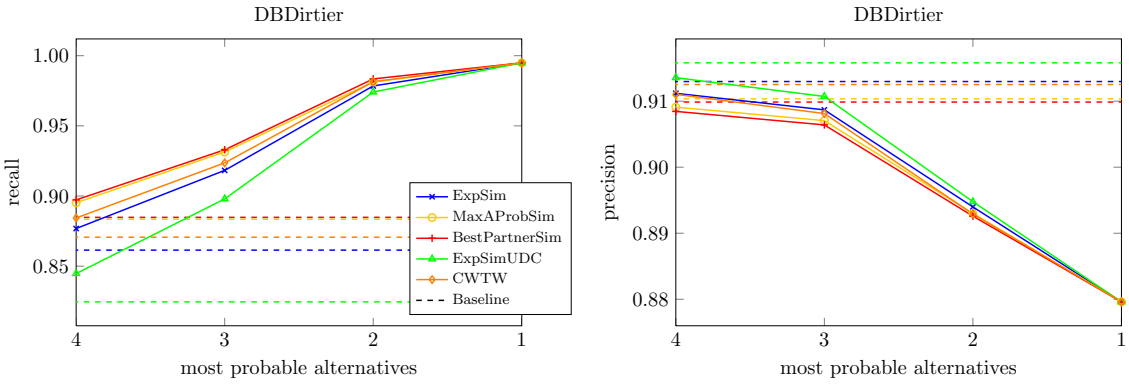
Measure	*ExpSim*	*MaxAProbSim*	*BestPartnerSim*	*ExpSimUDC*	*CWTW*
TruePositives	14269	14473	14497	13953	14352
FalsePositives	1434	1483	1497	1368	1452
TrueNegatives	102609	102560	102546	102675	102591
FalseNegatives	1270	1066	1042	1586	1187
Recall	0.91827	0.931398	0.932943	0.897934	0.923612
Precision	0.90868	0.907057	0.906402	0.910711	0.908125
F-measure	0.91345	0.919067	0.919481	0.904277	0.915803

2 Most Probable Alternatives (DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	2 most probable alternatives
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	454
SVM kernel	polynomial

Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15204	15248	15282	15137	15251
FalsePositives	1803	1827	1839	1780	1829
TrueNegatives	102240	102216	102204	102263	102214
FalseNegatives	335	291	257	402	288
Recall	0.978441	0.981273	0.983461	0.97413	0.981466
Precision	0.893985	0.893001	0.892588	0.89478	0.892916
F-measure	0.934308	0.935059	0.935824	0.932771	0.935099

Using the n Most Probable Alternatives – Comparison (DBDirtier)



C.3. Using a Single Alternative per Tuple

1 Most Probable Alternative (DBClean)

configuration	
database	DBClean
reducing alternatives	1 most probable alternative
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	138
SVM kernel	polynomial

results	
true positives	15527
false positives	2074
true negatives	101198
false negatives	12
recall	0.999228
precision	0.882166
F-measure	0.937055

Cry with the Wolves (DBClean)

configuration	
database	DBClean
reducing alternatives	Cry with the Wolves
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	138
SVM kernel	polynomial

results	
true positives	15528
false positives	2040
true negatives	101232
false negatives	11
recall	0.999292
precision	0.88388
F-measure	0.938049

1 Most Probable Alternative (DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	1 most probable alternative
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	139
SVM kernel	polynomial

results	
true positives	15460
false positives	2116
true negatives	101927
false negatives	79
recall	0.994916
precision	0.879609
F-measure	0.933716

Cry with the Wolves (DBDirtier)

configuration	
database	DBClean
reducing alternatives	Cry with the Wolves
reducing alternative pairs	-
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	139
SVM kernel	polynomial

results	
true positives	15469
false positives	2079
true negatives	101964
false negatives	70
recall	0.995495
precision	0.881525
F-measure	0.93505

C.4. Reducing Alternative Pairs with a Confidence Threshold

Confidence Threshold 0.1 (DBClean)

configuration	
database	DBClean
reducing alternatives	-
reducing alternative pairs	confidence threshold 0.1
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	490
SVM kernel	polynomial

Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15514	15519	15517	15512	15519
FalsePositives	1846	1872	1863	1832	1859
TrueNegatives	101426	101400	101409	101440	101413
FalseNegatives	25	20	22	27	20
Recall	0.998391	0.998713	0.998584	0.998262	0.998713
Precision	0.893664	0.892358	0.892808	0.894373	0.893026
F-measure	0.943129	0.942545	0.942738	0.943466	0.942917

Confidence Threshold 0.2 (DBClean)

configuration	
database	DBClean
reducing alternatives	-
reducing alternative pairs	confidence threshold 0.2
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	311
SVM kernel	polynomial

Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15518	15521	15519	15516	15521
FalsePositives	1918	1933	1929	1912	1936
TrueNegatives	101354	101339	101343	101360	101336
FalseNegatives	21	18	20	23	18
Recall	0.998649	0.998842	0.998713	0.99852	0.998842
Precision	0.889998	0.889252	0.889443	0.890291	0.889099
F-measure	0.941198	0.940866	0.940916	0.941305	0.940781

Confidence Threshold 0.3 (DBClean)

configuration	
database	DBClean
reducing alternatives	-
reducing alternative pairs	confidence threshold 0.3
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	215
SVM kernel	polynomial

Measure	*ExpSim*	*MaxAProbSim*	*BestPartnerSim*	*ExpSimUDC*	*CWTW*
TruePositives	15520	15523	15521	15520	15523
FalsePositives	1963	1975	1970	1957	1977
TrueNegatives	101309	101297	101302	101315	101295
FalseNegatives	19	16	18	19	16
Recall	0.998777	0.99897	0.998842	0.998777	0.99897
Precision	0.887719	0.88713	0.887371	0.888024	0.887029
F-measure	0.939979	0.939734	0.939812	0.94015	0.939677

Confidence Threshold 0.4 (DBClean)

configuration	
database	DBClean
reducing alternatives	-
reducing alternative pairs	confidence threshold 0.4
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	155
SVM kernel	polynomial

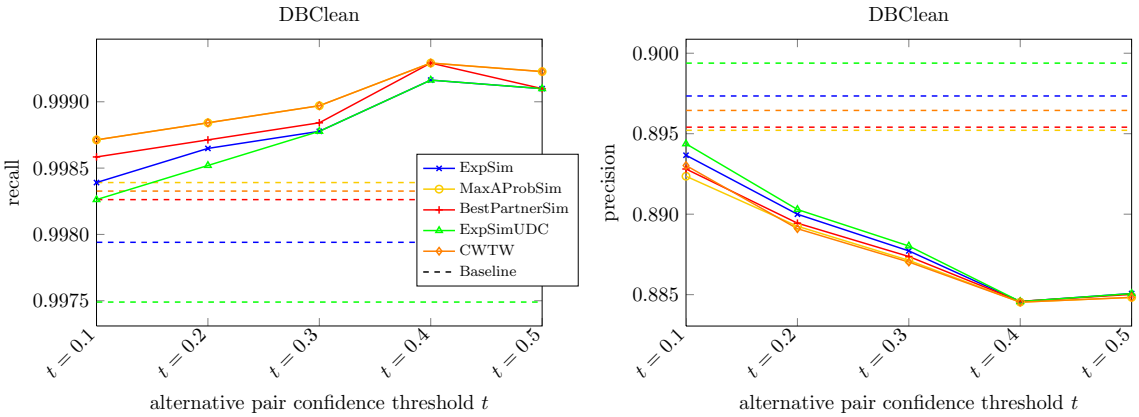
Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15526	15528	15528	15526	15528
FalsePositives	2026	2027	2026	2026	2027
TrueNegatives	101246	101245	101246	101246	101245
FalseNegatives	13	11	11	13	11
Recall	0.999163	0.999292	0.999292	0.999163	0.999292
Precision	0.884572	0.884534	0.884585	0.884572	0.884534
F-measure	0.938382	0.938418	0.938446	0.938382	0.938418

Confidence Threshold 0.5 (DBClean)

configuration	
database	DBClean
reducing alternatives	-
reducing alternative pairs	confidence threshold 0.5
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	148
SVM kernel	polynomial

Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15525	15527	15525	15525	15527
FalsePositives	2016	2021	2017	2016	2021
TrueNegatives	101256	101251	101255	101256	101251
FalseNegatives	14	12	14	14	12
Recall	0.999099	0.999228	0.999099	0.999099	0.999228
Precision	0.885069	0.88483	0.885019	0.885069	0.88483
F-measure	0.938634	0.938556	0.938605	0.938634	0.938556

Confidence Threshold – Comparison (DBClean)



Confidence Threshold 0.1 (DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	-
reducing alternative pairs	confidence threshold 0.1
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	493
SVM kernel	polynomial

Measure	*ExpSim*	*MaxAProbSim*	*BestPartnerSim*	*ExpSimUDC*	*CWTW*
TruePositives	15184	15231	15234	15125	15212
FalsePositives	1768	1808	1801	1744	1793
TrueNegatives	102275	102235	102242	102299	102250
FalseNegatives	355	308	305	414	327
Recall	0.977154	0.980179	0.980372	0.973357	0.978956
Precision	0.895706	0.89389	0.894276	0.896615	0.89456
F-measure	0.934659	0.935048	0.935347	0.933412	0.934857

Confidence Threshold 0.2 (DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	-
reducing alternative pairs	confidence threshold 0.2
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	312
SVM kernel	polynomial

Measure	*ExpSim*	*MaxAProbSim*	*BestPartnerSim*	*ExpSimUDC*	*CWTW*
TruePositives	15402	15415	15419	15392	15412
FalsePositives	1963	1993	1987	1953	1991
TrueNegatives	102080	102050	102056	102090	102052
FalseNegatives	137	124	120	147	127
Recall	0.991183	0.99202	0.992277	0.99054	0.991827
Precision	0.886957	0.885512	0.885844	0.887403	0.885594
F-measure	0.936178	0.935745	0.936045	0.936139	0.935705

Confidence Threshold 0.3 (DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	-
reducing alternative pairs	confidence threshold 0.3
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	217
SVM kernel	polynomial

Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15434	15442	15441	15428	15440
FalsePositives	2026	2043	2040	2020	2048
TrueNegatives	102017	102000	102003	102023	101995
FalseNegatives	105	97	98	111	99
Recall	0.993243	0.993758	0.993693	0.992857	0.993629
Precision	0.883963	0.883157	0.883302	0.884227	0.882891
F-measure	0.935422	0.935199	0.935251	0.935399	0.934993

Confidence Threshold 0.4 (DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	-
reducing alternative pairs	confidence threshold 0.4
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	156
SVM kernel	polynomial

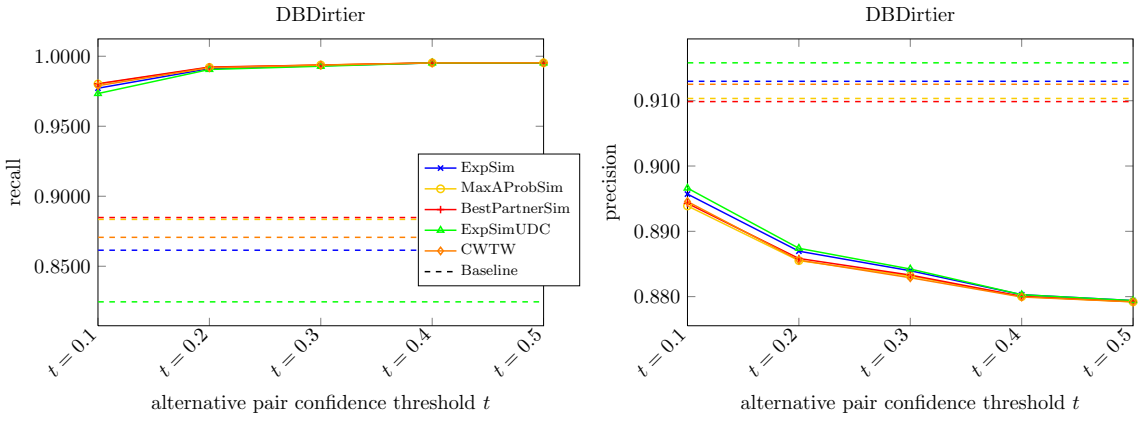
Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15465	15466	15465	15465	15465
FalsePositives	2103	2109	2108	2103	2110
TrueNegatives	101940	101934	101935	101940	101933
FalseNegatives	74	73	74	74	74
Recall	0.995238	0.995302	0.995238	0.995238	0.995238
Precision	0.880294	0.88	0.880043	0.880294	0.879943
F-measure	0.934244	0.934106	0.934102	0.934244	0.934046

Confidence Threshold 0.5 (DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	-
reducing alternative pairs	confidence threshold 0.5
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	149
SVM kernel	polynomial

Measure	*ExpSim*	*MaxAProbSim*	*BestPartnerSim*	*ExpSimUDC*	*CWTW*
TruePositives	15465	15466	15465	15465	15466
FalsePositives	2121	2125	2124	2121	2125
TrueNegatives	101922	101918	101919	101922	101918
FalseNegatives	74	73	74	74	73
Recall	0.995238	0.995302	0.995238	0.995238	0.995302
Precision	0.879393	0.8792	0.879243	0.879393	0.8792
F-measure	0.933736	0.933655	0.933651	0.933736	0.933655

Confidence Threshold – Comparison (DBDirtier)



C.5. Using the n Most Probable Alternative Pairs

8 Most Probable Alternative Pairs (DBClean)

configuration	
database	DBClean
reducing alternatives	-
reducing alternative pairs	8 most probable alternative pairs
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	696
SVM kernel	polynomial

Measure	*ExpSim*	*MaxAProbSim*	*BestPartnerSim*	*ExpSimUDC*	*CWTW*
TruePositives	15509	15514	15511	15506	15514
FalsePositives	1802	1833	1830	1756	1810
TrueNegatives	101470	101439	101442	101516	101462
FalseNegatives	30	25	28	33	25
Recall	0.998069	0.998391	0.998198	0.997876	0.998391
Precision	0.895904	0.894333	0.89447	0.898274	0.895521
F-measure	0.944231	0.943502	0.943491	0.945459	0.944162

6 Most Probable Alternative Pairs (DBClean)

configuration	
database	DBClean
reducing alternatives	-
reducing alternative pairs	6 most probable alternative pairs
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	638
SVM kernel	polynomial

Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15510	15515	15513	15509	15515
FalsePositives	1817	1850	1836	1781	1825
TrueNegatives	101455	101422	101436	101491	101447
FalseNegatives	29	24	26	30	24
Recall	0.998134	0.998455	0.998327	0.998069	0.998455
Precision	0.895135	0.893464	0.894173	0.896992	0.894752
F-measure	0.943833	0.943046	0.943384	0.944835	0.943763

4 Most Probable Alternative Pairs (DBClean)

configuration	
database	DBClean
reducing alternatives	-
reducing alternative pairs	4 most probable alternative pairs
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	534
SVM kernel	polynomial

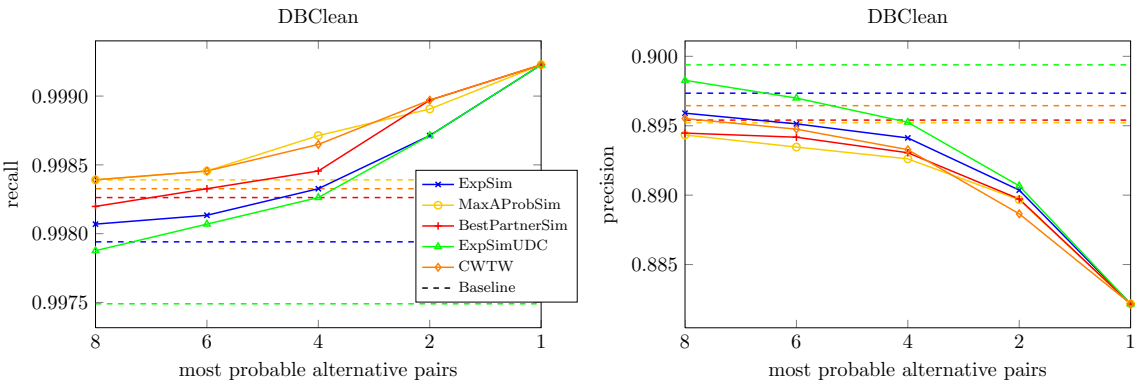
Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15513	15519	15515	15512	15518
FalsePositives	1837	1867	1858	1815	1854
TrueNegatives	101435	101405	101414	101457	101418
FalseNegatives	26	20	24	27	21
Recall	0.998327	0.998713	0.998455	0.998262	0.998649
Precision	0.894121	0.892615	0.893052	0.89525	0.893277
F-measure	0.943355	0.942688	0.942817	0.943954	0.943028

2 Most Probable Alternative Pairs (DBClean)

configuration	
database	DBClean
reducing alternatives	-
reducing alternative pairs	2 most probable alternative pairs
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	321
SVM kernel	polynomial

Measure	*ExpSim*	*MaxAProbSim*	*BestPartnerSim*	*ExpSimUDC*	*CWTW*
TruePositives	15519	15522	15523	15519	15523
FalsePositives	1911	1925	1924	1905	1945
TrueNegatives	101361	101347	101348	101367	101327
FalseNegatives	20	17	16	20	16
Recall	0.998713	0.998906	0.99897	0.998713	0.99897
Precision	0.890361	0.889666	0.889723	0.890668	0.888654
F-measure	0.94143	0.941127	0.941187	0.941601	0.940588

Using the n Most Probable Alternative Pairs – Comparison (DBClean)



8 Most Probable Alternative Pairs (DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	-
reducing alternative pairs	8 most probable alternative pairs
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	704
SVM kernel	polynomial

Measure	*ExpSim*	*MaxAProbSim*	*BestPartnerSim*	*ExpSimUDC*	*CWTW*
TruePositives	14263	14483	14478	13932	14355
FalsePositives	1437	1490	1498	1372	1458
TrueNegatives	102606	102553	102545	102671	102585
FalseNegatives	1276	1056	1061	1607	1184
Recall	0.917884	0.932042	0.93172	0.896583	0.923805
Precision	0.908471	0.906718	0.906234	0.91035	0.907797
F-measure	0.913153	0.919205	0.918801	0.903414	0.915731

6 Most Probable Alternative Pairs (DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	-
reducing alternative pairs	6 most probable alternative pairs
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	644
SVM kernel	polynomial

Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	14628	14773	14786	14412	14696
FalsePositives	1544	1585	1586	1489	1555
TrueNegatives	102499	102458	102457	102554	102488
FalseNegatives	911	766	753	1127	843
Recall	0.941373	0.950705	0.951541	0.927473	0.945749
Precision	0.904526	0.903106	0.903127	0.906358	0.904314
F-measure	0.922582	0.926294	0.926702	0.916794	0.924567

4 Most Probable Alternative Pairs (DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	-
reducing alternative pairs	4 most probable alternative pairs
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	538
SVM kernel	polynomial

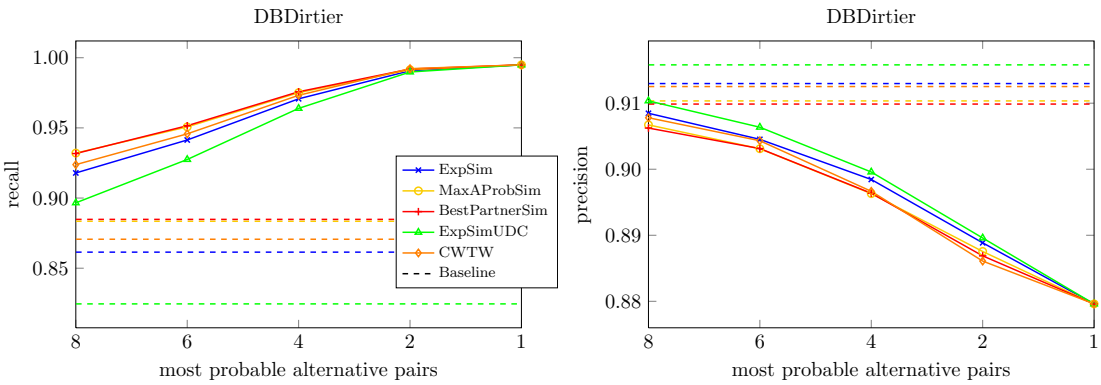
Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15084	15152	15162	14978	15123
FalsePositives	1705	1753	1753	1672	1743
TrueNegatives	102338	102290	102290	102371	102300
FalseNegatives	455	387	377	561	416
Recall	0.970719	0.975095	0.975738	0.963897	0.973229
Precision	0.898445	0.896303	0.896364	0.89958	0.896656
F-measure	0.933185	0.93404	0.934369	0.930628	0.933374

2 Most Probable Alternative Pairs (DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	-
reducing alternative pairs	2 most probable alternative pairs
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	323
SVM kernel	polynomial

Measure	*ExpSim*	*MaxAProbSim*	*BestPartnerSim*	*ExpSimUDC*	*CWTW*
TruePositives	15395	15406	15414	15381	15418
FalsePositives	1925	1952	1966	1909	1982
TrueNegatives	102118	102091	102077	102134	102061
FalseNegatives	144	133	125	158	121
Recall	0.990733	0.991441	0.991956	0.989832	0.992213
Precision	0.888857	0.887545	0.886881	0.889589	0.886092
F-measure	0.937034	0.93662	0.93648	0.937037	0.936155

Using the n Most Probable Alternative Pairs – Comparison (DBDirtier)



C.6. Best Partner Reduction

Best Partner Reduction (DBClean)

configuration	
database	DBClean
reducing alternatives	-
reducing alternative pairs	Best Partner
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	574
SVM kernel	polynomial

Measure	*ExpSim*	*MaxAProbSim*	*BestPartnerSim*	*ExpSimUDC*	*CWTW*
TruePositives	15520	15523	15521	15520	15523
FalsePositives	1963	1975	1970	1957	1977
TrueNegatives	101309	101297	101302	101315	101295
FalseNegatives	19	16	18	19	16
Recall	0.998777	0.99897	0.998842	0.998777	0.99897
Precision	0.887719	0.88713	0.887371	0.888024	0.887029
F-measure	0.939979	0.939734	0.939812	0.94015	0.939677

Best Partner Reduction (DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	-
reducing alternative pairs	Best Partner
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	579
SVM kernel	polynomial

Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15434	15442	15441	15428	15440
FalsePositives	2026	2043	2040	2020	2048
TrueNegatives	102017	102000	102003	102023	101995
FalseNegatives	105	97	98	111	99
Recall	0.993243	0.993758	0.993693	0.992857	0.993629
Precision	0.883963	0.883157	0.883302	0.884227	0.882891
F-measure	0.935422	0.935199	0.935251	0.935399	0.934993

C.7. Representative Comparison Vector

Representative Comparison Vector (DBClean)

configuration	
database	DBClean
reducing alternatives	-
reducing alternative pairs	-
reducing comparison vectors	expected similarity values
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	138
SVM kernel	polynomial

results	
true positives	6731
false positives	202
true negatives	103070
false negatives	8808
recall	0.433168
precision	0.970864
F-measure	0.599057

Representative Comparison Vector (DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	-
reducing alternative pairs	-
reducing comparison vectors	expected similarity values
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	139
SVM kernel	polynomial

results	
true positives	650
false positives	66
true negatives	103977
false negatives	14889
recall	0.04183
precision	0.907821
F-measure	0.079975

C.8. Alternative Confidence Threshold and Best Partner Reduction

Confidence Threshold 0.1 and Best Partner Reduction (DBClean)

configuration	
database	DBClean
reducing alternatives	confidence threshold 0.1
reducing alternative pairs	Best Partner
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	506
SVM kernel	polynomial

Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15514	15517	15517	15513	15518
FalsePositives	1847	1869	1864	1829	1863
TrueNegatives	101425	101403	101408	101443	101409
FalseNegatives	25	22	22	26	21
Recall	0.998391	0.998584	0.998584	0.998327	0.998649
Precision	0.893612	0.8925	0.892756	0.894534	0.892814
F-measure	0.9431	0.942566	0.94271	0.943584	0.94277

Confidence Threshold 0.2 and Best Partner Reduction (DBClean)

configuration	
database	DBClean
reducing alternatives	confidence threshold 0.2
reducing alternative pairs	Best Partner
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	409
SVM kernel	polynomial

Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15515	15520	15518	15515	15521
FalsePositives	1886	1902	1898	1881	1905
TrueNegatives	101386	101370	101374	101391	101367
FalseNegatives	24	19	21	24	18
Recall	0.998455	0.998777	0.998649	0.998455	0.998842
Precision	0.891615	0.890828	0.89102	0.891872	0.890681
F-measure	0.942016	0.941719	0.941769	0.942159	0.941665

Confidence Threshold 0.3 and Best Partner Reduction (DBClean)

configuration	
database	DBClean
reducing alternatives	confidence threshold 0.3
reducing alternative pairs	Best Partner
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	293
SVM kernel	polynomial

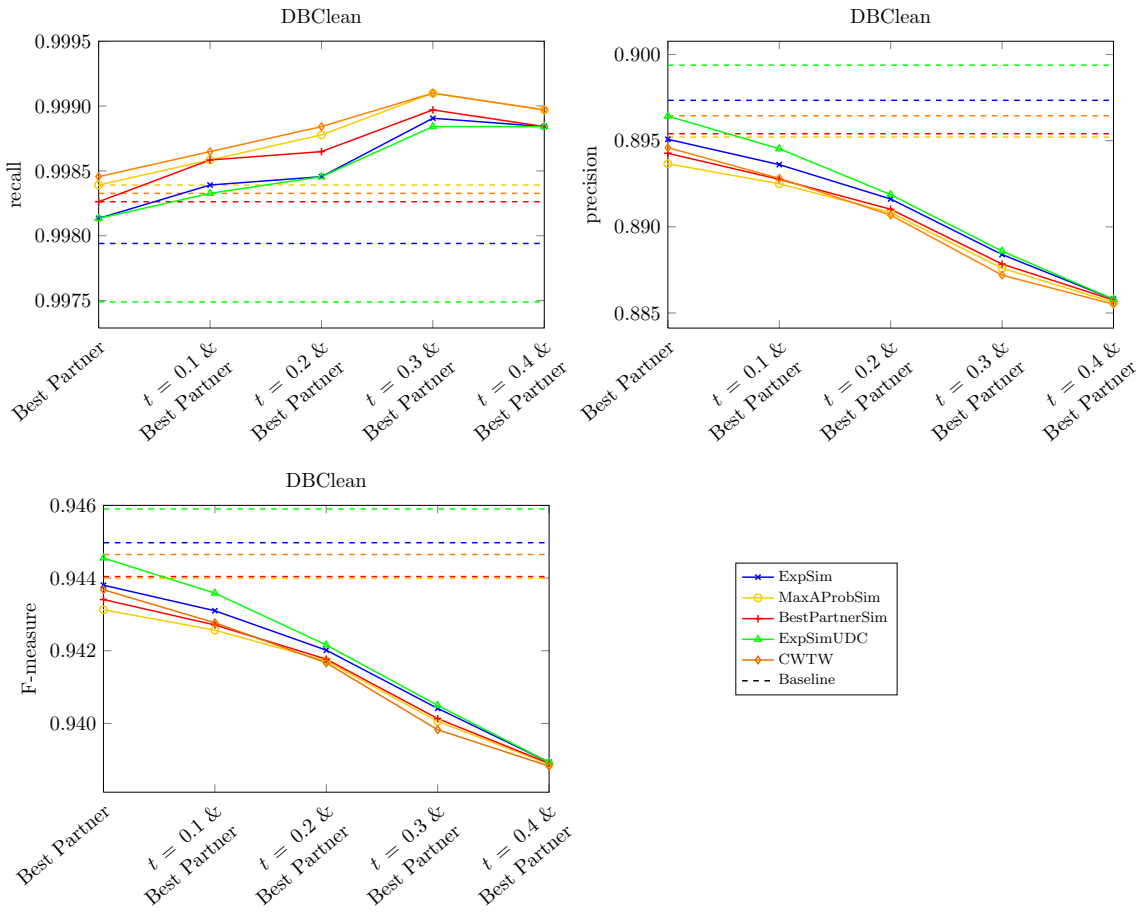
Measure	*ExpSim*	*MaxAProbSim*	*BestPartnerSim*	*ExpSimUDC*	*CWTW*
TruePositives	15522	15525	15523	15521	15525
FalsePositives	1950	1966	1961	1946	1974
TrueNegatives	101322	101306	101311	101326	101298
FalseNegatives	17	14	16	18	14
Recall	0.998906	0.999099	0.99897	0.998842	0.999099
Precision	0.888393	0.887599	0.88784	0.88859	0.887194
F-measure	0.940414	0.940054	0.940133	0.940496	0.939827

Confidence Threshold 0.4 and Best Partner Reduction (DBClean)

configuration	
database	DBClean
reducing alternatives	confidence threshold 0.4
reducing alternative pairs	Best Partner
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	170
SVM kernel	polynomial

Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15521	15523	15521	15521	15523
FalsePositives	2001	2005	2002	2001	2007
TrueNegatives	101271	101267	101270	101271	101265
FalseNegatives	18	16	18	18	16
Recall	0.998842	0.99897	0.998842	0.998842	0.99897
Precision	0.885801	0.885612	0.88575	0.885801	0.885511
F-measure	0.938931	0.938882	0.938903	0.938931	0.938825

Confidence Threshold and Best Partner Reduction – Comparison (DBClean)



Confidence Threshold 0.1 and Best Partner Reduction (DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	confidence threshold 0.1
reducing alternative pairs	Best Partner
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	509
SVM kernel	polynomial

Measure	*ExpSim*	*MaxAProbSim*	*BestPartnerSim*	*ExpSimUDC*	*CWTW*
TruePositives	15203	15236	15240	15145	15231
FalsePositives	1773	1800	1799	1751	1797
TrueNegatives	102270	102243	102244	102292	102246
FalseNegatives	336	303	299	394	308
Recall	0.978377	0.980501	0.980758	0.974644	0.980179
Precision	0.895558	0.894341	0.894419	0.896366	0.894468
F-measure	0.935138	0.935441	0.935601	0.933868	0.935364

Confidence Threshold 0.2 and Best Partner Reduction (DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	confidence threshold 0.2
reducing alternative pairs	Best Partner
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	411
SVM kernel	polynomial

Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15310	15335	15339	15285	15328
FalsePositives	1848	1878	1878	1836	1866
TrueNegatives	102195	102165	102165	102207	102177
FalseNegatives	229	204	200	254	211
Recall	0.985263	0.986872	0.987129	0.983654	0.986421
Precision	0.892295	0.890896	0.890922	0.892763	0.891474
F-measure	0.936477	0.936431	0.936561	0.936007	0.936547

Confidence Threshold 0.3 and Best Partner Reduction (DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	confidence threshold 0.3
reducing alternative pairs	Best Partner
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	295
SVM kernel	polynomial

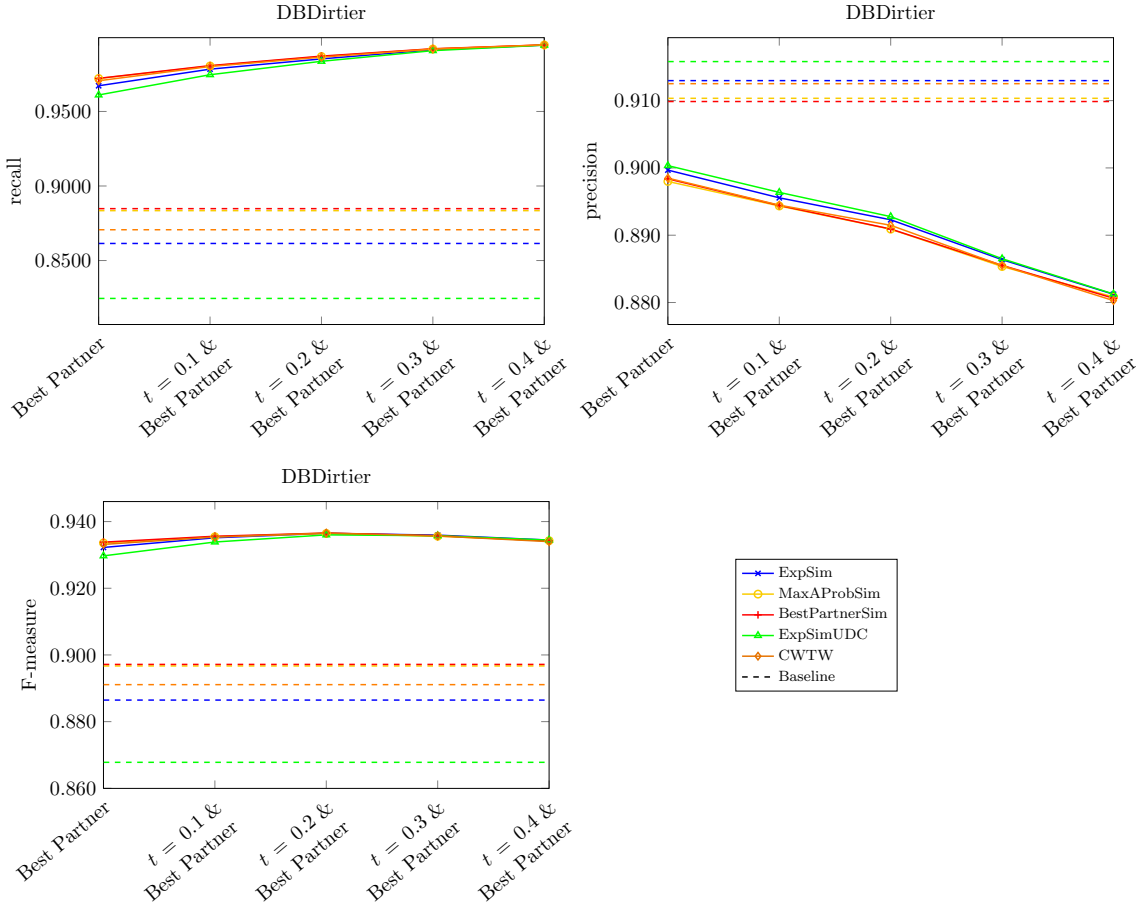
Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15405	15413	15416	15396	15413
FalsePositives	1975	1996	1993	1971	1993
TrueNegatives	102068	102047	102050	102072	102050
FalseNegatives	134	126	123	143	126
Recall	0.991377	0.991891	0.992084	0.990797	0.991891
Precision	0.886364	0.885347	0.885519	0.886509	0.885499
F-measure	0.935934	0.935595	0.935778	0.935756	0.935681

Confidence Threshold 0.4 and Best Partner Reduction (DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	confidence threshold 0.4
reducing alternative pairs	Best Partner
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	171
SVM kernel	polynomial

Measure	*ExpSim*	*MaxAProbSim*	*BestPartnerSim*	*ExpSimUDC*	*CWTW*
TruePositives	15454	15455	15456	15453	15456
FalsePositives	2083	2091	2095	2082	2102
TrueNegatives	101960	101952	101948	101961	101941
FalseNegatives	85	84	83	86	83
Recall	0.99453	0.994594	0.994659	0.994466	0.994659
Precision	0.881223	0.880828	0.880634	0.881266	0.880282
F-measure	0.934454	0.93426	0.93418	0.93445	0.933982

Confidence Threshold and Best Partner Reduction – Comparison (DBDirtier)



C.9. Using the n Most Probable Alternatives and Best Partner Reduction

4 Most Probable Alternatives and Best Partner Reduction (DBClean)

configuration	
database	DBClean
reducing alternatives	4 most probable alternatives
reducing alternative pairs	Best Partner
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	556
SVM kernel	polynomial

Measure	*ExpSim*	*MaxAProbSim*	*BestPartnerSim*	*ExpSimUDC*	*CWTW*
TruePositives	15511	15515	15513	15511	15516
FalsePositives	1826	1855	1845	1805	1841
TrueNegatives	101446	101417	101427	101467	101431
FalseNegatives	28	24	26	28	23
Recall	0.998198	0.998455	0.998327	0.998198	0.99852
Precision	0.894676	0.893207	0.893709	0.895761	0.893933
F-measure	0.943606	0.942903	0.943126	0.944209	0.943337

3 Most Probable Alternatives and Best Partner Reduction (DBClean)

configuration	
database	DBClean
reducing alternatives	3 most probable alternatives
reducing alternative pairs	Best Partner
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	504
SVM kernel	polynomial

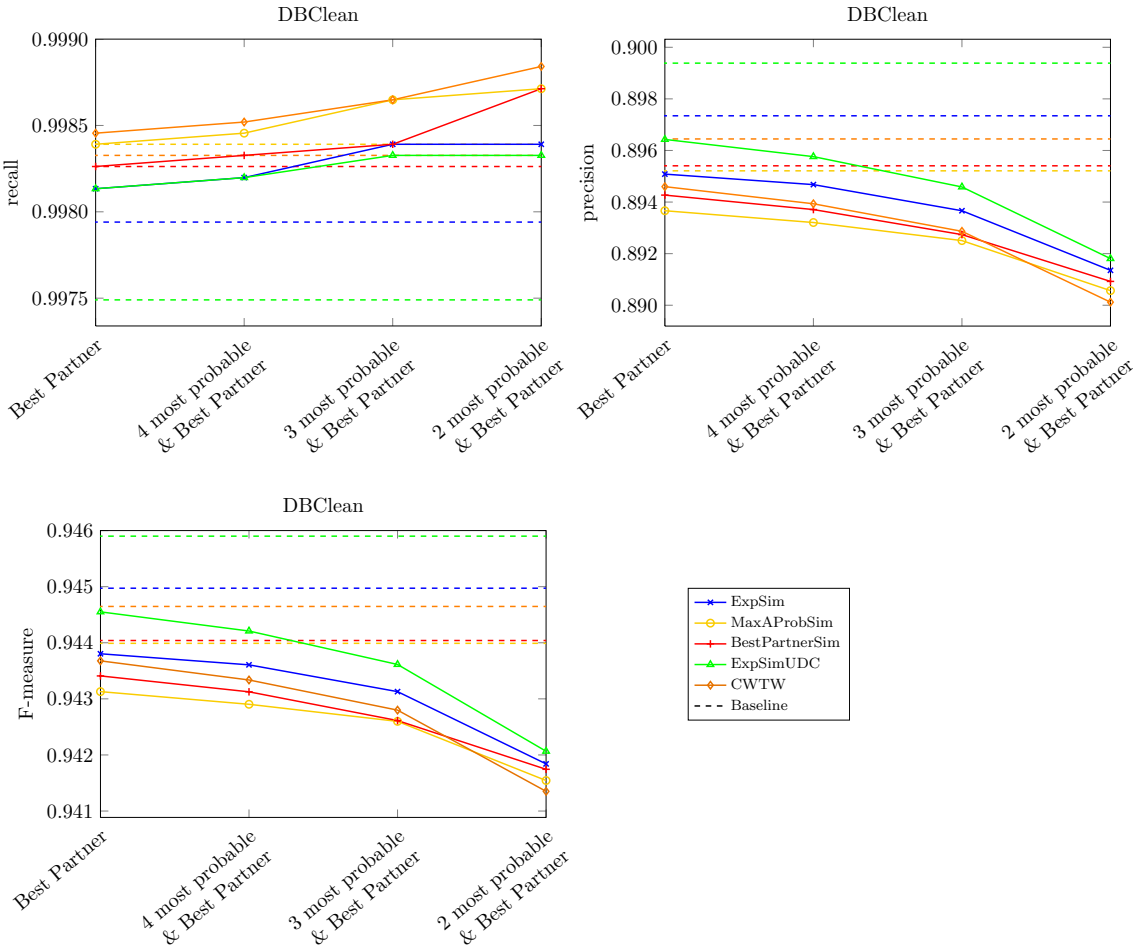
Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15514	15518	15514	15513	15518
FalsePositives	1846	1869	1864	1828	1862
TrueNegatives	101426	101403	101408	101444	101410
FalseNegatives	25	21	25	26	21
Recall	0.998391	0.998649	0.998391	0.998327	0.998649
Precision	0.893664	0.892506	0.892738	0.894585	0.892865
F-measure	0.943129	0.942599	0.942613	0.943613	0.942799

2 Most Probable Alternatives and Best Partner Reduction (DBClean)

configuration	
database	DBClean
reducing alternatives	2 most probable alternatives
reducing alternative pairs	Best Partner
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	385
SVM kernel	polynomial

Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15514	15519	15519	15513	15521
FalsePositives	1891	1907	1900	1882	1916
TrueNegatives	101381	101365	101372	101390	101356
FalseNegatives	25	20	20	26	18
Recall	0.998391	0.998713	0.998713	0.998327	0.998842
Precision	0.891353	0.890566	0.890924	0.891808	0.890119
F-measure	0.941841	0.941544	0.941744	0.942066	0.941351

Most Probable Alternatives and Best Partner Reduction – Comparison (DBClean)



4 Most Probable Alternatives and Best Partner Reduction (DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	4 most probable alternatives
reducing alternative pairs	Best Partner
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	561
SVM kernel	polynomial

Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15102	15161	15164	15012	15141
FalsePositives	1716	1752	1749	1689	1744
TrueNegatives	102327	102291	102294	102354	102299
FalseNegatives	437	378	375	527	398
Recall	0.971877	0.975674	0.975867	0.966085	0.974387
Precision	0.897966	0.896411	0.896588	0.898868	0.896713
F-measure	0.933461	0.934365	0.934549	0.931266	0.933938

3 Most Probable Alternatives and Best Partner Reduction (DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	3 most probable alternatives
reducing alternative pairs	Best Partner
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	508
SVM kernel	polynomial

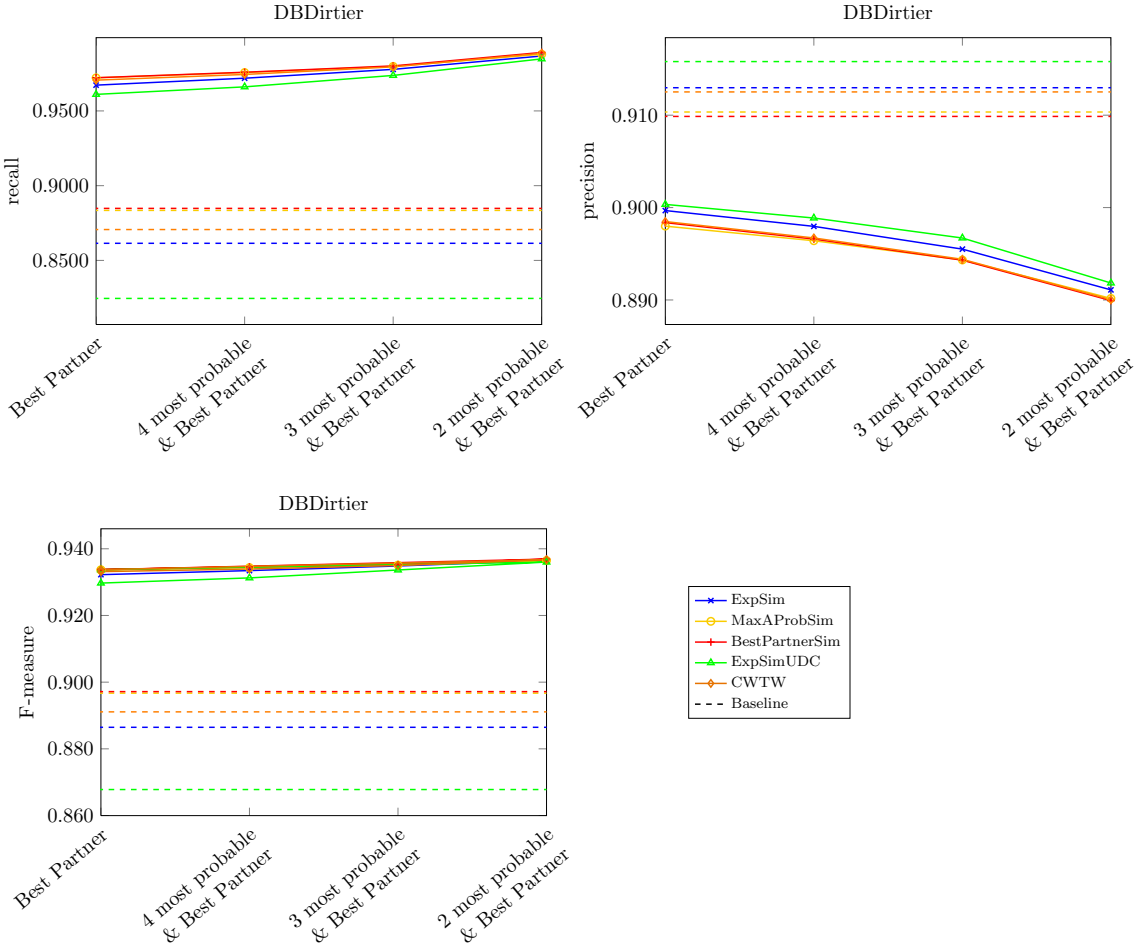
Measure	ExpSim	MaxAProbSim	BestPartnerSim	ExpSimUDC	CWTW
TruePositives	15194	15225	15230	15131	15220
FalsePositives	1773	1799	1800	1743	1797
TrueNegatives	102270	102244	102243	102300	102246
FalseNegatives	345	314	309	408	319
Recall	0.977798	0.979793	0.980115	0.973743	0.979471
Precision	0.895503	0.894326	0.894304	0.896705	0.8944
F-measure	0.934843	0.93511	0.935245	0.933638	0.935004

2 Most Probable Alternatives and Best Partner Reduction (DBDirtier)

configuration	
database	DBDirtier
reducing alternatives	2 most probable alternatives
reducing alternative pairs	Best Partner
reducing comparison vectors	-
similarity metric	SoftTFIDF – JaroWinkler
trainings vector selection	nearest-based
initial match vectors	100
initial unmatch vectors	387
SVM kernel	polynomial

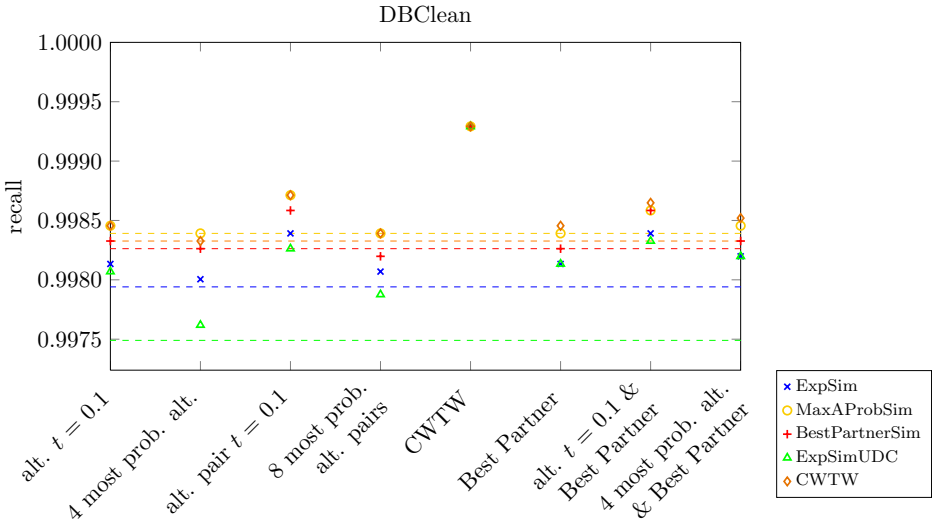
Measure	*ExpSim*	*MaxAProbSim*	*BestPartnerSim*	*ExpSimUDC*	*CWTW*
TruePositives	15333	15351	15370	15303	15361
FalsePositives	1874	1894	1901	1856	1898
TrueNegatives	102169	102149	102142	102187	102145
FalseNegatives	206	188	169	236	178
Recall	0.986743	0.987901	0.989124	0.984812	0.988545
Precision	0.891091	0.890171	0.889931	0.891835	0.890028
F-measure	0.936481	0.936493	0.936909	0.936021	0.936703

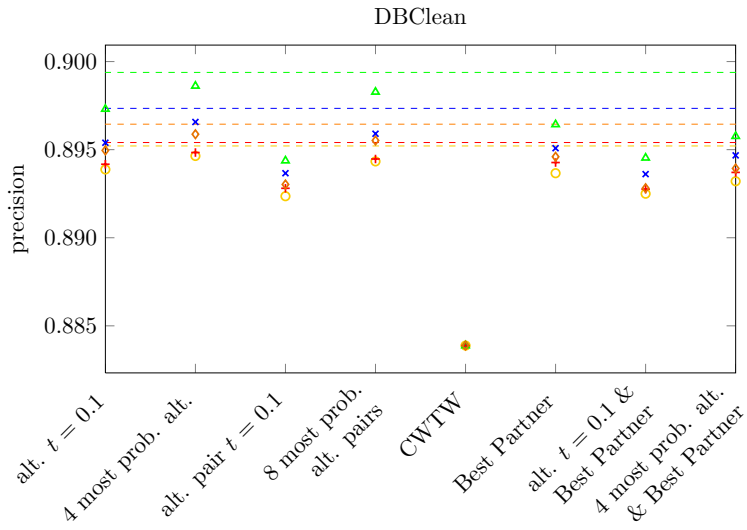
Most Probable Alternatives and Best Partner Reduction – Comparison (DBDirtier)



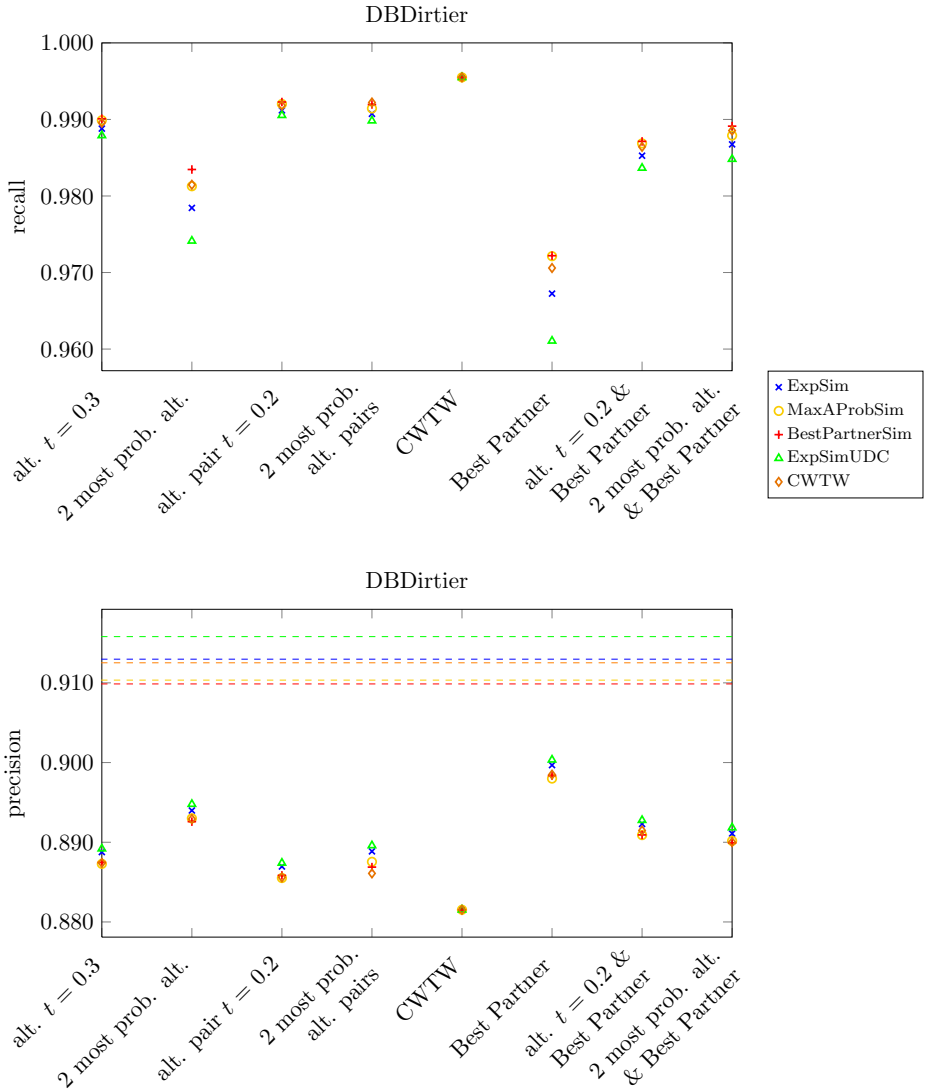
C.10. Final Evaluation

DBClean





DBDirtier


DBDirtier
recall
1.000
0.990
0.980
0.970
0.960
alt. $t = 0.3$
2 most prob. alt.
alt. pair $t = 0.2$
2 most prob. alt. pairs
CWTW
Best Partner
alt. $t = 0.2$ & Best Partner
2 most prob. alt. & Best Partner
ExpSim
MaxAProbSim
BestPartnerSim
ExpSimUDC
CWTW
DBDirtier
precision
0.910
0.900
0.890
0.880
alt. $t = 0.3$
2 most prob. alt.
alt. pair $t = 0.2$
2 most prob. alt. pairs
CWTW
Best Partner
alt. $t = 0.2$ & Best Partner
2 most prob. alt. & Best Partner

Bibliography

[ACG02] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 586–597, 2002.

[BDM+05] Jihad Boulos, Nilesh Dalvi, Bhushan Mandhani, Shobhit Mathur, Chris Re, and Dan Suciu. Mystiq: a system for finding more answers by using probabilities. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, SIGMOD '05, pages 891–893, New York, NY, USA, 2005. ACM.

[BM03] Mikhail Bilenko and Raymond J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Knowledge Discovery and Data Mining*, pages 39–48, 2003.

[Bre84] L. Breiman. *Classification and regression trees*. Chapman & Hall/CRC, 1984.

[BS12] Yukino Baba and Hisami Suzuki. How are spelling errors generated and corrected? a study of corrected and uncorrected spelling errors using keystroke logs. In *ACL (2)*, pages 373–377, 2012.

[BSHW06] Omar Benjelloun, Anish Das Sarma, Alon Halevy, and Jennifer Widom. Uldbs: Databases with uncertainty and lineage. In *VLDB*, pages 953–964. ACM, 2006.

[BSS03] Paola Bertolazzi, Luca De Santis, and Monica Scannapieco. Automatic record matching in cooperative information systems. *Proceedings of the ICDT*, (i), 2003.

[CCM+10] Nicoletta Calzolari, Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner, and Daniel Tapias, editors. *Proceedings of the International Conference on Language Resources and Evaluation, LREC 2010, 17-23 May 2010, Valletta, Malta*. European Language Resources Association, 2010.

[CG91] Kenneth W. Church and William A. Gale. Probability scoring for spelling correction. *Statistics and Computing*, 1(2):93–103, December 1991.

[CH90] J. B. Copas and F. J. Hilton. Record linkage: statistical models for matching computer records. *Journal of the Royal Statistical Society. Series A (Statistics in Society)*, 153(3):287–320, 1990.

[Chr05] Peter Christen. Probabilistic data generation for deduplication and data linkage. *Intelligent Data Engineering and Automated Learning-IDEAL 2005*, pages 109–116, 2005.

[Chr08a] Peter Christen. Automatic record linkage using seeded nearest neighbour and support vector machine classification. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 151–159, New York, NY, USA, 2008. ACM.

[Chr08b] Peter Christen. Febrl: an open source data cleaning, deduplication and record linkage system with a graphical user interface. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 1065–1068, Las Vegas, Nevada, USA, 2008. ACM. ACM ID: 1402020.

[Chr12] Peter Christen. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection.* Data-Centric Systems and Applications. Springer, 2012.

[CKLS01] Munir Cochinwala, Verghese Kurien, Gail Lalk, and Dennis Shasha. Efficient data reconciliation. *Information Sciences*, 137(1-4):1–15, September 2001.

[Coh00] William W. Cohen. Data integration using similarity joins and a word-based information representation language. *ACM Transactions on Information Systems (TOIS)*, 18(3):288–321, 2000.

[CP09] Peter Christen and Agus Pudjijono. Accurate synthetic generation of realistic personal information. In *Proceedings of the 13th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, PAKDD '09, pages 507–514, Berlin, Heidelberg, 2009. Springer-Verlag.

[CRF03] William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. A comparison of string distance metrics for name-matching tasks. pages 73–78, 2003.

[Dam64] Fred J. Damerau. A technique for computer detection and correction of spelling errors. *Commun. ACM*, 7(3):171–176, March 1964.

[DSD98] D. Dey, S. Sarkar, and P. De. Entity matching in heterogeneous databases: a distance-based decision model. In *System Sciences, 1998., Proceedings of the Thirty-First Hawaii International Conference on*, volume 7, pages 305–313, 1998.

[EIV07] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Trans. on Knowl. and Data Eng.*, 19:1–16, January 2007.

[EVE02] M. G. Elfeky, Vassilios S. Verykios, and A. K. Elmagarmid. TAILOR: a record linkage toolbox. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 17–28, 2002.

[FS69] Ivan Peter Fellegi and Alan Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64:1183–1210, 1969.

[FW10] Steffen Friedrich and Wolfram Wingerath. Search-space reduction techniques for duplicate detection in probabilistic data. Bachelor's thesis, Universität Hamburg, 2010.

[GB06] Lifang Gu and Rohan Baxter. Decision models for record linkage. In Graham Williams and Simeon Simoff, editors, *Data Mining*, volume 3755 of *Lecture Notes in Computer Science*, pages 146–160. Springer Berlin / Heidelberg, 2006.

[Gil01] L. Gill. *Methods for automatic record matching and linkage and their use in national statistics*. Office for National Statistics, 2001.

[GKMS04] S. Guha, N. Koudas, A. Marathe, and D. Srivastava. Merging the results of approximate match operations. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 636–647, 2004.

[HMMS12] Razia Haider, Federica Mandreoli, Riccardo Martoglia, and Simona Sassatelli. Fast on-line summarization of rfid probabilistic data streams. In Sumeet Dua, Aryya Gangopadhyay, Parimala Thulasiraman, Umberto Straccia, Michael A. Shepherd, and Benno Stein, editors, *ICISTM*, volume 285 of *Communications in Computer and Information Science*, pages 211–223. Springer, 2012.

[HS95] Mauricio A. Hernández and Salvatore J. Stolfo. The merge/purge problem for large databases. In *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 127–138, New York, NY, USA, 1995. ACM.

[HS98] Mauricio A Hernández and Salvatore J Stolfo. Real-world data is dirty: Data cleansing and the Merge/Purge problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998. 10.1023/A:1009761603038.

[HW79] J. A. Hartigan and M. A. Wong. A k-means clustering algorithm. *Journal of the Royal Statistical Society C*, 28(1):100–108, 1979.

[Jar95] Matthew A Jaro. Probabilistic linkage of large public health data files. *Statistics in Medicine*, 14(5-7):491–498, March 1995.

[KBS08] Nodira Khoussainova, Magdalena Balazinska, and Dan Suciu. Probabilistic event extraction from rfid data. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE '08, pages 1480–1482, Washington, DC, USA, 2008. IEEE Computer Society.

[Koc08] Christoph Koch. Maybms: A system for managing large uncertain and probabilistic databases. In *Managing and Mining Uncertain Data, chapter 6.* Springer-Verlag, 2008.

[LLL01] Wai Lup Low, Mong Li Lee, and Tok Wang Ling. A knowledge-based approach for duplicate elimination in data cleaning. *Information Systems*, 26(8):585–606, 2001.

[ME96] A. E Monge and C. Elkan. The field matching problem: Algorithms and applications. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 267–270, 1996.

[MR10] Muhammad Muzammal and Rajeev Raman. On probabilistic models for uncertain sequential pattern mining. In *ADMA (1)*, pages 60–72, 2010.

[NKAJ59] H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James. Automatic linkage of vital records. *Science*, 130:954–959, October 1959.

[Pet86] James L. Peterson. A note on undetected typing errors. *Commun. ACM*, 29(7):633–637, July 1986.

[PvKdKR09] Fabian Panse, Maurice van Keulen, Ander de Keijzer, and Norbert Ritter. Duplicate detection in probabilistic data. In *Proceedings of the 2nd International Workshop on New Trends in Information Integration (NTII 2010)*, number TR-CTIT-09-44 in CTIT technical report series, Enschede, December 2009. Centre for Telematics and Information Technology, University of Twente. Extended version of NTII2010 workshop paper.

[PZ84] Joseph J. Pollock and Antonio Zamora. Automatic spelling correction in scientific and scholarly text. *Commun. ACM*, 27(4):358–368, April 1984.

[Qui83] J. R Quinlan. Learning efficient classification procedures and their application to chess end games. *Machine learning: An artificial intelligence approach*, 1:463–482, 1983.

[Qui93] J. R Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, 1993.

[Rus18] R.C. Russell. a method of phonetic indexing. *U.S. Patent 1,261,167*, 1918.

[Rus22] R.C. Russell. a method of phonetic indexing. *U.S. Patent 1,435,663*, 1922.

[SCH09] Dan Suciu, Andrew Connolly, and Bill Howe. Embracing uncertainty in large-scale computational astrophysics. *MUD*, pages 63–67, 2009.

[SM64] J. A Sonquist and J. N Morgan. *The detection of interaction effects*. 1964.

[SS02] Bernhard Schölkopf and Alexander J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. Adaptive computation and machine learning. MIT Press, 2002.

[Ste09] Dan Steinberg. *The Top Ten Algorithms in Data Mining*, pages 179–201. Chapman & Hall/CRC Data Mining and Knowledge Discovery Series. Chapman and Hall/CRC, 2009.

[Taf70] R. L. Taft. *Name search techniques*. Number 1. Bureau of Systems Development, New York State Identification and Intelligence System, 1970.

[TZS09] John R. Talburt, Yinle Zhou, and Savitha Yalanadu Shivaiah. Sog: A synthetic occupancy generator to support entity resolution instruction and research. In *ICIQ'09*, pages 91–105, 2009.

[VC74] V. N. Vapnik and A. Ya Chervonenkis. Theory of pattern recognition. 1974.

[VEH00] V. S. Verykios, A. K. Elmagarmid, and E. N. Houstis. Automating the approximate record-matching process. *Information sciences*, 126(1-4):83–98, 2000.

[vKdK09] Maurice van Keulen and Ander de Keijzer. Qualitative effects of knowledge rules and user feedback in probabilistic data integration. *The VLDB Journal*, July 2009.

[VME03] V. S. Verykios, G. V. Moustakides, and M. G. Elfeky. A bayesian decision model for cost optimal record matching. *The VLDB Journal*, 12:28–40, 2003. 10.1007/s00778-002-0072-y.

[Wid08] Jennifer Widom. Trio: A system for data, uncertainty, and lineage. In *Managing and Mining Uncertain Data*. Springer, 2008.

[Win88] William E. Winkler. Using the em algorithm for weight computation in the fellegi-sunter model of record linkage. *American Statistical Association, Proceedings of the Section on Survey Research Methods*, pages 667–671, 1988.

[Win99] William E. Winkler. The state of record linkage and current research problems. In *Statistical Research Division, US Census Bureau*, 1999.

[Win00] William E. Winkler. Using the em algorithm for weight computation in the fellegi-sunter model of record linkage. In *Proceedings of the Section on Survey Research Methods, American Statistical Association*, pages 667–671, 2000.

[Win02] William E. Winkler. Methods for record linkage and bayesian networks. Technical report, Series RRS2002/05, U.S. Bureau of the Census, 2002.

[WM89] J. R. Wang and S. E Madnick. The inter-database instance identification problem in integrating autonomous systems. In *Data Engineering, 1989. Proceedings. Fifth International Conference on*, pages 46–55, 1989.

List of Figures

List of Tables

Listings

Statutory Declaration

We hereby declare that we have developed and written the enclosed thesis entirely by ourselves and have not used sources or means without declaration in the text. Any thoughts or quotations which were inferred from these sources are clearly marked as such. Individual responsibilities for certain sections are as follows:

chapter	Steffen Friedrich	together	Wolfram Wingerath
1		entirely	
2	2.2.3–2.2.7		2.1–2.2.2, 2.3, 2.4
3		3.3	3.1, 3.2
4	entirely		
5		entirely	
6		entirely	

This thesis was not submitted in the same or in a substantially similar version, not even partially, to any other authority to achieve an academic grading and was not published elsewhere.

We agree that a copy of this thesis may be made available in the Informatics Library of the Universität Hamburg.

Steffen Friedrich Wolfram Wingerath

Hamburg, November 30$^{\text{th}}$, 2012