KEY-BASED BLOCKING OF DUPLICATES IN ENTITY-INDEPENDENT PROBABILISTIC DATA

(Research-in-Progress)

Fabian Panse
University of Hamburg, Germany
panse@informatik.uni-hamburg.de

Wolfram Wingerath
University of Hamburg, Germany
wingerath@informatik.uni-hamburg.de

Steffen Friedrich
University of Hamburg, Germany
friedrich@informatik.uni-hamburg.de

Norbert Ritter
University of Hamburg, Germany
ritter@informatik.uni-hamburg.de

Abstract: Currently, in many application areas the demand on probabilistic data grows. Duplicate entity representations are an essential problem of data quality, for certain databases as well as for probabilistic databases. Traditional duplicate detection approaches are based on pairwise comparisons. For dealing with large data sets, however, a comparison of all entity representation pairs is impractical and the search space is usually reduced by blocking techniques. The majority of blocking techniques is based on the usage of keys created from the original representations. These techniques, however, are only designed to deal with certain keys and hence cannot be used for probabilistic data without any adaptation. In this paper, we propose an adaptation of existing blocking techniques to data uncertainty based on the creation of certain keys from the probabilistic data. Moreover, we discuss some approaches for adapting the techniques' core functionalities to handle probabilistic keys. A final set of experiments evaluates the quality of our certain key based approaches in terms of pairs completeness and pairs quality.

Key Words: Probabilistic Data, Duplicate Detection, Blocking, Sorted Neighborhood Method

1. INTRODUCTION

Today, a large amount of real-life applications [1] [2] naturally produce uncertain, imprecise or vague information. For accurately storing such imperfect information probabilistic databases [3] [4] [5] have been developed. For meaningfully integrating probabilistic data originating from different sources or for cleaning a single probabilistic database, duplicate entity representations[1] need to be identified. Techniques for duplicate detection are usually based on pairwise comparisons of entity representations [6] [7]. However, for detecting duplicates in large data sets, a pairwise comparison of all representations is by far too expensive in storage as well as in time. Instead the search space has to be initially reduced to a manageable size by the usage of blocking techniques [8] [9] (also known as indexing [10]) as for example the Sorted Neighborhood Method [11]. The most of these blocking techniques are based on the usage of key values which are generated by the entity representations' data (in the following, we use the words 'key value' and 'key' synonymously). In probabilistic entity representations, however, the data used for key value creation can be uncertain. Thus, from applying a traditional key definition function probabilistic keys, i.e. keys with multiple possible instances, can result. Existing blocking variants are not designed to deal with probabilistic keys and hence cannot be used for probabilistic data without any adaptation.

[1] In certain relational data, an entity representation corresponds to an ordinary tuple, but in probabilistic data an entity is usually represented by more complex constructs as x-tuples [5] or tuple-blocks [3].

In this paper, we consider an adaptation of existing blocking techniques to the uncertainty and impreciseness modeled in probabilistic data in two ways. First, by resolving uncertainty during key value creation, i.e. by applying methods for creating certain keys from probabilistic entity representations, and second by adapting the core functionality of the blocking technique to probabilistic keys. The advantage of creating certain keys is that the core functionality of the blocking technique remains substantially unchanged and blocking can be applied as usual. In contrast, an adaptation to probabilistic keys implies a reimplementation of the whole blocking technique and hence adaptations to one blocking technique cannot be simply adopted to other ones. Due to a variety of probabilistic data applications restrict themselves to the usage of entity-independent data models, i.e. representation systems in which the uncertainties of different entity representations are not correlated, as BID-tables [3] or ULDBs [5] without lineage, we restrict ourselves to this class of probabilistic data models as well. A consideration of probabilistic data with entity dependencies is planned for future research.

The main contributions of this paper are:

- Strategies to adapt existing blocking techniques to entity-independent probabilistic data by creating certain keys from the uncertain (probabilistic) data,
- First discussions on adapting the core functionality of the Sorted Neighborhood Method to probabilistic keys,
- An exhaustive experimental evaluation on the effectiveness and the accuracy of the proposed adaptations.

1.1. Motivating Example

As a motivating example, we consider the probabilistic entity representations of the three Movies 1-3 presented in Figure 1. Assume that the key of each movie is generated by concatenating the first three characters of its title and the last two digits of its production year. The title and the production year of Movie 1 are certain values and hence creating a certain key does not pose a problem. Although the title of Movie 2 is uncertain, for each of its possible instances the same key result, i.e. ’Bat01’. In contrast, the title’s three first characters of Movie 3 are either ’Bat’ or ’Ret’ and hence are uncertain. A simple idea to solve this problem is to create a single certain key for each movie, but it is not clear which certain key represents Movie 3 at best. One intuitive solution is to take the key of the movie’s most probable instance, which is ’Ret95’. Nevertheless, to take the key which is most probable at all (in this case ’Bat95’) is maybe more appropriate. Another option is to represent Movie 3 by multiple certain keys, i.e. both ’Ret95’ and ’Bat95’. We also could initially create a probabilistic key, but then the retained uncertainty has to be resolved during the remaining steps of the considered blocking technique. In summary, there are a lot of potentialities for handling this problem, but it is unclear which of them solves the problem at best.

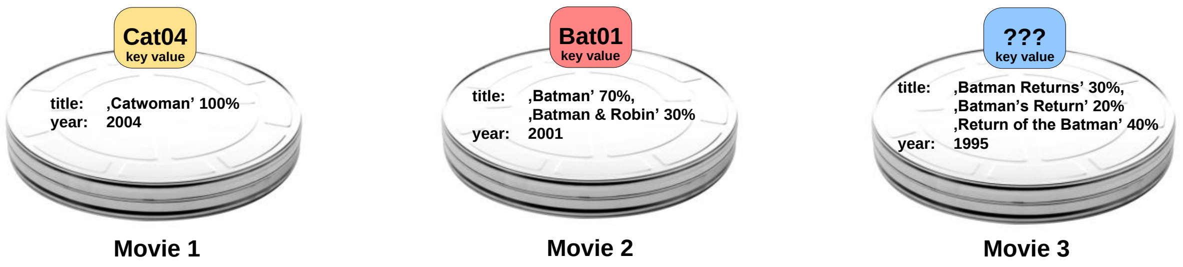

Cat04
key value
title: ‚Catwoman’ 100%
year: 2004
Movie 1
Bat01
key value
title: ‚Batman’ 70%, ‚Batman & Robin’ 30%
year: 2001
Movie 2
???
key value
title: ‚Batman Returns’ 30%, ‚Batman’s Return’ 20% ‚Return of the Batman’ 40%
year: 1995
Movie 3

Figure 1: Probabilistic entity representations of three sample movies containing uncertain information

1.2. Outline

The paper is structured as follows: We start with some basics on probabilistic data, duplicate detection, and existing blocking techniques in Section 2. Then we present our strategies to adapt blocking to probabilistic data in Section 3. First we discuss some approaches based on certain keys (Section 3.1). In Section 3.2 we then propose some adaptations of the Sorted Neighborhood Method to probabilistic keys. We evaluate our newly defined strategies experimentally in Section 4. Finally, we examine related work in Section 5. Section 6 concludes the paper.

2. Basics

In this section we give a short overview on probabilistic data and introduce some basics on duplicate detection and search space reduction (blocking). Moreover, we will go into detail with the Sorted Neighborhood Method which we will use as a blocking technique representative throughout this paper.

2.1. Probabilistic Data

A probabilistic relational database is defined on an ordinary relational database schema. According to the possible world semantics [12] the instantiation of a probabilistic database is theoretically defined as $PDB = (\mathbf{W},\mathbf{P})$ where $\mathbf{W} = \{W_1, \dots, W_n\}$ is a finite set of possible instances of this database (also called as possible worlds) and $\mathbf{P}{:}\mathbf{W} \rightarrow (0,1]$, $\sum_{W \in \mathbf{W}} \mathbf{P}(W) = 1$ is the probability distribution over these instances.

	title	year	studio	p	
t_1	Batman	1989	20th Century Fox	0.56	?
	Batman & Robin	1997	Columbia Pictures	0.34	
t_2	Catwoman	2004	Warner Bros.	1.00	
t_3	Return of Batman	1995	20th Century Fox	0.35	
	Catman	1995	Republic Pictures	0.25	
	Batman Returns	1992	Warner Bros.	0.20	
	Batman Returns	1992	Republic Pictures	0.20	

Possible World	***Probability***
$W_1 = \{t_{2,1}, t_{3,1}\}$	$P(W_1) = 0.1 \times 0.35 = 0.035$
$W_2 = \{t_{1,1}, t_{2,1}, t_{3,1}\}$	$P(W_2) = 0.56 \times 0.35 = 0.196$
$W_3 = \{t_{1,2}, t_{2,1}, t_{3,1}\}$	$P(W_3) = 0.34 \times 0.35 = 0.119$
$W_4 = \{t_{2,1}, t_{3,2}\}$	$P(W_4) = 0.1 \times 0.25 = 0.025$
$W_5 = \{t_{1,1}, t_{2,1}, t_{3,2}\}$	$P(W_5) = 0.56 \times 0.25 = 0.14$
$W_6 = \{t_{1,2}, t_{2,1}, t_{3,2}\}$	$P(W_6) = 0.34 \times 0.25 = 0.085$
$W_7 = \{t_{2,1}, t_{3,3}\}$	$P(W_7) = 0.1 \times 0.2 = 0.02$
$W_8 = \{t_{1,1}, t_{2,1}, t_{3,3}\}$	$P(W_8) = 0.56 \times 0.2 = 0.112$
$W_9 = \{t_{1,2}, t_{2,1}, t_{3,3}\}$	$P(W_9) = 0.34 \times 0.2 = 0.068$
$W_{10} = \{t_{2,1}, t_{3,4}\}$	$P(W_{10}) = 0.1 \times 0.2 = 0.02$
$W_{11} = \{t_{1,1}, t_{2,1}, t_{3,4}\}$	$P(W_{11}) = 0.56 \times 0.2 = 0.112$
$W_{12} = \{t_{1,2}, t_{2,1}, t_{3,4}\}$	$P(W_{12}) = 0.34 \times 0.2 = 0.068$

Figure 2: Sample x-relation (left) and its corresponding set of possible worlds (right)

Entity-independent probabilistic data models are specific probabilistic representation systems that restrict the possible world space to databases in which the instance and existence of one entity representation is independent from the instance and existence of any other entity representation. Although entity-independent probabilistic data models are no complete representation systems, i.e. there are sets of possible worlds which cannot be represented by such a data model; they are commonly used, because they are easier to manage than a complete one.

The simplest entity-independent probabilistic data model is a tuple-independent probabilistic data model [25] in which each entity is represented by a single tuple that is assigned with a probability score. In this representation system only the uncertainty on an entity's existence can be modeled in the probabilistic data. In this paper, we focus on entity-independent probabilistic data models that also allow a representation of the uncertainty on the entities' instantiations as ULDBs [5] and BID-tables [3] in which an entity is represented by an x-tuple or a block of disjoint tuples respectively.

Without any loss of generality, we use the ULDB model as a representative throughout this paper. The ULDB model [5] based on the x-tuple concept. Each x-tuple consists of a set of mutually exclusive alternatives each defined as a certain tuple which is assigned with a confidence score (attribute p). In the following, the set of alternatives (possible instances) of an x-tuple t is denoted as $pI(t)$. Moreover, the j^{th} alternative of an x-tuple t_i can be expressed by the form $t_{i,j}$. Maybe x-tuples (tuples for which non-existence is possible, i.e., for which the sum of its alternatives' probabilities is smaller than 1) are indicated by '?'. In the ULDB model different interpretations of the confidence values exist [5]. In our work we focus on probabilistic data, therefore, we always interpret confidence as probability. Relations containing one or more x-tuples are called x-relations. A sample movie x-relation with three x-tuples along with its possible world space is shown in Figure 2. Since x-tuple t_1 is a maybe tuple with two alternatives and x-tuple t_3 is a non-maybe tuple with four alternatives, the movie x-relation represents a set of twelve possible worlds.

When clear from the context, we sometimes simply use 'tuple' to refer to x-tuples (and hence entity representations in general) and 'relation' to refer to x-relations.

2.2. Duplicate Detection

Duplicate detection means the problem of identifying multiple entity representations that refer to the same real-world entities. The most approaches for duplicate detection are based on pairwise entity representation comparisons [7] [6] [13] [14]. Such approaches can be conceptually decomposed into four phases [6]:

1. **Search Space Reduction:** Since a comparison of all pairs of tuples is mostly too inefficient, the search space is usually reduced using heuristic blocking techniques (see Section 2.2.1).
2. **Attribute Value Matching:** Similarity of tuples is usually based on the similarity of their corresponding attribute values. Despite data preparation, syntactic as well as semantic irregularities remain. Thus, attribute value similarity is quantified by syntactic and semantic means [6]. From comparing two tuples, we obtain a comparison vector $\vec{c} = \langle c_1, \ldots, c_n \rangle$, where c_i represents the value similarity of the *i*th attribute.
3. **Decision Model:** The comparison vector is input to a decision model [13] which determines which set a tuple pair (t_1, t_2) is assigned to: matching tuples (M) or unmatching tuples (U).
4. **Duplicate Clustering:** Decision models only made decisions for single tuple pairs. To get a globally consistent result a clustering technique [6] needs to be applied.

For delimiting from the cheap comparison methods done by search space reduction techniques described later, we call the combined execution of the attribute value matching and the decision model as an in-depth comparison. We proposed methods for in-depth comparisons of x-tuples in [15].

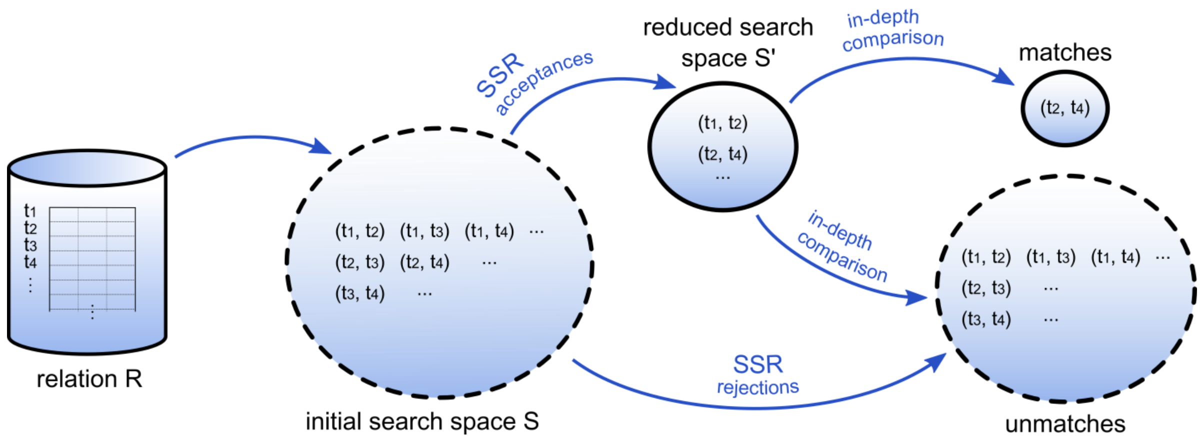


Figure 3: The principal functionality of a search space reduction for duplicate detection. The dashed boundaries of the initial search space and of the set of unmatches indicate that these sets are never materialized.

2.2.1. Search Space Reduction

Without reduction, the search space of a duplicate detection on an input relation $R = \{t_1, t_2, \ldots, t_n\}$ based on pairwise comparisons is principally the set of all possible pairs of tuples belonging to R (see Figure 3):

$$S = \{(t_i, t_j) \mid t_i, t_j \in R \ \wedge i < j\}$$

Since two tuples only need to be compared once and a tuple does not need to be compared with itself, the initial search space consists of $\frac{|R|\times(|R|-1)}{2} = \frac{n^2-n}{2}$ tuple pairs (complexity $O(n^2)$). In large data sets with millions or more tuples, the number of tuple pairs to be compared explodes and hence the duplicate detection process becomes infeasible. For that reason, the search space has to be initially reduced before comparing tuples in-depth. Reduction is realized by rejecting pairs of tuples being no duplicates for sure and adding them to the set of unmatches.

2.2.2. Evaluation Measures

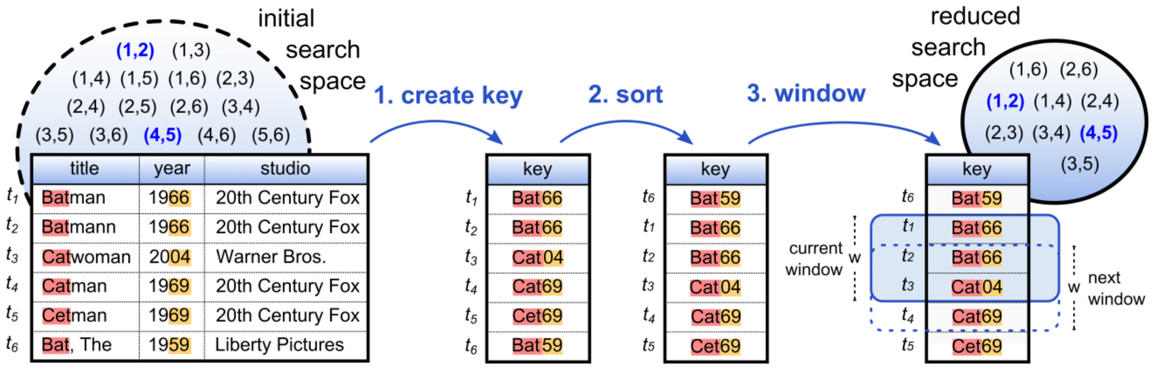
Blocking is effective, if the number of rejected tuple pairs is high. Nevertheless, it is only accurate, if no true duplicate pair is rejected. In general, blocking is based on cheap comparisons and hence is known to cause two kinds of errors: false acceptance (short FA), i.e. leaving an actual unmatch in the search space, and – even worse – false rejection (short FR), i.e. removing an actual match from the search space by assigning it to the set of unmatches.
False rejection is worse than false acceptance, because an actual match that is rejected is not considered again and therefore changes the duplicate detection result for the worse, whereas a false acceptance is eventually corrected during the in-depth comparison.
To score accuracy and effectiveness we use the two measures pairs completeness (PC) and pairs quality (PQ) as proposed by Christen [10]. Pairs completeness represents the share of true acceptance (short TA) in all duplicate pairs ($TA \cup FR$), and pairs quality represents the share of true acceptances in the accepted tuples pairs ($TA \cup FA$):

$$PC = \frac{|TA|}{|TA| + |FR|} \qquad\qquad PQ = \frac{|TA|}{|TA| + |FA|}$$

Note, compared to the pairs quality (also known as precision) achieved by in-depth comparison, a pairs quality of around 0.02 usually resulting from blocking is rather low, but compared to the pairs quality of the initial search space ($\simeq 2 \times 10^{-6}$), the percentage of increase is really high.

2.3. Existing Blocking Techniques for Certain Data

Currently several blocking techniques have been proposed (see [10] for a survey). The most of these techniques based on the usage of key values. The goal of this paper is not to present adaptations to probabilistic data for all of the key-based blocking techniques, but rather to point out different approaches for adaptation and to compare them with each other. In this paper, we consider three blocking techniques. We use the Sorted-Neighborhood Method (short SNM), which is a state-of-the-art blocking technique, to illustrate our adaptation strategies based on certain keys and discuss ways to adapt the SNM to probabilistic keys in Section 3. In our experiments in Section 4, we additionally use Standard Blocking [8] [14] (short SB) and Robust Suffix-Array Blocking [16] (short SAB).



	title	year	studio
t_1	Batman	1966	20th Century Fox
t_2	Batmann	1966	20th Century Fox
t_3	Catwoman	2004	Warner Bros.
t_4	Catman	1969	20th Century Fox
t_5	Cetman	1969	20th Century Fox
t_6	Bat, The	1959	Liberty Pictures

	key
t_1	Bat66
t_2	Bat66
t_3	Cat04
t_4	Cat69
t_5	Cet69
t_6	Bat59

	key
t_6	Bat59
t_1	Bat66
t_2	Bat66
t_3	Cat04
t_4	Cat69
t_5	Cet69

	key
t_6	Bat59
t_1	Bat66
t_2	Bat66
t_3	Cat04
t_4	Cat69
t_5	Cet69

Figure 4: The three steps of the Sorted Neighborhood Method

2.3.1. The Sorted Neighborhood Method

The standard variant of the SNM [11] reduces the search space in three steps (for illustration see Figure 4):

1. **Key Creation:** First, for each tuple t a key $\kappa(t)$ is computed by concatenating characters of some identifying attributes as for example identification numbers, names, addresses, etc.. In our example, we concatenate the first three non-space characters of the title and the last two digits of the production year.
2. **Sorting:** Second, the tuples are sorted - usually lexicographically - by their respective keys.
3. **Windowing:** Finally, a window of fixed size *w* (in our example *w*=3) slides sequentially over the sorted tuples. All tuples being within the window at the same time are paired with each other and added to the resultant search space. Due to the fixed window size, each tuple is compared with at most 2*w*-2 tuples from its immediate neighborhood.

The underlying assumption of the SNM is that duplicate tuples have similar keys and hence are sorted close together. According to [11], large window sizes do not lead to a high pairs completeness, but the rate of false acceptances grows very fast with the window size. For that reason, pairs completeness is often increased by using a multi-pass approach [11]. In this approach instead one, multiple key definition functions are used, each function in one pass. The final search space results in all candidate pairs detected for at least one pass (or more than k passes respectively). It is obvious that the resultant pairs quality is lower than in a single pass approach. However, the risk of not choosing the best key definition function is lowered and the result is usually more accurate.
Assuming a data set with n tuples and a window of a fixed size w, the total number of tuple pair comparisons resulting from using the SNM with a single pass is $O(wn)$ [11].

3. Blocking Approaches For Probabilistic Data

In the previous sections we introduced the ULDB model, described the process of duplicate detection in certain data and went into detail with the SNM. This section is devoted to the adaptation of blocking to the ULDB model. The one big issue here is that probabilistic entity representations may result in probabilistic keys. So in order to make blocking applicable to probabilistic data, the uncertainty of the keys has to be resolved. There are basically two approaches to those adaptations: generating only certain keys and thus resolving uncertainty during the key value creation, or generating probabilistic keys and so resolving the uncertainty during the remaining steps of the respective blocking technique (e.g. the sorting step or the windowing step of the SNM). For each of both approaches, we identified several strategies. With respect to the ULDB model, certain keys as well as probabilistic keys can be considered as non-maybe x-tuples defined on a single attribute. Whereas, a certain key has exactly one alternative, a probabilistic key can have multiple alternatives.
An important fact is that if the source data is certain each variant of our proposed adaptations (based on certain keys as well as probabilistic keys) lead to the same results as the original variants of the corresponding blocking techniques, i.e. our strategies are generalizations of the already existing techniques.

3.1. Adaptations based on Certain Keys

An adequate strategy for building certain keys from x-tuples is by far not so straight forward as already illustrated in our motivating example in Section 1.1, because all the uncertainty in the tuple's data needs to be resolved. For certain key creation, we discuss four strategies. In the multi-pass over possible worlds (Section 3.1.1) a separate pass is applied to some of the database's possible worlds (each a certain relation). In key-per-tuple (Section 3.1.2) for each x-tuple a certain key is built by applying a traditional key definition function on a certain tuple representative. In key-per-alternative (Section 3.1.3) we create a key per x-tuple alternative (each a certain tuple). In key-per-representative (Section 3.1.4) we first compute a set of certain representatives for each x-tuple and then create a key for each of them. Some variants of these strategies can be also applied to immediately created probabilistic keys instead of the original x-tuples (concept *Uncertain Keys First*, see Section 3.1.5).

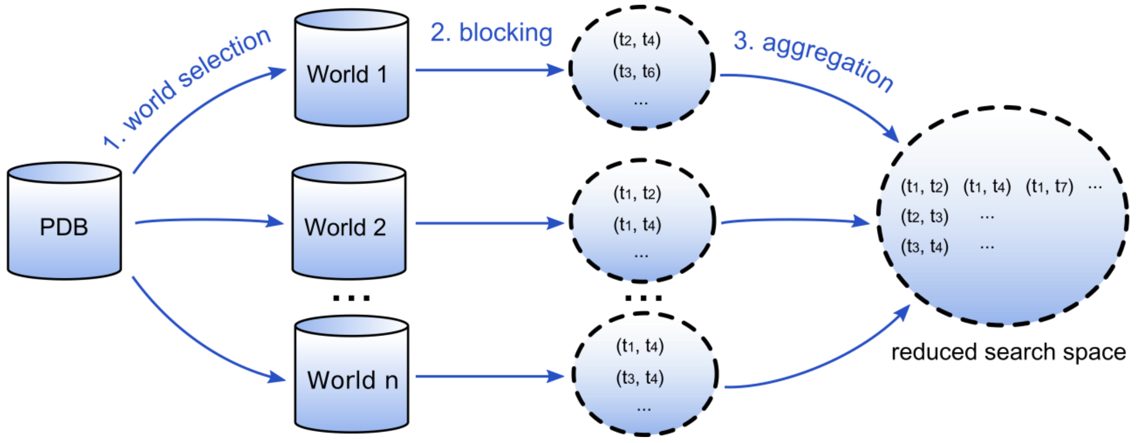


Figure 5: Basic concept of the multi-pass over possible worlds

3.1.1. Multi-Pass over Possible Worlds

The idea for the first strategy is based on the fact that each possible world of an x-relation is an ordinary relation on which blocking can be applied as usual. Thus, a conceptually simple way to perform blocking with certain keys on an x-relation is to construct its corresponding set of possible worlds (see Section 2.1), to apply the conventional blocking technique to each world individually, and to aggregate the resulting search spaces to a single one by the set union operator or by a voting strategy. The basic concept of the *multi-pass over possible worlds* strategy is illustrated in Figure 5.

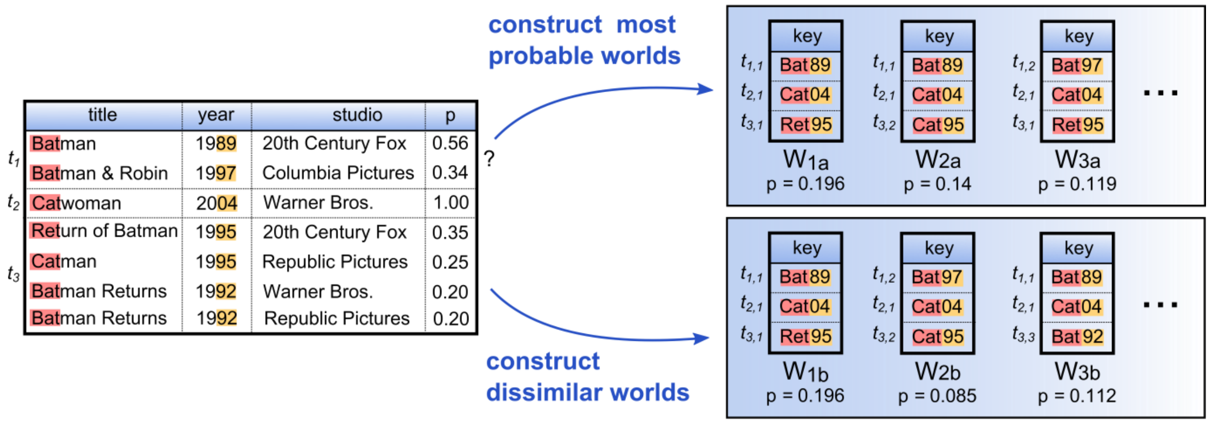


Figure 6: The variants of constructing the most probable worlds or constructing some dissimilar worlds respectively

The problem is that the number of possible worlds of large x-relations is usually tremendous and running passes on all possible worlds is infeasible in practice. Moreover, some tuples are not present in some worlds and thus cannot be paired with other tuples for later in-depth comparison (for instance, in the sample of Figure 2 tuple t_1 is missing in world W_1). Therefore, instead to all blocking is only applied to a set of selected worlds.
The decision which possible worlds should be used is not easy to make; once the first run has been performed on the most probable world, additional passes over the next few most probable worlds will not improve the result very much, because the most probable worlds are usually very similar. For a better result, worlds should be considered that have not only a rather high probability, but are also as dissimilar from one another as possible. We implemented two variants of this strategy. One is to construct the k-most probable worlds and the other is to construct a set of k highly dissimilar possible worlds (see Figure 6). Both variants consider only worlds with all x-tuples present.

```
Input: x-relation $R$, numerical value $k$
1. Let $W_{MP} = \{argmax_{I \in pI(t)} p(I) \mid t \in R\}$
2. Compute for remaining alternatives $t_{i,j}$: $w(t_{i,j}) = p(t_{i,j}) / max_{I \in pI(t)} p(I)$
3. Rank remaining alternatives $t_{i,j}$ into list $L_{alt}$ by $w(t_{i,j})$
4. Let $MostProbableWorlds = \{W_{MP}\}$
5. While $|MostProbableWorlds| < k$
    (a) Remove top element $t_{i,j}$ from $L_{alt}$
    (b) $NewWorlds = \emptyset$
    (c) For each world $W \in MostProbableWorlds$:
        $W_{New} = (W - pI(t_i)) \cup t_{i,j}$ with $P(W_{New}) = P(W) \times w(t_{i,j})$
        Add $W_{New}$ to $NewWorlds$
    (d) Add all $NewWorlds$ to $MostProbableWorlds$
6. Rank $MostProbableWorlds$ by probability into list $L_{worlds}$
Output: First $k$ elements of $L_{worlds}$
```

Algorithm 1: Compute the k most probable worlds

Since all x-tuples are independent to each other, the k-most probable worlds can be built as described in Algorithm 1: First the most probable world is created by taking the most probable alternative from each x-tuple. Second the remaining alternatives are sorted into the list L_{alt} by a weight which is computed from the alternatives probabilities in descending order. Then as long as we have less than k worlds, we make copies from all already created worlds, remove the top element $t_{i,j}$ of L_{alt} and replace the current alternative of x-tuple t_i in each copied world by $t_{i,j}$. Finally we rank the set of created worlds[2] by their probabilities and take the k most probable ones.

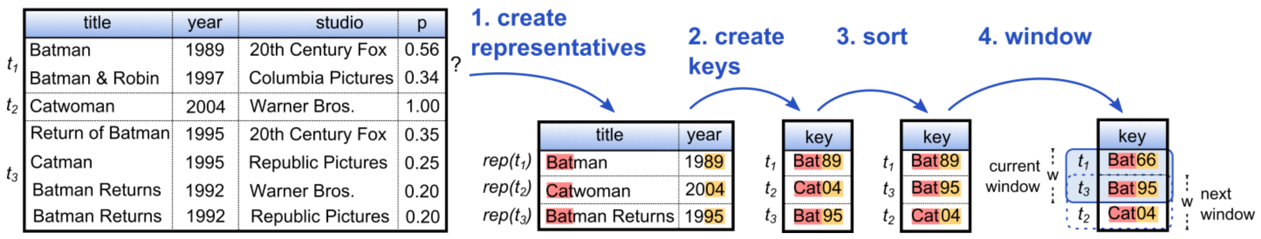
```
Input: x-relation $R$, numerical value $k$
1. Let $DissimilarWorlds = \emptyset$
2. For $i = 1, \dots, k$:
    (a) Let $CurrentWorld = \emptyset$
    (b) For each x-tuple $t \in R$:
        If ($i \le |pI(t)|$)
            Add the $i$th most probable alternative of $t$ to $CurrentWorld$
        Else
            Add the most probable alternative of $t$ to $CurrentWorld$
    (c) Add $CurrentWorld$ to $DissimilarWorlds$
Output: World Set $DissimilarWorlds$
```

Algorithm 2: Compute *k* dissimilar worlds

The basic idea of the second variant is to perform blocking on several possible worlds that are very dissimilar from each other. The probabilities of the constructed worlds are only of secondary importance. Here (see Algorithm 2), the most probable world with all tuples present is constructed for the first pass. Afterwards, a possible world is built by using only the second most probable alternative of each tuple. Accordingly, a possible world is then built from the third most probable tuple alternatives, and so on. This procedure is repeated, until all alternatives have been used or the user-defined threshold k is reached. If for any constructed world a tuple has no more new alternative, the most probable one is used for the remaining worlds. The number of worlds constructed by this procedure is rather small, as it cannot be greater than the maximum number of alternatives per tuple. Furthermore, each additional pass is likely to add many new tuple pairs and thus to improve the result much. So, this variant of the possible world strategy seems by far more promising than constructing the most probable worlds.

The biggest handicap of the multi-pass over possible worlds is its execution time. However, because all passes are independent to each other, we plan to reduce execution time by a parallel implementation using the Map-Reduce framework. The idea is to push each world to another mapper so that all passes can be done at the same time. Finally we use one reducer per x-tuple pair to decide if this pair belongs to the reduced search space or not. A similar approach has been already implemented by Kolb et al. [17] who perform a parallel multi-pass Sorted Neighborhood Method on certain data.

Theoretically, each variant of the multi-pass over possible worlds strategy is identical to a multi-pass over some variants of the key-per-tuple strategy (see Section 3.1.2), by using a different function for computing an x-tuple representative in each pass. However, finding a set of functions leading to the same results as the variants presented above is not trivial. For that reason, we consider this concept as an own strategy.



	title	year	studio	p
t_1	Batman	1989	20th Century Fox	0.56
	Batman & Robin	1997	Columbia Pictures	0.34
t_2	Catwoman	2004	Warner Bros.	1.00
t_3	Return of Batman	1995	20th Century Fox	0.35
	Catman	1995	Republic Pictures	0.25
	Batman Returns	1992	Warner Bros.	0.20
	Batman Returns	1992	Republic Pictures	0.20

	title	year
rep(t_1)	Batman	1989
rep(t_2)	Catwoman	2004
rep(t_3)	Batman Returns	1995

Figure 7: Key-per-tuple: In this example, a tuple's representative is computed from its most probable attribute values

[2] Note, by this algorithm a same world can be result from changing different worlds, but since we use a set of worlds we consider such duplicate worlds to be automatically removed. Moreover, the probability computation of the new worlds is only correct for the copy of the world with the most probable alternative of the considered x-tuple. Thus, we retain the highest probability when removing duplicate worlds.

3.1.2. Key-per-Tuple

This strategy resolves the uncertainty by computing exactly one certain key value for each x-tuple. As illustrated in Figure 7 and Algorithm 3, this strategy is composed of two steps. The simple idea is to compute a certain tuple for every x-tuple as a representative (Step 1) and then to create a key from this representative (Step 2).
For computing a certain x-tuple representative, metadata such as probabilities as well as the actual attribute values can be used. Of course, when computing a representative for key value creation, only key attributes have to be considered. Each x-tuple alternative corresponds to a certain tuple. Thus, computing a certain x-tuple representative from a set of alternatives is similar to computing a representative for multiple conflicting duplicate tuples in the fusion of certain data [18]. The only difference here is that x-tuple alternatives are per definition complete and no handling of null values is required. Moreover, x-tuple alternatives are assigned with probabilities and hence additional meta data for computing a representative is available. Following Bleiholder et al. [18], there are basically two strategies of computing a single representative of a whole tuple set: deciding strategies, in which simply one of the already existing tuples is chosen as a representative, or mediating strategies, in which from the given tuples a new representative is computed, i.e. the resultant representative does not necessarily belong to the input set.

```
Input: x-relation $R$, key definition $\kappa$, representative definition $rep$, blocking technique $B$
1. Let $KeyTuplePairs = \emptyset$
2. For each x-tuple $t \in R$:
    (a) Create the tuple representative $rep(t)$
    (b) Add $(\kappa(rep(t)), t)$ to $KeyTuplePairs$
3. Let $S$ be the search space that results from performing $B$ on $KeyTuplePairs$
Output: Search Space $S$
```

Algorithm 3: key-per-tuple

Deciding Strategies: A very simple deciding strategy is to pick the most probable alternative for each x-tuple. This is equivalent to perform blocking on just the most probable world without missing tuples (see Section 3.1.1). A more complex deciding strategy is based on the Distributional Cluster Feature (DCF). Andritsos et al. [19] use the DCFs to compute a tuple representative which in turn is used for computing a probability for each tuple of a duplicate cluster. Since by using this approach, the computed representative is not an element of the considered domain, the representative itself cannot be used for key value creation, but rather the x-tuple's alternative having the lowest distance to the x-tuple representative has to be used. Since this approach is most likely too time consuming for the blocking purpose and since our experiments showed that using a single key per x-tuple do not lead to best blocking qualities, we did not implement this variant so far.

function	*type*	*description*
cry with the wolves	dec.	take the most often occurring value
most probable value	dec.	take the most probable value
roll the dice	dec.	pick a value randomly
longest value	dec.	take the longest value
median/average	med.	compute the median/average of all values
expectation value	med.	compute the expected value

Table 1: Conflict resolution functions which can be used for mediating strategies

Mediating Strategies: In many situations an alternative computed with a mediating strategy represents an x-tuple better than one of the already existing ones. Mediating strategies are usually applied on an attribute-by-attribute basis. In other words, the tuple representative results from computing a single value representative for each of its attributes. Functions for merging single attributes are denoted as conflict resolution functions [18], because a representative is computed from multiple conflicting input values. To each attribute a different resolution function can be applied. Like the whole strategies, resolution functions can be of a deciding or a mediating style. By using a deciding function one of the existing values is chosen. Two typical deciding functions are *cry with the wolves* where the most often occurring value is taken or *roll the dice* where one of the given values is picked randomly. By mediating functions from a set of given values a new value is created. A typical mediating function is *meet in the middle*, by which the average value or the median is computed. Since x-tuple alternatives are assigned with probabilities, additional conflict resolution functions are possible and often more convenient, e.g., a deciding function in which the most probable value is chosen or a mediating function in which the expected value is computed. A set of conflict resolution functions which can be used for computing a representative of an uncertain attribute value is listed in Table 1.

Naturally, different techniques may be used for different attributes, e.g. the median or the expectation value can be used for numbers, while string values can be processed with taking the most probable value. Moreover, we generally use the roll the dice function as a fallback strategy, when the primary used function delivers an ambiguous result.

To illustrate the difference between deciding strategies and mediating strategies only consisting of deciding functions, we consider the x-tuple t_3 with its four alternatives presented in Figure 6. By choosing the most probable alternative, t_3 is represented by $rep(t_3) = t_{3,1}$. In contrast, by choosing the most probable value for each attribute (mediating strategy with deciding functions) the representative $rep(t_3) =$ ('Batman Returns', 1995) results, which is not equal to any alternative of the considered x-tuple.

3.1.3. Key-per-Alternative

In our third strategy, we do not create a single key for each tuple, but for tuple alternatives, so that tuples may have more than one key computed for them. As a consequence, a tuple can appear several times in the sorted list as shown in Figure 8. Since tuples may appear several times in one window, the number of different x-tuples per window can vary. In order to prevent this effect, we redefine the window size as the number of different x-tuples per window instead of the number of (key,tuple) pairs per window.

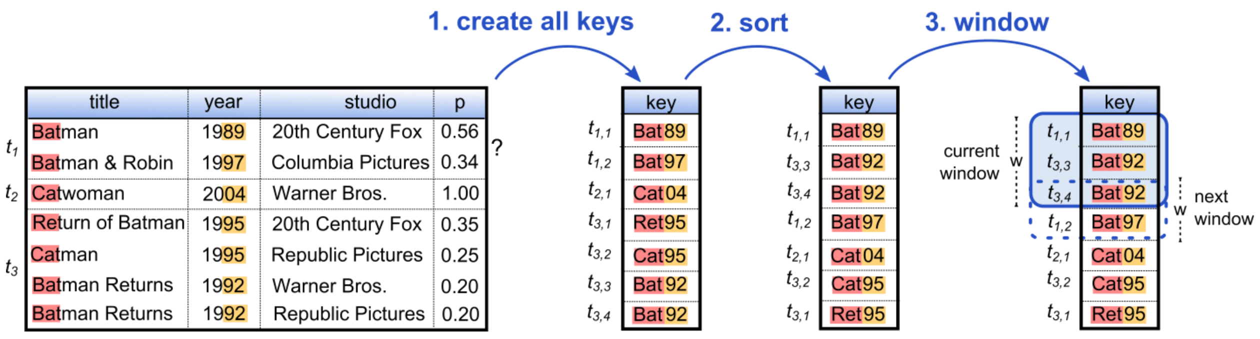


Figure 8: The standard variant of the key-per-alternative strategy

There are many approaches to decide which alternatives are used for key value creation. One of them is to simply use all alternatives. Another idea is to use only a predefined number of alternatives per tuple or to use the most probable alternative of every tuple and, in addition, a share of the remaining alternatives, e.g. the 100,000 most probable remaining alternatives in the database.

In this paper, we consider two variants: (a) the standard variant (KpA-All) which creates a key for all alternatives, and (b) the Top-*k*-variant (see Algorithm 4), which creates a key for the *k* most probable alternatives of each x-tuple.

```
Input: x-relation $R$, numerical value $k$, key definition $\kappa$, blocking technique $B$
1. Let $KeyTuplePairs = \emptyset$
2. For each x-tuple $t \in R$:
    (a) For each $i \in 1, \dots, \min(k, |pI(t)|)$:
        i. Let $I$ be the $i$ most probable alternative of $t$
        ii. Add $(\kappa(I), t)$ to $KeyTuplePairs$
3. Let $S$ be the search space that results from performing $B$ on $KeyTuplePairs$
Output: Search Space $S$
```

Algorithm 4: Top-*k*-variant of key-per-alternative

3.1.4. Key-per-Representative

Our fourth and newest strategy is basically a generalization of key-per-tuple and key-per-alternative and hence is a mixture of both concepts. The underlying idea is to create multiple key values per tuple (as in key-per-alternative)

each derived from a generated tuple representative (as in key-per-tuple). Thus, in key-per-representative we combine the concepts of key-per-tuple and key-per-alternative.
Key-per-representative can be specialized to key-per-tuple by generating only a single x-tuple representative, and it can be specialized to key-per-alternative by generating x-tuple representatives only with deciding strategies.

3.1.5. Concept of *Uncertain Keys First*

In the concept of *Uncertain Keys First*, instead of working on the original set of x-tuples, the proposed methods for certain key value creation are applied on intermediately generated probabilistic keys, each being an x-tuple with one attribute. Since multiple, maybe each less probable, alternatives of an x-tuple can have the same keys, the most probable alternative of an x-tuple's probabilistic key can differ from the key of the most probable alternative of this x-tuple. As a result, the keys created by using the *Uncertain Keys First* concept can be more representative for the considered x-tuples than the keys resulting from applying the key creation strategy commonly.
For illustrating the *Uncertain Keys First* concept and for demonstrating the differences to the standard approach, we consider the tuple t_3 from Figure 7. Assume that we apply the Top-2-variant of the key-per-alternative strategy. Instead of creating certain keys for the two most probable alternatives of each x-tuple, we choose the two most probable alternatives of each x-tuple's probabilistic key. For that purpose, in a first step, for each x-tuple a probabilistic key is created. Since the alternatives $t_{3,3}$ and $t_{3,4}$ of tuple t_3 have the same key 'Bat92', the probability of the corresponding alternative of the probabilistic key is equal to the sum $p(t_{3,3}) + p(t_{3,4}) = 0.4$. In the second step, the intended Top-2-selection is applied to the probabilistic keys. Thus, in our example, the third x-tuple is represented by the keys 'Bat92' and 'Ret95' instead by the keys 'Ret95' and 'Cat95'.
Before evaluating the quality of this concept by our experiments in Section 4, we first discuss the feasibility of the *Uncertain Keys First* concept for the different variants of our certain key based strategies:

- **Key-per-Tuple:** In deciding strategies the key of one (e.g. the most probable) alternative is taken. Choosing the most representative alternative of an x-tuple's probabilistic key seems more qualified than choosing the key of the most representative x-tuple's alternative. In contrast, mediating whole instances (x-tuple alternatives) seems more qualified than mediating single attribute values (probabilistic key alternatives), because keys are composed by proportion of different attributes and hence have no inherent semantics. For that reason, we suggest to take the *Uncertain Keys First* concept for variants only based on deciding strategies and not to use this concept for mediating variants or mixed ones.
- **Multi-Pass of Possible Worlds:** The *Uncertain Keys First* concept should improve the accuracy of the Top-k variant, because more representative worlds are selected. In contrast, in the variant of dissimilar worlds, worlds are arbitrarily selected. Thus, we cannot make any appropriate forecast for that variant.
- **Key-per-Alternative:** For the Top-k-variant the *Uncertain Keys First* concept should improve accuracy. If keys for all alternatives are created; the results of both concepts are equivalent.

3.2. The Sorted Neighborhood Method with Probabilistic Keys

In this section, we shortly discuss in which ways the core functionality of the SNM can be adapted to probabilistic keys. Sorting tuples by their key values corresponds to a tuple rank scenario where the keys serve as ranking scores and the lexicographic order serves as ranking order. Thus, we consider existent techniques for ranking probabilistic tuples [20] to resolve the uncertainty in the sorting step by building a sorted list of x-tuples based on their probabilistic keys or to resolve the uncertainty in the windowing step by sliding the window over a set of possible sorting lists.

3.2.1. Single Ranking Approaches

In single ranking approaches from probabilistic keys a single certain ranking is computed in the sorting phase.

- **Most Probable Ranking (SNM$_{\mathbf{MPR}}$) :** The base idea of this adaptation is to rank (sort) the probabilistic tuples by the most probable ranking of their key values. By using a key definition function κ, this can be realized by sorting based on the two relations '$<_P$' and '$=_P$', which are defined as:

  $$t_1 <_p t_2 \Leftrightarrow p\big(\kappa(t_1) < \kappa(t_2)\big) > p\big(\kappa(t_2) < \kappa(t_1)\big) \text{ and } t_1 =_p t_2 \Leftrightarrow \neg\big(t_1 <_p t_2\big) \wedge \neg\big(t_2 <_p t_1\big)$$

  Since just another order relation is used, complexity is dominated by the sorting time ($\Rightarrow O(nlog(n))$).

- **Expected Position Ranking (SNM_{ExpR}) :** This approach based on the idea to compute the expected rank position per x-tuple and then to rank all tuples by this position. For a finite set of possible ranking scores per tuple this computation can be done in $O(nlog(n))$ [20].
- **Expected Score Ranking (SNM_{ExpS}) :** This approach based on the idea to transform the blocking key into a numerical value, to use this value as a ranking score and then to rank the tuples by their expected score. For simple transformations this approach is dominated by the ranking time ($\Rightarrow O(nlog(n))$).
- **Uncertain Rank Aggregation (SNM_{URA}) :** In this approach, a ranking is computed which has the minimal average (expected) distance to all possible rankings. For attribute uncertainty models such an aggregation based on the footrule distance can be done in polynomial time ($O(n^{2.5})$), whereas a computation based on the Kendall tau distance is known to be NP-Hard [20].

3.2.2. Multiple Ranking Approaches

As multiple ranking approaches, we consider approaches which do not produce a single ranking result, but resolve uncertainty in the windowing phase.

- **Sorted U-Rank Neighborhood (SNM_{URN}) :** This approach is based on the rank function *l-UTop-Rank(i,j)* which is defined by Ilyas et. al [20]. This rank function returns the *l* most probable x-tuples that appear at the rank position $i \dots j$. Let *w* be the used window size, in the Sorted U-Rank Neigborhood we pair all x-tuples that result from *l_1-UTop-Rank(i,i)* with all x-tuples that result from *l_2-UTop-Rank(i-w,i+w)*, where intuitively $l_2 > l_1$ (for example $l_2 = w \times l_1$).

3.2.3. Comparison

Since the most probable sorting should be more representable than the sorting resulting from the most probable world, the SNM_{MPR} is expected to supply a better blocking quality than the Top-1 variant of key-per-alternative.
However, by using a single ranking approach each x-tuple is represented only once in the sorted list. Thus, an x-tuple t_i is only close to a second x-tuple t_j, if t_j is similar to the other neighbors of t_i, too. Therefore, similar to key-per-tuple, x-tuple uncertainty can be only restrictedly considered, because there exist no single sort position for an x-tuple with dissimilar alternatives which is appropriate to find all of its duplicate candidates. As a consequence, from single ranking approaches we can expect a blocking quality which is similar to the quality of key-per-tuple. First experiments for SNM_{MPR} and SNM_{ExpS} confirmed that intuition.

4. Experimental Evaluations

In our experimental evaluations, we analyzed the differences in blocking quality of our proposed adaptations based on creating certain keys. Hereby, we especially focused on the robustness against a varying data dirtiness and a varying data uncertainty. Moreover, we evaluated for which variants the *Uncertain Keys First* concept was actually valuable. Finally, we compared the quality results of our adaptations for different blocking techniques.

4.1. Probabilistic Test Data

Getting large sets of unclean probabilistic real-life data being labeled, i.e. each duplicate pair is exactly known, is nearly impossible. For that purpose, we produced some synthetic data sets for revalidating the quality of our proposed strategies. In order to make the data as realistic as possible, we decided to use real-life data from an existing certain database. So we extracted title, production year, studio and director of about 300,000 movies from the online movie database IMDb[3] with the Java application JMDb[4] and stored the data to an HSQLDB[5].
For generating probabilistic data from the duplicate-free certain data, we programmed a Java application named ProbDataGen[6]. With ProbDataGen it is possible to choose among several HSQL databases holding certain movie data to generate a probabilistic movie database with duplicates, where the user can make several adjustments, e.g. the number of duplicates, the maximal number of alternatives per tuple, or the datas' degree of dirtiness.
To improve the reliability of our experimental results further on, we use a standard data setting for the movie tables in our experiments. The characteristic of this standard setting is adopted from the characteristic of a real-life CD-

[3] The Internet Movie Database (http://www.imdb.com)
[4] Java Movie Database (http://www.jmdb.de)
[5] HyperSQL DataBase (http://hsqldb.org)
[6] http://vsis-www.informatik.uni-hamburg.de/projects/QloUD/ProbDataGen

dataset[7] with duplicates. We adjust the percentage of duplicates, the average duplicate cluster size and the average similarity of the true duplicates to this real-life data set. In experiments where data characteristics are modified for experimental reasons, we used this setting as a fixed point and only changed the analyzed characteristic. We think that these adjustments make our experiments as realistic as possible, even though synthetic data sets are used.
All the data sets (along with descriptions of their characteristics) we used in our experiments are available at *http://vsis-www.informatik.uni-hamburg.de/projects/QloUD/ICIQ2012/TestData*.

4.2. Experimental Settings

We performed five experiments. For space limitations, for the first four experiments we show only the results for the SNM which in our mind were most illustrative. In the last experiment, we also used Standard Blocking (SB) and Robust Suffix-Array Blocking (SAB) to make an overall comparison between different blocking techniques.

1. In the first experiment, we made an overall comparison of the certain key based variants proposed in this paper. We evaluated and compared their quality in terms of pairs completeness, pairs quality and runtime. In this experiment, we used the SNM with a fixed window size w=10 and a key built by the first 12 non-space characters of the movie title (parameter k_l) and the last two digits of the production year. Moreover, we used movie tables generated with our standard data setting.

2. Duplicate detection is especially required to work on dirty data, i.e. source data with poor quality. Thus, in Experiment 2, we evaluated the robustness of our variants against a varying dirtiness of the source data. For that purpose we used six sets of movie tables each generated with different settings for dirtiness. Since we consider duplicate detection, we measure quality as the average similarity of the true duplicate pairs (the lower the average duplicate similarity, the dirtier the data). For measuring similarity, we took the Monge-Elkan distance [6], which is known to work well for most domains. In this experiment, we used k_l =12. Moreover we used the SNM with a specifically chosen w for each strategy so that all strategies produced a search space of similar size (this should enable a fair comparison of pairs completeness).

3. In the third experiment, we evaluated the robustness against a varying data uncertainty. For that purpose, we changed the average number of alternatives per x-tuple. We used the SNM with w=10 and k_l =12.

4. In the fourth experiment, we evaluated the impact of the *Uncertain Keys First* concept on the resultant blocking quality. In this experiment, we used the SNM with w=10 and k_l =12.

5. In our final experiment, we compared the results from the SNM with the results from Standard Blocking (SB) and Robust Suffix-Array Blocking (SAB). For comparison, we conducted runs with two different experimental objectives. First, we executed the KpA-All variant on several databases with varying quality to test the robustness of the blocking techniques against poor data quality (Objective 1). Then, we compared the results for a selected set of adaptation approaches w.r.t. these three techniques on our standard data set (Objective 2). For the first objective, we took the KpA-All variant, because it was the adaptation approach performing best for all three techniques. For the second objective, we took our standard data set and performed for each blocking technique KpT, KpA-All, Diss(10) and Top-1.

For our experiments we consider the adaptation variants listed in Table 2.

shorthand	***variant description***
Top-1	a single pass over the most probable world (identical with MPW-1 and KpA-Top-1)
MPW-10	a multi-pass over the 10 most probable worlds
Diss(k)	a multi-pass over k dissimilar worlds
KpT	a key-per-tuple variant which build a representative by using the most probable value of each attribute
KpA-All	the standard variant of key-per-alternative using all alternatives for key value creation
KpA-Top-k	a key-per-alternative variant which uses the k most probable alternatives for key value creation
KpR	a key-per-representative variant which takes all alternatives plus a tuple built by the most probable attribute values as representatives

Table 2: The variants (along with their shorthand symbols) of our certain key based approaches used in the experiments

[7] http://www.hpi.uni-potsdam.de/naumann/projekte/repeatability/datasets/cd_datasets.html

We applied each experiment on generated data sets of 102,692 x-tuples with 4,380 duplicate pairs. If not stated otherwise, each x-tuple has at most 10 alternatives (5.46 alternatives in average). All experiments were performed on a machine with an Intel(R) 3.1GHz quad-core processor, 8GB main memory, and a 64-bit operating system.

4.3. Experimental Results

4.3.1. Experiment 1: Overall Comparison of Adaptation Strategies using the SNM

The absolute values of pairs completeness and pairs quality are shown in Figure 9. Table 3 shows the blocking quality of different variants in relation to the blocking quality produced by KpA-All. Figure 10 shows the runtime of the different variants.

As expected and shown by the experimental results, a multi-pass over the *k* most probable worlds with *k* > 1 did not bring any advantage, because no new candidate pairs result from the subsequent passes, but runtime increased linear with growing *k*. In contrast, a multi-pass over dissimilar worlds was extremely beneficial. Already for small window sizes and short keys a good pairs completeness (PC > 0.9) was achieved. The goodness lacked with fewer worlds to be constructed, but was still of good quality by using 5 dissimilar worlds (see Diss(5) in Figure 9 and in Table 3).

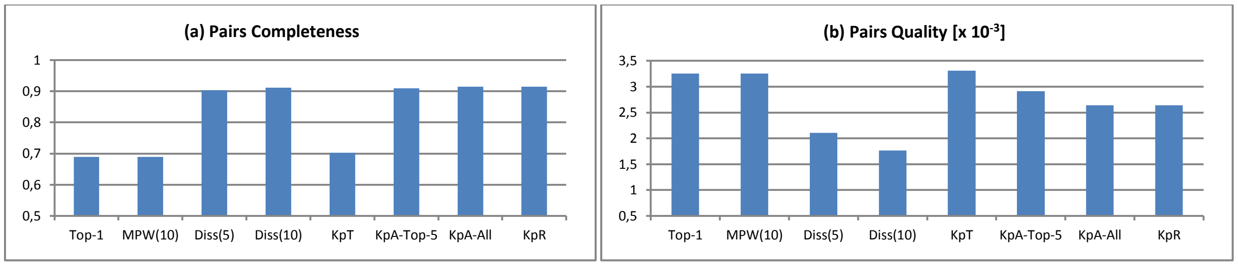


Figure 9: Pairs completeness and pairs quality of different adaptation variants, each performed with the SNM

Interestingly, the used KpT variant which creates an x-tuple representative by using the most probable value of each attribute performs a little bit better than using the most probable alternative as the representative (Top-1). That shows that mediating strategies can be useful to create an x-tuple representative. Combining mediating strategies and deciding strategies for creating a set of x-tuple representatives, as we did it with the KpR variant, was not successful, i.e. it did not improve the KpA-All variant in any of the performed experimental runs.

The conclusion of this experiment is that for the SNM producing multiple keys per tuple turned out to be more accurate than creating a single one. Of course, the resultant search space grows with the number of alternatives used for key value creation, but the resultant values of pairs quality are all of an acceptable size. The trade-off between accuracy and effectiveness is perfectly illustrated by the results shown in Table 3. The strategies using a single key per x-tuple (KpT, Top-1) are most effective (smallest search space and lowest runtime), but less accurate than the strategies using multiple keys per x-tuple (KpA-All, KpA-Top-5, Diss(5), Diss(10)).

In summary, due to the higher priority of pairs completeness, the variants which produce multiple key per tuple (KpA, Diss(*k*)) turned out to be best suitable to adapt the SNM to probabilistic data.

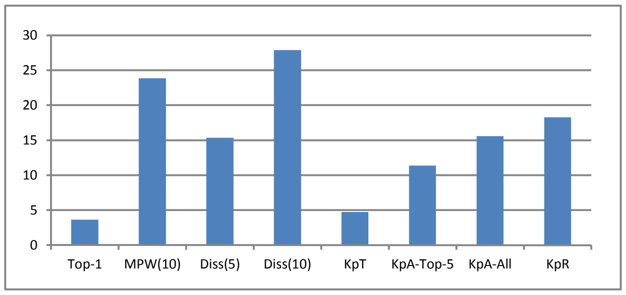


Figure 10: Runtime [in sec] of different adaptation strategies based on certain key values performed with the SNM

strategy:	*selected true duplicate pairs:*	*size of search space:*	*runtime:*
KpA-All	**100%**	**100%**	**100%**
KpA-Top-5	<u>99.45%</u>	90.26%	73.07%
KpT	76.82%	<u>61.3%</u>	<u>30.33%</u>
Top-1	75.45%	<u>61.3%</u>	<u>23.30%</u>
Diss(5)	<u>98.83%</u>	124.07%	95.58%
Diss(10)	<u>99.75%</u>	149.55%	179.02%

Table 3: Comparison of different variants to KpA-All (best results are underlined)

4.3.2. Experiment 2: Robustness against a varying Dirtiness of the Source Data

Since we set all strategies so that they produced search spaces of similar sizes, we present only results on pairs completeness in Figure 11. As you can see, all the variants produced a result of good quality if the source data were of good quality (similarity of 0.93), too. Nevertheless, the blocking quality shrank rapidly when the source data became dirtier. In general, it is easy to see that the five considered variants can be grouped into two classes. The first class contains KpA-All, KpA-Top-5 and Diss(10). These variants worked acceptable for the three cleanest data sets and became only bad for the data sets with the poorest quality. The second class contains KpT and Top-1. The blocking quality of these variants was bad in the most cases. This experiment shows that using multiple keys for x-tuples makes the blocking process more robust against a varying dirtiness of the source data.

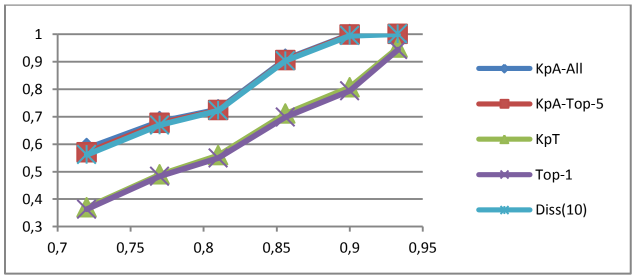


Figure 11: Pairs completeness for different variants of the adapted SNM w.r.t. a varying quality of the source data.

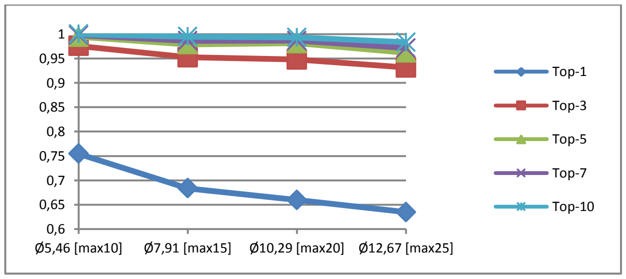


Figure 12: Pairs completeness for KpA Top-*k* w.r.t. a growing number of x-tuple alternatives

4.3.3. Experiment 3: Robustness against a varying Uncertainty of the Source Data

In the results of the previous experiments, the KpA-All variant shows the best performance on pairs completeness. However, creating a key for each alternative can be very ineffective for databases with a high degree of uncertainty, i.e. the average number of alternatives per x-tuple is very high. For that reason, we were interested in the loss of quality we will suffer, if we use only the *k* most probable x-tuple alternatives instead all of them. To evaluate that fact, we conducted a set of experiments with different settings for *k* on four different sets of movie tables, each with another degree of uncertainty. The experimental results on pairs completeness are depicted in Figure 12. The notation Øa [max b] on the x-axis denotes that in the corresponding movie table the average number of alternatives per x-tuple was a and the maximal number of alternatives an x-tuple can have was b. The values of the individual variants are computed in relation to the result of the variant KpA-All, i.e. a result of 1.0 for a setting *k* means that the Top-*k* variant detected all the duplicates which have been detected by the KpA-All variant.
The Top-1 variant performed significantly worse than the KpA-All variant, but for *k >2* the loss of true positives compared to KpA-All is less than 5%, even if the maximal number of alternatives per x-tuple is up to 25. Certainly, the relative number of correctly detected duplicate pairs shrank, if data uncertainty grew, but this loss of quality is of an acceptable size. To show the complexity which comes along with a high setting of *k*, we also compared the absolute size of the resultant search space and the execution time (see Figure 13). The higher *k*, the more the search space grew proportional with the uncertainty of the data. In contrast, for low values of *k*, e.g. *k* = 1 or *k* = 3, the size of the search space was mostly independent from the degree of uncertainty. Moreover, the runtime of KpA-All grew extremely with a growing number of x-tuple alternatives, whereas the runtime for the other variants grew less significantly, the lower *k*.

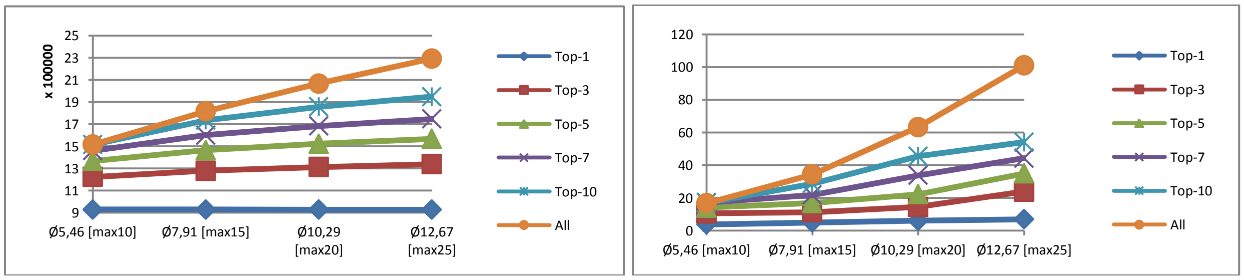


Figure 13: Absolute search space sizes and runtimes [sec] for different variants of KpA Top-*k* w.r.t. a growing number of x-tuple alternatives

4.3.4. Experiment 4: Uncertain Keys First

To test the idea of *Uncertain Keys First*, we conducted a set of experiments and tested different variants of the key-per-alternative strategy and the multi-pass over possible worlds strategy. Recall, these are the two strategies for which we expected that *Uncertain Keys First* could have a positive impact (see Section 3.1.5).
As expected, *Uncertain Keys First* improved the pairs completeness of the KpA-Top-*k* variants as well as the pairs completeness of the multi-pass over the *k* most probable worlds, but surprisingly decreases pairs completeness of the multi-pass over *k* dissimilar worlds. We detect that this impact is substantially independent from the window size and the quality of the data. The most interesting effect of *Uncertain Keys First* was observed for the KpA-Top-*k* variant. The degree of improvement decreased with growing *k*, i.e. is maximal for *k* = 1, and increased with the number of alternatives per x-tuple. The average amount of improvement (scored in percentage of pairs completeness) w.r.t. different settings of *k* as well as the average amount of improvement w.r.t. a growing number of alternatives per x-tuple are shown in Figure 14. In both cases, we aggregated over the remaining dimension.

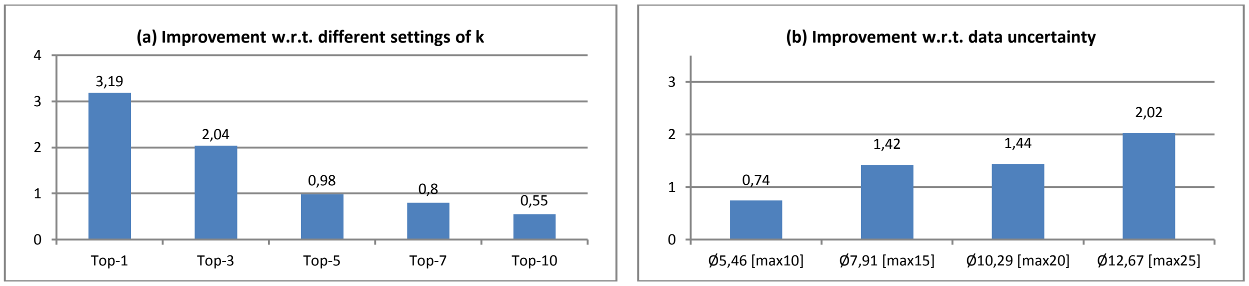


Figure 14: The improvement achieved by using Uncertain Keys First with respect to (a) different settings of the KpA Top-*k* variant and (b) a growing number of alternatives per x-tuple

4.3.5. Experiment 5: Overall Comparison of Different Blocking Techniques

The results of the robustness test are shown in Figure 15. In databases of good quality (similarity > 0.9) all three techniques achieved an outstanding pairs completeness close to 1. In contrast pairs completeness shrank significantly for databases with poor quality. SAB was by far the most robustness technique. Even for an average duplicate similarity of 0.72 SAB achieved a pairs completeness of nearly 0.9. In contrast the pairs completeness of SB and SNM decreased down to 0.74 (SB) or 0.61 (SNM) respectively. Surprisingly, SB performs better than SNM. Moreover, SAB achieved by far the highest pairs quality and produced the smallest search space. The pairs quality of SB and SNM were nearly identical. In general, pairs quality shrank, if the duplicate pairs became more dissimilar.
The results of our second objective are depicted in Figure 16. They show that SAB performed best for all of the adaptation variants. Second in quality was SB. SNM achieved the poorest results. You can see that the differences in blocking quality of the certain key variants are the same for all three techniques: KpT performed better than Top-1 what shows that using the most probable alternative is generally not the best variant to create a tuple representative. Moreover, the resultant qualities of the different blocking techniques vary at most in the variants producing a single key. In contrast, for KpA-All and Diss(10) all three techniques produced similar results. The single key strategies as KpT produce a smaller search space and hence had a better pairs quality than the strategies producing multiple keys.

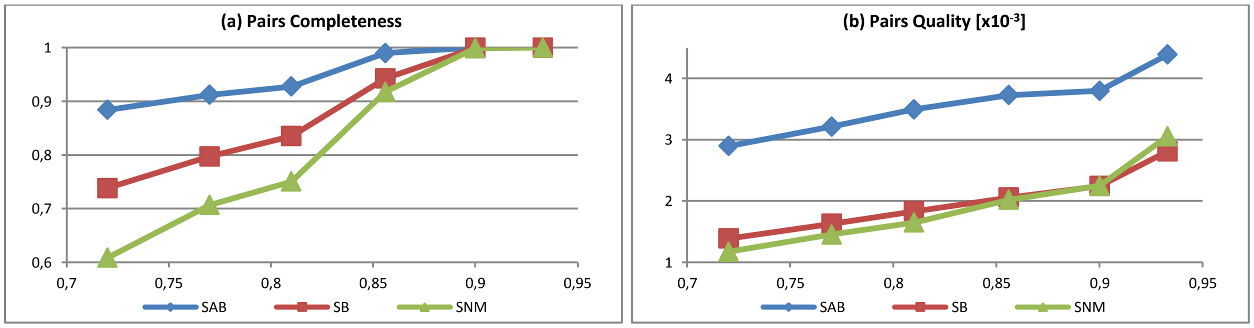
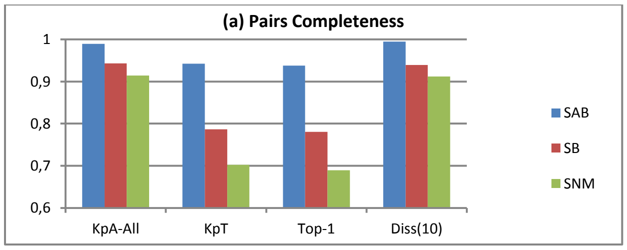


Figure 15: Pairs completeness and pairs quality of KpA-All performed with SAB, SB and the SNM w.r.t. databases of different qualities (measured by the average similarity of all true duplicates)



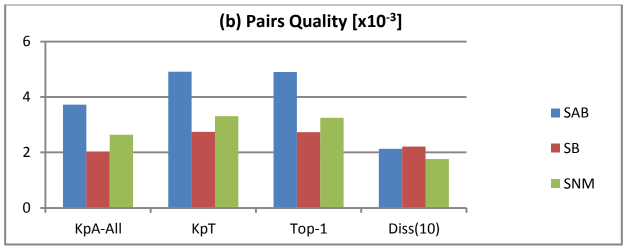


Figure 16: Pairs completeness and pairs quality of some adaptation strategies performed with SAB, SB and the SNM

4.4. Experimental Conclusions

The experiments presented above show the feasibility of our approaches. Moreover, with key-per-alternative and the multi-pass over dissimilar worlds, they lift out two adaptation strategies which were best fitting for all three considered blocking techniques. Moreover, they were most robust against a poor quality of the source data. Only in scenarios where the search space must be as small as possible, a single key approach as key-per-tuple is maybe a better choice. The critical point of KpA is the one discussed in Experiment 4. Using all the alternatives for key value creation can affect the efficiency of this approach negatively. For that reason a Top-k variant with k>2 is sometimes better suitable. In that case the concept of *Uncertain Key First* can improve the effectiveness further on, but slightly increases the search space. The drawback of Diss(k) is its long runtime for high settings of k. However, this weak point should be erased by a parallel implementation as we plan it in future research.

5. Related Work

Duplicate detection in general [6] [7] [14] [21] [13] and blocking in particular [10] are handled in several works. Existing blocking techniques that are based on the use of key values are Standard Blocking [8] [22], the Sorted Neighborhood Method [11] [23] [24], Q-gram Indexing [9], Suffix-Array Blocking [25] [16], K-way Sorting [26], Similarity-Aware Inverted Indexing [27], Sorted Blocks [28], String Map based Indexing [29], Priority Queue [30], TI-similarity [31], and Adaptive Filtering [32]. Further blocking techniques are Locality-Sensitive Hashing [33], Fuzzy Blocking [34], Canopy Clustering [35] [36], Spectral Neighborhood Blocking [37], and blocking with MFIBlocks [38]. Kolb et al. [17] consider a parallelization of duplicate blocking using the Map-Reduce programming model. Approaches for blocking based on semantic relationships between data items are proposed in [39] (tuple relationships given by foreign keys) and [40] (hierarchical relationships in XML documents). In [41] blocking data items with heterogeneous data structures is considered. Further interesting and useful work on blocking can be found in [42] [43] [44] and [45].
Some duplicate detection approaches produce probabilistic data as result data for modeling ambiguous duplicate decisions [46] [47] or for modeling uncertain merging results [19]. None of these studies, however, handle probabilistic data as source data. In contrast, in current research on the integration of uncertain data [48], deduplication is not considered. To the best of our knowledge, we are the first who consider the problem of blocking in the context of duplicate detection in probabilistic data. Nevertheless, to adapt blocking to probabilistic data we make recourse to techniques already used in the fusion of certain data tuples as proposed in [18] [19]. Moreover, we made some first proposals about the in-depth comparison of x-tuples in [15].

6. Conclusion

Duplicate tuples are pervasive problems of data quality. To efficiently apply duplicate detection on large data sets, the search space has to be initially reduced by a blocking technique. Until now, duplicate detection, and especially blocking, has only be considered for certain data. Nevertheless, duplicates are a quality problem in probabilistic databases, too. In this paper we propose different strategies to adapt the Sorted Neighborhood Method, which is a state-of-the-art blocking technique, to probabilistic source data. We present strategies based on certain keys created from probabilistic entity representations and shortly discuss possible strategies based on probabilistic keys. The benefit of using certain keys is that these strategies can also be applied to other key-based blocking techniques without any specific adaptation. In contrast, strategies based on probabilistic keys need to be tailor-made for each blocking technique. Our experimental evaluations of the certain key approaches show that creating multiple certain keys per entity representation is more effective than creating a single certain key per entity representation. Moreover, using multiple keys turned out to be more robust against a varying dirtiness or uncertainty of the source

data than using a single key. Finally, we observe that intermediately created probabilistic keys can improve the efficiency of the approaches based on multiple certain keys further on.
In future research, we aim to accelerate our blocking approaches, especially the multi-pass over possible world approaches, by using the Map-Reduce framework. Moreover, we plan to focus on strategies for probabilistic key based blocking adaptations in more detail.

REFERENCES

[1] D. Suciu, A. Connolly and B. Howe, "Embracing Uncertainty in Large-Scale Computational Astrophysics," *MUD Workshop,* pp. 63-77, 2009.

[2] D. Z. Wang, E. Michelakis, M. J. Franklin, M. Garofalakis and J. M. Hellerstein, "Probabilistic declarative information extraction," *ICDE,* pp. 173-176, 2010.

[3] D. Suciu, D. Olteanu, C. Re and C. Koch, Probabilistic Databases, Morgan & Claypool Publishers, 2011.

[4] T. J. Green and V. Tannen, "Models for incomplete and probabilistic information," *EDBT Workshops,* pp. 278-296, 2006.

[5] O. Benjelloun, A. D. Sarma, A. Y. Halevy and J. Widom, "Uldbs: Databases with uncertainty and lineage," *PVLDB,* pp. 953-964, 2006.

[6] F. Naumann and M. Herschel, An Introduction to Duplicate Detection, Morgan & Claypool Publishers, 2010.

[7] A. K. Elmagarmid, P. G. Ipeirotis and V. S. Verykios, "Duplicate Record Detection: A Survey," *TKDE,* pp. 1-16, 2007.

[8] M. Jaro, "Advances in Record Linkage Methodologies as Applied to Matching the 1985 Census of Tampa Bay, Florida," *Journal of American Statistical Society 84,* pp. 414-420, 1985.

[9] R. Baxter, P. Christen and T. Churches, "A comparison of fast blocking methods for record linkage," pp. 25-27, 2003.

[10] P. Christen, "A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication," *TKDE,* pp. 1537-1555, 2012.

[11] M. A. Hernandez and S. J. Stolfo, "The Merge/Purge Problem for Large Databases," *SIGMOD Conference,* pp. 127-138, 1995.

[12] S. Abiteboul, P. C. Kanellakis and G. Grahne, "On the representation and querying of sets of possible worlds," pp. 158-187, 1991.

[13] C. Batini and M. Scannapieco, Data Quality: Concepts, Methodologies and Techniques. Data-Centric Systems, Berlin: Springer, 2006.

[14] J. R. Talburt, Entity Resolution and Information Quality, Morgan Kaufmann Publishers, 2011.

[15] F. Panse, M. van Keulen, A. de Keijzer and N. Ritter, "Duplicate Detection in Probabilistic Data," *ICDE Workshops,* pp. 179-182, 2010.

[16] T. de Vries, H. Ke, S. Chawla and P. Christen, "Robust record linkage blocking using suffix arrays," *CIKM,* pp. 305-314, 2009.

[17] L. Kolb, A. Thor and E. Rahm, "Multi-pass sorted neighborhood blocking with mapreduce," *Computer Science - R&D,* pp. 45-63, 2012.

[18] J. Bleiholder and F. Naumann, "Data fusion," *ACM Comput. Surv.,* pp. 1-41, 2008.

[19] P. Andritsos, A. Fuxman and R. J. Miller, "Clean Answers over Dirty Databases: A Probabilistic Approach," pp. 30-41, 2006.

[20] I. Ilyas and M. A. Soliman, Probabilistic Ranking Techniques in Relational Databases, Morgan & Claypool Publishers, 2011.

[21] P. Christen, Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection, Springer, 2012.

[22] P. Lehti and P. Fankhauser, "A precise blocking method for record linkage," *DaWaK,* pp. 210-220, 2005.

[23] S. Yan, D. Lee, M.-Y. Kan and C. L. Giles, "Adaptive sorted neighborhood methods for efficient record linkage," *JCDL,* pp. 185-194, 2007.

[24] U. Draisbach, F. Naumann, S. Szott and O. Wonneberg, "Adaptive Windows for Duplicate Detection," pp. 1073-1083, 2012.

[25] A. N. Aizawa and K. Oyama, "A fast linkage detection scheme for multi-source information integration," *WIRI*, pp. 30-39, 2005.

[26] A. Feekin and Z. Chen, "Duplicate detection using k-way sorting method," *SAC*, p. 323–327, 2000.

[27] P. Christen and R. Gayler, "Towards scalable real-time entity resolution using a similarity-aware inverted index approach," *AusDM*, pp. 51-60, 2008.

[28] U. Draisbach and F. Naumann, "A Generalization of Blocking and Windowing Algorithms for Duplicate Detection," *ICDKE*, pp. 18-24, 2011.

[29] L. Jin, C. Li and S. Mehrotra, "Efficient record linkage in large data sets," *DASFAA*, pp. 137-152, 2003.

[30] A. E. Monge and C. Elkan, "An efficient domain-independent algorithm for detecting approximately duplicate database records," *DMKD*, pp. 0-7, 1997.

[31] S. Y. Sung, Z. Li and S. Peng, "A fast fltering scheme for large database cleansing," *CIKM*, pp. 76-83, 2002.

[32] L. Gu and R. A. Baxter, "Adaptive filtering for efficient record linkage," *SDM*, pp. 477-481, 2004.

[33] H. sik Kim and D. Lee, "Harra: fast iterative hashed record linkage for large-scale data collections," *EDBT*, pp. 525-536, 2010.

[34] J. Nin and V. Torra, "Blocking anonymized data," *AGOP*, pp. 83-87, 2007.

[35] W. W. Cohen and J. Richman, "Learning to match and cluster large high-dimensional data sets for data integration," *KDD*, pp. 475-480, 2002.

[36] A. McCallum, K. Nigam and L. H. Ungar, "Efficient clustering of high-dimensional data sets with application to reference matching," *KDD*, pp. 169-178, 2000.

[37] L. Shu, A. Chen, M. Xiong and W. Meng, "Efficient spectral neighborhood blocking for entity resolution," *ICDE*, pp. 1067-1078, 2011.

[38] B. Kenig and A. Gal, "Efficient entity resolution with mfiblocks," Technion, 2011.

[39] J. Nin, V. Muntes-Mulero, N. Martinez-Bazan and J. Larriba-Pey, "On the use of semantic blocking techniques for data cleansing," pp. 190-198, 2007.

[40] S. Puhlmann, M. Weis and F. Naumann, "Xml duplicate detection using sorted neighborhoods," pp. 773-791, 2006.

[41] G. Papadakis, E. Ioannou, C. Niederee and P. Fankhauser, "Efficient entity resolution for large heterogeneous information spaces," pp. 535-544, 2011.

[42] M. Bilenko, B. Kamath and R. J. Mooney, "Adaptive blocking: Learning to scale up record linkage," pp. 87-96, 2006.

[43] P. Christen, "Towards parameter-free blocking for scalable record linkage," 2007.

[44] M. Michelson and C. A. Knoblock, "Learning blocking schemes for record linkage," 2006.

[45] S. E. Whang, D. Menestrina, G. Koutrika, M. Theobald and H. Garcia-Molina, "Entity resolution with iterative blocking," pp. 219-232, 2009.

[46] G. Beskales, M. A. Soliman, I. F. Ilyas and S. Ben-David, "Modeling and Querying Possible Repairs in Duplicate Detection," *PVLDB*, pp. 598-609, 2009.

[47] M. van Keulen, A. de Keijzer and W. Alink, "A Probabilistic XML Approach to Data Integration," *ICDE*, pp. 459-470, 2005.

[48] P. Agrawal, A. D. Sarma, J. Ullman and J. Widom, "Foundations of uncertain-data integration," pp. 1080-1090, 2010.