I/O-Efficient Similarity Join*

Rasmus Pagh, Ninh Pham, Francesco Silvestri**, and Morten Stöckel***

IT University of Copenhagen, Denmark

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, 14], 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 [14], inverted index-based filtering [3]) are based on *filtering* techniques that often, but not always, succeed in reducing computational costs. If we let $N = |R| + |S|$ 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.

* 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.
*** Supported by the Danish National Research Foundation / Sapere Aude program.

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 monotonic if* $Pr[h(x) = h(y)]$ *is a non-increasing function of the distance function* $d(x, y)$*. We also say that* $\mathcal{H}$ *uses space* s *if a function* $h \in \mathcal{H}$ *can be stored and evaluated using space* s*.*

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 monotonic* (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*

$$\tilde{\mathcal{O}}\left(\left(\frac{N}{M}\right)^{\rho}\left(\frac{N}{B} + \frac{|R \underset{\leq r}{\bowtie} S|}{MB}\right) + \frac{|R \underset{\leq cr}{\bowtie} S|}{MB}\right) \text{ I/Os.}^{1}$$

Discussion. We now conjecture that the bound in Theorem 1 is close to the best possible for the class of "signature based" algorithms that work by generating a set of LSH values (from a black-box and monotonic family) and checking all pairs that collide. For simple arguments, we split the I/O complexity of our algorithms

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

in two parts:

$$T_1 = (N/M)^{\rho}\big(N/B + |R \underset{\leq r}{\bowtie} S|/(MB)\big),$$
$$T_2 = |R \underset{\leq cr}{\bowtie} S|/(MB).$$

We now argue that T_1 I/Os is necessary. First, notice that we need $\mathcal{O}(N/B)$ I/Os per hash function for transferring data between memories, computing and writing hash values to disk to find collisions. Second, since each I/O brings at most B points in order to find collisions with M points residing in the internal memory, we need N/B I/Os to find collisions of MN pairs. This means that when $p_2 \leq M/N$, the number of collisions of far pairs is at most MN in expectation and we need N/B I/Os to detect such far pairs.

Now consider the case where there are $\Omega\left(N^2\right)$ pairs at distance cr. Then the collision probability for each pair must be $\mathcal{O}(M/N)$ to ensure at most MN collisions of far pairs due to the monotonicity of LSH family. In turn, this means that the collision probability for near pairs at distance r must be at most $\mathcal{O}((M/N)^{\rho})$. So $\Omega((N/M)^{\rho})$ repetitions (different hash functions) are needed before we expect a near pair to collide at least once.

Then, a worst-case data set can be given so that we might need to examine, for each of the $\Omega((N/M)^{\rho})$ hash functions, a constant fraction the pairs in $R \bowtie_{\leq r} S$ whose collision probability is constant. For example, this can happen if R and S include two clusters of close points. One could speculate that some pairs could be marked as "finished" during computation such that we do not have to compare their hash values again. However, it seems hard to make this idea work for an arbitrary distance measure where there may be very little structure to the output set, hence the $\mathcal{O}(|R \bowtie_{\leq r} S|/(MB))$ additional I/Os per repetition is needed.

To argue that the term T_2 is needed, consider the case where all pairs in $R \bowtie_{\leq cr} S$ have distance $r + \varepsilon$ for a value εsmall enough to make the collision probability of pair at distance $r+\varepsilon$ indistinguishable from the collision probability of pair at distance r. Then every pair in $R \bowtie_{\leq cr} S$ must be brought into the internal memory to ensure a correct result, which requires T_2 I/Os. This holds for *any* algorithm enumerating or listing the near pairs. Therefore, there does not exist an algorithm that beats the quadratic dependency on N for such worst-case input sets, unless the distribution of the input is known beforehand. A possible approach to get subquadratic dependency on N in expectation when the number of reporting pairs is subquadratic is the one that can filter the pairs based on their distances — currently LSH-based methods are the only way to do this.

Note that when $M = N$ we have $T_1 = \mathcal{O}(N/B)$ as we would expect, since just reading the input is optimal. At the other extreme, when $B = M = 1$ our bound matches the time complexity of internal memory techniques. When $|R \bowtie_{\leq cr} S|$ are bounded by MN then our algorithm achieves subquadratic dependency on N/M. Such an assumption is realistic in some real-world datasets as shown in the experimental evaluation section.

It is worth noting that whereas most methods in the literature focus on a single (or a few) distance measure, our method works 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 (i.e., pairs with distance not greater than r), 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 $\Theta\left(|R \bowtie_{\leq r} S|/B\right)$ 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 shows some experimental results and Section 5 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., $\mathcal{O}\left(N^{\rho}\right)$) 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.

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 *monotonic*. 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 [13]). 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., [6, 3]). 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

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 the
       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;
```

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 with 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. 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 monotonic (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].

ASimJoin assumes that each point in R and S is associated with a counter initially set to 0. This counter can be thought as another dimension of the point which hash functions and comparisons do not take into account. 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. We let R_v and S_v denote the buckets respectively containing points of R and S with the same hash value v. 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 $R_v \times S_v$. The double nested loop loads consecutive chunks of R_v and S_v of size at most $M/2$: the outer loop runs on the smaller set (say R_v), while the inner one runs on the larger one (say S_v). For each pair (x, y), the algorithm emits near pairs $d(x, y) \leq r$, ignores c-near pairs $r < d(x, y) \leq cr$ and far pairs $d(x, y) > cr$, and increases counters associated with x and y of the far pairs by 1. Every time the counter of a point exceeds $4LM$, the point is removed from the bucket. Chunks will be moved back in memory when they are no more needed. The following theorem shows the I/O bounds of the cache-aware approach.

Theorem 3. *Consider $R, S \subseteq \mathbb{U}$ and let $N = |R| + |S|$ be sufficiently large. Assume there exists a monotonic (r, cr, p'_1, p'_2)-sensitive family of functions with respect to distance measure d with $p'_1 = (M/N)^\rho$ and $p'_2 = M/N$, for a suitable value $0 < \rho < 1$. With probability $1 - 1/N$, the* ASimJoin *algorithm enumerates all near pairs using*

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

Proof. We first observe that the I/O cost of Steps 3-4, that is of partitioning sets R and S according to a hash function h'_i, is $L \cdot \mathrm{sort}(N) = \tilde{\mathcal{O}}\left((N/M)^\rho N/B\right)$[2] for $L \leq (N/M)^\rho$ repetitions.

Consider now the I/O cost of an iteration of the loop in Step 5 for a given hash value v and suppose that the size of the smaller bucket is smaller than $M/2$. Then, the I/O cost is at most $(|R_v| + |S_v|)/B$. Therefore, the total I/O cost of the L iterations of Step 5 among all possible hash values where at least one bucket has size smaller than $M/2$ is at most $LN/B = (N/M)^\rho N/B$ I/Os.

Next, we consider the I/O cost of an iteration the loop in Step 5 for a given hash value v when both buckets R_v and S_v are larger than $M/2$. In this case the I/O cost is $2|R_v||S_v|/(BM)$, from which follows that the amortized cost of each pair in $R_v \times S_v$ is $2/(BM)$. Therefore the I/O cost of all iterations of Step 5, when there are no bucket size less than $M/2$, can be upper bounded by multiplying the total number of generated pairs for the amortized I/O cost. It is clear that generated pairs can be partitioned into three groups: near pairs, c-near pairs and far pairs. We denote with C_n, C_{cn} and C_f the respective size of each group.

1. *Number of near pairs.* By definition, LSH gives a lower bound on the probability of collision of near pairs. It may happen that the collision probability of near pairs is 1. Thus, two near points can be in the same hash value in each one of the L repetitions of Step 2. This means that $C_n \leq L|R \bowtie_{\leq r} S|$ (note that this is a deterministic worst case bound).

[2] We let $\mathrm{sort}(N) = \mathcal{O}\left((N/B)\log_{M/B}(N/B)\right)$ be shorthand for the I/O complexity [13] of sorting N points.

2. *Number of c-near pairs.* A pair from $R \bowtie_{\leq cr} S$ appears in a bucket with probability at most p'_1 due to monotonicity of our LSH family. Since we repeat for $L = 1/p'_1$ hash functions, the each c-near pair collide at most once in expectation. By using the Chernoff bound on $3 \log N$ independent L repetitions and applying the union bound, we get $C_{cn} \leq 3 \log N |R \bowtie_{\leq cr} S|$ with probability $1 - 1/N$.
3. *Number of far pairs.* For each point $x \in R \cup S$ we throw x away as described above if it exceeds $4LM$ collisions with points that are more than distance cr away. We have that the number of far away collisions examined per point is at most $4LM$, and hence the total number of far pairs is $C_f \leq 4NLM$.

Therefore by summing the number of near pairs C_n, c-near pairs C_{cn}, and far pairs C_f and multiplying these quantities by the amortized I/O complexity $2/(BM)$, we get that the I/O cost of all iterations of Step 5, when there are no buckets of size less than $M/2$, is

$$\left(\frac{N}{M}\right)^{\rho} \left(\frac{8N}{B} + \frac{2|R \bowtie_{\leq r} S|}{BM}\right) + \frac{6 \log N |R \bowtie_{\leq cr} S|}{BM},$$

with probability at least $1 - 1/N$. By summing all the previous bounds, we get the claimed bound with high probability. □

As already mentioned in the introduction, a near pair (x, y) can be emitted many times during the algorithm since points x and y can be hashed on the same value in $p(x, y)L$ rounds of Step 2, where $p(x, y) \geq p'_1$ denotes the actual collision probability. A simple approach for avoiding duplicates is the following: for each near pair found during the i-th iteration of Step 2, the pair is emitted only if the two points did not collide by all hash functions used in the previous $i - 1$ rounds. The check starts from the hash function used in the previous round and backtracks until a collision is found or there are no more hash functions. This approach increases the worst case complexity by a negligible constant factor (e.g, by setting $M = B = \mathcal{O}(1)$). Section 3.4 shows a more efficient randomized algorithm that reduces the number of replica per near pair to a constant. It also applies to the cache-oblivious algorithm described in the next section.

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 constructed 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 computes the similarity join even without knowing the values of the internal memory size M and the block length B. OSimJoin uses as a

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';
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 $(R_v, S_v, \psi + 1)$;

black-box a given monotonic (r, cr, p_1, p_2)-sensitive family of functions[3]. 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 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 is empty, 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'

[3] 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 monotonic 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 notional 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. Lemma 1 says that two properties of the choice of random sample in Step 3 are almost certain, and the proof relies on Chernoff bounds on the choice of $\tilde{S}'$. In the remainder of the paper, we assume that Lemma 1 holds and refer to this event as $\mathcal{A}$ holding with probability $1 - \mathcal{O}(1/N)$.

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:*

$$|\tilde{R}' \underset{\leq cr}{\bowtie} \tilde{S}| > \frac{|\tilde{R}'||\tilde{S}|}{6} \quad , \tag{1}$$

$$|(\tilde{R}\backslash\tilde{R}') \underset{>cr}{\bowtie} \tilde{S}| > \frac{5|\tilde{R}\backslash\tilde{R}'||\tilde{S}|}{6} \quad . \tag{2}$$

Proof. Both claims follow from Chernoff bounds. Let $x \in \tilde{R}$ be a point which is c-near to at most one sixth of the points in $\tilde{S}$, i.e. $|x \underset{\leq cr}{\bowtie} \tilde{S}| \leq |\tilde{S}|/6$. The point x enters $\tilde{R}'$ if there are at least 9Δ c-near points in $\tilde{S}'$ and this happens with probability at most $1/N^4$ since the entries in $\tilde{S}'$ are randomly and independently chosen from $\tilde{S}$ and there is probability $1/6$ that an entry in $\tilde{S}$ is near to x.

Each point of $R \cup S$ appears in at most in $2\sum_{i=0}^{i=\Psi-1} L^i < 2L^\Psi < 2N^2$ subproblems. Because there are at most N points in $R \cup S$, by an union bound we get that, in any subproblem $\text{OSimJoin}(\tilde{R}, \tilde{S}, \psi)$, with probability $1 - 2N^3N^{-4} = 1 - 2N^{-1}$ every point in $\tilde{R}'$ has at least $|\tilde{S}|/6$ c-near points in $\tilde{S}$, and thus the bound in Equation 1 follows. Similarly, we have that all points in $\tilde{R}\backslash\tilde{R}'$ are far from at least $5/6$ points of $\tilde{S}$ with probability at least $1 - 2N^{-1}$, getting Equation 2. □

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 i-th 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 i-th 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 i-th 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 $0 \leq i < \Psi$. We will later upper bound the expected size of these quantities in Lemma 3 and then get the claim of Theorem 1.

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*

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

Proof. To ease the analysis we assume that no more than $1/3$ of internal memory is used to store blocks containing elements of R and S, respectively. Since the cache-oblivious model assumes an optimal cache replacement policy this cannot decrease the I/O complexity. Also, internal memory space used for other things

than data (input and output buffers, the recursion stack of size at most Ψ) is less than $M/3$ by our assumption that $M \geq 18\Delta + 3B = \Omega(\log N)$. As a consequence, we have that the number of I/Os for solving a subproblem OSimJoin $(\tilde{R}, \tilde{S}, \cdot)$ where $|\tilde{R}| \leq M/3$ and $|\tilde{S}| \leq M/3$ is $\mathcal{O}\left((|\tilde{R}| + |\tilde{S}|)/B\right)$, including all recursive calls. This is because there is space $M/3$ dedicated to both input sets and only I/Os for reading the input are required. By charging the cost of such subproblems to the writing of the inputs in the parent problem, we can focus on subproblems where the largest set (i.e., $\tilde{S}$) has size more than $M/3$. We notice that the cost of Steps 3 and 4 is dominated by other costs by our assumption that the set $\tilde{S}'$ fits in internal memory, which implies that it suffices to scan data once to implement these steps. This cost is clearly negligible with respect to the remaining steps and thus we ignore them.

We first provide an upper bound on the I/O complexity required by all subproblems at a recursive level above ℓ. Let OSimJoin $(\tilde{R}, \tilde{S}, i)$ be a recursive call at the i-th recursive level, for $0 \leq i \leq \ell$. The I/O cost of the nested loop join in Step 5 in OSimJoin $(\tilde{R}, \tilde{S}, i)$ is $\mathcal{O}\left(|\tilde{S}|/B + |\tilde{R}'||\tilde{S}|/(MB)\right)$ by Theorem 2. We can ignore the $\mathcal{O}\left(|\tilde{S}|/B\right)$ term since it is asymptotically negligible with respect to the cost of each iteration of Step 6, which is upper bounded later. By Equation 1, we have that $\tilde{R}' \bowtie_{\leq cr} \tilde{S}$ contains more than $|\tilde{R}'||\tilde{S}|/6$ collisions, and thus the cost of Step 5 in OSimJoin$(\tilde{R}, \tilde{S}, i)$ is $\mathcal{O}\left(|\tilde{R}' \bowtie_{\leq cr} \tilde{S}|/(MB)\right)$. This means that we can bound the total I/O cost of all executions of Step 5 at level i of the recursion with $\mathcal{O}\left(C_i\left(R \bowtie_{\leq cr} S\right)/(MB)\right)$ since each near pair (x, y) appears in $C_i((x, y))$ subproblems at level i.

The second major part of the I/O complexity is the cost of preparing recursive calls in OSimJoin$(\tilde{R}, \tilde{S}, i)$ (i.e., Steps 7-8). In fact, in each iteration of Step 6, the I/O cost is $\tilde{\mathcal{O}}\left((|\tilde{R}| + |\tilde{S}|)/B\right)$, which includes the cost of hashing and of sorting to form buckets. Since each point of $\tilde{R}$ and $\tilde{S}$ is replicated in L subproblems in Step 6, we have that each point of the initial sets R and S is replicated L^{i+1} times at level i. Since the average cost per entry is $\tilde{\mathcal{O}}(1/B)$, we have that the total cost for preparing recursive calls at level i is $\tilde{\mathcal{O}}\left(NL^{i+1}/B\right)$. By summing the above terms, we have that the total I/O complexity of all subproblems in the i-th recursive level is upper bounded by:

$$\tilde{\mathcal{O}}\left(\frac{C_i\left(R \underset{\leq cr}{\bowtie} S\right)}{MB} + \frac{NL^{i+1}}{B}\right). \tag{3}$$

We now focus our analysis to bound the I/O complexity required by all subproblems at a recursive level below ℓ. Let again OSimJoin$(\tilde{R}, \tilde{S}, i)$ be a recursive call at the i-th recursive level, for $\ell \leq i \leq \Psi$. We observe that (part of) the cost of a subproblem at level $i \geq \ell$ can be upper bounded by a suitable function of collisions among far points in OSimJoin $(\tilde{R}, \tilde{S}, i)$. More specifically, consider an iteration of Step 6 in a subproblem at level i. Then, the cost for

preparing the recursive calls and for performing Step 5 in each subproblem (at level $i+1$) generated during the iteration, can be upper bounded as

$$\tilde{\mathcal{O}}\left((|\tilde{R}\backslash\tilde{R}'| + |\tilde{S}|)/B + |(\tilde{R}\backslash\tilde{R}') \bowtie_{\leq cr} \tilde{S}|/(BM)\right),$$

since each near pair in $(\tilde{R}\backslash\tilde{R}') \bowtie_{\leq cr} \tilde{S}$ is found in Step 5 in at most one subproblem at level $i+1$ generated during the iteration. Since we have that $|(\tilde{R}\backslash\tilde{R}') \bowtie_{\leq cr} \tilde{S}| \leq |\tilde{R}\backslash\tilde{R}'||\tilde{S}|$, we easily get that the above bound can be rewritten as $\tilde{\mathcal{O}}\left(|\tilde{R}\backslash\tilde{R}'||\tilde{S}|/(B\min\{M, |\tilde{R}\backslash\tilde{R}'|\})\right)$. We observe that this bound holds even when $i = \Psi - 1$: in this case the cost includes all I/Os required for solving the subproblems at level Ψ called in the iteration and which are solved using the nested loop in Theorem 2 (see Step 2). By Lemma 1, we have that the above quantity can be upper bounded with the number of far collisions between $\tilde{R}$ and $\tilde{S}$, getting $\tilde{\mathcal{O}}\left((|\tilde{R}\backslash\tilde{R}' \bowtie_{>cr} \tilde{S}|)/(B\min\{M, |\tilde{R}\backslash\tilde{R}'|\})\right)$.

Recall that $C_{i,k}(Q)$ denotes the number of times a pair in Q is in a call to OSimJoin at the i-th 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 $k = \log M$. Then, the total cost for preparing the recursive calls in Step 7-8 in all subproblems at level i and for performing Step 5 in all subproblems at level $(i+1)$ is:[4]

$$\tilde{\mathcal{O}}\left(\sum_{k=0}^{\log M} \frac{C_{i,k}(R \bowtie_{>cr} S)\, L}{B2^k}\right). \tag{4}$$

The L factor in the above bound follows since far collisions at level i are used for amortizing the cost of Step 5 for each one of the L iterations of Step 6.

To get the total I/O complexity of the algorithm we sum the I/O complexity required by each recursive level. We bound the cost of each level as follows: for a level $i < \ell$ we use the bound in Equation 3; for a level $i > \ell$ we use the bound in Equation 4; for level $i = \ell$, we use the bound given in Equation 4 to which we add the first term in Equation 3 since the cost of Step 5 at level ℓ is not included in Equation 4 (note that the addition of Equations 3 and 4 gives a weak upper bound for level ℓ). The lemma follows. □

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 i-th recursive level as a *Galton-Watson branching process* [10].

[4] We note that the true input size of a subproblem is $|\tilde{R}|$ and not $|\tilde{R}\backslash\tilde{R}'|$. However, the expected value of $C_{i,k}(R \bowtie_{>cr} S)$ is computed assuming the worst case where there are no close pairs an thus $\tilde{R}' = \emptyset$.

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

Proof. Let $x \in R$ and $y \in S$. We are interested in upper bounding the number of collisions of the pair at the i-th recursive level. We envision the problem as *branching process* (more specifically a GaltonWatson process, see e.g. [10]) where the expected number of children (i.e., recursive calls that preserve a particular collision) is $\Pr[h(x) = h(y)]/p_1$ for random $h \in \mathcal{H}$. It is a standard fact from this theory that the expected population size at generation i (i.e., number of times (x, y) is in a problem at recursive level i) is $(\Pr[h(x) = h(y)]/p_1)^i$ [10, Theorem5.1]. If $d(x, y) > cr$, we have that $\Pr[h(x) = h(y)] \leq p_2$ and each far pair appears $(p_2/p_1)^i$ times in expectation at level i, from which follows Equation 1. Moreover, since the probability of collisions is monotonic in the distance, we have that $\Pr[h(x) = h(y)] \leq 1$ if $d(x, y) \leq r$, and $\Pr[h(x) = h(y)] \leq 1/p_1$ if $r < d(x, y) \leq cr$, from which follow Equations 2 and 3.

In order to get the last bound we observe that each entry of R and S is replicated $L^i = p_1^{-i}$ times at level i. Thus, we have that $N 2^{k+1} L^i$ is the total maximum number of far collisions in subproblems at level i where the smallest input set has size in $[2^k, 2^{k+1})$. Each one of these collisions survives up to level i with probability p_2^i, and thus the expected number of these collisions is $N 2^{k+1} (p_1/p_2)^i$. □

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

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

where $\ell = \lceil \log_{1/p_2}(N/M) \rceil$. By noticing $C_{i,\log M}(R \bowtie_{>cr} S) \leq C_i(R \bowtie_{>cr} S)$ we have $|R \bowtie_{>cr} S| \leq N^2$ and $C_i(R \bowtie_{\leq cr} S) = C_i(R \bowtie_{\leq r} S) + C_i(R \bowtie_{>r,\leq cr} S)$, 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 i-th 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 $\tilde{\mathcal{O}}\left(\kappa \frac{|R \bowtie_{\leq r} S|}{B}\right)$ 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 i-th 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 $L' = (N/M)^\rho$.

Theorem 4. *The above approaches guarantee that each near pair is emitted with constant probability in* ASimJoin *and in* OSimJoin.

Proof. The claim easily follows for the cache-aware algorithm: indeed the two points of a near pair (x, y) have the same hash value in $p(x, y)L$ (in expectation) of the $L' = (N/M)^\rho$ repetitions of Step 2. Therefore, by emitting the pair with probability $1/(p(x, y)L)$ we get the claim.

We now focus on the cache-oblivious algorithm, where the claim requires a more articulated proof. Consider a near pair (x, y). Let G_i and H_i be random variables denoting respectively the number of subproblems at level i containing the pair (x, y), and the number of subproblems at level i where (x, y) is not found by the cache-oblivious nested loop join algorithm in Theorem 2. Let also K_i be a random variable denoting the actual number of times the pair (x, y) is emitted at level i. We have: (1) $\mathbb{E}[K_i|G_i, H_i] = (G_i - H_i)/(p(x, y)L)^i$ since a near pair is emitted with probability $1/(p(x, y)L)^i$ only in those subproblems where the pair is found by the join algorithm; (2) $\mathbb{E}[G_i] = (p(x, y)L)^i$ since a near pair is in the same bucket with probability $p(x, y)^i$ (it follows from the previous analysis based on standard branching); (3) $G_0 = 1$ since each pair exists at the beginning of the algorithm; (4) $H_\Psi = 0$ since each pair surviving up to the last recursive level is found by the nested loop join algorithm.

We are interested in upper bounding $\mathbb{E}\left[\sum_{i=0}^{\Psi} K_i\right]$. We prove by induction that

$$\mathbb{E}\left[\sum_{i=0}^{l} K_i\right] = 1 - \frac{\mathbb{E}[H_l]}{(p(x, y)L)^l},$$

for any $0 \leq l \leq \Psi$. For $l = 0$ (i.e., the first call to OSimJoin) the equality is verified since

$$\mathbb{E}[K_0] = \mathbb{E}[\mathbb{E}[K_0|G_0, H_0]] = \mathbb{E}[G_0 - H_0] = 1 - \mathbb{E}[H_0],$$

since $\mathbb{E}[G_0] = G_0 = 1$. Consider now a generic level $l > 0$. Since a pair propagates in a lower recursive level with probability $p(x, y)$, we have

$$\mathbb{E}[G_l] = \mathbb{E}[\mathbb{E}[G_l|H_{l-1}]] = p(x, y)L\mathbb{E}[H_{l-1}].$$

Thus

$$\mathbb{E}[K_l] = \mathbb{E}[\mathbb{E}[K_l|G_l, H_l]] = \mathbb{E}\left[\frac{G_l - H_l}{(p(x,y)L)^l}\right] = \frac{\mathbb{E}[H_{\ell-1}]}{(p(x,y)L)^{l-1}} - \frac{\mathbb{E}[H_\ell]}{(p(x,y)L)^l}$$

By exploiting the inductive hypothesis, we get

$$\mathbb{E}\left[\sum_{i=0}^{l} K_i\right] = \mathbb{E}[K_l] + \mathbb{E}\left[\sum_{i=0}^{l-1} K_i\right] = 1 - \frac{\mathbb{E}[H_l]}{(p(x,y)L)^l}.$$

Since $H_\Psi = 0$, we have $\mathbb{E}\left[\sum_{i=0}^{\Psi} K_i\right] = 1$ and the claim follows. □

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. Thus, the probability of emitting a pair is at least $\Omega\left(1/\sqrt{\Psi}\right)$ as shown in the second part of Section 3.3 and $\mathcal{O}\left(\log^{3/2} N\right)$ repetitions of OSimJoin suffices to find all pairs with high probability (however, the expected number of replica of a given near pair becomes $\mathcal{O}\left(\log^{3/2} N\right)$, even with the proposed approach).

4 Experiments

As discussed above, we informally argue that our upper bounds are close to the best possible for the class of "signature based" algorithms. We again split the I/O complexity of our algorithms in two parts:

$$T_1 = (N/M)^\rho\big(N/B + |R \underset{\leq r}{\bowtie} S|/(MB)\big)$$
$$T_2 = |R \underset{\leq cr}{\bowtie} S|/(MB)$$

and carry out experiments to demonstrate that the first term T_1 often dominates the second term T_2 in real datasets. In particular, we depict the cumulative distribution function (cdf) in log-log scale of all pairwise distances (i.e., ℓ_1, ℓ_2) and all pairwise similarities (i.e., Jaccard and cosine) on two commonly used datasets: Enron Email[5] and MNIST[6], as shown in Figure 1. Since the Enron data set does not have a fixed data size per point our considerations consider a version of the data set where the dimension has been reduced such that each vector has a fixed size.

[5] https://archive.ics.uci.edu/ml/datasets/Bag+of+Words
[6] http://yann.lecun.com/exdb/mnist/

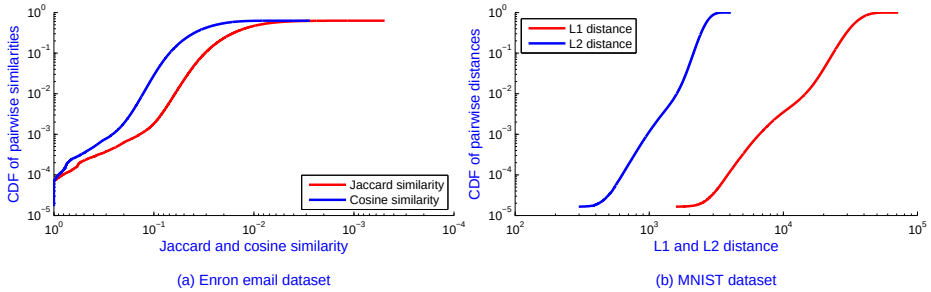

Fig. 1. The cumulative distributions of pairwise similarities and pairwise distances on samples of 10,000 points from Enron Email and MNIST datasets. We note that values decrease on the x-axis of Figure 1.a, while they increase in Figure 1.b.

Figure 1.a shows an inverse *polynomial* relationship with a small exponent m between similarity threshold s and the number of pairwise similarities greater than s. The degree of the polynomial is particularly low when $s > 0.5$. This setting $s > 0.5$ is commonly used in many applications for both Jaccard and cosine similarities [2, 3, 14]. Similarly, Figure 1.b also shows a *monomial* relationship between the distance threshold r and the number of pairwise distances smaller than r. In turn, this means that the number of c-near pairs $|R \bowtie_{\leq cr} S|$ is not much greater than $c^m|R \bowtie_{\leq r} S|$. In other words, the second term T_2 is often much smaller than the first term T_1.

Finally, for the same data sets and metrics, we used the cache-aware algorithm with explicit constants and examined the I/Os used compared to a standard nested loop method (Section 2) and a lower bound on the standard LSH method (Section 2). We used cache size $M = N/1000$, which is reasonable for judging number of cache misses since the size ratio between CPU caches and RAM is in that order of magnitude. In general this setting allows us to investigate what happens when the data size is much larger than fast memory. For simplicity we use $B = 1$ since all methods contain a multiplicative factor $1/B$ on the I/O complexity. The used values of ρ were computed using good LSH families for the specific metric, given r and c, which are picked according to Figure 1 such that the number of c-near pairs are only an order of magnitude larger than the number of near pairs. The I/O complexity used for nested loop join is $2N + N^2/MB$ (here we assume both sets have size N) and the complexity for the standard LSH approach is *lower bounded* by sort $\left(N^{1+\rho}\right)$. This complexity is a lower bound on the standard sorting based approaches as it lacks the additional cost that depends on how the LSH distributes the points. Since $M = N/1000$ we can bound the log-factor of the sorting complexity and use sort $(N) \leq 8N$ since $2N$ points read and written twice. The I/O complexity of our approach is stated in Theorem 3. The computed I/O-values in Figure 2 show that the complexity of our algorithm is lower on all instances examined. Nested loop suffers from quadratic dependency on N, while the standard LSH bounds lack the dependency on M. Overall the

Data set	Metric	r	cr	ρ	$\frac{\lvert R \bowtie_{\leq r} S\rvert}{N}$	$\frac{\lvert R \bowtie_{\leq cr} S\rvert}{N}$	Standard LSH	Nested loop	ASimJoin
Enron	Jaccard	0.5	0.1	0.30	$1.8 \cdot 10^3$	$16 \cdot 10^3$	$> 7.5 \cdot 10^9$ I/Os	$8 \cdot 10^9$ I/Os	$3.2 \cdot 10^9$ I/Os
Enron	Cosine	0.7	0.2	0.51	$1.6 \cdot 10^3$	$16 \cdot 10^3$	$> 212 \cdot 10^9$ I/Os	$8 \cdot 10^9$ I/Os	$6.6 \cdot 10^9$ I/Os
MNIST	L1	3000	6000	0.50	1.8	42	$> 29 \cdot 10^6$ I/Os	$60 \cdot 10^6$ I/Os	$12 \cdot 10^6$ I/Os

Fig. 2. Examples of similarity joins on our data sets. For each join a value of the approximation parameter c has been chosen based on estimates of the distance distribution to ensure that the number of c-near neighbors was not too much higher than the output set. The numbers stated are extrapolated from the sample, similarly to Figure 1, and expressed in join results per point. Corresponding ρ-values have been computed using standard optimal LSH families for the similarity measures. (We omit the L2 metric for this reason since there is no clear winner in choice of LSH for this measure. Since we want to explore the effect of data sets much larger than fast memory, the stated I/Os are under the assumption of $M = N/1000$ and $B = 1$ due to a $1/B$ factor being present in all terms. Our new algorithm uses the least I/Os on the examined data.)

computed values points towards that our algorithm is practical on the examined data sets.

5 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. Jeffrey Scott Vitter. *Algorithms and Data Structures for External Memory*. Now Publishers Inc., 2008.
14. 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.