Structure Prediction and Computation of Sparse Matrix Products

Edith Cohen
AT&T Labs–Research
Murray Hill, NJ 07974
edith@research.att.com

Revised: May 18, 1997

Abstract

We consider[1] the problem of predicting the nonzero structure of a product of two or more matrices. Prior knowledge of the nonzero structure can be applied to optimize memory allocation and to determine the optimal multiplication order for a chain product of sparse matrices. We adapt a recent algorithm by the author and show that the essence of the nonzero structure and hence, a near-optimal order of multiplications, can be determined in near-linear time in the number of nonzero entries, which is much smaller than the time required for the multiplications. An experimental evaluation of the algorithm demonstrates that it is practical for matrices of order 10^3 with 10^4 nonzeros (or larger). A relatively small pre-computation results in a large time saved in the computation-intensive multiplication.

1. Introduction

Large sparse matrices occur in the solutions of many practical problems. Many matrix operations that are computation intensive or infeasible on the full representation of the matrix become feasible when sparsity is exploited. Indeed, extensive literature exists on more efficient storage schemes and algorithms for matrix operations on sparse matrices. (See, e.g.,[12] [13] [6]). Since sparse matrices are easier to manipulate, much effort was made on minimizing *fill-in* (nonzeros introduced during some computation) when there is freedom in the choice of operations. (For example, minimizing fill-in in Gaussian eliminations.) It is also common to represent matrices as products of sparser matrices. To exploit sparsity, it is of value to be able to determine the nonzero structure of an output matrix of a matrix operation in advance (see Gilbert and Ng [9] and Gilbert [10] for a survey). Many matrix computations have an initial phase that predicts the nonzero structure of the output matrix, followed by the actual numerical computation. Prior knowledge of the nonzero structure is used to improve memory allocation, data structure setup, and running time.

We consider the problem of determining the nonzero structure of products of sparse matrices. The intention is to predict the structure using a low cost procedure, before performing the computation-intensive matrix multiplication. The nonzero structure can serve as a guide for efficient memory allocation for the output matrix when performing the actual multiplication. The

[1] A preliminary version of these paper appeared in IPCO 96 conference [2]

MATLAB system, for example, utilizes such predictions [8]. When the prediction is off (allocated size is too small) the system allocates another memory block, larger by a constant factor, copies over columns of the product that were computed so far, and frees the current block. Hence, good predictions on the product size are of value. Another application for prediction of the nonzero structure is to determine the *optimal order* of multiplications (that is, the association that minimizes the total number of operations) when computing a chain product of three or more matrices. Consider the product $\mathbf{A}_1 \cdots \mathbf{A}_k$ of $k \geq 3$ sparse matrices. The multiplication can be performed in many orders. The number of operations can vary significantly for different orders. The nonzero structure is used to determine the least-expensive order in which to perform the multiplications. For dense matrix multiplications, the optimal order is determined solely by the dimensions of the matrices, and can be obtained efficiently by dynamic programming [14]. For sparse multiplication algorithms, however, the optimal order varies significantly even for matrices of the same dimensions.

Throughout the paper we assume *no cancellation*. That is, the inner product of two vectors with overlapping nonzero entries never cancels to zero. This is a very common assumption in the mathematical programming literature. The justification is that cancellations are unlikely when computing over real numbers. Furthermore, even when (due to a singular structure) cancellation occurs, rounding errors may prevent the resulting computed value from being zero.

Even with the no cancellation assumption it is costly to determine the full nonzero structure (exact locations of all nonzero entries) of a matrix product. This operation amounts to performing a boolean matrix multiplication (transitive closure computation). Known boolean matrix multiplication algorithms require the same number of operations to predict the structure as to perform the actual multiplication. (To be precise, in practice each "operation" in a real matrix multiplication is a floating point multiply-add. These are much more expensive than boolean and-or operations and hence it is still much cheaper to perform a boolean multiplication. However, the cost of boolean sparse matrix multiplication still grows quadratically with the average number of nonzeros in each row.)

We consider a relaxed goal of determining the number of nonzeros in each row and column of the product. This partial knowledge is as valuable as the full structure for optimizing memory allocation, data structure setup, and determining the optimal association for computing chain products. We introduce an algorithm where each operation is a simple compare-store operation on small integers and also the number of operations is linear in the number of nonzeros.

The obvious method of quickly estimating row and column densities in a matrix product assumes random distributions of nonzero entries. This assumption, however, often does not hold in practice. The method we propose, based on an adaptation of a recent reachability-set size estimation algorithm [1], obtains accurate estimates for products of *arbitrary* matrices. We also provide better estimators and analysis. We estimate the densities of all rows and columns of the product matrix in time linear in the number of nonzero entries of the input matrices. We introduce an algorithm that for chain products of $k > 3$ matrices, proposes a multiplication-order that (nearly) minimizes the number of operations. The algorithm combines the estimation technique with dynamic programming and runs in $O(kZ + k^3 N)$ time, where Z is the total number of nonzeros in the input matrices and N is the maximum dimension (number of rows or columns) of an input matrix.

We experimentally tested our method on products of 3 randomly-generated large sparse matrices. We introduced correlations between nonzero locations that model patterns arising in "real" data. We also experimented with data obtained in a Text Retrieval application. For these matrices, there was a significant difference in the costs of the two multiplication orders. For comparison, we also applied other estimation methods that assume random nonzero distributions. Generally, these methods produced inferior estimates. We concluded that even for matrices of dimension 10^3 with 2×10^4 nonzeros it is cost-effective to use our algorithm prior to performing the multiplication. The number of operations performed by the estimation routine was small in comparison to the gain obtained by selecting the less-costly association. The potential savings are even larger for longer chain products and larger matrices.

In Section 2 we provide background material and review graph representation of matrices and sparse matrix multiplication. In Section 3 we present estimation algorithms for number of nonzeros in rows and columns of a product. In Section 4 we present algorithms for determining a near-optimal association for computing a chain product. Section 5 contains experimental performance study. Section 6 contains a summary.

2. Preliminaries

Extensive literature exists on storage schemes and implementing matrix operations on sparse matrices (see, e.g., [13, 16]). We review some relevant material. We denote by $[A]_{i,j}$ the entry at the ith row and jth column of a matrix $\mathbf{A}$, by $[A]_{i\cdot}$ the ith row, by $[A]_{\cdot j}$ the jth column, and by $|\mathbf{A}|$ the number of nonzero entries in the matrix $\mathbf{A}$. We use the term *size* of a column, a row, or a matrix to refer to the number of nonzero entries.

2.1. Graph representation of matrices and matrix-products

We represent a matrix $\mathbf{A}$ of dimension $n_1 \times n_2$ by a bipartite graph G_A, with n_1 nodes $\{v_1, \ldots, v_{n_1}\}$ at the lower level and n_2 nodes $\{v'_1, \ldots, v'_{n_2}\}$ at the upper level. For $1 \leq i \leq n_1$ and $1 \leq j \leq n_2$, there is a directed edge from v_i to v'_j if and only if $[A]_{ij} \neq 0$. It is easy to see that the outdegree of v_i (resp., indegree of v'_j) is the size of the ith row (resp., jth column) of $\mathbf{A}$.

We represent a product $\mathbf{A} = \mathbf{A}_1 \cdots \mathbf{A}_k$ as a layered graph with $(k+1)$ layers as follows. Suppose $\mathbf{A}_i$ is of dimensions $n_i \times n_{i+1}$ $(1 \leq i \leq k)$. For $1 \leq i \leq k$, the ith layer has n_i nodes and the edges between the ith and the $(i+1)$st layers are as is the bipartite graph G_{A_i}.

For $1 \leq r < r' \leq k$, consider the subproduct $\mathbf{B} = \mathbf{A}_r \mathbf{A}_{r+1} \cdots A_{r'}$. $\mathbf{B}$ is of dimension $n_r \times n_{r'+1}$. The following proposition is immediate.

Proposition 2.1 *Assuming no cancellation*

1. *The number of nonzero entries in the ith row of* $\mathbf{B}$ *equals the number of nodes in layer* $r'+1$ *that the ith node in layer* r *reaches via directed paths.*
2. *The number of nonzero entries in the ith column of* $\mathbf{B}$ *is equal to the number of nodes in layer* r *that can reach the ith node in layer* $r'+1$.

2.2. Storage schemes of sparse matrices

A good storage scheme that exploits sparsity should be both compact (memory usage of the order of the number of nonzero entries) and facilitate an efficient computation of the applicable matrix operations. (See, e.g. [13] for a comprehensive discussion and references.) The simplest representation is an array or a list of all nonzero entries, where for each entry we store its contents and location. Common representations that allow for efficient access to the content of specified rows or columns are the *sparse row-wise* and *sparse column-wise* formats. The sparse row-wise representation consists of an array of indexes to the contents of the rows of the matrix. Each row is stored as a linked list of the nonzero entries in the row. For each nonzero in the row the list contains a record of the content and the column number. (Most sparse matrix computation systems store a sparse row vector in a pair of matched arrays, one array for the nonzeros and the other for respective column indices [7, 8].) A column-wise format is similar to row-wise when the roles of rows and columns are reversed. For efficient implementation of sparse multiplication of two matrices $\mathbf{AB}$ it suffices that either the matrix $\mathbf{A}$ is represented in column-wise format or $\mathbf{B}$ is in row-wise format. Our algorithm for estimating column and row sizes of products of arbitrary matrices can be implemented efficiently even when each matrix in the product is represented as an unsorted array of all nonzero locations. (Naturally, the row-wise or column-wise formats are also suitable.)

2.3. Sparse matrix multiplication

Dense matrix multiplication algorithms perform near cubic number of operations in the dimension of the matrices (a practical subcubic algorithm is [15], the best known but impractical bounds is given by [3]). When the matrices have a relatively small number of nonzero entries, the sparsity can be exploited to significantly reduce the amount of storage and running time. Consider the product of the matrices $\mathbf{A}^{n_1\times n}$ and $\mathbf{B}^{n\times n_2}$. Every entry $[AB]_{i_1,i_2}$ in the product is expressed as the sum of products

$$[AB]_{i_1,i_2} = \sum_i [A]_{i_1,i}[B]_{i,i_2} \ .$$

Sparse multiplication considers the product of $[A]_{i_1,i}$ and $[B]_{i,i_2}$ only if both are nonzero. The total number of such pairs and a good measure for the number of operations performed by a sparse multiplication of (appropriately-represented) matrices is

$$\sum_{i=1}^{n} |[A]_{\cdot i}||[B]_{i\cdot}| = \sum_{i=1}^{n} |\{j|[A]_{ji} \neq 0\}||\{k|[B]_{ik} \neq 0\}| \ . \tag{1}$$

The sum over the columns of $\mathbf{A}$ (rows of $\mathbf{B}$), of the products of the size of the ith column of $\mathbf{A}$ and the size of the ith row of $\mathbf{B}$. Hence, the cost of multiplying $\mathbf{A} = \mathbf{A}_1 \cdots \mathbf{A}_i$ and $\mathbf{B} = \mathbf{B}_1 \cdots \mathbf{B}_j$ can be deduced from the number of nonzero entries in each column of $\mathbf{A}$ and each row of $\mathbf{B}$. To see this recall that the product of $\mathbf{A}$ and $\mathbf{B}$ can be expressed as the sum of n outer products: Sum, for $1 \leq i \leq n$, the outer product of the ith column of $\mathbf{A}$ and the ith row of $\mathbf{B}$ (see, e.g., [11]). For every nonzero entry $[A]_{i_1 i_2}$, the sparse multiplication procedure computes its scalar product with every nonzero in the i_2th row of $\mathbf{B}$. (Symmetrically, every nonzero of $\mathbf{B}$ is multiplied by all nonzeros in a corresponding column of $\mathbf{A}$.) Hence, for efficient implementation of sparse multiplication it is desirable to access in linear time in the size either all nonzero elements of each column of $\mathbf{A}$ (have $\mathbf{A}$ in column-wise format) or access all nonzeros of each row of $\mathbf{B}$ (have $\mathbf{B}$ in row-wise format). Sparse multiplication also necessitates a data structure for the output matrix that allows for values to be added to arbitrary locations. A dense ($O(n_1 n_2)$ space) representation is simple but not space and time efficient if the product matrix itself is not dense. Alternatively, a compact row-wise or column-wise format for the product may increase access time (a common approach that does not asymptotically increase time complexity is to use a dense representation only for the current "active" row or column of the output matrix [7, 8].) Our experimental evaluation compares the cost of structure prediction to the cost of multiplication (our structure prediction is effective when its

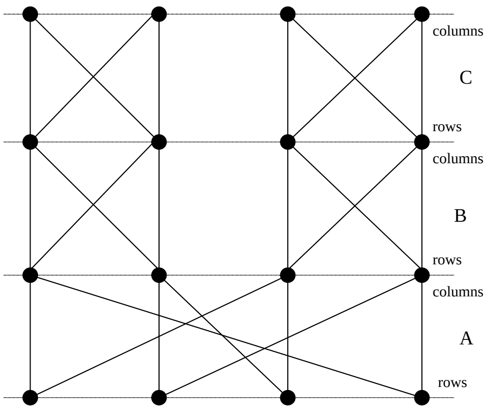


Figure 1: The layered graph of the product **ABC**

cost is small with respect to multiplication cost). We use Equation 1 (number of scalar multiply-adds performed by the sparse-multiplication algorithm.) as a measure of the cost. The cost of multiplying large matrices in practice is often dominated by the transfer of data between slow and fast memory. Hence, measuring the number of operations, although common, is rather simplistic. To justify it, we note that the same memory management procedures applied in the multiplication itself can also be carried out in our structure-prediction algorithms. Furthermore, the structure prediction algorithm fetches the data once. The multiplication algorithm on the other hand may require fetching the data many times. In light of this and the additional costs associated with the data structure for the output matrix (discussion above), Equation 1 is an optimistic measure of the cost. Therefore, the actual gap between the costs of structure prediction and multiplication is likely to be larger.

2.4. Example

Consider the following two nonzero structures (x denotes a nonzero entry).

$$T_1 = \begin{pmatrix} x & x & 0 & 0 \\ x & x & 0 & 0 \\ 0 & 0 & x & x \\ 0 & 0 & x & x \end{pmatrix} \quad T_2 = \begin{pmatrix} x & 0 & x & 0 \\ 0 & x & 0 & x \\ 0 & x & x & 0 \\ x & 0 & 0 & x \end{pmatrix}$$

Let $\mathbf{B}$ and $\mathbf{C}$ be matrices with nonzero structure in the form of T_1. Let $\mathbf{A}$ have the structure of T_2. The goal is to compute the product $\mathbf{ABC}$. See Figure 1 for the graph representation of the product. The product $\mathbf{ABC}$ is evaluated using two matrix multiplications. There are two possible associations (orders of evaluation), either $(\mathbf{AB})\mathbf{C}$ (where $\mathbf{AB}$ is the first multiplication) or $\mathbf{A}(\mathbf{BC})$ (where $\mathbf{BC}$ is the first multiplication). All three matrices have the same column and row sizes (size 2). Hence, computing each of the products $\mathbf{AB}$ or $\mathbf{BC}$ takes 16 operations. (The cost of $\mathbf{AB}$ is the sum of the products of the indegree and outdegree of each node in the second layer and the cost of $\mathbf{BC}$ is the sum for nodes in the third layer.) Note, however, that the product $\mathbf{BC}$ has the form T_1 whereas $\mathbf{AB}$ has all entries nonzero. Hence, the multiplication $\mathbf{A}(\mathbf{BC})$ takes 16 operations but $(\mathbf{AB})\mathbf{C}$ takes 32 operations. Therefore, the evaluation order $\mathbf{A}(\mathbf{BC})$ requires 32 scalar multiplication operations and $(\mathbf{AB})\mathbf{C}$ requires 48 operations. This example demonstrates that knowledge of the row sizes of the product $\mathbf{BC}$ and the column sizes of $\mathbf{AB}$ is needed in order to determine the costs of the second multiplications.

3. Estimating column sizes in a product matrix

The algorithm is based on a reachability-set size estimation algorithm [1]. The size estimation algorithm in [1] produces estimates on the sizes of the reachability sets for all nodes in an arbitrary directed graph. In the context of matrix products, we designate a subset of the nodes and estimate for all other nodes, the number of ancestors or descendents they have from the designated set. We utilize the layered structure of the matrix product graphs to obtain a simpler specialized algorithm.

Consider a matrix product $\mathbf{A}_1 \cdots \mathbf{A}_k$ represented as a $k+1$ layer graph (as in Subsection 2.1). Let $\mathbf{A}_i$ be of dimension $n_i \times n_{i+1}$ $(1 \le i \le k)$. For $1 \le i \le k+1$, denote by V_i the nodes in layer i. We have $|V_i| = n_i$. We use an adapted single application of the estimation algorithm of [1] to produce for each $1 < i \le k$ and each column of the product matrix $\mathbf{A}'_i = \mathbf{A}_1 \cdots \mathbf{A}_i$, an estimate on the size of the column. Each column of the product $\mathbf{A}'_i$ corresponds to a node $v \in V_{i+1}$. The size of the column equals the number of V_1-nodes that can reach v. The size of each row in each of the subproducts $\mathbf{A}_i \cdots \mathbf{A}_k$ $(1 \le i < k)$ can be estimated using a symmetric procedure.

The estimation algorithm gets as a parameter the *number of rounds* r. Both estimate quality and running time increase with r, and hence, it should be set according to the desired performance. We state a column-size estimation algorithm. (The row-size estimation algorithm performs essentially a symmetric computation on the graph with the layers in reverse order.) Each round amounts to first assigning independent random numbers (*keys*) to all V_1 nodes (the first layer). Following that, for all other nodes, the least key of a V_1 node that reaches them is determined. This is performed in time linear in the size of the layered graph, by traversing the layers sequentially and propagating the smallest keys. The intuition, is that for every node $v \in V_i$ $(i > 1)$, the distribution and the expected value of the key of the minimum-ranked V_1-node that reaches v depends solely on the number of V_1 nodes that reach v. Each round provides a single sample for every such node v. The estimation algorithm applies a "reversed" strategy to deduce the size from the least-key samples. The accuracy and confidence of the estimate increase with the number of samples (rounds). Note that the rounds may be performed independently on a parallel platform. The algorithm assumes in Step 2 that the matrices are represented in a column-wise format. That is, every node $v \in V_i$ has an associated list of its neighbors from layer V_{i-1}. An alternate statement for Step 2 that is suitable for matrices represented as an arbitrary list of their nonzero entries is also provided.

3.1. Estimation algorithm for column-sizes

1. Assign for each node in V_1 an r-vector that consists of random samples from the Exponential distribution with parameter $\lambda = 1$ (that is, density function $f(x) = \lambda e^{-\lambda x}$ and distribution function $F(x) = 1 - e^{-\lambda x}$.)

2. For layers $i = 2, \ldots, k+1$ do as follows:
   - For each node $v \in V_i$:
     Assign to v the r-vector that equals the coordinate-wise
     minima of the r-vectors of all the V_{i-1} neighbors of v.

3. At this point, every node in the graph has an r-vector associated with it. To estimate the number of nonzeros of a column in $\mathbf{A}'_j$, apply an *estimator* to the r-vector of the corresponding node from layer $j+1$. For a vector $(a_1, \ldots, a_r)$ we apply the estimator

$$\frac{r-1}{\sum_{i=1}^{r} a_i}$$

($r-1$ divided by the sum of the coordinates.)

We remark that the analysis holds when we use an exponential distribution with $0 < \lambda \neq 1$ in step 1, and divide the estimator (step 3) by λ.

The algorithm above can be implemented efficiently when matrices are provided in column-wise format. When the matrices are provided as unsorted lists of their nonzero locations (that is, for every i, we are provided with a list of all edges between V_i and V_{i+1} in arbitrary order), we replace Step 2 by the following computation:
For layers $i = 2, \ldots, k+1$ do as follows:

- For each node $v \in V_i$ initialize all entries in the r-vector of v to $+\infty$:
- For each edge (v, u) where $v \in V_{i-1}$ and $u \in V_i$: (nonzero entry in $\mathbf{A}_i$)
  Assign to u the r-vector that equals the coordinate-wise minimum
  of u's current vector and the r-vector of v.

3.2. Time bounds

Obtaining the r-vectors requires rn_1 random samples and $r\sum_{i=1}^{k}|\mathbf{A}_i|$ compare and store operations. Computing each estimate amounts to $r-1$ additions and a division. Hence, estimating the column sizes of all submatrices $\mathbf{A}'_i$ requires $\sum_{i=2}^{k+1} n_i$ divisions and $(r-1)\sum_{i=2}^{k+1} n_i$ additions. The total number of operations is dominated by the term $r\sum_{i=1}^{k}|\mathbf{A}_i|$.

3.3. Accuracy and time tradeoffs

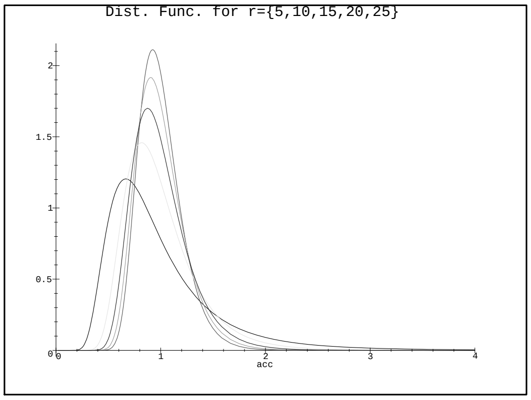
We analyze the quality of each estimate $\hat{z}$ on the number of nonzeros z (in some row or column). Let the *r.v.* $M^{(z)}$ be the minimum key for a (product) column (or row) of size z. $M^{(z)}$ distributes like the minimum of z independent Exponential *r.v.*'s. The sum $S^{(r,z)}$ of r independent *r.v.*'s drawn from $M^{(z)}$ has density and distribution functions

$$\begin{aligned} g_{r,z}(x) &= z\frac{(zx)^{r-1}}{(r-1)!}e^{-zx} \\ G_{r,z}(x) &= 1 - e^{-zx}\left(1 + \sum_{i=1}^{r-1}\frac{(zx)^i}{i!}\right) \ , \end{aligned}$$

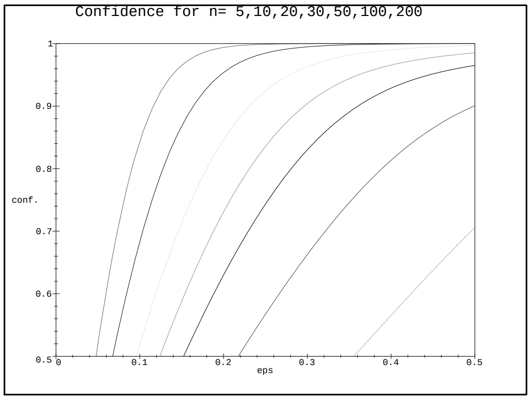
for $x \geq 0$ (see, e.g. [5]). We estimate z with the estimator $\hat{z} = (r-1)/S^{(r,z)}$. Consider the *r.v.* $y = \hat{z}/z$. The density and distribution functions of y are independent of z and are given by

$$\begin{aligned} f_r(y) &= -\frac{(r-1)^r}{(r-1)!y^{r+1}}e^{-\frac{r-1}{y}} \\ F_r(y) &= e^{-\frac{(r-1)}{y}}\left(1 + \sum_{i=1}^{r-1}\frac{(r-1)^i}{y^i i!}\right) \end{aligned}$$

for $y \geq 0$. A plot of the distribution functions of y for $r = \{5, 10, 15, 20, 25\}$ and of the confidence/accuracy tradeoffs for $r = \{5, 10, 20, 30, 50, 100, 200\}$ are provided in Figure 2. We obtain



Distribution functions



Confidence/Accuracy

Figure 2: Estimate quality vs. rounds

the following expressions for the relative error

$$\begin{aligned}
\text{Prob}\{\hat{z} \geq z(1+\epsilon)\} &= \text{Prob}\{y \geq (1+\epsilon)\} = 1 - e^{-\frac{r-1}{1+\epsilon}}\left(1 + \sum_{i=1}^{r-1} \frac{(r-1)^i}{(1+\epsilon)^i i!}\right) \\
\text{Prob}\{\hat{z} \leq z(1-\epsilon)\} &= \text{Prob}\{y \leq (1-\epsilon)\} = e^{-\frac{r-1}{1-\epsilon}}\left(1 + \sum_{i=1}^{r-1} \frac{(r-1)^i}{(1-\epsilon)^i i!}\right)
\end{aligned}$$

Hence,

$$\text{Prob}\left\{\frac{|\hat{z}-z|}{z} \geq \epsilon\right\} = 1 - e^{-\frac{r-1}{1+\epsilon}}\left(1 + \sum_{i=1}^{r-1} \frac{(r-1)^i}{(1+\epsilon)^i i!}\right) + e^{-\frac{r-1}{1-\epsilon}}\left(1 + \sum_{i=1}^{r-1} \frac{(r-1)^i}{(1-\epsilon)^i i!}\right) .$$

Further analysis (see the full version of [1]) establishes the asymptotic behavior

$$\text{Prob}\left\{\frac{|z-\hat{z}|}{z} \geq \epsilon\right\} = \exp(-\Omega(\epsilon^2 r)) .$$

(The probability that an estimate is ϵ fraction off the true value decreases exponentially with $r\epsilon^2$.) The bias of the estimator $\hat{z}$ is

$$E(y) - 1 = \int_0^\infty \frac{f_r(y)}{y} dy - 1 = 0 .$$

(The estimator is unbiased). The variance is

$$E((1-y)^2) = \int_0^\infty (1-y)^2 f_r(y) dy = \frac{1}{r-2} .$$

The expected relative error is

$$E(|1-y|) = \int_0^\infty |1-y| f_r(y) dy = \frac{2(r-1)^{(r-2)}}{(r-2)! e^{r-1}} \approx \sqrt{\frac{2}{\pi(r-2)}} .$$

(using Stirling's formula.) The accuracy and confidence levels are guaranteed for each column, but are correlated for different columns. The amount of correlation increases with the amount of overlap (locations of nonzeros). Note that for any fixed $\epsilon > 0$, a logarithmic number of rounds is sufficient to guarantee that with a very high probability (at least $(1 - \frac{1}{\text{poly}})$) all (polynomially many) estimates are ϵ-accurate.

3.4. Other estimation methods

We discuss simpler estimation algorithms that are based on the assumption that nonzero locations are randomly and independently spread in rows and columns. We apply elementary probability theory considerations. Later on, we compare the performance of these straightforward methods to the scheme suggested above. Consider the matrix product $\mathbf{AB}$, where $\mathbf{A}$ has dimensions $n_1 \times n$ and $\mathbf{B}$ has dimensions $n \times n_2$.

Coarse estimates in $O(1)$ time. Suppose that nonzero locations in the matrices $\mathbf{A}$ and $\mathbf{B}$ are spread randomly and are independent. (Essentially, the corresponding bipartite graphs are expanders.) Suppose $\mathbf{A}$ has a nonzeros and $\mathbf{B}$ contains b nonzeros. When $a \ll n_1 n$ and $b \ll n_2 n$ (the matrices are sparse) the expected size of the product $\mathbf{AB}$ is roughly ab/n. Since nonzero locations are spread randomly in $\mathbf{AB}$ as well, the expected sizes of each row and column of $\mathbf{AB}$ are $ab/(n_1 n)$ and $ab/(n_2 n)$, respectively.

More refined estimates in $O(n \log n)$ time. We make the following (weaker) assumption. Suppose that sizes of rows of $\mathbf{A}$ and columns of $\mathbf{B}$ are arbitrary but positions of the nonzero entries in each row of $\mathbf{A}$ are independent of the corresponding positions of the nonzero entries in the columns of $\mathbf{B}$. That is, if the ith row of $\mathbf{A}$ has r nonzeros and the jth column of $\mathbf{B}$ has r' nonzeros. The probability that the ijth entry in the product $\mathbf{AB}$ is nonzero is

$$p_{ij} \equiv \text{Prob}\{[A]_{i,j} \neq 0\} = p(n, r, r') = 1 - \frac{(n-r)!(n-r')!}{(n-r-r')!n!} \ .$$

Hence, the expected size of the jth column (resp., ith row) of the product $\mathbf{AB}$ is $\sum_i p_{ij}$ (resp., $\sum_j p_{ij}$). The expected size is used as an estimate of the true size. Computing these estimates for all columns of $\mathbf{AB}$ amounts to $O(n_1 n_2)$ evaluations of $p(n, r, r')$. for various values of r and r'. Moreover, evaluating $p(n, r, r')$ are expensive operations. Hence, these suggested estimates are slower to compute than our more accurate estimates presented above. We suggest the following relaxations. Firstly, the quantities $p(n, r, r')$ can be approximated as

$$p(n, r, r') \approx \min\{1, \frac{rr'}{n}\}$$

(at least in the ranges $rr' \ll n$ and $rr' \gg n$.) Secondly, the sizes $r_1 \leq \cdots \leq r_{n_1}$ of the rows of $\mathbf{A}$ can be sorted and grouped according to sizes. We maintain only the number of rows in each of the $\lceil \log_{1+\epsilon} n_1 \rceil$ size classes $[(1+\epsilon)^i, (1+\epsilon^{i+1})]$. Suppose there are s_i rows in the ith size class. We use the estimate

$$\sum_i s_i \min\{1, \frac{r(1+\epsilon)^i}{n}\} \ .$$

Note that this reduces the complexity of estimating all column sizes to $O(n_2 \epsilon^{-1} \log n)$.

Comparing estimates. Later on, we experimentally compare these estimates to the ones produced by the size estimation framework. The main advantage of these estimates is that they can be computed quickly. The main disadvantages are: (i) Worse accuracy. (Even when the independence assumption holds, the estimation error increases with the number of matrices in the product.) (ii) The independence assumption of nonzero locations is not justified in many real instances. (In contrast, the size estimation framework is suitable for products of *arbitrary* matrices.)

4. Near-optimal association for chain-products

We present algorithms that utilize column and row-size estimates to determine a near-optimal association (order of multiplications) for evaluating a *chain product* (product of 3 or more matrices). The cost of multiplying two matrices $\mathbf{A}$, $\mathbf{B}$ can be computed from the sizes $\mathbf{c}_{\mathbf{A}}$ of the columns of $\mathbf{A}$ and $\mathbf{r}_{\mathbf{B}}$ of the rows of $\mathbf{B}$ and is given in Equation 1. These sizes are likely to be readily available with the representation of the matrices. Consider computing a product $\mathbf{ABC}$ of three matrices. There are two possible orders to compute the product. The first multiplication is either $\mathbf{AB}$ or $\mathbf{BC}$. The second is either $(\mathbf{AB})\mathbf{C}$ or $\mathbf{A}(\mathbf{BC})$ accordingly. The costs of the two possible first multiplications are $\mathbf{c}_{\mathbf{A}}^T\mathbf{r}_{\mathbf{B}}$ and $\mathbf{c}_{\mathbf{B}}^T\mathbf{r}_{\mathbf{C}}$ (obtained using Equation 1). To compute the costs of the two possible second multiplications, we use *estimates* $\hat{\mathbf{c}}_{\mathbf{AB}}$ on the column sizes of the matrix $\mathbf{AB}$ and $\hat{\mathbf{r}}_{\mathbf{AB}}$ on the row sizes of the matrix $\mathbf{BC}$. Our estimates for the cost of the second multiplication, $\hat{\mathbf{c}}_{\mathbf{AB}}^T\mathbf{r}_{\mathbf{C}}$ and $\mathbf{c}_{\mathbf{A}}^T\hat{\mathbf{r}}_{\mathbf{BC}}$, are obtained by applying Equation 1 to the *estimated* sizes. The algorithm of Section 3 produces estimates for the column sizes of $(\mathbf{AB})$ using $r(|\mathbf{A}|+|\mathbf{B}|)$ operations and for the row sizes of $(\mathbf{BC})$ using $r(|\mathbf{B}|+|\mathbf{C}|)$ operations. An optimal order of multiplications (relative to the estimates) is obtained by comparing the two quantities

$$\mathbf{c}_{\mathbf{A}}^T\mathbf{r}_{\mathbf{B}} + \hat{\mathbf{c}}_{\mathbf{AB}}^T\mathbf{r}_{\mathbf{C}} \text{ and } \mathbf{c}_{\mathbf{B}}^T\mathbf{r}_{\mathbf{C}} + \mathbf{c}_{\mathbf{A}}^T\hat{\mathbf{r}}_{\mathbf{BC}} .$$

4.1. Algorithm for longer products

We present an algorithm that computes a near-optimal multiplication order for a chain product $\mathbf{A}_1\cdots\mathbf{A}_k$ of $k > 3$ matrices. For $1 \le i \le k$, $\mathbf{A}_i$ has dimensions $n_i \times n_{i+1}$. We apply a two-stage algorithm. The first stage consists of using several applications of the algorithm of Section 3 to compute estimated column and row sizes for all the subproducts. The second stage is a dynamic programming algorithm that utilizes these estimates and produces an optimal order of multiplication with respect to these estimates. The use of dynamic programming to optimally associate products of dense matrices (of varying dimensions) is standard (see [4]). Our use of dynamic programming is similar, but is presented for the sake of completeness. We elaborate on the two stages.

1. For $j > i > 1$ we produce estimates $\mathbf{c}_{i,j}$ for the column sizes of each subproduct $\mathbf{A}_i\cdots\mathbf{A}_j$. For $i < j < k$ we produce estimates $\mathbf{r}_{i,j}$ for the row sizes of each subproduct. (Note that the $\hat{\mathbf{c}}$ and $\hat{\mathbf{r}}$ notation was dropped.) These estimates are used in the dynamic programming stage and are obtained as follows:
   - For $i = 1, \ldots, k-2$:
     Apply the column-size estimation algorithm (Section 3)
     on the product $\mathbf{A}_i\cdots\mathbf{A}_{k-1}$.
     This provides us with estimates $\mathbf{c}_{i,i+1}, \ldots, \mathbf{c}_{i,k-1}$ of the column sizes of the
     subproducts $\mathbf{A}_i\cdots\mathbf{A}_j$ (for $i < j \le k-1$).

- For $i = 3, \ldots, k$:
  Apply the row-size estimation algorithm (Section 3)
  on the product $\mathbf{A}_2 \cdots \mathbf{A}_i$.
  This provides us with estimates $\mathbf{r}_{2,i}, \ldots, \mathbf{r}_{i-i,i}$ of the row sizes of the subproducts $\mathbf{A}_j \cdots \mathbf{A}_i$ (for $2 \leq j < i$).

For $1 \leq i \leq k$, let $\mathbf{c}_{i,i}$ (resp., $\mathbf{r}_{i,i}$) be the vector whose entries are the column (resp., row) sizes of the matrix $\mathbf{A}_i$.

2. We use dynamic programming to compute an optimal association with respect to the *estimated* row and column sizes of the subproducts. The dynamic programming computation is performed in phases, where the ℓth phase computes optimal multiplication order for all subproducts of size ℓ. In the last phase $\ell = k$ and the algorithms produces the optimal order for computing the whole product. For $1 \leq i < j \leq k$ denote by $\mathrm{cost}(i, j)$ the cost of computing the subproduct $\mathbf{A}_i \cdots \mathbf{A}_j$ using the least-cost ordering. The cost $\mathrm{cost}(i, j)$ is computed at the $(j-i+1)$st phase as the minimum of the $(j-i)$ quantities $\mathrm{cost}(i,h)+\mathrm{cost}(h+1,j)+\mathbf{c}_{i,h}^T\mathbf{r}_{h+1,j}$ $(i \leq h < j)$. Note that these intermediary costs are computed in previous phases. Finally, $\mathrm{cost}(1, k)$ is the cost of the least-cost order for computing $\mathbf{A}_1 \cdots \mathbf{A}_k$.

   - For $1 \leq i \leq k$, let $\mathrm{cost}(i, i) \leftarrow 0$.
   - Compute the cost of all subproducts of size 2:
     For $i = 1, \ldots, k-1$, let $\mathrm{cost}(i, i+1) \leftarrow \mathbf{c}_{i,i}^T \mathbf{r}_{i+1,i+1}$
     (apply Equation 1 to the column sizes of $\mathbf{A}_i$ and the row sizes of $\mathbf{A}_{i+1}$).
   - For $\ell = 3, \ldots, k$: (compute costs of all subproducts of size ℓ)
     For $i = 1, \ldots, k - \ell + 1$: (compute cost of $\mathbf{A}_i \cdots \mathbf{A}_{i+\ell-1}$)

$$\mathrm{cost}(i, i+\ell-1) = \min_{i \leq h < i+\ell-1} \left(\mathrm{cost}(i,h) + \mathrm{cost}(h+1, i+\ell-1) + \mathbf{c}_{i,h}^T \mathbf{r}_{h+1,i+\ell-1}\right) .$$

4.2. Time bounds

The time bounds for the first stage follow from Subsection 3.2. For $i = 1, \ldots, k-2$, the algorithm of subsection 3.1 is applied to estimate column sizes for all subproducts $\mathbf{A}_i \cdots \mathbf{A}_j$ $(i < j < k)$. This takes $O(r \sum_{p=i}^{k} |\mathbf{A}_p|)$ operations, where r is the number of rounds used in the estimation. Similarly, for $3 \leq i \leq k$, computing estimates on the row sizes of all subproducts $\mathbf{A}_j \cdots \mathbf{A}_i$ $(1 < j < i)$ takes $O(r \sum_{p=1}^{i} |\mathbf{A}_p|)$ operations. It follows that the number of operations (per round) performed in the first stage of the algorithm is:

$$\begin{array}{ll} \text{for } k = 3 & |\mathbf{A}_1| + 2|\mathbf{A}_2| + |\mathbf{A}_3| \\ \text{for } k \geq 4 & |\mathbf{A}_1| + |\mathbf{A}_k| + k(|\mathbf{A}_2| + |\mathbf{A}_{k-1}|) + (k+1)\sum_{i=3}^{k-2} |\mathbf{A}_i| \end{array}$$

The number of operation in the second stage (dynamic programming) is dominated by computing, for every i_1, i_2, i_3 such that $1 \leq i_1 \leq i_2 < i_3 \leq k$, an estimate on the cost of multiplying the two subproducts $\mathbf{A}' = \mathbf{A}_{i_1} \cdots \mathbf{A}_{i_2}$ and $\mathbf{A}'' = \mathbf{A}_{i_2+1} \cdots \mathbf{A}_{i_3}$. This estimate is obtained by a dot product of the estimate vectors (both of dimension n_{i_2+1}) on sizes of columns of $\mathbf{A}'$ and rows of $\mathbf{A}''$ (takes n_{i_2+1} operations). Hence, the second stage requires $\sum_{i=2}^{k} n_i(i-1)(k+1-i)$ operations. It follows that the asymptotic cost for r rounds of the estimation method is

$$O(kr \sum_{i=1}^{k} |\mathbf{A}_i| + k^3 N)$$

where $N = \max_i n_i$ is the maximum dimension of the matrices.

4.3. Accuracy

The algorithm computes an optimal multiplication order *with respect to the estimated sizes* of rows and columns. Accuracy and running time increase with the number of rounds used. Hence, the number of rounds should be set carefully such that time spent on selecting the order is small relatively to the time saved by identifying a better order.

The cost estimate of multiplying two products $\mathbf{A}'$,$\mathbf{A}''$ is the sum of products of estimates on column sizes of $\mathbf{A}'$ with respective row sizes of $\mathbf{A}''$. The estimates on sizes of columns of $\mathbf{A}'$ and on rows of $\mathbf{A}''$ are independent. The estimates on sizes of different columns of $\mathbf{A}'$ (or rows of $\mathbf{A}''$), however, are not independent and are skewed the same way for columns with similar patterns. The convergence of estimates is computed analytically for estimates on individual column or row sizes. The convergence for multiplication cost and near-optimal order estimates, however, depends on the structure of the matrices. Since estimates on column and row sizes are unbiased, it is easy to see that the estimates on multiplication cost are unbiased. We note that when $r = O(\epsilon^{-2} \log N)$ rounds, then with very high probability $((1 - 1/\operatorname{poly} N))$ all estimates used have a relative error smaller than ϵ (ϵ-accurate). Hence, estimates computed by the dynamic programming algorithm on the cost of multiplying two subproducts, are 2ϵ-accurate. (Therefore, the multiplication order produced by the algorithm is within $(1 + 2\epsilon)$ of optimal.) A more careful analysis shows that estimates on multiplication cost can *in the worst case* converge as estimates on sizes of single rows and columns (see Subsection 3.3). The worst case occurs when many rows (or columns) in the appropriate subproduct have similar nonzero patterns (and hence size estimates are biased the same way). Generally (not in dense matrices or pathological worst-cases), our experiments confirm that relative errors tend to "cancel out" and much fewer rounds are needed.

4.4. Dynamic programming with other estimates

Any estimation algorithm of sizes of columns and rows of subproducts can be substituted for the first stage. In particular, the approaches described in Subsection 3.4 which assume independence on nonzero locations. However, when there are no statistical guarantees on estimates on column and row sizes, the multiplication cost estimates deteriorate even further for longer products.

5. Experimental evaluation

We tested our algorithm on products of 2 and 3 randomly generated square matrices of various patterns and sizes and on products of Document-Term matrices that arise in Information Retrieval. We studied the performance of our algorithm for the following two objectives.

1. There are 2 evaluation orders for the matrix product $\mathbf{ABC}$: Either multiply $(\mathbf{AB})$ first and then $(\mathbf{AB})\mathbf{C}$, or compute $(\mathbf{BC})$ first and then $\mathbf{A}(\mathbf{BC})$. We estimate the costs of the two evaluation orders to determine the less costly one. We study the dependence of the accuracy of our estimates on the number of rounds. We explored the effectiveness of the estimation procedure by comparing the cost of estimation to the cost of multiplication and to the possible saving by selecting the less costly order.

2. We studied the distribution of the relative error of estimates on individual column and row sizes of products of two matrices $\mathbf{AB}$. The tradeoffs of accuracy and cost of obtaining the estimate are evaluated. We conclude that the estimates obtained experimentally using pseudo random numbers have the distribution predicted by the theory (Subsection 3.3). The cost of computing the product $\mathbf{AB}$ dominates the cost of estimation as data size increases.

5.1. Computing and comparing costs

We denote by $\text{cost}(\mathbf{A}*\mathbf{B})$ the cost according to Equation 1 (the number of floating-point multiply-add operations), of a sparse multiplication of the two matrices $\mathbf{A}$ and $\mathbf{B}$. Recall that Equation 1 is an "optimistic" measure in the sense that it does not account for possible additional required computation. We measured the cost of computing the two evaluation orders of $\mathbf{ABC}$ as follows. The costs $\text{cost}(\mathbf{A}*\mathbf{B})$ and $\text{cost}(\mathbf{B}*\mathbf{C})$ are computed directly using Equation 1 from the sizes of rows of $\mathbf{B}$ and $\mathbf{C}$ sizes of columns of $\mathbf{A}$ and $\mathbf{B}$ (provided with the representation of the matrices). We estimate $\text{cost}(\mathbf{AB}*\mathbf{C})$ (multiplying ($\mathbf{AB}$) by $\mathbf{C}$) and $\text{cost}(\mathbf{A}*\mathbf{BC})$) (multiplying ($\mathbf{BC}$) by $\mathbf{A}$) by estimating the column sizes of $\mathbf{AB}$ and row sizes of $\mathbf{BC}$ and applying Equation 1. For comparison purposes, we compute $\text{cost}(\mathbf{AB}*\mathbf{C})$ and $\text{cost}(\mathbf{A}*\mathbf{BC})$ exactly by performing a (time consuming) boolean matrix multiplication of the respective nonzero structures and determining exact column and row sizes. The costs of the two evaluation orders are $\text{cost}(\mathbf{A}*\mathbf{B})+\text{cost}(\mathbf{AB}*\mathbf{C})$ and $\text{cost}(\mathbf{B}*\mathbf{C})+\text{cost}(\mathbf{A}*\mathbf{BC})$. We used r rounds, for several values of r, to estimate $\text{cost}(\mathbf{AB}*\mathbf{C})$ and $\text{cost}(\mathbf{A}*\mathbf{BC})$. The number of operations per round for estimating $\text{cost}(\mathbf{AB}*\mathbf{C})$ (resp., $\text{cost}(\mathbf{A}*\mathbf{BC})$) is $|\mathbf{A}|+|\mathbf{B}|$ (resp., $|\mathbf{B}|+|\mathbf{C}|$). Each operation in the estimation procedure is a compare-store on numbers of size at most $O(\log N)$, where N is the largest dimension of the matrices. The additional computation (obtaining random samples and computing final estimates) is subsumed by the compare-store operations. Hence, the total cost per round is $c_r = |\mathbf{A}| + 2|\mathbf{B}| + |\mathbf{C}|$.

We measure the effectiveness of multiplication-costs estimation by comparing its cost rc_r to the cost $\min\{\text{cost}(\mathbf{A}*\mathbf{B})+\text{cost}(\mathbf{AB}*\mathbf{C}), \text{cost}(\mathbf{B}*\mathbf{C})+\text{cost}(\mathbf{A}*\mathbf{BC})\}$ of multiplication and to the gain $|\text{cost}(\mathbf{A}*\mathbf{B})+\text{cost}(\mathbf{AB}*\mathbf{C})-\text{cost}(\mathbf{B}*\mathbf{C})-\text{cost}(\mathbf{A}*\mathbf{BC})|$ by selecting the less costly order. We note that on some computing platforms, an "operation" of the multiplication algorithm is slower that an "operation" of the estimation algorithm. (Modern workstations, however, pipe floating-point multiply-adds and can perform one per clock-cycle.) The exact relation is platform-dependent. Our results support the use of the estimation scheme even if the two units are comparable.

5.2. Generating the random test matrices

We selected the matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ randomly according to some patterns. To generate the matrix $\mathbf{A}$ we used a small set of column densities (probability for an entry to be nonzero.). The matrix $\mathbf{A}$ is generated column by column. To generate a column we first select a probability p uniformly at random from the set of densities. Each entry in the column is determined to be zero or nonzero independently with probability p. The matrices $\mathbf{B}$ and $\mathbf{C}$ are generated using a "block like" pattern that also introduces correlations between the nonzero locations in $\mathbf{B}$ and $\mathbf{C}$. Each column of $\mathbf{C}$ and each row of $\mathbf{B}$ were generated independently according to the following procedure. A *block-type* that consists of a set of rows (resp., columns for $\mathbf{C}$) is selected uniformly at random from a set of block-types. Each block-type has two different probabilities $\{p_{in}, p_{out}\}$ associated with it. The probability p_{in} (resp., p_{out}) is the probability of an entry inside (resp., outside) the block is nonzero. Entries that are inside the block have higher probability of being nonzero ($p_{in} > p_{out}$).

In Table 1 we list the parameters used in each of 9 experiments: The dimension of the matrices, the number and size of blocks, the probabilities to be inside and outside the block, and the set of densities used for selecting the columns of matrix **A**. Figures 5 and 6 visualize the structures of all subproducts of **ABC** in experiments E_0 and E_1. Figure 5 helps visualize the "block" patterns we used to simulate correlations in matrices **B** and **C**. The patterns are not obvious in the larger matrices in Figure 6.

5.3. Results of experiments

We evaluated our method using 10 experiments on matrices of different sizes, density, and forms. In 9 experiments we used randomly generated matrices (E_0, E_1, F_1–F_7). We also experimented with matrices that arise in Information Retrieval, where the matrix **A** relates a set of documents D_1 and set of terms T_1, **B** is a relation between T_1 and documents D_2, and **C** relates D_2 and T_2. The product **ABC** relates D_1 and T_2. The sets D_1, D_2, T_1, and T_2 had size (dimension of matrices) roughly 5000.

In Tables 2 and 3 we provide the data from a single execution of each experiment. In Table 2 we list the number of nonzeros in each matrix and subproduct and the (easy to compute) multiplication costs $\mathrm{cost}(\mathbf{A} * \mathbf{B})$ and $\mathrm{cost}(\mathbf{B} * \mathbf{C})$. In Table 3 we list the cost c_r of a single round of the estimation algorithm, the exact costs of the two multiplications $\mathrm{cost}(\mathbf{A} * \mathbf{BC})$ and $\mathrm{cost}(\mathbf{AB} * \mathbf{C})$, and the estimates to these costs using $r = \{5, 10, 15, 20\}$ rounds of the estimation algorithm. For comparison, we applied the two simple estimation methods described in Subsection 3.4 (*coarse* and *refined*). The coarse estimates on column and row sizes, when applied to multiplication costs, yield the estimate $\frac{|\mathbf{A}||\mathbf{B}||\mathbf{C}|}{n_2 n_3}$ for both $\mathrm{cost}(\mathbf{AB} * \mathbf{C})$ and $\mathrm{cost}(\mathbf{A} * \mathbf{BC})$. (Where **A**, **B**, and **C** have dimensions $n_1 \times n_2$, $n_2 \times n_3$, and $n_3 \times n_4$, respectively.) The refined multiplication costs estimates utilize the refined row and column size estimates as described in Subsection 3.4. Another conceivable method to select a multiplication order is *greedy*, which selects the order by the cheaper first multiplication. It is easy to see that *greedy* would not perform well on our data.

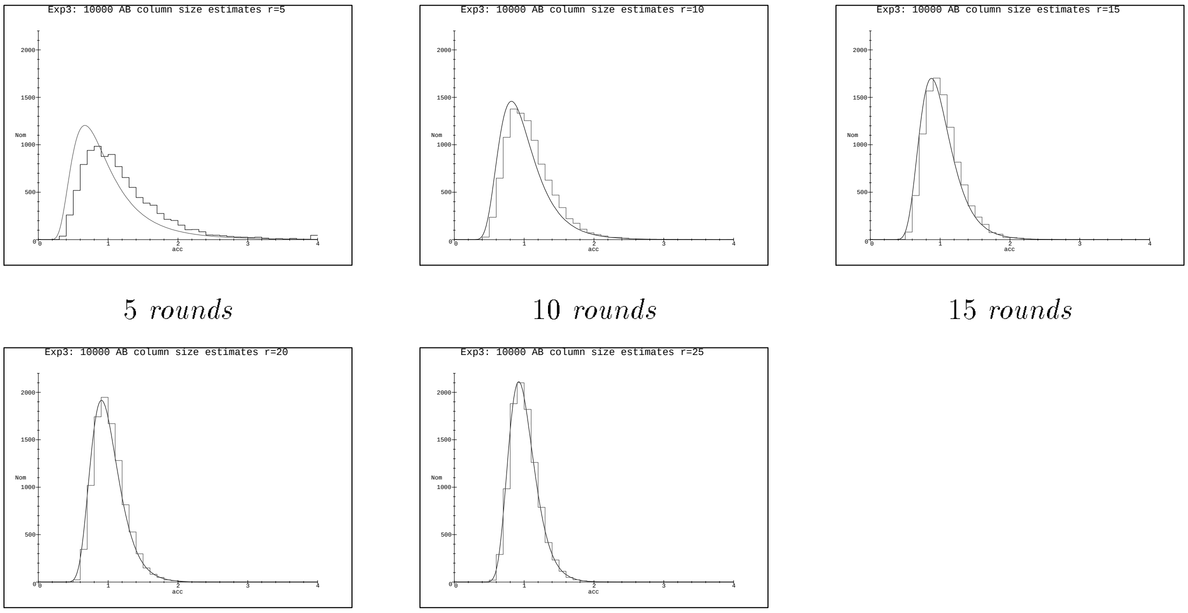
In experiments F_1–F_7, the number of operations required for a sufficiently good estimate of the multiplication costs is small compared to the gap

$$|(\mathrm{cost}(\mathbf{A} * \mathbf{B}) + \mathrm{cost}(\mathbf{AB} * \mathbf{C})) - (\mathrm{cost}(\mathbf{B} * \mathbf{C}) + \mathrm{cost}(\mathbf{A} * \mathbf{BC}))|$$

between the best and the lesser multiplication orders. The number of rounds needed for a sufficiently good estimate decreases for larger matrices. For experiments F_4–F_7, only 5 rounds suffice.

The coarse estimate (obtained in $O(1)$ time) provides an order-of-magnitude estimate to the multiplication cost, but have very limited accuracy. The refined estimate is reasonably accurate on the experiments where the independence assumptions hold. The estimates are far off for experiment *IR*. For the other experiments, recall that nonzeros in rows of matrix **A** are selected independently at random but the matrices **B** and **C** are correlated. Hence, the refined estimates are more accurate for predicting the structure of **AB** than **BC**.

We studied the accuracy of the multiplication-cost estimates as a function of the number of rounds in different experiments. Figures 7,8, 9, and 10 contain plots of the ratio of the estimated cost of multiplication and the actual cost. It plots the dependence of this ratio on the number of rounds performed for Experiments E_0, E_1, F_3–F_7. Each plot contains the accuracy curve for 10 repetitions of the experiment with $r = [2, \ldots, 25]$ rounds. For each experiment we included the curves for both products $(\mathbf{AB})\mathbf{C}$ (cost of multiplying the product **AB** by the matrix **C**) and



5 *rounds* 10 *rounds* 15 *rounds*

20 *rounds* 25 *rounds*

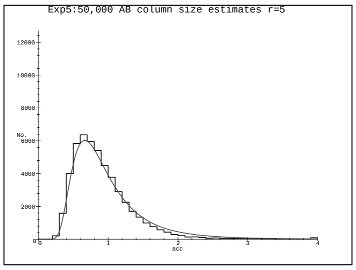
Figure 3: Exp.F_3:Histograms for accuracy of estimates on **AB** column sizes

A(**BC**) (cost of multiplying the matrix **A** by the product **BC**). As expected, the accuracy of the multiplication-cost estimates increases with input size and for a larger number of rounds. The cost of multiplication and the potential savings of selecting the less-costly order grow much faster than input size (and the cost of estimating multiplication-costs).

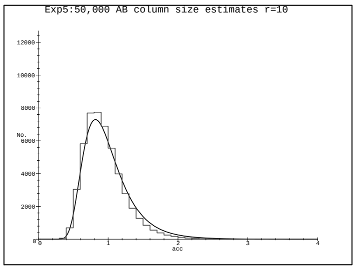
5.4. Estimates of sizes of rows and columns

Prior knowledge of (approximate) sizes of the columns or rows in a matrix product can be used to optimize memory and time usage by the multiplication procedure. For example, by allocating memory and setting up data structures of appropriate sizes. We included here histograms that plot the distribution of the estimate accuracies for various numbers of rounds. Figures 3 and 4 provide plots of the distribution functions and histograms for the accuracy (ratio of estimate to the value estimated) of estimates on the column sizes of the product **AB** in 10 repetitions of experiments F_3 and F_5. (Hence, the histograms contain $10,000$ and $50,000$ different estimates.). The histograms plot the accuracy distribution for $r = \{5, 10, 15, 20, 25\}$ rounds. Recall that the estimates are produced using $r(|A| + |B|)$ operations, (see Table 2). The close fit of the histograms and the respective distribution functions establish that the pseudo random number generator we used was sufficient. Observe that even for only 5 rounds, it is very unlikely that the estimate is over 3 times or below 0.4 times the actual value.

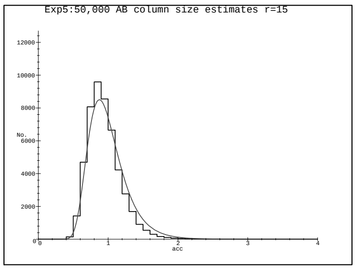
Table 4 lists the per-round operation counts of estimation, the operation count of multiplication, and the ratios of the number of estimation operations (comparisons) to multiplication operations (multiply-adds) for various numbers of rounds for the product **AB** in each of the experiments. The effectiveness of estimation increases with the size of the data. In all of our experiments, the cost of multiplication dominated the cost of estimation with 5 rounds.



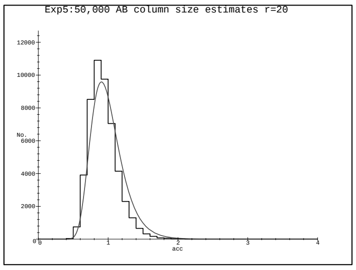
5 *rounds*



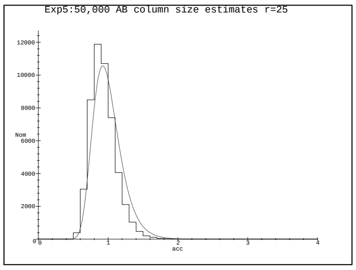
10 *rounds*



15 *rounds*



20 *rounds*



25 *rounds*

Figure 4: Exp.F_5:Histograms for accuracy of estimates on **AB** column sizes

5.5. Some implementation issues

5.5.1. Using thresholds

Our implementation was in *C* language, using the pseudo random number generator provided by the standard library. The estimation algorithm tracks the *minimum* elements in large sets of random keys. The estimate is essentially the inverse of the average of r such minima. Hence, particularly for very small values of r, the estimate is sensitive to skews from a random distribution or to very small values. Moreover, the variance of the estimator is not bounded for $r = 2$. Our experimental setting demonstrate that $r \geq 5$ produced adequate estimates.

To decrease the standard deviation of the estimates for very small number of rounds, we applied a *low-threshold cutoff* to the random keys. That is, values below the threshold were replaced by the threshold. The threshold utilize the upper bound on the quantities estimated and rules out large overestimates. The use of a threshold, however, introduces a bias (towards underestimation). We observed that the thresholds indeed improved the estimate quality for $r = \{2, 3\}$. We suggest using a threshold of the order of 1 over the maximum row or column-size expected in the product. In the absence of a good estimate for the latter, the dimension (an obvious upper bound) can be used. (This increases the accuracy, but introduces a bias, in the column/row size estimates).

We used a low threshold or no threshold for the multiplication cost estimates. Figures 8 and 9 contain plots where the threshold is 1 over twice the dimension, that is, 1/2000 for Experiment F_3, 1/10000 for Experiments F_4 and F_5, and 1/20000 for Experiments F_6, F_7 and no threshold for Experiment *IR*. To visualize the effects of a threshold, Figure 12 contains the accuracy/rounds tradeoffs for repetitions of experiment F_3 with no threshold and with a high threshold 1/600. Figure 11 contains experiment *IR* with threshold 1/2000. The behavior depicted there for Experiments F_3 and *IR* is representative. As expected, the accuracy of the multiplication cost estimates for small number of rounds increases, the variance of the estimator decreases, and a bias is introduced.

5.5.2. Representing the keys

It suffices to use a fixed small number of *significant bits* for representing the random keys (to be determined according to the desired accuracy $O(\log \epsilon^{-1})$. Note that for Exponential samples, the average size of the exponent part is fixed. If a simple representation is used (no exponent) we need $O(\log N + \log \epsilon^{-1})$ bits.

5.5.3. Determining the number of rounds

For some applications it is desirable to determine the number of rounds online. The general guidelines are to balance the cost of estimation by the potential savings of choosing a cheaper order of multiplications. (The potential savings are the difference in the costs of the two possible orders.) For example, if after 5 rounds our estimate for the cost of one order is 50% higher than the other order, we may stop the estimation procedure. On the other hand, if our estimates to the costs of the two orders are within 10% of each other and we expect 10% accuracy we may choose to use more rounds (if the cost of using more rounds is small with respect to the current gap between the two orders.)

6. Summary

Multiplications of sparse matrices arise often in practice, both as stand alone operations and as part of more complex computation, such as solving linear programs. We considered predicting the nonzero structure of a product of 2 or more sparse matrices as a preprocessing step prior to the multiplication. In particular, we considered the prediction of the size (number of nonzero elements) of each columns and row of the product. Prior knowledge of the nonzero structure may improve storage utilization and running time and could be used to determine the least-costly order to multiply 3 or more sparse matrices.

We outlined and tested naive approaches that predict the structure assuming that nonzero entries are spread randomly. This assumption is often not justified. We adapted a reachability-set size estimation method [1] to produce accurate estimates of row and column sizes of arbitrary matrix products. These enabled us to estimate the cost of multiplying two matrix products. Our estimation procedure runs in linear time in the number of nonzero entries, which (asymptotically) is much smaller than the cost of computing the matrix product.

We experimented with an implementation of the method on products of large sparse matrices. We used matrices arising in Information Retrieval and random matrices. The random matrices were generated using patterns which we believe reflects correlations arising in some real applications. We observed that the estimation-cost is small relatively to the multiplication-cost even for relatively small matrices. We conclude that the method is practical and can both aid in multiplying pairs of matrices and considerably reduce the cost of computing chain-products of matrices.

Acknowledgement. I would like to thank David Applegate, David Lewis, Fernando Pereira, and Mauricio Resende for discussions and pointers to bibliography and applications, and to David Lewis for the IR data. I am grateful to the anonymous referees for helpful comments and references.

References.

[1] E. Cohen. Estimating the size of the transitive closure in linear time. In *Proc. 35th IEEE Annual Symposium on Foundations of Computer Science*, pages 190–200. IEEE, 1994. full version submitted to JCSS.

[2] E. Cohen. Optimizing multiplications of sparse matrices. In W. H. Cunningham, S. T. McCormick, and M. Queyranne, editors, *Proc. of the 5th International Conference on Integer Programming and Combinatorial Optimization*, pages 219–233. Springer-Verlag, Lecture Notes in Computer Science Vol. 1084, 1996.

[3] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9:251–280, 1990.

[4] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to algorithms.* McGraw-Hill Book Co., New York, 1990.

[5] W. Feller. *An introduction to probability theory and its applications*, volume 2. John Wiley & Sons, New York, 1971.

[6] A. George, J. Gilbert, and J.W.H. Liu, editors. *Graph theory and sparse matrix computation*, volume 56 of *The IMA volumes in Mathematics and its Applications.* Springer-Verlag, 1993.

[7] A. George and J. W. H. Liu. *Computer Solution of Large Sparse Positive Definite Systems.* Prentice-Hall, 1981.

[8] J. Gilbert, C. Moler, and R. Schreiber. Sparse matrices in Matlab: Design and implementation. *SIAM J. Matrix Anal. Appl.*, 13:333–356, 1992.

[9] J. Gilbert and E. G. NG. Predicting structure in nonsymmetric sparse matrix factorizations. In A. George, J. Gilbert, and J.W.H. Liu, editors, *Graph theory and sparse matrix computation The IMA volumes in Mathematics and its Applications*, volume 56, pages 107–140. Springer-Verlag, 1993.

[10] J. R. Gilbert. Predicting structure in sparse matrix computations. *SIAM J. Matrix Anal. Appl.*, 15(1):62–79, 1994.

[11] G. Golub and C. Van Loan. *Matrix Computations.* The Johns Hopkins U. Press, Baltimore, MD, 1989.

[12] A. Jennings and J. J. McKeown. *Matrix computations.* John Wiley & Sons, New York, second edition, 1992.

[13] S. Pissanetzky. *Sparse matrix technology.* Academic Press, New York, 1984.

[14] R. Sedgewick. *Algorithms.* Addison-Wesley Publishing Co., Reading, MA, 1988.

[15] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 14(3):345–356, 1969.

[16] R. P. Tewarson. *Sparse matrices.* Academic Press, New York, 1973.

Exp.	dimension	# blocks	block-size	p_{in}	p_{out}	**A** densities
E_0	256	8	15	0.8	0.02	{0.03}
E_1	512	8	60	0.25	0.01	{0.02}
F_1	1,000	20	50	0.3	0.005	{0.0150.0350.0200.0100.020}
F_2	1,000	10	30	0.4	0.005	{0.0150.0030.0200.0100.020}
F_3	1,000	10	60	0.2	0.005	{0.01}
F_4	5,000	100	40	0.5	0.0015	{0.0040.0040.0050.0050.004}
F_5	5,000	50	60	0.4	0.001	{0.0040.0030.0050.0050.003}
F_6	10,000	100	60	0.4	0.001	{0.0020.0010.0030.0030.002}
F_7	10,000	500	20	0.7	0.001	{0.002}

Table 1: Parameter choices for the experiments

Exp.	$\|\mathbf{A}\|$	$\|\mathbf{B}\|$	$\|\mathbf{C}\|$	$\|\mathbf{AB}\|$	$\|\mathbf{BC}\|$	$\|\mathbf{ABC}\|$	$\text{cost}(\mathbf{A} * \mathbf{B})$	$\text{cost}(\mathbf{B} * \mathbf{C})$
E_0	2,042	4,317	4,242	23,780	32,489	63,975	34,325	111,465
E_1	5,245	9,906	10,047	83,937	98,419	255,678	101,590	202,547
F_1	19,377	19,697	19,700	316,806	198,822	971,975	381,391	388,186
F_2	13,923	16,753	16,916	185,205	218,316	944,800	233,781	612,263
F_3	9,948	16,572	16,800	147,979	209,317	866,913	164,809	369,591
F_4	110,421	137,176	137,524	2,830,071	1,946,403	20,254,461	3,028,574	4,276,973
F_5	100,294	144,729	144,908	2,669,364	1,773,636	18,678,073	2,900,704	6,117,018
F_6	219,762	339,549	339,583	7,099,568	6,589,297	75,692,609	7,458,357	15,343,737
F_7	199,865	240,035	240,155	4,685,728	3,929,670	53,909,513	4,799,504	5,760,134
IR	82,670	85,793	85,155	5,787,707	1,174,978	12,305,303	8,870,366	2,412,967

Table 2: Matrix sizes

Exp.	c_r $(\times 10^3)$	cost: $(\times 10^3)$	exact	5 rnds	10 rnds.	15 rnds.	20 rnds.	coarse	refined
E_0	14.9	$\text{cost}(\mathbf{AB} * \mathbf{C})$	582	382	465	520	489	571	780
		$\text{cost}(\mathbf{A} * \mathbf{BC})$	257	237	248	265	268	"	491
E_1	35.1	$\text{cost}(\mathbf{AB} * \mathbf{C})$	1,709	1,887	1,613	1,563	1,652	1,991	2,075
		$\text{cost}(\mathbf{A} * \mathbf{BC})$	1,009	822	929	870	930	"	1,949
F_1	78.5	$\text{cost}(\mathbf{AB} * \mathbf{C})$	6,256	7,937	6,934	6,321	6,317	7,519	7,522
		$\text{cost}(\mathbf{A} * \mathbf{BC})$	3,860	5,168	4,163	4,235	4,246	"	7,513
F_2	64.3	$\text{cost}(\mathbf{AB} * \mathbf{C})$	6,456	7,760	7,416	6,465	6,647	3,946	8,484
		$\text{cost}(\mathbf{A} * \mathbf{BC})$	3,058	3,956	3,307	3,345	3,331	"	3,955
F_3	59.9	$\text{cost}(\mathbf{AB} * \mathbf{C})$	3,274	3,963	3,489	3,288	3,310	2,789	3,677
		$\text{cost}(\mathbf{A} * \mathbf{BC})$	2,084	2,690	2,222	2,287	2,286	"	2,769
F_4	522.3	$\text{cost}(\mathbf{AB} * \mathbf{C})$	88,001	93,130	95,477	88,911	83,709	83,324	94,453
		$\text{cost}(\mathbf{A} * \mathbf{BC})$	42,964	41,033	40,830	42,984	42,927	"	83,301
F_5	534.7	$\text{cost}(\mathbf{AB} * \mathbf{C})$	112,208	115,820	121,008	111,493	105,116	84,136	122,699
		$\text{cost}(\mathbf{A} * \mathbf{BC})$	35,510	33,486	33,106	35,031	352,821	"	84,068
F_6	1,238.4	$\text{cost}(\mathbf{AB} * \mathbf{C})$	319,546	298,115	306,531	313,904	322,776	253,396	337,196
		$\text{cost}(\mathbf{A} * \mathbf{BC})$	144,820	141,311	143,162	143,812	146,296	"	253,271
F_7	920.1	$\text{cost}(\mathbf{AB} * \mathbf{C})$	112,476	106,704	108,683	112,046	112,839	115,213	115,125
		$\text{cost}(\mathbf{A} * \mathbf{BC})$	78,607	74,261	76,961	76,949	78,113	"	115,265
IR	339.4	$\text{cost}(\mathbf{AB} * \mathbf{C})$	127,798	92,766	110,700	104,433	116,506	33,203	39,581
		$\text{cost}(\mathbf{A} * \mathbf{BC})$	68,892	45,018	50,595	58,875	65,785	"	71,851

Table 3: Estimation cost and results

Exp.	mult. cost $\text{cost}(\mathbf{A} * \mathbf{B})$	cost per rnd. $\|\mathbf{A}\| + \|\mathbf{B}\|$	5 rnds	10 rnds.	15 rnds.	20 rnds.
E_0	34,325	6,359	0.93	1.85	2.78	3.71
E_1	101,590	15,151	0.75	1.49	2.24	2.98
F_1	381,391	39,074	0.51	1.02	1.54	2.05
F_2	233,781	30,676	0.66	1.31	1.97	2.62
F_3	164,809	26,520	0.80	1.61	2.41	3.22
F_4	3,028,574	247,597	0.41	0.82	1.23	1.64
F_5	2,900,704	245,023	0.42	0.84	1.27	1.69
F_6	7,458,357	559,311	0.37	0.75	1.12	1.50
F_7	4,799,504	439,900	0.46	0.92	1.37	1.83
IR	8,870,366	168,463	0.09	0.19	0.28	0.38

Table 4: Column-size estimations costs as fractions of multiplication costs of **AB**

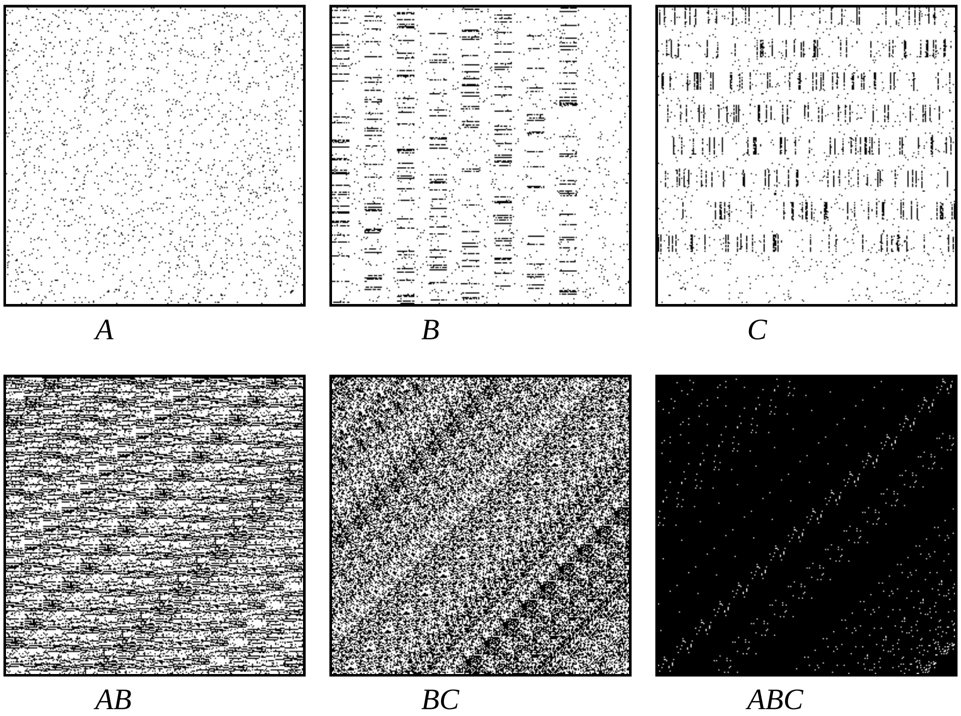


Figure 5: Visualization of matrices in experiment E_0

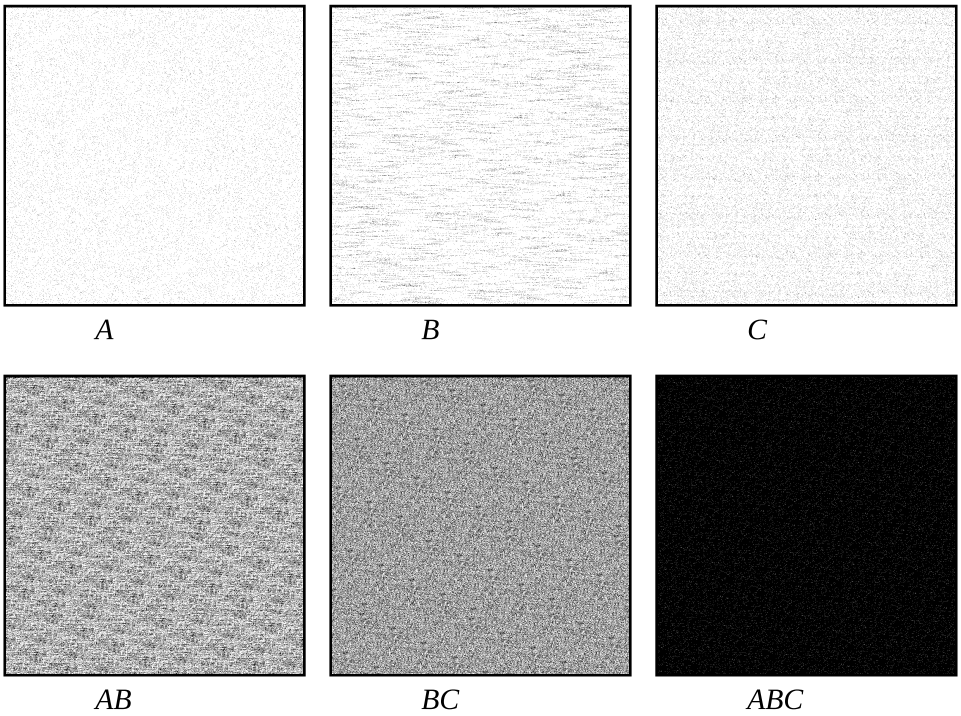


Figure 6: Visualization of matrices in experiment E_1

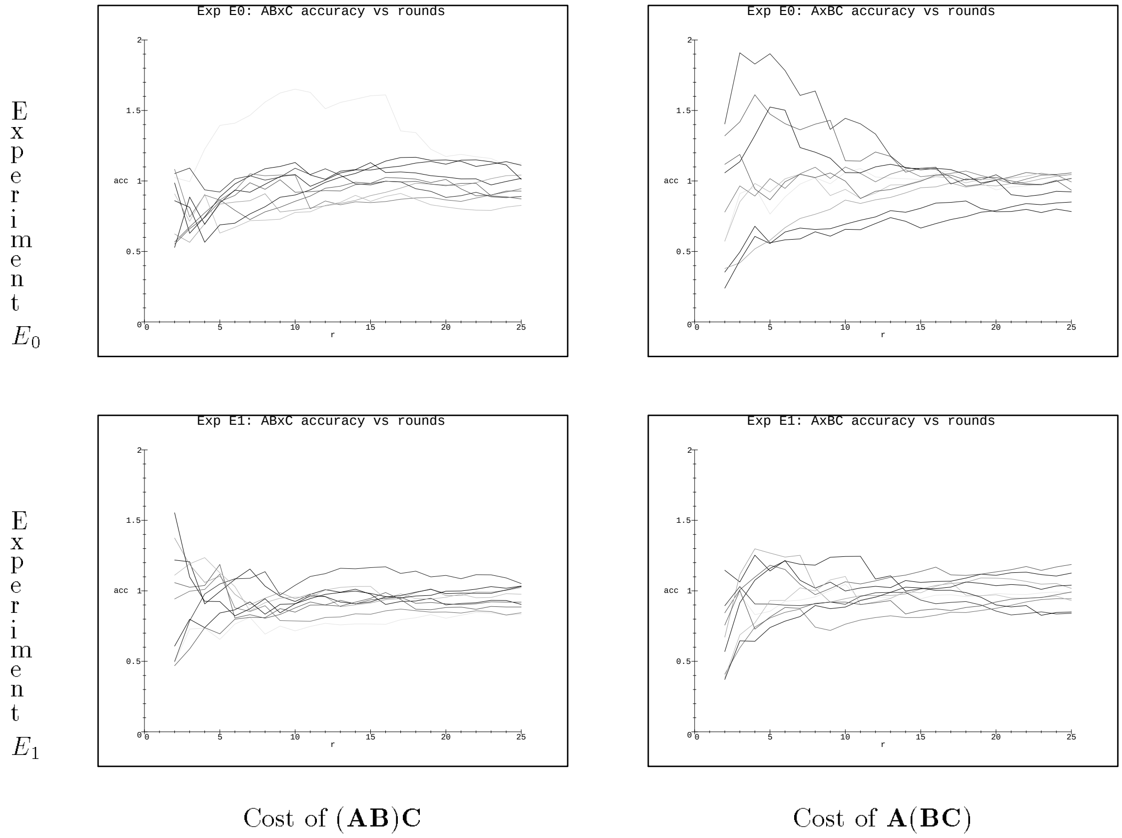


Figure 7: Accuracy vs. Rounds for 10 repetitions of Exp. E_0, E_1

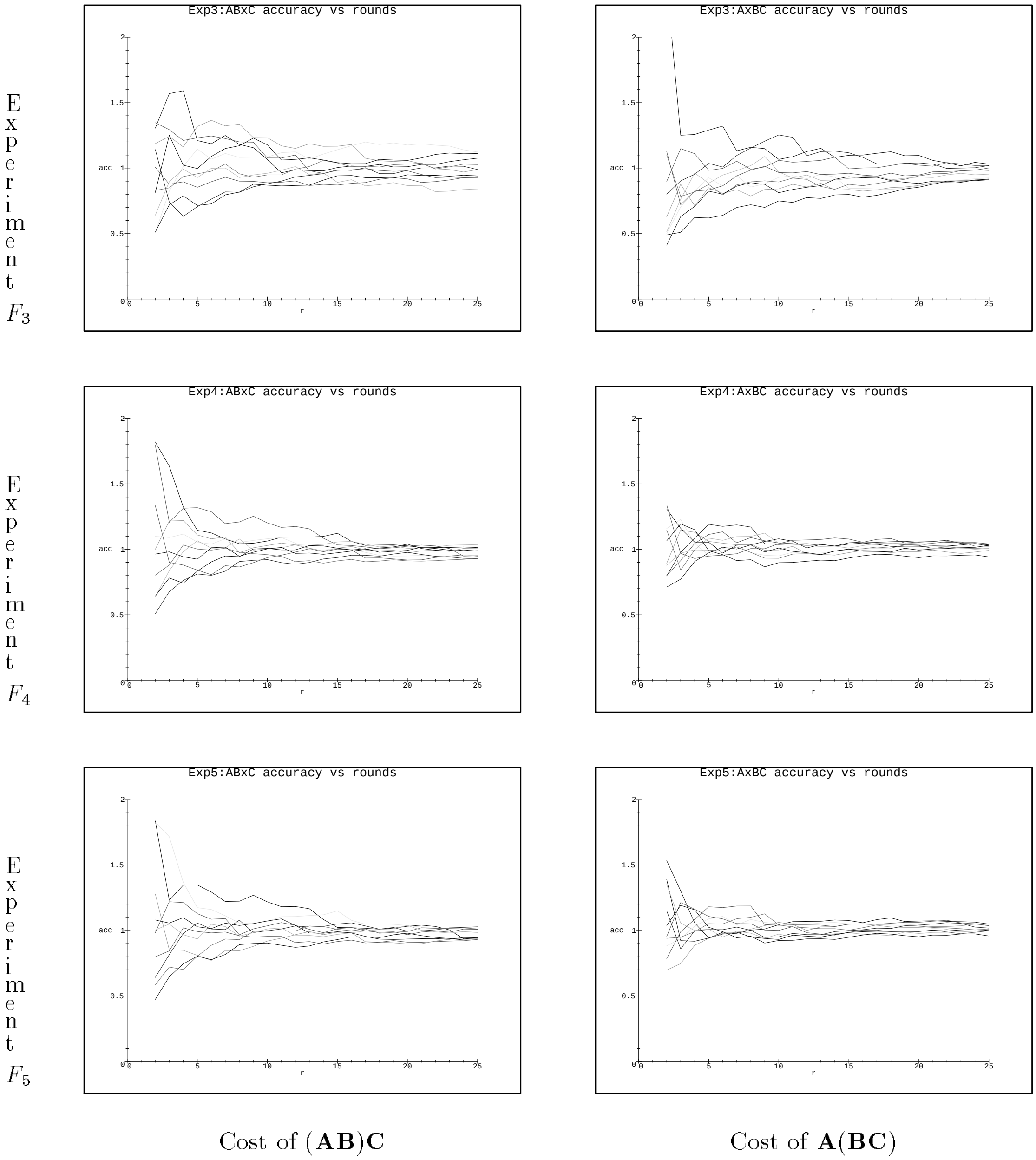
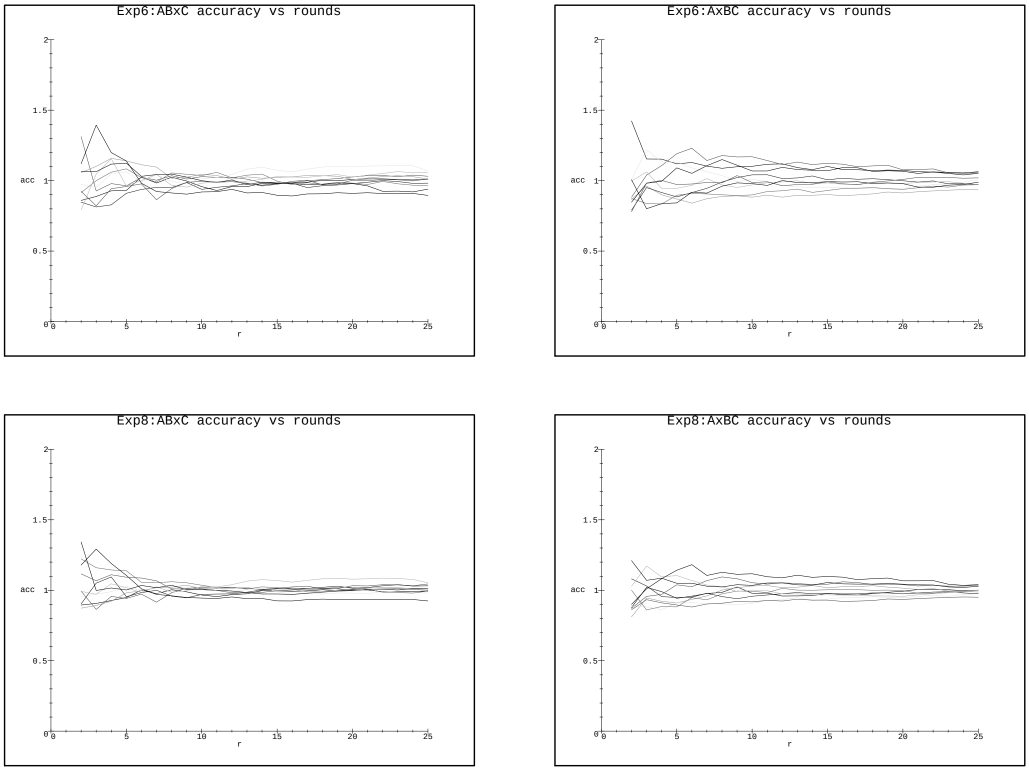


Figure 8: Accuracy vs. Rounds for 10 repetitions of Exp. F_3,F_4,F_5

Experiment F_6

Experiment F_7

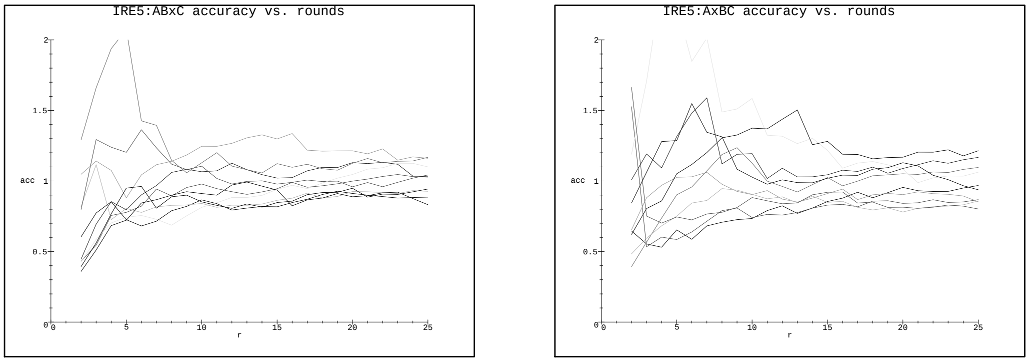


Cost of $(\mathbf{AB})\mathbf{C}$

Cost of $\mathbf{A}(\mathbf{BC})$

Figure 9: Accuracy vs. Rounds for 10 repetitions of Exp. F_6, F_7

Experiment IR

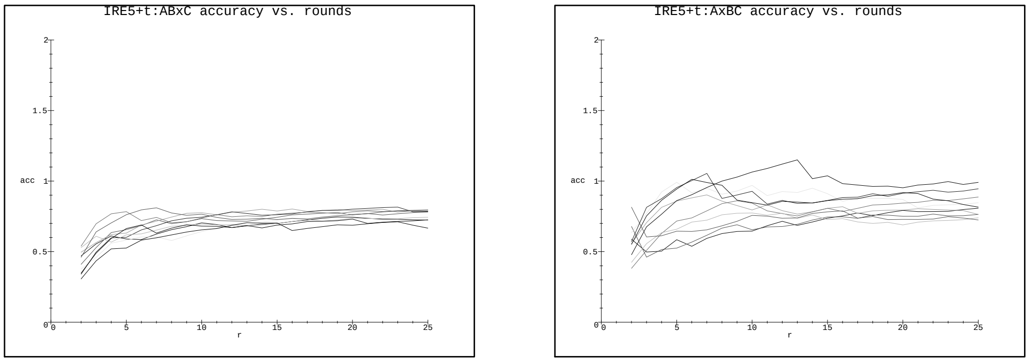


Cost of $(\mathbf{AB})\mathbf{C}$

Cost of $\mathbf{A}(\mathbf{BC})$

Figure 10: Accuracy vs. Rounds for 10 repetitions of Exp. IR

Experiment IR

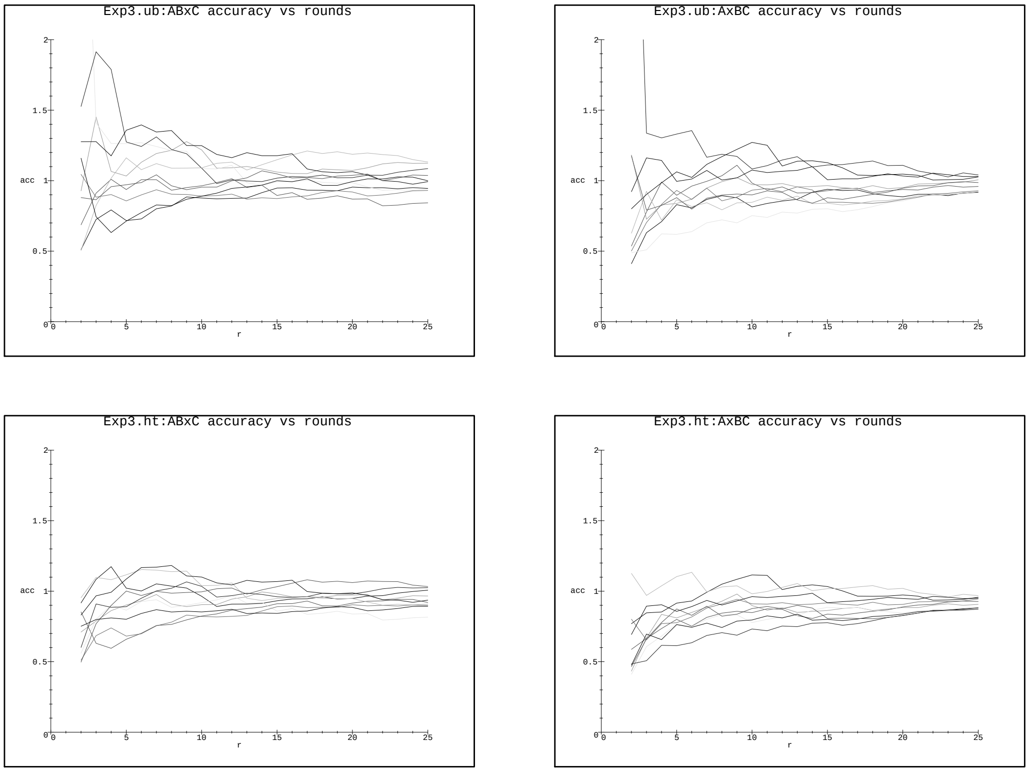


Cost of $(\mathbf{AB})\mathbf{C}$ Cost of $\mathbf{A}(\mathbf{BC})$

Figure 11: Accuracy vs. Rounds for 10 repetitions of Exp. *IR* with threshold 0.0005

Thresh 0.0000

Thresh 0.0017



Cost of $(\mathbf{AB})\mathbf{C}$ Cost of $\mathbf{A}(\mathbf{BC})$

Figure 12: Accuracy vs. Rounds for 10 repetitions of Exp. F_3 with thresholds $\{0, 1/600\}$