I/O-Efficient Similarity Join*

Rasmus Pagh[1], Ninh Pham[1], Francesco Silvestri[1]**, and Morten Stöckel[2]***

[1] IT University of Copenhagen, Denmark
{pagh,ndap,fras}@itu.dk
[2] University of Copenhagen, Denmark
most@di.ku.dk

Abstract. We present an I/O-efficient algorithm for computing similarity joins based on locality-sensitive hashing (LSH). In contrast to the filtering methods commonly suggested our method has provable sub-quadratic dependency on the data size. Further, in contrast to straightforward implementations of known LSH-based algorithms on external memory, our approach is able to take significant advantage of the available internal memory: Whereas the time complexity of classical algorithms includes a factor of N^ρ, where ρ is a parameter of the LSH used, the I/O complexity of our algorithm merely includes a factor $(N/M)^\rho$, where N is the data size and M is the size of internal memory. Our algorithm is randomized and outputs the correct result with high probability. It is a simple, recursive, cache-oblivious procedure, and we believe that it will be useful also in other computational settings such as parallel computation.

1 Introduction

The ability to handle noisy or imprecise data is becoming increasingly important in computing. In database settings this kind of capability is often achieved using similarity join primitives that replace equality predicates with a condition on similarity. To make this more precise consider a space $\mathbb{U}$ and a distance function $d : \mathbb{U} \times \mathbb{U} \to \mathbf{R}$. The *similarity join* of sets $R, S \subseteq \mathbb{U}$ is the following: Given a radius r, compute the set $R \bowtie_{\leq r} S = \{(x, y) \in R \times S \mid d(x, y) \leq r\}$. This problem occurs in numerous applications, such as web deduplication [3, 15], document clustering [4], data cleaning [2, 6]. As such applications arise in large-scale datasets, the problem of scaling up similarity join for different metric distances is getting more important and more challenging.

Many known similarity join techniques (e.g., prefix filtering [2, 6], positional filtering [15], inverted index-based filtering [3]) are based on *filtering* techniques that often, but not always, succeed in reducing computational costs. If we let

* The research leading to these results has received funding from the European Research Council under the EU 7th Framework Programme, ERC grant agreement no. 614331.
** In part supported by University of Padova project CPDA121378 and by MIUR of Italy project AMANDA while working at the University of Padova.
*** This work was done while at IT University of Copenhagen. Supported by the Danish National Research Foundation / Sapere Aude program and VILLUM FONDEN.

$N = |R| + |S|$ these techniques generally require $\Omega(N^2)$ comparisons for worst-case data. Another approach is *locality-sensitive hashing* (LSH) where candidate output pairs are generated using collisions of carefully chosen hash functions. The LSH definition is as follows.

Definition 1. *Fix a distance function $d : \mathbb{U} \times \mathbb{U} \to \mathbf{R}$. For positive reals r, c, p_1, p_2, and $p_1 > p_2, c > 1$, a family of functions $\mathcal{H}$ is* (r, cr, p_1, p_2)-sensitive *if for uniformly chosen $h \in \mathcal{H}$ and all $x, y \in \mathbb{U}$:*

- *If $d(x, y) \leq r$ then $Pr[h(x) = h(y)] \geq p_1$;*
- *If $d(x, y) > cr$ then $Pr[h(x) = h(y)] \leq p_2$.*

We say that $\mathcal{H}$ is monotone if $Pr[h(x) = h(y)]$ is a non-increasing function of the distance function $d(x, y)$.

LSH is able to break the N^2 barrier in cases where for some constant $c > 1$ the number of pairs in $R \bowtie_{\leq cr} S$ is not too large. In other words, there should not be too many pairs that have distance within a factor c of the threshold, the reason being that such pairs are likely to become candidates, yet considering them does not contribute to the output. For notational simplicity, we will talk about *far* pairs at distance greater than cr (those that should not be reported), *near* pairs at distance at most r (those that should be reported), and *c-near* pairs at distance between r and cr (those that should not be reported but affect the I/O cost).

In this paper we study I/O-efficient similarity join methods based on LSH. That is, we are interested in minimizing the number of I/O operations where a block of B points from $\mathbb{U}$ is transferred between an external memory and an internal memory with capacity for M points from $\mathbb{U}$. Our main result is the first *cache-oblivious* algorithm for similarity join that has *provably* sub-quadratic dependency on the data size N and at the same time inverse polynomial dependency on M. In essence, where previous methods have an overhead factor of either N/M or $(N/B)^\rho$ we obtain an overhead of $(N/M)^\rho$, where $0 < \rho < 1$ is a parameter of the LSH employed, strictly improving both. We show:

Theorem 1. *Consider $R, S \subseteq \mathbb{U}$, let $N = |R|+|S|$, assume $18 \log N + 3B \leq M < N$ and that there exists a monotone (r, cr, p_1, p_2)-sensitive family of functions with respect to distance measure d, using space B and with $p_2 < p_1 < 1/2$. Let $\rho = \log p_1 / \log p_2$. Then there exists a cache-oblivious randomized algorithm computing $R \bowtie_{\leq r} S$ (wrt. d) with probability $1 - \mathcal{O}(1/N)$ using*

$$\tilde{\mathcal{O}}\left(\left(\frac{N}{M}\right)^\rho \left(\frac{N}{B} + \frac{|R \underset{\leq r}{\bowtie} S|}{MB}\right) + \frac{|R \underset{\leq cr}{\bowtie} S|}{MB}\right) \text{ I/Os.}^3$$

At first, the given I/O complexity may look somewhat strange, but we argue that the bound is really quite natural. The first term matches the complexity of the standard LSH method if we set $M = 1$, and becomes essentially linear

[3] The $\tilde{\mathcal{O}}(\cdot)$-notation hides polylog$(N)$ factors.

when $M = N$ (i.e., when we know that the I/O complexity is linear). Regarding the second term, we need to use $|R \bowtie_{\leq r} S|/(MB)$ I/Os to compute distances of $|R \bowtie_{\leq r} S|$ pairs since in a single I/O we can introduce at most MB pairs in memory. We get a factor $(N/M)^\rho$ because we may introduce some very near pairs this number of times in memory. We can make a similar argument for the last term. Because it is very hard to distinguish distance r from distance slightly above r we expect to have to get some, possibly all, pairs in $R \bowtie_{\leq cr} S$ into fast memory in order to conclude that they should not be output. A more detailed discussion and experimental evaluations can be seen in the full version [13].

It is worth noting that whereas most methods in the literature focus on a single (or a few) distance measure, our methods work for an arbitrary space and distance measure that allows LSH, e.g., Hamming, Manhattan (ℓ_1), Euclidean (ℓ_2), Jaccard, and angular metric distances. A primary technical hurdle in the paper is that we cannot use any kind of strong concentration bounds on the number of points having a particular value, since hash values of an LSH family may be correlated *by definition*. Another hurdle is *duplicate elimination* in the output stemming from pairs having multiple LSH collisions. However, in the context of I/O-efficient algorithms it is natural to not require the *listing* of all near pairs, but rather we simply require that the algorithm *enumerates* all such near pairs. More precisely, the algorithm calls for each near pair (x, y) a function $\texttt{emit}(x, y)$. This is a natural assumption in external memory since it reduces the I/O complexity. In addition, it is desired in many applications where join results are intermediate results pipelined to a subsequent computation, and are not required to be stored on external memory. Our upper bound can be easily adapted to list all instances by increasing the I/O complexities of an *unavoidable* additive term of $\Theta(|R \bowtie_{\leq r} S|/B)$ I/Os.

The organization of the paper is as follows. In Section 2, we briefly review related work. Section 3 describes our algorithms including a warm-up cache-aware approach and the main results, a cache-oblivious solution, its analysis, and a randomized approach to remove duplicates. Section 4 concludes the paper.

2 Related Work

Because Locality-sensitive hashing (LSH) is a building block of our I/O-efficient similarity join, we briefly review LSH, the computational I/O model, and some state-of-the-art similarity join techniques.

Locality-sensitive hashing (LSH). LSH was originally introduced by Indyk and Motwani [12] for similarity search problem in high dimensional data. This technique obtains a sublinear (i.e., $\mathcal{O}(N^\rho)$) time complexity by increasing the gap of collision probability between near points and far points using the LSH family as defined in Definition 1. Such gap of collision probability is polynomial, with an exponent of $\rho = \log p_1 / \log p_2$ dependent on c, and $0 < \rho < 1$.

In this work we will use LSH as a black box for the similarity join problem. It is worth noting that the standard LSHs for metric distances, including Hamming [12], ℓ_1 [7], ℓ_2 [1, 7], Jaccard [4] and angular distances [5] are *monotone*. These common

LSHs are space-efficient, and use space comparable to that required to store a point, except the LSH of [1] which requires space $N^{o(1)}$. We did not explicitly require the hash values themselves to be particularly small. However, using universal hashing we can always map to small bit strings while introducing no new collisions with high probability. Thus we assume that B hash values fit in one memory block.

Computational I/O model. We study algorithms for similarity join in the *external memory model*, which has been widely adopted in the literature (see, e.g., the survey by Vitter [14]). The external memory model consists of an internal memory of M words and an external memory of unbounded size. The processor can only access data stored in the internal memory and move data between the two memories in blocks of size B. For simplicity we will here measure block and memory size in units of points from $\mathbb{U}$, such that a block can contain B points.

The *I/O complexity* of an algorithm is defined as the number of input/output blocks moved between the two memories by the algorithm. The *cache-aware* approach makes use of the parameter M explicitly to achieve its I/O complexity whereas the *cache-oblivious* one [8] does not explicitly use any model parameters. The latter is a desirable property as it implies optimality on all levels of the memory hierarchy and does not require parameter tuning when executed on different physical machines. The cache-oblivious model assumes that the internal memory is *ideal* in the sense that it has optimal cache-replacement policy that can evict the block that is used the farthest in the future, and also that a block can be placed anywhere in the cache (full associativity).

Similarity join techniques. We review some state-of-the-art of similarity join techniques most closely related to our work.

- **Index-based similarity join.** A popular approach is to make use of indexing techniques to build a data structure for one relation, and then perform queries using the points of the other relation. The indexes typically perform some kind of *filtering* to reduce the number of points that a given query point is compared to (see, e.g., [3, 6]). Indexing can be space consuming, in particular for LSH, but in the context of similarity join this is not a big concern since we have many queries, and thus can afford to construct each hash table "on the fly". On the other hand, it is clear that index-based similarity join techniques will not be able to take significant advantage of internal memory when $N \gg M$. Indeed, the query complexity stated in [9] is $\mathcal{O}\left((N/B)^{\rho}\right)$ I/Os. Thus the I/O complexity of using indexing for similarity join will be high.
- **Sorting-based.** The indexing technique of [9] can be adapted to compute similarity joins more efficiently by using the fact that many points are being looked up in the hash tables. This means that all lookups can be done in a batched fashion using sorting. This results in a dependency on N that is $\tilde{\mathcal{O}}\left((N/B)^{1+\rho}\right)$ I/Os, where $\rho \in (0; 1)$ is a parameter of the LSH family.
- **Generic joins.** When N is close to M the I/O-complexity can be improved by using general join operators optimized for this case. It is easy to see that when N/M is an integer, a nested loop join requires $N^2/(MB)$ I/Os. Our cache-oblivious algorithm will make use of the following result on cache-oblivious nested loop joins:

Theorem 2. *(He and Luo [11]) For an arbitrary join condition, the join of relations R and S can be computed in $\mathcal{O}\left((|R|+|S|)/B + (|R||S|)/(MB)\right)$ I/Os by a cache-oblivious algorithm. This number of I/Os suffices to generate the result in memory, but may not suffice to write it to disk.*

3 Our Algorithms

In this section we describe our I/O efficient algorithms. We start in Section 3.1 with a warm-up cache-aware algorithm. It uses an LSH family where the value of the collision probability is set to be a function of the internal memory size. Section 3.2 presents our main result, a recursive and cache-oblivious algorithm which uses the LSH as a black-box approach and does not make any assumption on the value of collision probability. Section 3.3 describes the analysis and Section 3.4 shows how to reduce the expected number of times each near pair is emitted.

3.1 Cache-aware algorithm: ASimJoin

We will now describe a simple cache-aware algorithm called ASimJoin, which achieves the worst case I/O bounds as stated in Theorem 1. Due to the limit of space, we will sketch some intuitions of the algorithm and refer to [13] for the full discussion and omitted proof.

ASimJoin relies on an (r, cr, p'_1, p'_2)-sensitive family $\mathcal{H}'$ of hash functions with the following properties: $p'_2 \leq M/N$ and $p'_1 \geq (M/N)^\rho$, for a suitable value $0 < \rho < 1$. Given an arbitrary (r, cr, p_1, p_2)-sensitive family $\mathcal{H}$, the family $\mathcal{H}'$ can be built by concatenating $\lceil \log_{p_2}(M/N) \rceil$ hash functions from $\mathcal{H}$. For simplicity, we assume that $\log_{p_2}(M/N)$ is an integer and thus the probabilities p'_1, p'_2 can be exactly obtained. However, the algorithm and the analysis can be extended to the general case by increasing the I/O complexity by a factor at most p_1^{-1} in the worst case; in practical scenarios, this factor is a small constant [4, 7, 9].

Let R and S denote the input sets. The algorithm repeats $L = 1/p'_1$ times the following procedure. A hash function is randomly drawn from the (r, cr, p'_1, p'_2)-sensitive family, and it is used for partitioning the sets R and S into buckets of points with the same hash value. Then, the algorithm iterates through every hash value and, for each hash value v, it uses a double nested loop for generating all pairs of points in $R_v \times S_v$, where R_v and S_v denote the buckets respectively containing points of R and S with hash value v. A pair is emitted only if it is a near pair. For each input point $x \in R \cup S$, we maintain a counter that is increased every time a pair (x, y) is generated and y is far from x. The counter is maintained over all the L repetitions and keeps track of the number of collisions of x with its far points. As soon as the counter of a point exceeds $4LM$, the point is removed from the input set[4]. We refer to the pseudocode for more details.

[4] We observe that removing points that collide with at least $4LM$ points is only required for getting the claimed I/O complexity with high probability. The algorithm can be simplified by removing this operation, and yet obtaining the same I/O bound in expectation.

Algorithm ASimJoin(R, S)**:** R, S are the input sets.

```
1  Associate to each point in S a counter initially set to 0;
2  Repeat L = 1/p'_1 times
3      Choose h'_i ∈ H' uniformly at random;
4      Use h'_i to partition (in-place) R and S in buckets R_v, S_v of points with hash
       value v;
5      For each hash value v generated in the previous step
6          /* For simplicity we assume that |R_v| ≤ |S_v|                          */
7          Split R_v and S_v into chunks R_{i,v} and S_{i,v} of size at most M/2;
8          For every chunk R_{i,v} of R_v
9              Load in memory R_{i,v};
10             For every chunk S_{i,v} of S_v do
11                 Load in memory S_{i,v};
12                 Compute R_{i,v} × S_{i,v} and emit all near pairs. For each far pair,
                   increment the associated counters by 1;
13                 Remove from S_{i,v} and R_{i,v} all points with the associated counter
                   larger than 4LM, and write S_{i,v} back to external memory;
14         Write R_{i,v} back to external memory;
```

By using the (r, cr, p'_1, p'_2)-sensitive family, ASimJoin guarantees that each point collides with at most M far points. Therefore, each point in $R_v \cup S_v$ is far from at most M points in $R_v \cup S_v$ (note that points in R and S are in the same universe and then collision probabilities apply independently of the belonging set). This implies that, if there are not too many near and c-near pairs, $R_v \bowtie_{\leq r} S_v$ can be efficiently computed with $\mathcal{O}(M/B)$ I/Os. On the other hand, if there are many near or c-near points, the I/O complexity can be upper bounded in an output sensitive way. In particular, we observe that a point cannot collide with too many far points since it is removed from the set after $4LM$ collisions with far points. Moreover, since each near pair has probability p'_1 to be emitted by partitioning R and S with LSH, the process is repeated $L = 1/p'_1$ times.

3.2 Cache-oblivious algorithm: OSimJoin

The above cache-aware algorithm uses an (r, cr, p'_1, p'_2)-sensitive family of functions, with $p'_1 \sim (M/N)^\rho$ and $p'_2 \sim M/N$, for partitioning the initial sets into smaller buckets, which are then efficiently processed in the internal memory using the nested loop algorithm. As soon as the internal memory size M is known, this family of functions can be built by concatenating $\lceil \log_{p_2} p'_2 \rceil$ hash functions from any given primitive (r, cr, p_1, p_2)-sensitive family. However, in the cache-oblivious settings the value of M is not known and such family cannot be built. Therefore, we propose in this section an algorithm, named OSimJoin, that efficiently compute the similarity join even without knowing the values of the internal memory size M and the block length B, and uses as a black-box a given

Algorithm OSimJoin(R, S, ψ)**:** R, S are the input sets, and ψ is the recursion depth.

If $|R| > |S|$, **then** swap (the references to) the sets such that $|R| \leq |S|$;
If $\psi = \Psi$ or $|R| \leq 1$, **then** compute $R \bowtie_{\leq r} S$ using the algorithm of Theorem 2 and return;
Pick a random sample S' of 18Δ points from S (or all points if $|S| < 18\Delta$);
Compute R' containing all points of R that have distance smaller than cr to at least half points in S'. Permute R such that points in R' are in the first positions;
Compute $R' \bowtie_{\leq r} S$ using the algorithm of Theorem 2;
Repeat $L = 1/p_1$ times
    Choose $h \in \mathcal{H}$ uniformly at random;
    Use h to partition (in-place) $R \backslash R'$ and S in buckets R_v, S_v of points with hash value v;
    For each v where R_v and S_v are nonempty, recursively call OSimJoin $(R_v, S_v, \psi + 1)$;

monotonic (r, cr, p_1, p_2)-sensitive family of functions[5]. The value of p_1 and p_2 can be considered constant in practical scenario.

As common in the cache-oblivious settings, we use a recursive approach for splitting the problem into smaller and smaller subproblems that at some point will fit the internal memory, although this point is not known in the algorithm. We first give a high level description of the cache-oblivious algorithm and an intuitive explanation. We then provide a more detailed description and analysis.

OSimJoin receives in input the two sets R and S of similarity join, and a parameter ψ denoting the depth in the recursion tree (initially, $\psi = 0$) that is used for recognizing the base case. Let $|R| \leq |S|$, $N = |R| + |S|$, and denote with $\Delta = \log N$ and $\Psi = \lceil \log_{1/p_2} N \rceil$ two global values that are kept invariant in the recursive levels and computed using the initial input size N. For the sake of simplicity, we assume that $1/p_1$ and $1/p_2$ are integers, and further assume without loss of generality that the initial size N is a power of two. Note that, if $1/p_1$ is not integer, that the last iteration can be performed with probability $1/p_1 - \lfloor 1/p_1 \rfloor$, such that $L \in \{\lfloor 1/p_1 \rfloor, \lceil 1/p_1 \rceil\}$ and $\mathbb{E}[L] = 1/p_1$.

OSimJoin works as follows. If the problem is currently at recursive level $\Psi = \lceil \log_{1/p_2} N \rceil$ or $|R| \leq 1$, the recursion ends and the problem is solved using the cache-oblivious nested loop described in Theorem 2. Otherwise the following operations are executed. By exploiting sampling, the algorithm identifies a subset R' of R containing (almost) all points that are near or c-near to a constant fraction of points in S. More specifically, the set R' is computed by creating a random sample S' of S of size 18Δ and then adding to R' all points in R that have distance at most cr to at least half points in S'. The join $R' \bowtie_{\leq r} S$ is computed by using the cache-oblivious nested-loop of Theorem 2 and then points in R'

[5] The monotonicity requirement can be relaxed to the following: $\Pr[h(x) = h(y)] \geq \Pr[h(x') = h(y')]$ for every two pairs (x, y) and (x', y') where $d(x, y) \leq r$ and $d(x', y') > r$. A monotone LSH family clearly satisfies this assumption.

are removed from R. Subsequently, the algorithm repeats $L = 1/p_1$ times the following operations: a hash function is extracted from the (r, cr, p_1, p_2)-sensitive family and used for partitioning R and S into buckets, denoted with R_v and S_v with any hash value v; then, the join $R_v \bowtie_{\leq r} S_v$ is computed recursively.

The explanation of our approach is the following. By recursively partitioning input points with hash functions from an (r, cr, p_1, p_2)-sensitive family, the algorithm decreases the probability of collision between two far points. In particular, the collision probability of two far points is p_2^i at the i-th recursive level. On the other hand, by repeating the partitioning $1/p_1$ times in each level, the algorithm guarantees that a pair of near points is enumerated with constant probability since the probability that two near points collide is p_1^i at the i-th recursive level. It deserves to be noticed that the collision probability of far and near points at the recursive level $\log_{1/p_2}(N/M)$ is $\Theta(M/N)$ and $\Theta((M/N)^\rho)$, respectively, which are asymptotically equivalent to the values in the cache-aware algorithm. In other words, the partitioning of points at this level is equivalent to the one in the cache-aware algorithm, being the expected number of colliding far points is M. Finally, we observe that, when a point in R becomes close to many points in S, it is more efficient to detect and remove it, instead of propagating it down to the base cases. Indeed, it may happen that the collision probability of these points is large (close to 1) and the algorithm is not able to split them into subproblems that fit in memory.

3.3 I/O Complexity and Correctness of OSimJoin

Analysis of I/O Complexity. We will bound the *expected* number of I/Os of the algorithm rather than the worst case. This can be converted to a fixed time bound by a standard technique of restarting the computation when the expected number of I/Os is exceeded by a factor 2. To succeed with probability $1 - 1/N$ it suffices to do $\mathcal{O}(\log N)$ restarts to complete within twice the expected time bound, and the logarithmic factor is absorbed in the $\tilde{\mathcal{O}}$-notation. If the computation does not succeed within this bound we fail to produce an output, slightly increasing the error probability.

For notational simplicity, in this section we let R and S denote the initial input sets and let $\tilde{R}$ and $\tilde{S}$ denote the subsets given in input to a particular recursive subproblem (note that $\tilde{R}$ can be a subset of R but also of S; similarly for $\tilde{S}$). We also let $\tilde{S}'$ denote the sampling of $\tilde{S}$ in Step 3, and with $\tilde{R}'$ the subset of $\tilde{R}$ computed in Step 4. Our first lemma says that two properties of the choice of random sample in Step 3 are almost certain. The proof relies on Chernoff bounds on the choice of $\tilde{S}'$. See [13] for the omitted proof.

Lemma 1. *Consider a run of Steps 3 and 4 in a subproblem* OSimJoin$(\tilde{R}, \tilde{S}, \psi)$*, for any level* $0 \leq \psi \leq \Psi$*. Then with probability at least* $1 - \mathcal{O}(1/N)$ *over the choice of sample* $\tilde{S}'$ *we have:*

$$|\tilde{R}' \underset{\leq cr}{\bowtie} \tilde{S}| > \frac{|\tilde{R}'||\tilde{S}|}{6} \qquad \text{and} \qquad |(\tilde{R} \backslash \tilde{R}') \underset{> cr}{\bowtie} \tilde{S}| > \frac{5|\tilde{R} \backslash \tilde{R}'||\tilde{S}|}{6} \ .$$

In the remainder of the paper, we assume that Lemma 1 holds and refer to this event as $\mathcal{A}$. By the above, $\mathcal{A}$ holds with probability $1 - \mathcal{O}(1/N)$.

To analyze the number of I/Os for subproblems of size more than M we bound the cost in terms of different types of *collisions*, i.e., pairs in $R \times S$ that end up in the same subproblem of the recursion. We say that (x, y) *is in* a particular subproblem $\text{OSIMJOIN}(\tilde{R}, \tilde{S}, \psi)$ if $(x, y) \in (\tilde{R} \times \tilde{S}) \cup (\tilde{S} \times \tilde{R})$. Observe that a pair (x, y) is in a subproblem if and only if x and y have colliding hash values on every step of the call path from the initial invocation of OSIMJOIN.

Definition 2. *Given $Q \subseteq R \times S$ let $C_i(Q)$ be the number of times a pair in Q is in a call to* OSIMJOIN *at the ith level of recursion. We also let $C_{i,k}(Q)$, with $0 \leq k \leq \log M$ denote the number of times a pair in Q is in a call to* OSIMJOIN *at the ith level of recursion where the smallest input set has size in $[2^k, 2^{k+1})$ if $0 \leq k < \log M$, and in $[M, +\infty)$ if $k = \log M$. The count is over all pairs and with multiplicity, so if (x, y) is in several subproblems at the ith level, all these are counted.*

Next we bound the I/O complexity of OSIMJOIN in terms of $C_i(R \bowtie_{\leq cr} S)$ and $C_{i,k}(R \bowtie_{>cr} S)$, for any $0 \leq i < \Psi$. These quantities are then bounded in Lemma 3. Due to the space constraint, refer to [13] for the proof details.

Lemma 2. *Let $\ell = \lceil \log_{1/p_2}(N/M) \rceil$ and $M \geq 18 \log N + 3B$. Given that $\mathcal{A}$ holds, the I/O complexity of* $\text{OSIMJOIN}(R, S, 0)$ *is*

$$\tilde{\mathcal{O}}\left(\frac{NL^\ell}{B} + \sum_{i=0}^{\ell} \frac{C_i\left(R \underset{\leq cr}{\bowtie} S\right)}{MB} + \sum_{i=\ell}^{\Psi-1} \sum_{k=0}^{\log M} \frac{C_{i,k}\left(R \underset{>cr}{\bowtie} S\right) L}{B 2^k}\right)$$

Proof. (Sketch) The proof of this lemma consists of bounding the I/O complexity of each step as a function of the number of c-near or far collisions. The first two terms give the cost of all subproblems at levels above ℓ: the first term is due to Step 5 and follows by expressing the cost in Theorem 2 in terms of c-near collision through Lemma 1; the second term follows from a simple analysis of Steps 7-8. The last term is the cost of levels below ℓ and follows by expressing the I/O complexity in terms of far collisions within subproblems of size in $[2^k, 2^{k+1})$ for any $k \geq 0$. The cost of level ℓ is asymptotically negligible compared to the other cases. □

We will now analyze the expected sizes of the terms in Lemma 2. Clearly each pair from $R \times S$ is in the top level call, so the number of collisions is $|R||S| < N^2$. But in lower levels we show that the expected number of times that a pair collides either decreases or increases geometrically, depending on whether the collision probability is smaller or larger than p_1 (or equivalently, depending on whether the distance is greater or smaller than the radius r). The lemma follows by expressing the number of collisions of the pairs at the ith recursive level as a *Galton-Watson branching process* [10]. See [13] for proof details.

Lemma 3. *Given that $\mathcal{A}$ holds, for each $0 \leq i \leq \Psi$ we have*

1. $\mathbb{E}\left[C_i\left(R \underset{>cr}{\bowtie} S\right)\right] \leq |R \underset{>cr}{\bowtie} S|\,(p_2/p_1)^i$;
2. $\mathbb{E}\left[C_i\left(R \underset{>r,\leq cr}{\bowtie} S\right)\right] \leq |R \underset{>r,\leq cr}{\bowtie} S|$;
3. $\mathbb{E}\left[C_i\left(R \underset{\leq r}{\bowtie} S\right)\right] \leq |R \underset{\leq r}{\bowtie} S|\,L^i$;
4. $\mathbb{E}\left[C_{i,k}\left(R \underset{>cr}{\bowtie} S\right)\right] \leq N2^{k+1}\,(p_2/p_1)^i$, *for any* $0 \leq k < \log M$.

We are now ready to prove the I/O complexity of OSimJoin as claimed in Theorem 1. By the linearity of expectation and Lemma 2, we get that the expected I/O complexity of OSimJoin is

$$\tilde{\mathcal{O}}\left(\frac{NL^\ell}{B} + \sum_{i=0}^{\ell}\frac{\mathbb{E}\left[C_i\left(R \underset{\leq cr}{\bowtie} S\right)\right]}{MB} + \sum_{i=\ell}^{\Psi-1}\sum_{k=0}^{\log M}\frac{\mathbb{E}\left[C_{i,k}\left(R \underset{>cr}{\bowtie} S\right)\right]L}{B2^k}\right) ,$$

where $\ell = \lceil \log_{1/p_2}(N/M)\rceil$. By noticing $C_{i,\log M}\left(R \bowtie_{>cr} S\right) \leq C_i\left(R \bowtie_{>cr} S\right)$ we have $|R \bowtie_{>cr} S| \leq N^2$ and $C_i\left(R \bowtie_{\leq cr} S\right) = C_i\left(R \bowtie_{\leq r} S\right) + C_i\left(R \bowtie_{>r,\leq cr} S\right)$, and by plugging in the bounds on the expected number of collisions given in Lemma 3, we get the claimed result.

Analysis of Correctness. We now argue that a pair (x, y) with $d(x, y) \leq r$ is output with good probability. Let $X_i = C_i((x, y))$ be the number of subproblems at level i containing (x, y). By applying Galton-Watson branching process, we get that $\mathbb{E}[X_i] = (\Pr[h(x) = h(y)]/p_1)^i$. If $\Pr[h(x) = h(y)]/p_1 > 1$ then in fact there is positive constant probability that (x, y) survives indefinitely, i.e., does not go extinct [10]. Since at every branch of the recursion we eventually compare points that collide under all hash functions on the path from the root call, this implies that (x, y) is reported with positive constant probability.

In the *critical case* where $\Pr[h(x) = h(y)]/p_1 = 1$ we need to consider the variance of X_i, which by [10, Theorem 5.1] is equal to $i\sigma^2$, where σ^2 is the variance of the number of children (hash collisions in recursive calls). If $1/p_1$ is integer the number of children in our branching process follows a binomial distribution with mean 1. This implies that $\sigma^2 < 1$. Also in the case where $1/p_1$ is not integer it is easy to see that the variance is bounded by 2. That is, we have $\mathrm{Var}(X_i) \leq 2i$, which by Chebychev's inequality means that for some integer $j^* = 2\sqrt{i} + \mathcal{O}(1)$:

$$\sum_{j=j^*}^{\infty}\Pr[X_i \geq j] \leq \sum_{j=j^*}^{\infty}\mathrm{Var}(X_i)/j^2 \leq 1/2 .$$

Since we have $\mathbb{E}[X_i] = \sum_{j=1}^{\infty}\Pr[X_i \geq j] = 1$ then $\sum_{j=1}^{j^*-1}\Pr[X_i \geq j] > 1/2$, and since $\Pr[X_i \geq j]$ is non-increasing with j this implies that $\Pr[X_i \geq 1] \geq$

$1/(2j^*) = \Omega\left(1/\sqrt{i}\right)$. Since recursion depth is $\mathcal{O}(\log N)$ this implies the probability that a near pair is found is $\Omega\left(1/\sqrt{\log N}\right)$. Thus, by repeating $\mathcal{O}\left(\log^{3/2} N\right)$ times we can make the error probability $\mathcal{O}\left(1/N^3\right)$ for a particular pair and $\mathcal{O}(1/N)$ for the entire output by applying the union bound.

3.4 Removing duplicates

The definition of LSH requires the probability $p(x, y) = \Pr[h(x) = h(y)]$ of two near points x and y of being hashed on the same value is at least p_1. If $p(x, y) \gg p_1$, our OSimJoin algorithm can emit (x, y) many times. As an example suppose that the algorithm ends in one recursive call: then, the pair (x, y) is expected to be in the same bucket for $p(x, y)L$ iterations of Step 6 and thus it is emitted $p(x, y)L \gg 1$ times in expectation. Moreover, if the pair is not emitted in the first recursive level, the expected number of emitted pairs increases as $(p(x, y)L)^i$ since the pair (x, y) is contained in $(p(x, y)L)^i$ subproblems at the ith recursive level. A simple solution requires to store all emitted near pairs on the external memory, and then using a cache-oblivious sorting algorithm [8] for removing repetitions. However, this approach requires $\tilde{\mathcal{O}}(\kappa|R \bowtie_{\leq r} S|/B)$ I/Os, where κ is the expected average replication of each emitted pair, which can dominate the complexity of OSimJoin. A similar issue appears in the cache-aware algorithm ASimJoin as well: however, a near pair is emitted in this case at most $L' = (N/M)^\rho$ since there is no recursion and the partitioning of the two input sets is repeated only L' times.

If the collision probability $\Pr[h(x) = h(y)]$ can be explicitly computed in $\mathcal{O}(1)$ time and no I/Os for each pair (x, y), it is possible to emit each near pair once in expectation without storing near pairs on the external memory. We note that the collision probability can be computed for many metrics, including Hamming [12], ℓ_1 and ℓ_2 [7], Jaccard [4], and angular [5] distances. For the cache-oblivious algorithm, the approach is the following: for each near pair (x, y) that is found at the ith recursive level, with $i \geq 0$, the pair is emitted with probability $1/(p(x, y)L)^i$ and is ignored otherwise. For the cache-aware algorithm, the idea is the same but a near pair is emitted with probability $1/(p(x, y)L')$ with $L' = (N/M)^\rho$. The proof of the claim is provided in [13].

We observe that the proposed approach is equivalent to use an LSH where $p(x, y) = p_1$ for each near pair. Finally, we remark that this approach does not avoid replica of the same near pair when the algorithm is repeated for increasing the collision probability of near pairs.

4 Conclusion

In this paper we examine the problem of computing the similarity join of two relations in an external memory setting. Our new cache-aware algorithm of Section 3.1 and cache-oblivious algorithm of Section 3.2 improve upon current state of the art by around a factor of $(M/B)^\rho$ I/Os unless the number of c-near pairs

is huge (more than NM). We believe this is the first cache-oblivious algorithm for similarity join, and more importantly the first subquadratic algorithm whose I/O performance improves significantly when the size of internal memory grows.

It would be interesting to investigate if our cache-oblivious approach is also practical — this might require adjusting parameters such as L. Our I/O bound is probably not easy to improve significantly, but interesting open problems are to remove the error probability of the algorithm and to improve the implicit dependence on dimension in B and M: In this paper we assume for simplicity that the unit of M and B is number of points, but in general we may get tighter bounds by taking into account the gap between the space required to store a point and the space for e.g., hash values. Also, the result in this paper is made with general spaces in mind and it is an interesting direction to examine if the dependence on dimension could be made explicit and improved in specific spaces.

References

1. Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Proceedings of FOCS'06*, pages 459–468, 2006.
2. Arvind Arasu, Venkatesh Ganti, and Raghav Kaushik. Efficient exact set-similarity joins. In *Proceedings of VLDB'06*, pages 918–929, 2006.
3. Roberto J. Bayardo, Yiming Ma, and Ramakrishnan Srikant. Scaling up all pairs similarity search. In *Proceedings of WWW'07*, pages 131–140, 2007.
4. Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. *Computer Networks*, 29(8-13):1157–1166, 1997.
5. Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of STOC'02*, pages 380–388, 2002.
6. Surajit Chaudhuri, Venkatesh Ganti, and Raghav Kaushik. A primitive operator for similarity joins in data cleaning. In *Proceedings of ICDE'06*, page 5, 2006.
7. Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of SOCG'04*, pages 253–262, 2004.
8. Matteo Frigo, Charles E Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *Proceedings of FOCS'99*, pages 285–297, 1999.
9. Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of VLDB'99*, pages 518–529, 1999.
10. Theodore E Harris. *The theory of branching processes*. Courier Dover Publications, 2002.
11. Bingsheng He and Qiong Luo. Cache-oblivious nested-loop joins. In *Proceedings of CIKM'06*, pages 718–727, 2006.
12. Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of STOC'98*, pages 604–613, 1998.
13. Rasmus Pagh, Ninh Pham, Francesco Silvestri, and Morten Stöckel. I/O-efficient similarity join. Full version, arXiv:1507.00552, 2015.
14. Jeffrey Scott Vitter. *Algorithms and Data Structures for External Memory*. Now Publishers Inc., 2008.
15. Chuan Xiao, Wei Wang, Xuemin Lin, and Jeffrey Xu Yu. Efficient similarity joins for near duplicate detection. In *Proceedings of WWW'08*, pages 131–140, 2008.