A Randomized Blocking Structure for Streaming Record Linkage

Dimitrios Karapiperis  
International Hellenic University  
Greece  
dkarapiperis@ihu.edu.gr

Christos Tjortjis  
International Hellenic University  
Greece  
c.tjortjis@ihu.edu.gr

Vassilios S. Verykios  
Hellenic Open University  
Greece  
verykios@eap.gr

ABSTRACT
A huge amount of data, in terms of streams, are collected nowadays via a variety of sources, such as sensors, mobile devices, or even raw log files. The unprecedented rate at which these data are generated and collected calls for novel record linkage methods to identify matching records pairs, which refer to the same real-world entity. Towards this direction, blocking methods are used in order to reduce the number of candidate record pairs while still maintaining high levels of accuracy. This paper introduces ExpBlock, a randomized record linkage structure, which guarantees that both the most frequently accessed and recently used blocks remain in main memory and, additionally, the records within a block are renewed on a rolling basis. Specifically, the probability of inactive blocks and older records to remain in main memory decays in order to make room for more promising blocks and fresher records, respectively. We implement these features using random choices instead of utilizing cumbersome sorting data structures in order to favour simplicity of implementation and efficiency. We showcase, through the experimental evaluation, that ExpBlock scales efficiently to data streams by providing accurate results in a timely fashion.

PVLDB Reference Format:
doi:10.14778/3611479.3611487

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/dimkar121/ExpBlock.

1 INTRODUCTION
The vast amount of records which is collected using several sources need first to be combined and cleaned before undergoing any data analysis task. Assume for example a stream of records that originate from various hospitals or healthcare centers which send data regarding symptoms and medication of individuals to a central national repository. These records are, then, integrated in order to identify possible matches among the underlying individuals, who should be alerted in cases of dangerous contagious diseases. Assume also a central system that collects passenger records from airline companies with the aim to match these records with criminal suspects, whose records originate from a criminal or a forensic database. This suite of algorithms that perform this kind of integration tasks belong to the family of record linkage algorithms, which lie at the core of data integration and analysis.

The area of record linkage has received a lot of attention during the last decades due to its numerous real-world applications. In the literature, a plethora of such methods have been proposed for identifying similar pairs of records [2, 4]. A record linkage task involves two distinct phases, namely the blocking and the matching phase. During blocking, the algorithm formulates blocks, which aim to group together records that share some common characteristics of certain attributes, such as the initial letters of surname, or the same zip code. The purpose of blocking is to reduce the quadratic complexity $n^2$ of the number of possible evaluations of record pairs, during the matching phase, that originate from a pair of data sources. We refer the interested reader to [3] which surveys exact and approximate blocking algorithms.

The task of record linkage given a stream of records has its own characteristics, challenges, and features. Due to the velocity and high volume of records, it is infeasible and unrealistic to match a query record over all those records that have been processed from the initialization of the stream [21]. In fact, this can be also useless, since a real-time integration focuses on fresh records in terms of time. All the works presented thus far [1, 6, 13, 14, 16], in this context, maintain a bounded number of blocks in main memory, which should be continuously renewed by evicting older and unused blocks (or records) in favour of fresh incoming records that belong to new blocks. Gazzari and Herschel in [6] use techniques such as block pruning and elimination of certain comparisons, which are based solely on empirical estimations and heuristics that lack completely any theoretical ground. For example, the authors propose a block pruning mechanism that (a) discards whole oversized blocks, and (b) ignores, during matching, record pairs which originate from blocks that are significantly larger than the smallest block found therein, in terms of the number of records. However, this abrupt discontinuation of blocks and comparisons may affect negatively the accuracy of the results. It is not also practical in streaming settings to specify an oversized block by simply estimating a parameter, because if an estimate is poorly made, a considerable memory overhead may occur. Araujo et al. [1] in order to reduce the memory consumption apply a time window during blocking so that only the most recent records remain in main memory, discarding blindly records, which fall out of the window but might be good candidates to formulate matching pairs.

In this work, we introduce ExpBlock, a randomized record linkage streaming algorithm, which utilizes an in-memory blocking structure that maintains those blocks that are both the most frequently accessed and most recently used. ExpBlock calculates the potential future access of a block, based both on the frequency of access and its period of inactivity, where the probability of inactive
blocks to remain in the blocking structure decreases as these blocks remain in main memory. Additionally, ExpBlock renews the records stored in each block on a rolling basis, where the probability of a record to remain in a block exponentially decays, in order to make room for fresh records. ExpBlock is exempt from auxiliary data structures or cumbersome sorting operations, as it relies on simple randomized techniques to evict blocks as well as records. Our experimental evaluation against three state-of-the-art methods on three real-world data sets indicate that ExpBlock achieves high levels of recall and extremely fast running times.

Our work makes the following contributions:

- It proposes a randomized algorithm to maintain in main memory the most promising blocks, according to the frequency of access and their period of inactivity.
- It introduces an algorithm to renew the records stored in each block on a rolling basis, where the probability of a record to remain exponentially decays in order to make room for fresher records.

The rest of this paper is structured as follows: Section 2 discusses the related work, while Section 3 provides the problem statement. Our proposed data structure is described in detail in Section 4. Section 5 presents our experimental evaluation. Conclusions are discussed in Section 6.

2 RELATED WORK

Various record linkage methods have been developed for online [9–13], progressive [5, 7, 15, 20, 22, 24], incremental [8], and topic-aware [21] settings. Progressive record linkage focuses on how to identify matching record pairs upfront the linkage process. The main idea of incremental record linkage is to identify matching record pairs in the presence of new records utilizing the available data structures used for blocking without repeating already executed comparisons. The online settings aim to link vast amounts of records in near real-time, whereas the progressive settings provide results upfront by focusing on record pairs that are more likely to be matches. Although, these methods must operate in a computationally and memory efficient manner, in order to provide accurate results very fast, they do not assume an unbounded number of incoming records. Their corresponding data structures are usually supported by a persistent data store, which aims to provide a complete result set including all possible matching record pairs with respect to a query record.

In the area of streaming record linkage, only a few works [1, 6, 13, 14, 16] have been introduced in the literature thus far. Karapikeris et al. [13] proposed a method to handle record linkage of streams of records, where the number of blocks might unexpectedly grow considerably. [13] bounds the number of blocks that are maintained in main memory and applies an eviction strategy to accommodate newly arrived blocking keys from the stream, when there are no empty slots. Nevertheless, a block due to a short period of inactivity might be immediately discarded, although it might have been frequently accessed. A privacy-preserving extension of [13] is presented in [16].

In [14], the authors propose another streaming algorithm, which maintains in main memory the most frequently accessed blocks for fast computations, in order to minimize any accesses to secondary storage. This approach is more efficient than the one presented in [13] because the algorithm does not rely on auxiliary data structures during the eviction process. However, this method suffers from scalability issues, because it does not take into consideration the growth of the number of records in each block, which might be partially stored in secondary storage.

Gazzari and Herschel in [6] assume the creation of a redundant block collection, where each record appears in more than one block depending on its tokens used as blocking keys. If the number of blocked records exceed a user-defined threshold, then this block is discontinued and, moreover, its blocking key is added to an in-memory quarantine list, so that if any records emerge that exhibit that key will be immediately discarded. Setting though the maximum admissible block size, without any other quality criterion, will incur a negative impact in the accuracy of the result set. During matching, the authors group the pairs per record, and discard those pairs that do not appear in an adequate number of blocks by setting the average count per group to be the corresponding threshold. Although, the aim of these techniques is to keep in main memory as fewer records as possible, the lack of theoretical grounds may lead to reduced accuracy and poor scalability.

Araujo et al. in [1] propose a schema-agnostic approach that relies on meta-blocking [18, 19], which investigates how to restructure the generated blocks with the aim of discarding any redundant comparisons. Their proposal for managing high volumes of records is a time window technique, which maintains the most recent records in main memory, to reduce memory consumption. However, the authors do not take into account the number of blocks which is unbounded and may lead to a memory overflow. This technique also suffers from low recall because the only criterion for discarding a record is its admission time into main memory.

3 PROBLEM STATEMENT

In this section, we define the problems and provide some definitions that have led to the development of ExpBlock.

Assume a pair of data sets A and B whose records are created in real time and formulate a data stream. Each of these records that belong to A (or B) should undergo a very fast resolution process to deliver a subset of its matching records in B (or A), which have appeared previously. The results do not need to be complete, because in streaming settings, one is usually more interested in getting very fast only a subset of the results.

There are two main problems that arise throughout a record linkage process, involving streams of records. The first problem is the very large number of blocking keys, whose storage in an unbounded inverted index would be problematic or even infeasible. A smart way is required so as to keep the most frequently used blocks in memory, and discard the ones, whose usage is less frequent. The second problem is how one can maintain a representative set of blocked records. The term ‘representative’ boils down to maintaining both newly inserted and older records, without resorting to any auxiliary data structures for storing the history of these records.

We use an in-memory data structure T, whose details are found in Section 4, to implement the resolution process. T refers to a certain block of T (the one with the blocking key k). We also use

---

1 The inverted index is unbounded in terms of the number of slots
The average number of hits per block is denoted by \( h \). The average number of hits per block as \( \frac{h}{n} \) should be measured based on both its hits \( h \) and its period of inactivity in \( T \).

The Average Number of Hits and Period of Inactivity

Our eviction strategy favours the most frequently and recently used blocks by minimizing the corresponding probability of choosing such blocks for eviction. For this purpose, ExpBlock maintains the number \( n_k \) of hits of each block seen so far during the current round, the global summation \( n = \sum_{k=1}^{b} n_k \), as well as the number of the last round that each block has been accessed. These are the only additional pieces of information, apart from the records and their blocking keys, maintained in \( T \).

Before the eviction process of round \( r \), ExpBlock calculates the average number of hits per block as \( \alpha = \left[ \frac{n}{b} \right] \). Then, it chooses uniformly at random a block from \( T \) and calculates its activity \( y_k \), normalized in the interval \((0, 1]\) as:

\[
y = \frac{r'}{r},
\]

where \( r' \) denotes the last round this block had been accessed due to an arrival of an incoming record, and always holds that \( r \geq r' \). Thus, \( y \) for active blocks will tend to 1, while for inactive blocks it will tend to 0. Using \( y \) and \( \alpha \), ExpBlock measures the performance \( \beta \) of a \( T_k \), termed as the degree of its potential future access, as:

\[
\beta = \left\lfloor \frac{n_k y}{\alpha} \right\rfloor.
\]

### 4.1 Eviction of blocks

Since \( T \) is bounded by \( b \), which is the maximum number of blocks, an eviction strategy should be applied in case of a miss. Therefore, whenever \( T \) is fully occupied and an incoming record should be placed in a block that does not exist in \( T \), ExpBlock should free some space by discarding some blocks, which exhibit both some period of inactivity and fewer hits compared to some other more active blocks. Although, the choice of these blocks is random, the corresponding probability will be rigorously quantified.

Our eviction strategy favours the most frequently and recently used blocks by minimizing the corresponding probability of choosing such blocks for eviction. For this purpose, ExpBlock maintains the number \( n_k \) of hits of each block seen so far during the current round, the global summation \( n = \sum_{k=1}^{b} n_k \), as well as the number of the last round that each block has been accessed. These are the only additional pieces of information, apart from the records and their blocking keys, maintained in \( T \).

Before the eviction process of round \( r \), ExpBlock calculates the average number of hits per block as \( \alpha = \left[ \frac{n}{b} \right] \). Then, it chooses uniformly at random a block from \( T \) and calculates its activity \( y_k \), normalized in the interval \((0, 1]\) as:

\[
y = \frac{r'}{r},
\]

where \( r' \) denotes the last round this block had been accessed due to an arrival of an incoming record, and always holds that \( r \geq r' \). Thus, \( y \) for active blocks will tend to 1, while for inactive blocks it will tend to 0. Using \( y \) and \( \alpha \), ExpBlock measures the performance \( \beta \) of a \( T_k \), termed as the degree of its potential future access, as:

\[
\beta = \left\lfloor \frac{n_k y}{\alpha} \right\rfloor.
\]

---

2For brevity, we drop subscript \( k \), which denotes block \( k \).
In doing so, the number of hits $n_k$ is weighted by the block’s activity factor, which acts as the counterbalance of the $n_k$; more specifically, if a block exhibits a large number of hits, but has not been accessed during the last rounds, then ExpBlock downgrades its degree of potential future access due to its reduced activity factor. On the other hand, if a block has been accessed during the current round, then its activity factor would be 1, because $r = r'$. If $\beta = 0$, then this block is discarded. Otherwise, the corresponding block remains in $T$, but is penalized by reducing its number $n_k$ of hits by $\alpha$. This eviction process continues until a certain ratio $\xi$, where $\xi < 1$, of slots has been released.

Using this scheme, we give another chance of survival to these blocks that exhibit a large number of recent hits, and simultaneously discard the blocks that have not been recently accessed, although they might exhibit large number of hits in the past.

**Lemma 4.1.** The probability of evicting a block, symbolized by $E_k$, is inversely proportional to its degree $\beta$ of potential future access:

$$\Pr(E_k) = \frac{1}{b^\beta r^\gamma}. \quad (3)$$

**Proof.** Assume that block $T_k$ exhibits $\alpha < n_k < 2\alpha$ hits, some of which have occurred during the the current round. So, the degree of its potential future access would be $\gamma = 1$. Although the probability of this block to be initially chosen is $\frac{1}{b^\gamma r^\gamma}$, it would not be immediately discarded, if chosen, because $1 < \beta < \frac{2\alpha}{\alpha} = 2$. Applying the floor function, $\beta$ becomes equal to 1. Hence, ExpBlock would grant another chance to this $T_k$, which will remain in $T$ with a reduced number $n_k$ of hits. Consequently, the probability to be chosen again, and be evicted, is $\frac{1}{b}$. Therefore, the probability of eviction for this block is $\Pr(E_k) = \frac{1}{b^\beta r^\gamma}$. By induction, we conclude the proof of the lemma. $\square$

Since $\xi$ is a very small constant, e.g., $\xi = 0.05$, the lower bound is quite tight. If $\beta \leq 0$ for a block, then its probability of eviction is merely $\frac{1}{b}$. The greater the number of hits of a block, the less its chance of eviction.

Algorithm 1 showcases the eviction of blocks, which exhibit low degree of potential future access. Initially in line 2, ExpBlock calculates the average number $\alpha$ of hits per block for the current round. Then, in line 4 the main loop begins, which terminates when a certain percentage $\xi$ of slots have been released. Inside the loop, a block is chosen uniformly at random to investigate its degree of potential future access. First, ExpBlock calculates its activity $\gamma$ (line 6), and then its degree of potential future access $\beta$ (line 7) by considering its hits. If $\beta$ is less than zero, then the chosen block is discarded and the corresponding slot is released (line 8). Otherwise, ExpBlock reduces its number $n_k$ of hits by $\alpha$ (line 12). In this way, ExpBlock gives another chance of survival to this block, due to its high selectivity by the incoming records.

**Algorithm 1 Eviction of blocks**

1. if ($T$.emptySlots = 0) then $\triangleright$ If a requested block is not found in $T$ and there are no slots available, then the eviction process is initiated.
2. $\alpha \leftarrow \left\lfloor \frac{n}{b} \right\rfloor \triangleright \alpha$ is the average number of hits per block.
3. $v \leftarrow 0$
4. while ($v \leq \left\lfloor \xi b \right\rfloor$) do $\triangleright$ The eviction process continues until a ratio $\xi$ of blocks has been discarded.
5. $i \leftarrow \text{Random}(1, b)$ $\triangleright$ Function $\text{Random}()$ uses a pregenerated sequence of random integers.
6. $T[i].\gamma \leftarrow \frac{T[i].\gamma'}{T[i].\gamma'}$ $\triangleright$ A random block is chosen to compute its activity $\gamma$.
7. $T[i].\beta = \frac{T[i].n_k}{\alpha}$ $\triangleright$ Then, the degree of its potential future access $\beta$ is computed.
8. if ($T[i].\beta = 0$) then
9. $T$.delete(i) $\leftarrow$ null $\triangleright$ The $i$-th block is discarded.
10. $v \leftarrow v + 1$
11. else
12. $T[i].n_k = T[i].n_k - \alpha \triangleright$ Its current number of hits $n_k$ is reduced by $\alpha$.
13. $r \leftarrow r + 1$
14. $T$.emptySlots $\leftarrow v$

<table>
<thead>
<tr>
<th>blocking key</th>
<th>$n_k$</th>
<th>$r'$</th>
<th>records</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1$</td>
<td>10</td>
<td>3</td>
<td>rec1, rec2, rec3</td>
</tr>
<tr>
<td>$k_2$</td>
<td>8</td>
<td>5</td>
<td>rec1, rec2, rec3</td>
</tr>
<tr>
<td>$k_3$</td>
<td>5</td>
<td>2</td>
<td>rec1, rec2, rec3</td>
</tr>
</tbody>
</table>

Table 1: A snapshot of $T$, with $b = 3$ and $w = 3$, that includes 3 blocking keys, when a miss occurred during the current round $r = 5$, which initiates the eviction process. Block $k_3$, although exhibits the largest number of hits $n_k = 10$, its most recent round that it has been accessed is $r' = 3$, which downgrades its degree $\beta$ of potential future access. In contrast, the current activity of $k_2$ combined with its number of hits results grant its survival. Block $k_2$ exhibits both a small number of hits and a long period of inactivity.

Algorithm 1 either will be evicted. If ExpBlock chooses $k_2$, then it will reduce $n_2$ by $\alpha$ and will grant $k_2$ another chance of survival.

In a near uniform distribution, the probability of a hit is $\frac{b}{n}$ and each block would exhibit the same probability of eviction, which is $\frac{b}{n} \leq \Pr(E_k) \leq \frac{1}{b}$. The only important point would be whether a block has been accessed during the current round or not, which would vary the value of $\beta$. In a skewed distribution of hits, some popular blocks, which are those with the highest degree $\beta$ of potential future access, will be found in $T$ with higher probability than some other blocks with fewer hits. Specifically, as Lemma 4.1 suggests, the probability of not evicting a block is $1 - \frac{1}{b^{\beta r^\gamma}}$, which is also the probability of a block to remain. This skew would result in less evictions, at the expense of a negligible overhead; ExpBlock might perform additional random tosses in order to choose less popular blocks to evict, when it is required to free some slots. The degree of potential future access of each block reflects the skew
of the distribution of hits. Although, the probability of a hit in a skewed distribution is also $\frac{b}{a}$, because a block occupies a single slot in $T$, the probability of evicting a popular block depends on $\beta$ according to Lemma 4.1.

The running time depends on the distribution of hits in the blocks. Therefore, in a near uniform distribution, the running time is $O(\zeta \times b)$. In a skewed distribution, by assuming $\zeta < b$, as the number of popular blocks, then the running time is $O(\zeta + (\zeta \times b))$. This small overhead during the eviction process pays dividends in minimizing the total blocking time, because by finding a popular block in $T$, we skip an eviction process.

### 4.2 Eviction of records within a block

Each block hosts a linked list of $w$ positions in order to store the corresponding blocked records. When these positions of a block are occupied, and an incoming record should be inserted, ExpBlock initiates the renewal process of records blocked therein.

During this process, ExpBlock essentially performs a Bernoulli process for the survival of each record with a fixed probability $p$. This process is memoryless, which means that there is no mechanism that remembers the outcome of the previous renewal process for a record that has survived.

Let us assume that there were $t$ such renewal processes thus far. The probability of a set of records, that had occupied a block, to survive from these $t$ processes, exponentially decays in order to make room for newer records to remain. This is achieved without utilizing any additional data structure to indicate the age of records or keep any other kind of tracking history.

**Lemma 4.2.** The probability of finding a record from the current round in $T$ is $\frac{b}{a} p$.

**Proof.** During a renewal process, each record of a block remains with probability $p$. After $t$ such renewal processes, each of the blocked records would survive with probability $p^t$. Thus, by taking into account the probability of finding a block in $T$, which is $\frac{b}{a}$, and by assuming independence, we conclude the proof. □

Therefore, during a renewal process, $p$ of the blocked records are expected to survive, while $p^2$ of the survived ones will undergo an additional renewal process and so forth.

We derive the number $w$ of positions of blocks according to the following lemma:

**Lemma 4.3.** By assuming $w = \left\lceil \frac{3 \ln(2/\delta)}{q e^2} \right\rceil$ positions, where $q$ is the probability of eviction, $\delta < 1$, and $\epsilon < 1$, the total number of evictions in each round is within $(1 \pm \epsilon)wq$ with probability at least $1 - \delta$.

**Proof.** Let $\psi_i$ represent the outcome of $\{0, 1\}$ of an attempt to evict a record with success probability $q = 1 - p$. Also, let $S = \sum_{i=1}^{w} \psi_i$, which is binomially distributed with expected value $wq$. By using the following Chernoff bound:

$$\Pr (|S - wq| > \epsilon wq) \leq 2e^{-\epsilon wq^2} < \delta = w > \frac{3 \ln(2/\delta)}{q e^2},$$

where $\epsilon < 1$ is a small multiplicative error factor and $\delta$ is a probability bound, we derive $w$ to ensure that $(1 - \epsilon)wq \leq S \leq (1 + \epsilon)wq$ with probability at least $1 - \delta$, which concludes the proof. □

If ExpBlock would have used a deterministic approach to evict the records, which would have required detailed tracking of the records inserted, then all the current records of a block in any point of time would have been discarded after exactly $\log_2(w) + 1$ successive evictions. Since ExpBlock relies on random choices, the probability of the total eviction of a set of records found in a block depends on $p$, as the following lemma suggests.

**Lemma 4.4.** The probability of the total eviction of a set of records, symbolized by $T$, is inversely proportional to the probability of survival $p$.

**Proof.** The probability of a record to remain after $\log_2(1/p) + 1$ renewal processes is $p^{\log_2(1/p) + 1}$, hence the probability of eviction for $w$ such records is $(1 - p^{\log_2(1/p) + 1})^w$. Therefore,

$$\Pr(T) = \left(1 - p^{\log_2(1/p) + 1}\right)^w = \left(1 - \frac{p}{w}\right)^w \approx \frac{1}{w^p e}$$

using the fact that $e^{-x} \geq 1 - x$, which concludes the proof. □

### 5 EXPERIMENTAL EVALUATION

We evaluated ExpBlock, termed as EXP, against TASK[6], TIME[1], and UNI[14]. All the baselines are discussed in Section 2 and were evaluated in terms of the recall and precision rates achieved as well as the time consumed to perform blocking and matching. The recall is the ratio of the number of matching record pairs that were correctly identified to the total number of matching record pairs that originally existed. The precision is the ratio of the number of matching record pairs that were correctly identified to the total number of comparisons. The results of both metrics lie in the interval $[0,1]$, where higher values indicate better performance.

**Data sets.** We have used three real-world data sets, namely (a) NCVR, which includes a registry of voters, (b) DBLP, which comprises bibliographic records between 2013 and 2020 from the domain of computer science, and (c) Open Academic Graph (OAG)[23], which includes includes details of multidisciplinary academic papers from Microsoft Academic Graph which has been paired with Aminer. For NCVR and DBLP, each record from the original data sets, tagged as $A$, was chosen with probability $\frac{1}{2}$, in order to generate 2 perturbed records to populate the counterpart data sets, tagged as $B$. For OAG, whose ground truth was available, we extracted 1M records from Microsoft Academic Graph, tagged as $A$, and 1M records from AMiner, tagged as $B$, which resulted in almost 300K matching entities. The attributes that played the role of the blocking keys were the concatenation of the last name and PO Box, for NCVR, and the name of the first author and the year of publication for OAG and DBLP. Table 2 summarizes the characteristics of the data sets used throughout the experimental evaluation.

**Implementation.** We applied MinHash[17] with murmur hashing to the initially formulated blocking keys to facilitate approximate matching. Our method, though, is orthogonal to the blocking/matching phases, hence, any such algorithm can be used. We performed the experiments using a virtual machine with a Xeon

---

*In this case, we have assumed $\frac{1}{2}$ as the rate of eviction.*


[2] https://dblp.org/xml

Table 2: \(A\) and \(B\) are the data sets to be linked, \(MP\) is the set of the truly matching record pairs, and \(G\) is the Cartesian product of \(A\) and \(B\).

<table>
<thead>
<tr>
<th></th>
<th>OAG</th>
<th>NCVR</th>
<th>DBLP</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>A</td>
<td>)</td>
<td>1M</td>
</tr>
<tr>
<td>(</td>
<td>B</td>
<td>)</td>
<td>1M</td>
</tr>
<tr>
<td>(</td>
<td>MP</td>
<td>)</td>
<td>300K</td>
</tr>
<tr>
<td>(</td>
<td>G</td>
<td>)</td>
<td>1T</td>
</tr>
<tr>
<td>#blocks</td>
<td>678K</td>
<td>960K</td>
<td>3.5M</td>
</tr>
</tbody>
</table>

Table 3: Average time of an eviction process (in millis) and blocking time (in secs) across all data sets using random choices (RND), iterative scanning (ITR), and sorting (SRT).

<table>
<thead>
<tr>
<th></th>
<th>(10^3)</th>
<th>(10^4)</th>
<th>(10^5)</th>
<th>(10^6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RND(^1)</td>
<td>0.9</td>
<td>40.1</td>
<td>9.2</td>
<td>35</td>
</tr>
<tr>
<td>ITR</td>
<td>1.2</td>
<td>43.5</td>
<td>13.5</td>
<td>39</td>
</tr>
<tr>
<td>SRT</td>
<td>12</td>
<td>89</td>
<td>130</td>
<td>77.6</td>
</tr>
</tbody>
</table>

\(^1\)Time of an eviction process vs blocking time

Table 4: Comparing total time (in seconds) and recall between EXP, standard blocking (SB), and sorted neighbourhood (SN).

<table>
<thead>
<tr>
<th></th>
<th>NCVR</th>
<th>OAG</th>
<th>DBLP</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXP</td>
<td>87</td>
<td>0.76</td>
<td>89</td>
</tr>
<tr>
<td>SB</td>
<td>960</td>
<td>0.84</td>
<td>1025</td>
</tr>
<tr>
<td>SN</td>
<td>760</td>
<td>0.87</td>
<td>820</td>
</tr>
</tbody>
</table>

Experimental results. First, we measured the clock time needed for blocking and matching by increasing exponentially the available number \(b\) of slots of \(T\). Figure 1a shows the corresponding time measured in terms of seconds. We initially observe a steep drop up to \(10^5\) slots, where we achieve nearly 15% reduction of time, because of the availability of the requested blocks in main memory, which results in fewer evictions. By further increasing the exponent, the earnings are limited to 2% – 4% of time reduction due to the higher memory overhead caused by the larger number of slots of \(T\). We also observe in Figure 1b a linear increase of the recall rates for all data sets as the number of slots increases. The slopes though of these lines is low, which indicates that the earnings of recall are not great, because even with \(b = 10^3\) slots, ExpBlock achieves a rate almost equal to 0.75. Any further increments of the exponent will incur diminishing earnings in terms of the recall.

We, also, evaluated the clock time of the eviction process of slots using our scheme, which is based on random choices (RND), iterative scanning (ITR), and quick sort (SRT) in order to calculate \(\beta\) and evict low-performing slots from \(T\). We fixed \(\xi = 0.1\) and increased the number of slots exponentially to obtain the results quoted in Table 3. We observe that RND scores on average 30% faster times than ITR and both are 10 times faster than SRT. We also note that using OAG, whose keys follow a near uniform distribution, ITR almost exhausts all slots in each iteration, which results in more running time, until completing the eviction process. In contrast, although RND is only slightly affected by a skewed distribution, it eventually accelerates the total blocking time, since popular blocks are found in \(T\).

Comparing with popular offline methods like standard blocking (SB) and sorted neighbourhood (SN) [3], we highlight the level of efficiency achieved by EXP. SB creates blocks and compares only the records within a block, while SN sorts the records and, then, sequentially moves a window of a fixed number of records over these records to perform the comparisons. These offline methods assume finite data sets, which can be managed, not always efficiently though, by the available computational resources. The clock time of both SB and SN is one order of magnitude worse than EXP, as clearly shown in Table 4. We did not obtain any results of DBLP for both SB and SN, because both crashed during blocking. On the other hand, the recall rates of the offline methods are on average 16% higher than EXP. In conclusion, EXP sacrifices some recall for the sake of speed using a small footprint of main memory.

We proceeded the evaluation process by comparing the performance of ExpBlock with our competitors. In this set of experiments, we had set \(b = 1000\) slots, \(\epsilon = 0.1\), and \(\delta = 0.1\), which resulted in \(w = 1349\) positions for creating \(T\).

We, then, measured the blocking time using different levels of skew, which has also been demonstrated in [14]. We have used the same synthetic data set, which relies on NCVR, that contains blocking keys, whose number of records follows a Zipf distribution of a certain skew specified by the exponent \(z\). Thus, the size of each block \(i = 1, 2, 3, \ldots, n\), in terms of the number of records, is proportional to \(i^{-z}\). We, then, measured, for each method, the time needed for conducting the blocking phase. Figure 2a shows the performance of each method, where we observe that UNI and EXP exhibit linearly decreased running times, much lower than TIME and TASK, as the skew increases, \(z = \{2, 3\}\); specifically EXP is on average 95% and 89% faster than TASK and TIME, respectively. This indicates that the number of evictions is smaller when skew is higher, which consequently results in shorter blocking times. EXP also maintains a steady gap from UNI, namely 33%. UNI, although...
deals with skew, it fails to be faster than EXP, because it conducts much restructuring, which is due to the early eviction of some blocks, which remained inactive for a short period but were immediately needed to host incoming records. For TASK and TIME, by observing their linear times, we concluded that the skew of blocking keys does not play any role in their mechanisms. This behavior is explained by the fact that both TIME and TASK either prune or quarantine the oversized, yet popular, blocks to save running time.

We also tested the performance of EXP and its competitors using OAG which exhibits some mild skew with dominating authors ‘Helmut Herrmann’, ‘Wei Wang’, ‘Richard J. Lewis’, and ‘Cristiano da Silva Teixeira’. Figure 2b displays the results, where we observe almost linear times but EXP exhibits the lowest slope, as the number of records increases, among all methods. On the other hand, TASK quarantines the oversized blocks but simultaneously retains in main memory all blocks, which might remain inactive for long periods of time. These blocks along with the auxiliary data structures, maintained in main memory, caused a severe bottleneck that resulted in costly running times. UNI utilizes the secondary storage for keeping the records of the oversized blocks, which incurs serious time delays to retrieve the corresponding records. In conclusion, EXP scores on average 28% (until 1M records), 81%, and 40% lower times than TASK, UNI, and TIME, respectively. The totally different distributions of blocks and records for OAG accounts for the faster times, as Figure 2d indicates. Blocking OAG resulted in a larger number of blocks than NCVR, but simultaneously, in fewer records in each block, which accelerated the resolution times.

We, then, evaluated the performance of each method. Figures 3a and 3b report the cumulative recall rates for both NCVR and OAG by varying the number of records. We immediately observe the increasing linear performance of EXP, which reaches almost 75%, despite its selectivity, in both data sets. This rate is achieved by...
Although, all methods start with exceptional rates after 10 almost linear performance, with a slight downward slope, as we observe in Figures 4a and 4b, EXP, TIME, and TASK display an interesting pattern of superiority, which is attributed to its effective renewal of records employed the same matching conditions for evaluating a record classification of many missed matching pairs.

mainly due to the unbounded growth of its blocks, which incurs costs in terms of the recall rates as shown in Figures 3c and 3d for NCVR and OAG, respectively. Some blocks that host the initial set of records are unconditionally evicted due to time constraints, which results in abrupt drops of the recall rates in each successive round. In contrast, EXP strikes a nice balance between time constraints and popularity of blocks and essentially manages to maintain in main memory those blocks that are expected to gather more hits. Although, all methods start with exceptional rates after 10 rounds, UNI and TIME drop their rates considerably as the whole process progresses. EXP increases the gap from its competitors after 10 and 40 rounds by 21% and 41% on average, respectively. Another important factor that accounts for the superiority of EXP is its renewal process of records within each block. This process guarantees that each block contains fresh and older records following the randomization scheme discussed in Section 4.2.

Precision focuses on the ability of a method to reduce efficiently the comparison space. In our experiments, although all methods employed the same matching conditions for evaluating a record pair, the precision rates exhibit some interesting findings. As we observe in Figures 4a and 4b, EXP, TIME, and TASK display an almost linear performance, with a slight downward slope, as the number of records increases. EXP, though, maintains a small margin of superiority, which is attributed to its effective renewal of records of each block. In contrast, UNI struggles with the precision rates, mainly due to the unbounded growth of its blocks, which incurs the classification of many missed matching pairs.

Lastly, we measured the recall, precision, and the elapsed time to block and match the records of the largest data set DBLP. EXP and UNI exhibited the same pattern of performance in recall as previously reaching almost 0.73 and 0.67, respectively, as Figure 5a shows. The precision rates, which are depicted in Figure 5b, of EXP and TIME was linear with a slight negative slope maintaining a large margin from UNI. Regarding clock time, EXP achieved robust performance, which was due to its bounded number of slots and block positions that kept the running time constant. For blocking, EXP and UNI performed almost equivalently, as Figure 5c suggests, while TIME, as the number of records was increasing, its response time was dramatically growing mainly due to the large number of generated blocks that remained in main memory. In matching, we verified EXP’s stability to manage large volumes of records, which reported around 350 seconds per 4M records, as Figure 5d indicates. We do not report the measurements of TASK, which crashed upfront due to a memory overflow.

6 CONCLUSIONS
In this paper, we presented ExpBlock, a randomized streaming record linkage structure, where simplicity lies at the core of its mechanism. ExpBlock relies its operation on random choices, without using any cumbersome sorting operations or utilizing complex data structures. Throughout the experiments, we concluded that by increasing the number of slots \( b \), the running time does not improve proportionally, because the memory overhead caused by the size of the data structure in main memory poses some time delay. We additionally observed that the recall rates do not exhibit any remarkable increase either. However, we have to note that the recall depends mainly on the point of time at which the corresponding records will potentially arrive. If these records are far apart, in terms of time, then the recall will be negatively affected. The skew of the blocking keys is another factor that favours the running time of EXP, since the number of evictions is smaller, which consequently results in faster blocking times. To sum up, EXP sacrifices some recall for the sake of speed using a small footprint of main memory to maintain fruitful blocks and records.

ACKNOWLEDGMENTS
This research is supported by the European Social Fund-SF in the context of the project “Support for International Actions of the International Hellenic University”, (MIS 5154651).
REFERENCES