Automatic Blocking Key Selection for Duplicate Detection based on Unigram Combinations

Tobias Vogel
Hasso Plattner Institute
Potsdam, Germany
tobias.vogel@hpi.uni-potsdam.de

Felix Naumann
Hasso Plattner Institute
Potsdam, Germany
felix.naumann@hpi.uni-potsdam.de

ABSTRACT

Duplicate detection is the process of identifying multiple but different representations of same real-world objects, which typically involves a large number of comparisons. Partitioning is a well-known technique to avoid many unnecessary comparisons. However, partitioning keys are usually handcrafted, which is tedious and the keys are often poorly chosen.

We propose a technique to find suitable blocking keys automatically for a dataset equipped with a gold standard. We then show how to re-use those blocking keys for datasets from similar domains lacking a gold standard. Blocking keys are created based on unigrams, which we extend with length-hints for further improvement. Blocking key creation is accompanied with several comprehensive experiments on large artificial and real-world datasets.

1. EFFICIENT DUPLICATE DETECTION

In a duplicate detection process the safe way to find *all* duplicates is to compare each element (record) with each other element. This approach is computationally expensive, and it is also pointless, because the vast majority of comparisons is performed on two totally different records that have little to nothing in common. Thus, efficient duplicate detection relies on a good pair selection algorithm that anticipates promising comparisons upfront and chooses only those pairs for similarity measure calculation. The *precision* of a duplicate detection run is largely driven by the ability of the similarity measure to tell non-duplicate records apart. On the other hand, high *recall* is achieved by sophisticated methods that are able to avoid comparisons among non-duplicates. In this paper we generalize from similarity measures and consider only recall, i. e., we propose a method that automatically partitions records according to duplicity with other records.

In general, pair selection divides the comparison space into either overlapping or non-overlapping partitions and performs the comparisons only within those partitions. One well-known, non-overlapping technique for pair selection is *blocking*. An expert usually selects one or more attributes that he expects to have common values amid similar records, the *blocking key*. The relation is partitioned according to each record's value, the *blocking key value*, for these attributes. In an address data scenario, this could be, for instance, the ZIP code or the family name (or the concatenation of both). It is probable that two records identifying the same person have the same ZIP code or the same family name. Other attributes, such as gender, social security number, or middle initial, do not serve well, because the partitions would become either too large, leading to very many unnecessary comparisons, or too small, separating duplicate records over multiple partitions. Depending on the choice of attributes and the actual value distribution in the relation, several attributes might be required to form the blocking/partitioning criterion to further refine and sub-partition the partitions.

A refinement of that blocking approach is to consider only parts of attribute values instead of their full value: First, mistakes in the unconsidered parts of the attribute are ignored and thus smoothed out, and second, partitions might become too small if the entire attribute was considered. Instead, only parts of the attributes, usually the first n characters or the first m vowels are used. A more sophisticated method is to apply hash functions on the values, such as calculating the Soundex [2] code for a value. However, selecting a proper value for n or m, or deciding whether and which hash functions make sense, requires domain-experience and considerable manual effort.

Additionally, several passes with different blocking criteria may be performed to overcome data defects in exactly those attributes. Since blocking saves significant computation time, multiple passes can be afforded that keep the overall recall considerably high.

We have observed that different positions in attributes serve differently well as blocking criterion. For example, a blocking key might use the first two characters of both the city and the ZIP code attribute. Assuming no typos, with such a key all address records with the same ZIP code end up in the same (large) partition, and the city refines this partition to finally create appropriately sized partitions. However for large cities, the ZIP code is the same (for this city) in the first few characters, thus in fact not refining the city partition[1]. I. e., using the first few characters in a ZIP code does not add any distinction to the blocking key. Each record

[1]For example, Stuttgart (Germany) has ZIP codes ranging from 70173 to 70619.

has to be mutually compared to all other records in the partition, which causes high computation costs. For example, having 100,000 records in a partition causes approximately 5 billion comparisons. Hence, this blocking key does not succeed in separating dissimilar records into different partitions.

In contrast, the third character in a ZIP code might be more relevant, because it changes for different districts of a (large) city. Not all digits occur equally frequently and resemble the actual distribution of addresses within this city. Our approach will identify those relevant parts of attributes and propose them for blocking keys.

We make the following contributions:

1. A technique to automatically choose blocking keys that are both a good estimation of the record similarity and do not create too large partitions

2. A comprehensive evaluation on a large dataset and high recall on related datasets

3. For our example domain, address data, a list of the top blocking keys

The remainder of this paper is structured as follows: Section 2 presents related work. Section 3 describes the general idea of unigram-based partitioning, gives formalized notions of pairs completeness, efficiency, and the overall blocking key quality, and depicts the overall workflow. It further introduces a novel technique to integrate attribute length-hints into unigrams. Section 4 contains evaluations on the presented techniques, and finally Section 5 concludes the paper and highlights further research directions.

2. RELATED WORK

Duplicate detection is a settled and broad field. Prior work dates back many decades of research. This section restricts briefly to duplicate detection in general and subsequently describes partitioning approaches and automation of partitioning in particular.

Duplicate detection

Duplicate detection (or record linkage, reference conciliation, instance identification, merge/purge, etc.) describes the process of identifying same real-world objects within one or several databases. Research started in the late 1960s [6] and was driven by many different groups, organizations, and purposes. That is why there are plenty of names for the same research area (which are themselves duplicates). Elmagarmid [5] and Naumann and Herschel [14] give overviews about the topic and present the fundamental steps and approaches.

Partitioning in general

Christen [3] compares several (so-called) indexing techniques (traditional blocking, Sorted Neighborhood [9], q-grams [7], canopy clustering [12], String-Map [10], and Suffix-array). Concluding that they all achieve similar results on the same dataset, he identifies the key definition as the most crucial decision (which attributes to take and which part of those), rather than the indexing technique. Draisbach [4] bridges blocking and windowing approaches into sorted blocks.

Automatic blocking

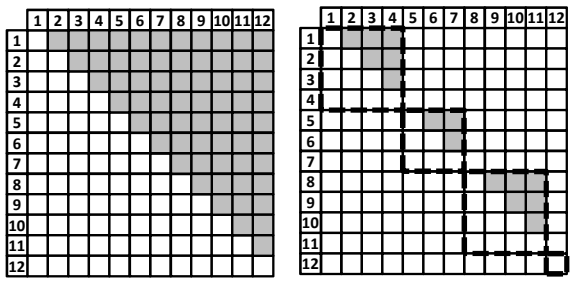
Bilenko et al. [1] propose to take two orthogonal steps at the same time. Not only do they decide on which attributes to consider, but they also decide on the very information taken from those attributes, e. g., "Same 1st Three Chars in LastName" or "... Year same or off-by-one" (see Section 1). They create disjunctive normal forms (DNFs) of different length and evaluate them against a gold standard. DNFs perform slightly better than canopy clustering. However, the features need to be manually selected and their applicability may differ over several domains.

Michelson and Knoblock [13] also use different DNFs that were greedily aggregated via a training dataset. They consider similarity measures from a small pool of methods combined with all the attributes from the dataset as atomic features and optimize the DNFs regarding reduction rate and pair completeness. Due to the greedy approach, they only find sub-optimal blocking keys and they also treat attributes values as a whole and rely on the availability of similarity measures.

Kenig and Gal [11] propose another technique that clusters tuples according to their overlap of common attributes. However, this approach relies on knowledge about the estimated size of the duplicate clusters and considers full attributes. We promote a more general approach using unigrams instead of n-grams, without explicit specification of attribute characteristics and without knowledge about the anticipated duplicates.

3. PARTITIONING

Partitioning is the key to efficiency when carrying out a duplicate detection process. Without partitioning, each record has to be compared pairwise with each other record, causing the effort to be squared regarding the number of comparisons (see Figure 1(a)). To reduce this effort, only records within the same partition are compared (intra-partition comparisons), eliminating all inter-partition comparisons (see Figure 1(b), blocks[2] are highlighted). Consequently, each partition contains few enough elements to ensure a relatively quick execution of the duplicate detection run and the total number of comparisons decreases dramatically. Moreover, the partitioning pre-classifies clusters of duplicates so that a high recall can be retained.



(a) All comparisons (b) Intra-Partition Comparisons

Figure 1: Comparison of computation effort for exhaustive comparisons (a) and blocking (b).

[2]We use the terms partition(ing) and block(ing) synonymously.

The crucial parameter for partitioning is the blocking key, the criterion for partitioning records into blocks. When using attribute values to create blocking keys, the smallest possible unit are the individual *unigrams* of attribute values. We specify a chosen unigram by so called *unikeys*; a unikey defines an attribute and a position within, for example, the third character of the family name attribute, written as `familyname-2`[3]. Hence, the value of a unikey for a given attribute value is a single character, the unigram. For instance, the aforementioned unikey and the family name attribute value "Sørensen" yield the unigram "r". A *blocking key* is then a list of m unikeys, for example `[city-3, date-2, givenname-2]`[4]; a corresponding blocking key value could look like "s 2 r".

Note that there are two objective functions (pairs completeness and efficiency) that could be optimized for. For example, find the most effective blocking key with a fixed efficiency (costs) of, say, 1 billion comparisons. In this paper, we formally present a (configurable) trade-off between both functions and optimize this trade-off.

Our goal is to find the "best" blocking key for a given dataset. The quality of a blocking key derives from both pairs completeness (how many duplicates are found) and efficiency (how many comparisons are performed to find those duplicates). Pairs completeness is measured in the achieved recall, efficiency is calculated using the absolute number of comparisons. A perfect blocking key would create partitions exactly according to the tuples' duplicity relations, thus no unnecessary comparisons are performed and all duplicates are found. In the opposite, a trivial blocking key would put every tuple into a giant single partition, achieving perfect pairs completeness, too, but disrupting efficiency by far too many unnecessary comparisons. The notion of blocking key quality is explained in more detail in Section 3.1.

Once good blocking keys are determined for a training dataset that has a gold standard, this knowledge can be used for other (test) datasets from a similar domain where no gold standard is available. Two datasets are of the same or similar domain if both schemata overlap to a certain degree; the higher the overlap, the higher the domain-similarity. The required overlap ratio depends on the actual application, user preferences (if available), value distribution (especially language), error characteristics, and the used blocking keys.

In such a test dataset, blocking keys still have to be "good" (i. e., effective and efficient), but also *valid*. This means, their unikeys have to be available in the schema of the test dataset. Validity is defined below.

Finally, the set of good blocking keys for training datasets can be applied to test datasets automatically without any human interaction. See Section 3.2 for the overall workflow. Section 3.3 describes the integration of length-hints for unigrams.

3.1 Problem Statement and Blocking Key Quality

Given a dataset and its schema, find a valid blocking key (or a set of k valid blocking keys) that achieves the optimal trade-off between pairs completeness and efficiency.

[3]By convention, we start indexing at 0.

[4]Note that the order of unikeys is irrelevant. We always state them in alphabetical order.

Validity

Usually, a dataset different from the training dataset will comprise other attributes. A given blocking key is called *valid* in a test dataset, iff all of its unikeys are available in the test dataset, both regarding the availability of the schema attributes as well as the attribute lengths. The attribute length is defined by the schema (e. g., a `CHAR(100)` in SQL) or is infinite for other data sources (e. g., CSV files).

Pairs completeness

The pairs completeness [8, 13] is the measure of how many of the duplicates can be found for a blocking key, i. e., how effective the blocking key is.

A blocking key is used to create a partitioning to preclassify duplicate records. Subsequently, a similarity measure is applied on each possible pair within each partition. If the pair's similarity is above a given threshold, it is treated as a duplicate, otherwise as a non-duplicate. The ratio of actual duplicates among the declared pairs divided by all duplicates is called recall and serves as the pairs completeness. In our experiments, we replace such a similarity measure by a lookup in the true matches.

Efficiency

A blocking key is efficient if it uses relatively few comparisons to achieve a given pairs completeness. Thus, the measure for efficiency is the average number of performed comparisons for each found duplicate.

In practice, however, the number of comparisons should not exceed a fixed threshold θ. We express efficiency by normalizing the number of comparisons c according to θ and subtract it from 1 to align it to the pairs completeness. Thus, efficiency is defined as $1-(\frac{c}{\theta}) \in [0,1]$, assuming $c \leq \theta$.

This measure resembles the term *Filtered Reduction Ratio* [8]. Yet using the actual number of potential comparisons ($5 \cdot 10^{10}$ for 100,000 tuples) in the denominator would usually create a value close to 1. Therefore, we adapt the notion by Gu and Baxter, but instead of a filtering step, we give the efficiency in relation to a baseline approach. In our case this is the number of comparisons, the Sorted Neighborhood approach would have created.

Overall Blocking Key Quality (BQ)

A good blocking key should be effective and efficient. Therefore, we define the *Overall Blocking Key Quality BQ* as the harmonic mean between pairs completeness and efficiency ($BQ = \frac{2 \cdot PC \cdot Ey}{PC+Ey}$), where PC is pairs completeness and Ey is efficiency.

3.2 Key Generation Workflow

Automatic blocking key generation is performed in two steps. First, for a training dataset with a given gold standard, all combinatorially possible blocking keys are evaluated. Second, for a test dataset, typically lacking a gold standard, the previously created list of blocking keys is iterated to find the best valid blocking key.

Training Phase

As the first step, good blocking keys are identified:

1. Generate all possible unikey combinations (i. e., blocking keys).

2. For each blocking key perform a duplicate detection experiment on the reference dataset:
   (a) If the number of comparisons exceeds the threshold θ, discard this blocking key.
   (b) Else, calculate the achieved overall blocking key quality (BQ) for the blocking key
3. Sort all non-discarded blocking keys descendingly by BQ.

Production Phase

The keys from the training phase can subsequently be used to find duplicates in test datasets of similar domains.

1. For each blocking key in the previously calculated list, check for validity for the current dataset.
2. For each remaining valid blocking key (still ordered by BQ), start a duplicate detection run.
   (a) If the number of comparisons exceeds a certain threshold, abort the run, keeping the so-far detected duplicates.
   (b) Else, finish the duplicate detection run until one of the following abortion criteria is fulfilled: the desired number of passes have been executed, the total number of actually performed comparisons over all runs exceeds a threshold θ, the overall efficiency sinks below a given threshold (i. e., no or not enough new duplicates are found), or the number of detected duplicates is sufficient. Note that the thresholds might be domain dependent or given by a user.

3.3 Incorporate Attribute Value Lengths into Unigrams

Many attributes allow for different lengths of their values, e. g., cities, names, or phone numbers. Thus, the higher the unikey position in an attribute, the higher the probability of reading an empty character. In blocking key values, empty characters are indistinguishable, no matter whether they appear just after the last non-empty character or far behind it. Unigrams do not carry information about the attribute value's length, they just contain a single character from a specified position. This section shows a technique to enhance unigram information by length-hints.

In the upper part of Figure 2, some strings with different string lengths are shown. On the right-hand side, their respective unigrams for the unikey `givenname-8` are given. They all consist of the empty character (represented as $\perp$). Hence, this unikey does not add any further refinement to a blocking key for those attribute values.

We propose to pad the missing positions (empty character) with special characters up to the highest unikey position. In the experiments and for illustration in this paper, we selected circled numbers (see Figure 2, lower part), because they consume only one character (and are thus compatible with the unigram concept) and they are very unlikely to occur in the dataset. Moreover, they have an inherent, human-readable order. Taking the same unikey as before (`givenname-8`) we can see that there are more different unigrams than in the traditional variant above and they capture similarity more closely.

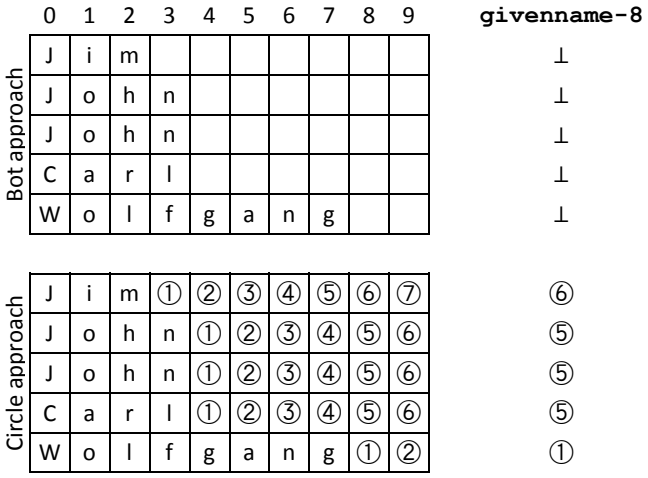


Figure 2: Replacing empty characters to allow more fine-grained partitions

Section 4.4 shows that the circle approach outperforms the traditional (bot[5]) approach in terms of BQ due to better refinement capabilities.

4. EVALUATION

Section 4.1 presents the datasets used for evaluation. Section 4.2 introduces the key figures while Section 4.3 evaluates on details and on how well blocking keys can be used for other datasets. Finally, Section 4.4 evaluates the length-hints approach.

4.1 Datasets and Settings

For evaluation, we chose as training dataset a confidential corporate address dataset (called "Corporate-1") with a random sample of 100,000 tuples consisting of 12 attributes, comprising names, city, date of birth, etc. Further, we took unikeys for all attributes and selected the first five positions, a generally sound value according to our experience. In total, this yields $12 \cdot 5 = 60$ unikeys. We further created blocking keys consisting of unikey combinations of sizes 3 to 5. This creates $\binom{60}{57} + \binom{60}{56} + \binom{60}{55} = 5,983,367$ unikey combinations. The sample contains 804 pairwise disjoint duplicates, roughly 1%.

To come up with a comparison threshold θ, we took the traditional Sorted Neighborhood pair selection algorithm on this dataset with a window size of 100. Such an algorithm performs exactly $\theta = 9,895,050$ comparisons. Thus, we use this θ when evaluating the six million blocking keys. The similarity measure for the corporate dataset was a perfect lookup in the gold standard, i. e., the similarity measure always achieves a precision of 100%. Therefore, we only report on recall (pairs completeness).

We chose two datasets as test datasets, one was another (disjoint) 100,000 sample from the corporate address dataset (called "Corporate-2"), the second was a 100,000 sample of a places dataset, integrated data from Facebook, Gowalla, and Foursquare about places such as shops, restaurants, etc. throughout the world (called "Places").

[5] $\perp$ is written as `\bot` in LaTeX

The experiments were performed on a many-core Linux CentOS (64 bit) machine. The multithreaded implementation was written in Java using 10 GB of main memory and 30 threads.

4.2 Overall Blocking Key Quality

Figure 3 shows the duplicate detection results for the training dataset regarding the number of found duplicates and the number of comparisons performed to find those duplicates. Each point shows the results for one blocking key. Only results for a pairs completeness of at least 82% (660 duplicates; achieved by 325 blocking keys) are shown.

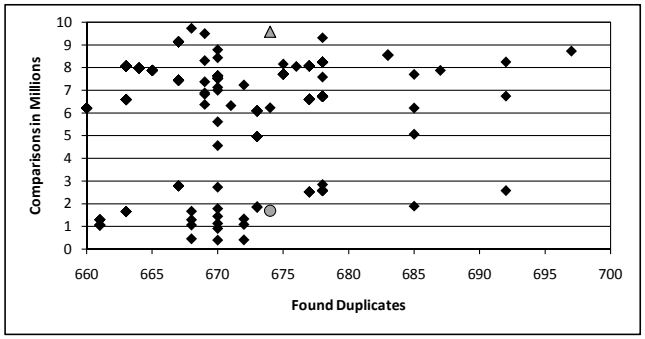


Figure 3: Performed comparisons against each number of found duplicates (showing only results with at least 82% pairs completeness).

For example, there are several blocking keys that find 674 duplicates; the most efficient key (`[familyname-0, familyname-1, zip-0, zip-1]`, gray circle) performed 1,690,163 comparisons and took 277 ms while the least efficient key (`[street-0, street-1, street-2, street-3]`, gray triangle) used 9,578,354 comparisons within 475 ms for the same number of duplicates. Yet, the threshold θ (9,895,050) prevented calculating far worse keys (e. g., `[phone-0, title-0, title-1, title-2]` for the same result by using 4,164,642,286 comparisons in 74,797 ms). Note that the partitioning itself (i. e., without time for comparison) only takes 314 ms on average for all of the three mentioned blocking keys (445 ms on average over all blocking keys).

For the 6 million blocking keys evaluated, the total calculation took one full day with our 30-thread implementation.

All of the three blocking keys have the same pairs completeness (82%), but they can be distinguished by their efficiency which is 83% for the first key and only 3% for the second key. The overall blocking key quality BQ mediates between both measures and is shown for the same 325 blocking keys in Figure 4.

The best blocking key has a BQ of 89.31%. It finds 672 duplicates using 407,232 comparisons. Among all blocking keys, 5,329 blocking keys have a BQ of at least 80%.

4.3 Detailed Experiment Results

Table 1 shows the most successful blocking keys with regard to the number of found duplicates, comparisons, pairs completeness, efficiency, and BQ. To compare, an "expert guess" – the ad-hoc blocking key `[city-0, familyname-0, givenname-0, zip-0]` a human expert might have come up with – only found 274 duplicates.

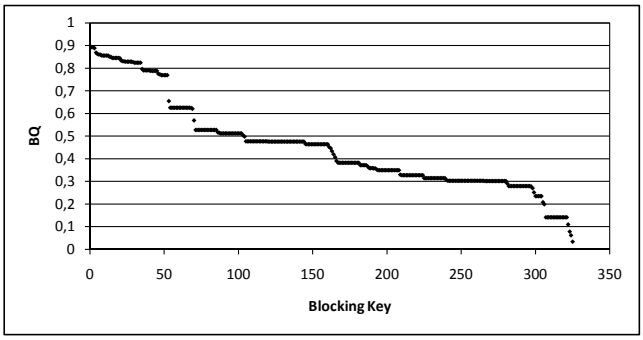


Figure 4: Overall blocking key quality (BQ) for the blocking keys from Figure 3, sorted by BQ.

Just a bit of derivation in the attribute positions as in `[city-0, familyname-0, givenname-3, zip-1]` had found 7 more duplicates comparing one fifth fewer records. The most successful blocking key found 86.69% of the duplicates, but used 12,521 comparisons for each duplicate. In contrast, the most efficient blocking key only performed 27 comparisons per duplicate revealing only a small fraction of all the duplicates. Finally, the overall best key was both, effective and efficient, as described above. The respective maximum values in the table are emphasized.

The Top 20 blocking keys (regarding to BQ) are depicted in Table 2 in the appendix. At the end of the day, 3,531,838 of the 5,983,367 blocking keys (59%) caused few enough comparisons and were considered for the training dataset.

The most frequent unikeys among the Top 300 blocking keys (Table 3) contain expected attributes (ZIP code, street, familyname) at the top and show that for most attributes the first position is not the most frequent one.

Domain Transfer

We used the Corporate-2 dataset to evaluate whether blocking keys can be taken over to another dataset from the same domain. Since both samples (Corporate-1 and Corporate-2) have been derived from the same dataset, the schema is the same and thus all blocking keys are valid. We chose the 300 best blocking keys (according to their BQ) from the training dataset and performed duplicate detection runs on them.

The absolute number of found duplicates marginally increased, because there are a few more duplicates in the second sample. However, all the relative measures (pairs completeness, efficiency, and BQ) remained nearly stable. The average overall blocking key quality among the top 300 blocking keys decreased slightly from 85.54% to 85.19%, while BQ even improved for 89 blocking keys. Table 4 in the appendix shows key figures for the first 10 blocking keys.

In another experiment, we applied the 300 blocking keys from before on the Places dataset (7,151 duplicates to find) which has a different schema (and value distribution), but a similar domain. Here, only 131 blocking keys are valid, however the first invalid blocking key had rank 50. The average overall blocking key quality is 94.29% due to a generally higher pairs completeness. This means that the duplicate characteristics resemble the blocking keys very well, even with data from different languages and domains. Table 5 in the appendix shows key figures for the first 10 blocking keys.

Description	Blocking key	Found duplicates	Comparisons	Pairs Completeness	Efficiency	BQ
Expert guess	`[city-0, familyname-0, givenname-0, zip-0]`	274	258,077	34.08%	97.39%	50.49%
Most duplicates and maximum pairs completeness	`[zip-0, zip-1, zip-2, zip-3]`	*697*	8,727,009	*86.69%*	11.80%	20.78%
Least comparisons per duplicate and most efficient	`[city-0, familyname-0, family-name-3, givenname-3, street-3]`	214	*5,781*	26.62%	*99.94%*	42.04%
Overall best	`[familyname-0, familyname-1, zip-0, zip-1, zip-2]`	672	407,232	83.58%	95.88%	*89.31%*

Table 1: Selected outstanding blocking keys

4.4 Attribute Lengths in Unigrams

For all the experiments in this section, we used the circle approach, preserving attribute lengths. In this section, we evaluate the usefulness of integrating the attribute length in a unigram. We ran each deduplication experiment/blocking key twice for the Corporate-1 dataset, first with the bot, second with the circle replacement strategy.

Among the 61 distinct top 50 blocking keys (both from circle and bot), 19 blocking keys have a higher quality when using the circle approach, 29 blocking keys have the same quality, and 13 blocking keys have a higher quality for bot.

The traditional replacement strategy (bot) creates equal-sized or larger partitions, because the character diversity is smaller. This, in turn, causes more comparisons and might increase the pairs completeness compared to the circle approach lowering the efficiency. Since in practice, the number of allowed comparisons has to be limited, some blocking keys cannot be considered when using the bot approach while the circle approach still allows those blocking keys. The best blocking key that causes too many comparisons when using bot has a BQ of 62.69% and finds 74.38% of the duplicates.

5. CONCLUSION

We presented a technique to discover high quality blocking keys on a given training dataset. The blocking keys were evaluated regarding pairs completeness and efficiency; both measures are joined in the notion of the overall blocking key quality. The experiments showed that it is possible to easily find high quality blocking keys for other datasets from similar domains lacking a gold standard by selecting them from the results of the training dataset. Length-hints provide useful help and even allow otherwise discarded blocking keys to be considered.

Our final goal is to create an autonomous system (a service) for the entire duplicate detection process, with no need for human configuration. In this work, we assume that some pre-processing is already done: Input data is structured and the schema has been analyzed. Moreover, each attribute in the schema is assigned to a class (e. g., city, family name, or ZIP code) [15], and with it to some similarity measure. As a next step, such a service has to find high quality blocking keys automatically. Our proposed automatic duplicate detection technique just needs one training dataset per domain to be efficient and autonomous. With the training dataset's gold standard, a list of good blocking keys can be compiled beforehand. This list is subsequently searched for the best valid blocking key for each customer dataset.

As a future research direction, true n-grams instead of unigrams can be taken as blocking key features. This dramatically increases the computational complexity. Hence, a good heuristics is needed to estimate the number of comparisons upfront without performing the whole partitioning. Many parameters were fixed in the evaluation, e. g., taking the first five attribute value positions. A good choice for that number derives from the attribute's class (see above). For gender, 1 might be enough, but dates might benefit from 8 to 10 positions.

Many of the best blocking keys resemble each other and only differ in one or two unikeys. Multiple passes with different blocking keys might be more effective if the blocking keys are more diverse. The trade-off between pairs completeness (e. g., by having a larger diversity) and efficiency (e. g., by saving comparisons due to quite similar blocking keys) should incorporate this and generate improved, more carefully aligned sets of blocking keys.

The BQ treats pairs completeness and efficiency equally. It is possible to place more weight on the pairs completeness to achieve better results, while using more computation power, enabling different cost models. Furthermore, test datasets usually offer attributes which are currently ignored, but might provide good unikeys. Finally, overlapping partitioning techniques are not yet considered by the blocking key creation technique, but have the potential to give even better results. However blocking keys face special requirements here, such as unikey order sensitivity.

6. REFERENCES

[1] M. Bilenko, B. Kamath, and R. J. Mooney. Adaptive blocking: Learning to scale up record linkage. In *Proceedings of the International Conference on Data Mining (ICDM)*, pages 87–96, 2006.

[2] C. P. Bourne and D. F. Ford. A study of methods for systematically abbreviating english words and names. *Journal of the ACM*, pages 538–552, 1961.

[3] P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2011.

[4] U. Draisbach and F. Naumann. A generalization of blocking and windowing algorithms for duplicate detection. In *Proceedings of the International Conference on Data and Knowledge Engineering (ICDKE)*, 2011.

[5] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2007.

[6] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 1969.

[7] A. Ferro, R. Giugno, P. Puglisi, and A. Pulvirenti. An efficient duplicate record detection using q-grams array inverted index. In *Proceedings of the International Conference on Data Warehousing and Knowledge Discovery (DaWaK)*, 2010.

[8] L. Gu and R. Baxter. Adaptive filtering for efficient record linkage. *Proceedings of the SIAM International Conference on Data Mining (SDM)*, 2004.

[9] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 127–138, 1995.

[10] L. Jin, C. Li, and S. Mehrotra. Efficient record linkage in large data sets. In *Proceedings of the International Conference on Database Systems for Advanced Applications (DASFAA)*, 2003.

[11] B. Kenig and A. Gal. Efficient entity resolution with mfiblocks. Technical report, Technion, 2011.

[12] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the ACM International Conference on Knowledge discovery and data mining (SIGKDD)*, pages 169–178, 2000.

[13] M. Michelson and C. A. Knoblock. Learning blocking schemes for record linkage. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)*, 2006.

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

[15] T. Vogel and F. Naumann. Instance-based "one-to-some" assignment of similarity measures to attributes. In *Proceedings of the International Conference on Cooperative Information Systems (CoopIS)*, 2011.

Appendix

Rank	Blocking key	BQ
1	[familyname-0, familyname-1, zip-0, zip-1, zip-2]	89.31%
2	[street-1, street-4, zip-0, zip-1, zip-2]	89.21%
3	[street-0, street-1, zip-0, zip-1, zip-2]	88.81%
4	[street-1, street-4, zip-1, zip-2]	86.88%
5	[familyname-0, familyname-1, zip-1, zip-2]	86.23%
6	[street-0, street-1, zip-1, zip-2]	86.07%
7	[street-1, street-4, zip-0, zip-2]	85.87%
8	[familyname-0, familyname-1, title-3, zip-1, zip-2]	85.60%
9	[familyname-0, familyname-1, title-2, zip-1, zip-2]	85.60%
10	[familyname-0, familyname-1, title-4, zip-1, zip-2]	85.60%
11	[familyname-0, familyname-1, title-1, zip-1, zip-2]	85.60%
12	[familyname-0, familyname-1, title-0, zip-1, zip-2]	85.60%
13	[familyname-0, familyname-1, zip-0, zip-2]	85.05%
14	[street-0, street-1, zip-0, zip-2]	84.94%
15	[familyname-0, familyname-1, title-3, zip-0, zip-2]	84.46%
16	[familyname-0, familyname-1, title-4, zip-0, zip-2]	84.46%
17	[familyname-0, familyname-1, title-2, zip-0, zip-2]	84.46%
18	[familyname-0, familyname-1, title-1, zip-0, zip-2]	84.46%
19	[familyname-0, familyname-1, title-0, zip-0, zip-2]	84.46%
20	[street-1, street-4, zip-0, zip-1]	84.36%

Table 2: Top 20 blocking keys for Corporate-1 training dataset.

Unikey	Frequency
zip-2	256
zip-1	187
street-1	162
zip-0	147
street-4	124
street-0	121
street-3	108
street-2	71
familyname-0	56
familyname-1	33
zip-3	27
hno-0	25
hno-1	15
pobox-4	14
pobox-1	14
pobox-3	14
pobox-0	14
pobox-2	14
title-3	14
title-4	14
title-2	14
title-1	14
title-0	14
familyname-4	2

Table 3: A histogram of the unikeys for the Top 300 blocking keys.

Blocking key	Corporate 1					Corporate 2				
	Found Duplicates	Comparisons	Pairs Completen.	Efficiency	BQ	Found Duplicates	Comparisons	Pairs Completen.	Efficiency	BQ
[familyname-0, familyname-1, zip-0, zip-1, zip-2]	672	407,232	83.58%	95.88%	89.31%	729	401,129	83.21%	95.94%	89.13%
[street-1, street-4, zip-0, zip-1, zip-2]	670	396,567	83.33%	95.99%	89.21%	732	396,333	83.56%	95.99%	89.34%
[street-0, street-1, zip-0, zip-1, zip-2]	668	455,230	83.08%	95.39%	88.81%	726	444,373	82.87%	95.50%	88.74%
[street-1, street-4, zip-1, zip-2]	670	914,682	83.33%	90.75%	86.88%	720	262,808	82.19%	97.34%	89.12%
[familyname-0, familyname-1, zip-1, zip-2]	672	1,082,543	83.58%	89.05%	86.23%	724	289,010	82.64%	97.07%	89.28%
[street-0, street-1, zip-1, zip-2]	668	1,060,006	83.08%	89.28%	86.07%	720	328,685	82.19%	96.67%	88.84%
[street-1, street-4, zip-0, zip-2]	670	1,129,843	83.33%	88.58%	85.87%	691	284,009	78.88%	97.12%	87.05%
[familyname-0, familyname-1, title-3, zip-1, zip-2]	661	1,060,663	82.21%	89.28%	85.60%	691	290,549	78.88%	97.06%	87.03%
[familyname-0, familyname-1, title-2, zip-1, zip-2]	661	1,060,666	82.21%	89.28%	85.60%	691	291,244	78.88%	97.05%	87.03%
[familyname-0, familyname-1, title-4, zip-1, zip-2]	661	1,060,668	82.21%	89.28%	85.60%	721	431,861	82.30%	95.63%	88.47%

Table 4: Comparison of the key figures for the first 10 best blocking keys in Corporate-1 (training) applied on Corporate-2.

Blocking key	Corporate 1					Places				
	Found Duplicates	Comparisons	Pairs Completen.	Efficiency	BQ	Found Duplicates	Comparisons	Pairs Completen.	Efficiency	BQ
[familyname-0, familyname-1, zip-0, zip-1, zip-2]	672	407,232	83.58%	95.88%	89.31%	7,151	270,706	100%	97.26%	98.61%
[street-1, street-4, zip-0, zip-1, zip-2]	670	396,567	83.33%	95.99%	89.21%	6,721	405,490	93.98%	95.90%	94.93%
[zip-0, zip-1, zip-2, street-0, street-1]	668	455,230	83.08%	95.39%	88.81%	6,753	617,040	94.43%	93.76%	94.09%
[zip-1, zip-2, street-1, street-4]	670	914,682	83.33%	90.75%	86.88%	6,717	465,847	93.93%	95.29%	94.60%
[familyname-0, familyname-1, zip-1, zip-2]	672	1,082,543	83.58%	89.05%	86.23%	6,721	432,902	93.98%	95.62%	94.79%
[street-0, street-1, zip-1, zip-2]	668	1,060,006	83.08%	89.28%	86.07%	6,717	379,379	93.93%	96.16%	95.03%
[street-1, street-4, zip-0, zip-2]	670	1,129,843	83.33%	88.58%	85.87%	6,731	545,091	94.12%	94.49%	94.30%
[familyname-0, familyname-1, title-3, zip-1, zip-2]	661	1,060,663	82.21%	89.28%	85.60%	6,736	993,987	94.19%	89.95%	92.02%
[familyname-0, familyname-1, title-2, zip-1, zip-2]	661	1,060,666	82.21%	89.28%	85.60%	6,727	1,241,669	94.07%	87.45%	90.64%
[familyname-0, familyname-1, title-4, zip-1, zip-2]	661	1,060,668	82.21%	89.28%	85.60%	6,717	388,312	93.93%	96.07%	94.99%

Table 5: Comparison of the key figures for the first 10 best blocking keys in Corporate-1 (training) applied on Places.