Künstl Intell (2018) 32:91–94
DOI 10.1007/s13218-017-0515-7

Randomized Primitives for Big Data Processing

Morten Stöckel[1]

Published online: 10 November 2017

Abstract A basic question on two pieces of data is: "What is the similarity of the data?" In this extended abstract we give an overview of new developments in randomized algorithms and data structures that relate to this question. In particular we provide new state of the art methods in three particular settings, that all relate to the computation of intersection sizes:

1. We give a new space-efficient summary data structure for answering set intersection size queries. The new summaries are based on one-permutation min-wise hashing, and we provide a lower bound that nearly matches our new upper bound.
2. For sparse matrix multiplication, we give new tight bounds in the I/O model, settling the I/O complexity a natural parameterization of the problem—namely where the complexity depends on the input sparsity N, the output sparsity Z and the parameters of the I/O model. In the RAM model we give a new algorithm that exploits output sparsity and which beats previous known results for most of the parameter space.
3. We give a new I/O efficient algorithm to compute the similarity join between two sets: two elements are members of this join if they are close according to a specified metric. Our new algorithm is based on locality-sensitive hashing and strictly improves on previous work.

This work is supported by the Danish National Research Foundation under the Sapere Aude program.

✉ Morten Stöckel
most@di.ku.dk

[1] IT University of Copenhagen, Copenhagen, Denmark

Keywords Set intersection · Matrix multiplication · Similarity join · min-wise hashing · Communication complexity

1 Introduction

The amount of data residing in the world is enormous and rapidly growing—we live in the age of *information explosion*. Performing meaningful actions on data, such as searching or sorting, generally takes computational power proportional to the amount of data that needs to be processed. As the amount of data grows, more computational power is required to process the data in a timely manner. This motivates research in more efficient algorithms and data structures for big data processing. The contribution of this work [12] is new and efficient data processing methods for a particular family of actions. Namely, we investigate the following basic, and for now intentionally informal, question: *What is the intersection between two pieces of data?* We consider several formal settings where intersections between data is the quantity of interest and we provide new algorithms and data structures that give stronger theoretical guarantees than previous state of the art. In this extended abstract we will highlight three such formal settings – multiple set intersection estimation (Sect. 2), sparse matrix multiplication (Sect. 3), and similarity join (Sect. 4).

The three highlighted settings mentioned above fall under the umbrella of *intersection* in the following manner. Consider two bit vectors $a, b \in \{0,1\}^n$, a standard metric for similarity between big strings is simply their dot product $\langle a, b\rangle = \sum_{i\in[n]} a_i b_i$. The bit strings a, b can be considered the indicator vectors for sets S_a and S_b, i.e we have $S_a, S_b \subseteq [n]$. The intersection size $|S_a \cap S_b|$ is then exactly equal to $\langle a, b\rangle$. All three considered settings deal with the computation of dot products and set intersection sizes. First, in Sect. 2 we will investigate how to *summarize* a collection of sets, such that all intersection sizes between subsets of the collection can be answered approximately. Then in Sect. 3 we will consider matrix multiplication, which can be seen as a batch of dot products. In particular we are interested in the case where either the input matrices or output matrix (or both) are sparse, i.e., contain a lot of zeroes. Motivated by the fact that the intersection size can be seen as a *distance metric*, we finally consider in Sect. 4 the case where we have two collections of points from a given metric space, and we wish to output all pairs with one from each set, that are close as defined by the distance metric. This is exactly standard set intersection , except two points belong to the intersection when they are close instead of being equal.

All three sections will aim to summarize our findings and we refer to the thesis [12] as well the subsumed papers [4, 6, 8–11] for the full technical details.

2 Multiple Set Intersection

In this work the aim is, given m input sets $\mathcal{S} = S_1, \ldots, S_m$, to pre-process $\mathcal{S}$ into a space-efficient *summary* of the set. From the summaries we must be able to output an estimate of the quantity $t = \left|\bigcap_{j\in q} S_j\right|$, where $q \subseteq \{1, \ldots, m\}$ is a query that is not known in advance of the pre-processing. The setting is illustrated in Fig. 1. Further, we require that all summaries are computed independently of each other, i.e., no information from other summaries can be used. This setting is motivated by the fact that data sizes can be so large that sub-linear space can be needed, and the

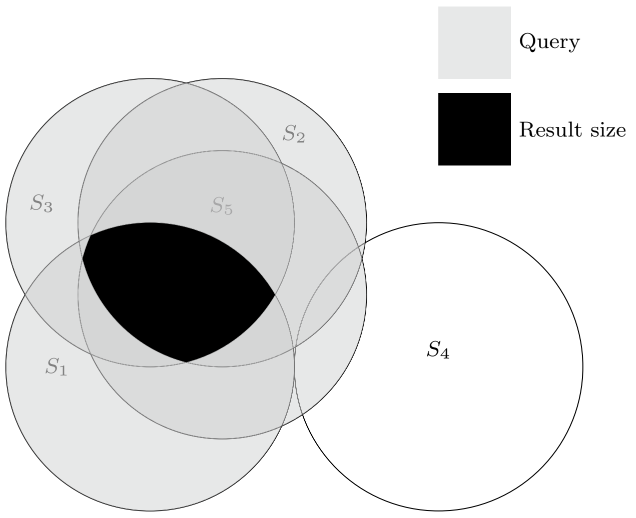

Fig. 1 The setting. We have multiple sets $S_1, \ldots, S_5$ with the sets S_1, S_2, S_3, S_5 being the query sets (grey color) and the return value is the size of the black region

independent computation of the summaries enables the method to be easily distributed. Even though we compute only an estimate of the intersection size, our method has applications in exact computation as well: conjunctive queries in databases use computational time proportional to the intersection between two relations. If a good estimate of this intersection size can be estimated beforehand, the order of computation can be optimized to gain computational speedups.

In this work we are interested in achieving an (ε, δ) -approximation of the intersection size t, that is, the probability that our output $\tilde{t}$ has relative error more than ε must be at most δ. Before our work, the state of the art to achieve a good (ε, δ)-approximation of t was a variant of the well-known min-wise hashing [2], namely b-bit min-wise hashing by Li and König (see [7] for an overview). The overall approach of Li and König can be summarized as follows. Say we choose to store k elements from each set in $S_i \in \mathcal{S}$. Then they compute k random hash functions independently, and for each hash function they store the element from S_i with the smallest hash value. We consider a simple twist on this approach—instead of k hash functions that each determine one element to store, we use one hash function that determines all k elements to store! That is, we select the elements with the k smallest hash values. The argument relies on the subtle technical observation that when using k hash functions, an element from the intersection gets added with probability roughly $t/|\cup_i S_i|$ while when using only one hash function this event occurs with probability roughly $t\,/\,n$, where n is the maximum set size. Queries are answered by computing the intersection size of the summaries of the query sets, and scaling by $n\,/\,k$—the intuition being that for a particular summary, an intersection element is added to it with probability roughly $t\,/\,n$, and we perform k such “repetitions”. Our results hold for other set sizes as well, in which case the dependence on n is replaced with a dependence on the largest set size.

Our results can be summarized as the answer to the question *Is b-bit min-wise hashing optimal for summarizing set intersection?* When the number of sets $m = 2$ the answer is yes. This space optimality is a consequence of a new lower bound shown in this work, and the lower bound holds for any summary. When $m \geq 2$ is larger, we show that the performance of b-bit min-wise hashing (and more generally any method based on “k-permutation” min-wise hashing) deteriorates as m grows. Our new summary almost matches our new lower bound and improves upon the space consumption of b-bit min-wise hashing (and other “k-permutation schemes”) by a factor of $\Omega(m/\log m)$. Our results are summarized in Table 1, which also includes previous work discussed in detail in [12].

Table 1 Comparison of estimators of intersection size t for relative error ε and constant error probability, with m sets of maximum size n. Bounds on the summary size s ignore constant factors

Method	Required space (bits)	Time
Inclusion–exclusion	$s \geq \varepsilon^{-2}(mn/t)^2 + \log n$	2^m
Subsampling	$s \geq \varepsilon^{-2}(n/t) \log m \log^2 n$	sm
b-bit min-wise hashing	$s \geq \varepsilon^{-2}(mn/t)$	sm
New upper bound	$s \leq \varepsilon^{-2}(n/t) \log(m) \log(n/\varepsilon t)$	sm
New lower bound	$s \geq \varepsilon^{-2}(n/t)$	-

3 Sparse Matrix Multiplication

We now turn to the classic problem of computing matrix products. This falls under our "data intersection" umbrella by observing that a boolean matrix product can be seen as a batch computation of inner products between vectors of indicator variables. Matrix multiplication is widely used in algorithm design, perhaps most notably in algorithmic graph theory where the performing matrix multiplications that involve the adjacency matrix of a graph is a powerful tool. Matrix multiplication is also widely used in practice, e.g. in computer graphics and computational linear algebra (see [12]).

A $U \times U$ matrix A can be seen as U row vectors each containing U values. The matrix multiplication problem can be stated as follows: Let the *input matrices* be A and C both of size $U \times U$. The task is then to multiply A and C to create the *output matrix* AC which is also of dimensions $U \times U$. If the values of the *entries* of A and C can only take values 0 and 1 we call it boolean matrix multiplication. Letting $[U]$ denote the set $\{0, \dots, U-1\}$, the output matrix AC is defined as

$$\forall i,j \in [U] \;:\; (AC)_{i,j} = \sum_{k=0}^{U-1} A_{i,k} C_{k,j} \,. \tag{1}$$

We will seek to compute (1) efficiently in both the standard RAM model and also in the I/O model. Recall that in the I/O model we have a fast memory of size M, a disk of infinite size, and a block transfer between disk and memory can hold B words. All computation takes place in the memory, but the cost of an algorithm is exclusively the maximum number of block transfers used by an algorithm.

In the RAM model the standard solution is to compute 1 directly using $\mathcal{O}(U^3)$ time. When we allow certain arithmetic operations over the ring, i.e. "Strassen-like" tricks [13], this can be improved to $\mathcal{O}(U^\omega)$, where currently $\omega < 2.3728639$ [3]. In the I/O model the basic solution is to divide the matrices into size $c\sqrt{M} \times c\sqrt{M}$ blocks where c is picked such that three $c\sqrt{M} \times c\sqrt{M}$ fit into internal memory of size M. This reduces the problem to $\mathcal{O}((U/\sqrt{M})^3)$ matrix products that fit in main memory, costing $\mathcal{O}(M/B)$ I/Os each, and hence $\mathcal{O}(U^3/B\sqrt{M})$ in total [5].

We consider the problem of *sparse* matrix multiplication. We say that parameter N is the number of non-zero entries in the input matrices, and Z is the number of non-zero entries in the output matrix. We are now interested in efficient algorithms that rely on this natural parameterization on N and Z instead of the dimension U.

Our results for this problem are two new state of the art algorithms in the RAM model and the I/O model, respectively. Our algorithms are Monte Carlo algorithms, they have polynomially small error probability, where the polynomial depends on the constant factor hidden in the running time.

In the I/O model we settle the I/O complexity of sparse matrix multiplication: We provide a new algorithm and a matching lower bound, both parameterized on input sparsity N and output sparsity Z. Our new upper bound improves previous state of the art [1] by a factor of $M^{3/8}$. We leave details to [12], but remark that the argument relies on partitioning the input into many small subproblems that each have a small number of non-zeroes in their output – on our way to achieve this we develop a new size estimation tool that estimates the number of non-zeroes in the output of a matrix product, and this method runs in roughly N / B I/Os. Our results in the I/O model can be summarized as:

1. We show that using $\tilde{\mathcal{O}}\left(\frac{N}{B} \min\left(\sqrt{\frac{Z}{M}}, \frac{N}{M}\right)\right)$ I/Os, AC can be computed with high probability. This is tight (up to polylogarithmic factors) when only semiring operations are allowed (to rule out Strassen-like algorithms), even for dense rectangular matrices:
2. We show a lower bound of $\Omega\left(\frac{N}{B} \min\left(\sqrt{\frac{Z}{M}}, \frac{N}{M}\right)\right)$ I/Os, hence closing the gap between lower and upper bounds for this problem.

In the RAM model we seek to use fast matrix multiplication, i.e., the $\mathcal{O}(U^\omega)$ method to compute fast matrix products. Due to the nature of the black box, it proves difficult to use the input sparsity N, and so our new algorithm depends on U and Z. Our algorithm uses the size estimation tool mentioned above to split the input matrices cleverly. We achieve a running time of $\tilde{\mathcal{O}}\big(U^2(Z/U)^{\omega-2} + Z + N\big)$, where $\tilde{\mathcal{O}}(\cdot)$ hides polylog factors in U. This bound is asymptotically better than previous state of the art when Z is asymptotically larger than U, and when $Z = U$ we coincide with $\mathcal{O}(U^\omega)$. This algorithm can be transfered to the I/O model, in which unless M is very large our new algorithm beats all previous methods.

4 Similarity Join

Finally, we consider the case where we have two sets, and two elements are a part of the intersection between two sets if they are "close enough", but not necessarily equal. The problem is closely related to similarity search, and indeed our contribution uses particular hash functions that have gained popularity due to similarity search. The problem of computing similarity joins has applications such as web deduplication, document clustering and click fraud detection.

For a space $\mathcal{U}$ and a distance function $d : \mathcal{U} \times \mathcal{U} \to \mathbf{R}$, the *similarity join* of sets $R, S \subseteq \mathcal{U}$ is the following: Given a radius r, compute the set

$$R \underset{\leq r}{\bowtie} S = \{(x, y) \in R \times S \mid d(x, y) \leq r\} .$$

Let the total number of points be $N = |R| + |S|$. Since there are $\mathcal{O}(N^2)$ number of pairs the naive approach is to just perform this number of distance evaluations and so the trivial time bounds for this problem depend on this number of pairs. We go below this barrier by using a special kind of hash function called a *locality sensitive hash function* (LSH): informally such a hash function will guarantee that two close points hash to the same value with probability p_1 and that two far away points hash to the same value with probability p_2 where $p_1 > p_2$. The rough idea is that since close points hash to the same value often and far away points hash to the same value rarely, then by examining only pairs that hash to the same value we avoid looking at many of the far away pairs.

We consider the I/O model only and we use the LSH idea as described above in order to avoid comparing far away pairs. Ideally one would like an algorithm that depends on the output size $|R \bowtie_{\leq r} S|$ and a subquadratic dependency in the number of points N. However it turns out that this requires that the "c-near" join $R \bowtie_{\leq cr} S$, for a constant c, has few pairs in it.

Our result for this problem is the first I/O-efficient (in fact cache-oblivious) algorithm for similarity join that has provably sub-quadratic dependency on the data size and at the same time inverse polynomial dependency on M. The I/O complexity of our algorithm is

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

Here $\rho \in (0;1)$ is a parameter of the LSH. Where previous methods have an overhead factor of either N / M or $(N/B)^{\rho}$ we obtain an overhead of $(N/M)^{\rho}$, strictly improving both.

References

1. Amossen RR, Pagh R (2009) Faster join-projects and sparse matrix multiplications. In: Proceedings of the 12th international conference on database theory, ICDT '09, St. Petersburg, Russia, 23–25 March, pp 121–126
2. Broder AZ (1997) On the resemblance and containment of documents. In: Proceedings of the compression and complexity of sequences. SEQUENCES '97. IEEE Computer Society, Washington, DC, p 21
3. Gall FL (2014) Powers of tensors and fast matrix multiplication. Int Symp Symb Algebraic Comput ISSAC 2014:296–303
4. Jacob, R, Stöckel M (2015) Fast output-sensitive matrix multiplication. In: Algorithms, ESA 2015: 23rd annual European symposium, Proceedings. Springer, Heidelberg, pp 766–778
5. Jia-Wei H, Kung HT (1981) I/o complexity: the red–blue pebble game. In: Proceedings of the thirteenth annual ACM symposium on theory of computing, STOC '81, ACM, pp 326–333
6. Knudsen MBT, Stöckel M (2015) Quicksort, largest bucket, and min-wise hashing with limited independence. In: Algorithms, ESA 2015, 23rd annual European symposium, Proceedings, pp 828–839
7. Li P, König AC (2011) Theory and applications of b-bit minwise hashing. Commun ACM 54(8):101–109
8. Pagh R, Pham N, Silvestri F, Stöckel M (2015) I/O-efficient similarity join. In: Algorithms, ESA 2015: 23rd annual European symposium, Proceedings. Springer, Heidelberg, pp 941–952
9. Pagh R, Stöckel M (2014) The input/output complexity of sparse matrix multiplication. In: Algorithms, ESA 2014: 22th annual European symposium, Proceedings. Springer, Heidelberg, pp 750–761
10. Pagh R, Stöckel M (2015) Association rule mining using maximum entropy. CoRR. arxiv:1501.02143
11. Pagh R, Stöckel M, Woodruff DP (2014) Is min-wise hashing optimal for summarizing set intersection? In: Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems, PODS'14. ACM, pp 109–120
12. Stöckel M (2015) Randomized primitives for big data processing. https://en.itu.dk/~/media/en/research/phd-programme/phd-defences/2015/morten-st%C3%B6ckel-phd-thesis-final-pdf.pdf?la=en
13. Strassen V (1969) Gaussian elimination is not optimal. Numer Math 13(4):354–356