Cent. Eur. J. Comp. Sci. • 4(1) • 2014 • 1-11
DOI: 10.2478/s13537-014-0201-x

VERSITA

Central European Journal of **Computer Science**

A note on the multiplication of sparse matrices

Research Article

Keivan Borna[12*], Sohrab Aboozarkhani Fard[1]

1 *Faculty of Mathematics and Computer Science, Kharazmi University, Tehran, Iran*

2 *School of Mathematics, Institute for Research in Fundamental Sciences (IPM), P.O. 19395-5746, Tehran, Iran.*

Received 07 December 2012; accepted 28 December 2013

Abstract: We present a practical algorithm for multiplication of two sparse matrices. In fact if A and B are two matrices of size n with m_1 and m_2 non–zero elements respectively, then our algorithm performs $O(\min\{m_1 n, m_2 n, m_1 m_2\})$ multiplications and $O(k)$ additions where k is the number of non–zero elements in the tiny matrices that are obtained by the columns times rows matrix multiplication method. Note that in the useful case, $k \leq m_2 n$. However, in Proposition 3.3 and Proposition 3.4 we obtain tight upper bounds for the complexity of additions. We also study the complexity of multiplication in a practical case where non-zero elements of A (resp. B) are distributed independently with uniform distribution among columns (resp. rows) of them and show that the expected number of multiplications is $O(m_1 m_2/n)$. Finally a comparison of number of required multiplications in the naïve matrix multiplication, Strassen's method and our algorithm is given.

Keywords: algorithms • matrix multiplication • sparse matrices • tiny matrices

1. Introduction

Let A be an $m \times n$ matrix and B a second matrix of size $n \times p$. Then the product AB is an $m \times p$ matrix whose entries are $(AB)_{i,j} = \sum_{k=1}^{n} A_{ik} B_{kj}$. In the following we recall some alternate descriptions of matrix multiplication; see [1, Section 2.4] for more details:

- A times columns of B: The j-th column of AB is the product of A and the j-th column of B.
- Rows of A times B: The i-th row of AB is the product of the i-th row of A and B.
- Columns of A times rows of B: AB is obtained as the sum of columns of A times rows of B. That is, $AB = A_{*1}B_{1*} + \cdots + A_{*n}B_{n*}$, where A_{*i} and B_{j*} stand for the i-th column of A and the j-th row of B respectively. We also use this method for multiplication in our algorithms.

* *E-mail: borna@khu.ac.ir*

In the following we give a very short review of some algorithms for matrix multiplication:

- The naïve matrix multiplication algorithm performs $O(n^3)$ operations using n^3 multiplications and n^3-n^2 additions.
- Strassen [2] gave a divide-and-conquer algorithm which runs in $O(n^{2.81})$ time. For example, for two 2×2 matrices, the naïve method takes 8 multiplications and 4 additions, while using the Strassen's method they can be multiplied using only 7 multiplications and 18 additions.
- Horowitz et al. [3, Section 2.4.4] gave an algorithm that runs in $O(m_1n + m_2n)$ where the matrices are stored in sparse storage model.
- Coppersmith and Winograd [4] provided the fastest known matrix multiplication algorithm, with a complexity of $O(n^{2.38})$.
- Yuster and Zwick [5] gave a new algorithm that multiplies A and B using $O(\min\{(m_1m_2)^{0.347}n^{1.2} + n^{2+o(1)}, m_1n, m_2n, n^{2.376+o(1)}\})$ algebraic operations (multiplications, additions and subtractions). In fact they split each of the given matrices into two dense and sparse matrices. Recall that an $n\times n$ matrix is called sparse (resp. dense) if the number of non–zero elements of it is $O(n^{1.37})$ (resp. $O(n^{1.68})$). Then multiply the dense parts using the fast dense rectangular matrix multiplication algorithm of Coppersmith [6] and multiply the sparse parts using the naïve sparse matrix multiplication algorithm. Finally they output the sum of theses two parts.

Note that if the given matrices are sparse, one has to avoid multiplying zeros. Now multiplying two sparse matrices using the matrix multiplication algorithms of Coppersmith and Winograd [4] for example, then it does not provide any improvements over the non–sparse matrix multiplication, as their concern is to multiply two matrices in general. Furthermore note that the result of Yuster and Zwick [5] is of theoretical importance (at least by now).
In this paper we give a practical algorithm for multiplication of two sparse matrices using sparse storage models. More precisely, let A and B be two sparse matrices of size n with m_1 and m_2 non–zero elements respectively then the complexity of multiplication of our algorithm is $O(\min\{m_1n, m_2n, m_1m_2\})$ and the complexity of additions is $O(k)$. This improves the complexity $O(m_1n + m_2n)$ mentioned in [3, Section 2.4.4]. We also study the complexity of multiplication where non-zero elements of A and B are distributed independently with uniform distribution (among columns of A and rows of B) respectively. In fact we then show that the expected number of multiplications is $O(m_1m_2/n)$. Furthermore in Section 3.2 we obtain tight upper bounds for the complexity of additions, k, as it is presented in the following:

$$k \leq \begin{cases} m_1m_2, & m_1, m_2 \leq n \\ m_1n, & m_1 \leq n, m_2 > n \\ m_2n, & m_1 > n, m_2 \leq n \end{cases}$$

and if $m_1, m_2 > n$ we have

$$k \leq \begin{cases} m_2n, & m_2 \leq \alpha n, \\ \alpha n^2 + (m_1 - \alpha n).\min\{m_2 - \alpha n, n\}, & m_2 > \alpha n \end{cases}$$

where $\alpha = \lfloor m_1/n \rfloor$.
The organization of this paper is as follows. In Section 2 the main results of the paper are given and in Section 3 the complexity analysis of our algorithms are presented. Section 4 is devoted to some conclusions and future works. Finally in Section 5 we give a comparision for the number of required multiplications in the naïve matrix multiplication, Strassen's method and our algorithm.

2. Main results

Let A and B be two square matrices of size n and with m_1 and m_2 non–zero elements. Our results could be generalized easily to rectangular matrices, but for the sake of simplicity we just present them for square matrices. We first recall two storage models for sparse matrices; see [3, Section 2.3] for more details. Note that these two sparse storage models keep the row and column numbers sorted respectively.
Storage Model 1. In this model we store any sparse matrix of size n in a matrix by $m+1$ rows and 3 columns, where m is the number of non–zero elements of it.

(a) In the first row we store the triple row count, column count and the number of non–zero elements.

(b) In the next rows we store the triple row number, column number and value for each non–zero element.

Storage Model 2. This model is essentially the same as Storage Model 1 with a difference that we first store the column numbers and then the row numbers.
In the following our main algorithm for multiplication of two sparse matrices is given. Our approach is essentially based on the "Columns times Rows" matrix multiplication method which states that $a_{ij}b_{jk}$ is the entry at row i and column k of the j-th tiny matrix.

2.1. Main algorithm

If $\boldsymbol{m_2 \leq n}$, multiply $\boldsymbol{A}$ and $\boldsymbol{B}$ using Partial Algorithm **1**.
Else multiply $\boldsymbol{A}$ and $\boldsymbol{B}$ using Partial Algorithm **2**.

2.2. Partial algorithm 1

Input: A and B two sparse matrices of size n and with m_1 and m_2 non-zero elements which are stored in Storage Model 1 as $\bar{A}$ and $\bar{B}$ respectively.
Output: The product $C := AB$ (which is a square matrix of size n) is given in its ordinary model.
Process:
$//Z$: a matrix with $k+1$ rows and 3 columns,
$//i$: index for non-zero elements of A,
$//j$: index for non-zero elements of B,
$//u$: index for non-zero elements of tiny matrices,
$//t$: counter of non-zero elements of tiny matrices.

$i \leftarrow t \leftarrow u \leftarrow 1$
$(Z(0,0)\,,\, Z(0,1)) \leftarrow (n\,,\, n)$
while $i \leq m_1$ **do**
  $j \leftarrow 1$
  while $j \leq m_2$ **do**
    if $\bar{A}(i,1) = \bar{B}(j,0)$ **then**
      $(Z(t,0)\,,\, Z(t,1)\,,\, Z(t,2)\,,\, t) \leftarrow (\bar{A}(i,0)\,,\, \bar{B}(j,1)\,,\, \bar{A}(i,2)\bar{B}(j,2)\,,\, t+1)$
$k \leftarrow t-1$
$Z(0,2) \leftarrow k$
while $u \leq k$ **do**
  $C(Z(u,0)\,,\, Z(u,1)) \leftarrow C(Z(u,0)\,,\, Z(u,1)) + Z(u,2)$

2.3. Partial algorithm 2

Let B be stored in Storage Model 1 as $\bar{B}$, then the first column of $\bar{B}$ is sorted. Using a one dimensional array of size n, say W, for each i, let $W[i]$ stand for the first row number whose first column is i and zero if i does not appear in the first column of $\bar{B}$. For example if $\bar{B}$ is $\begin{pmatrix} 3 & 3 & 3 \\ 0 & 1 & 4 \\ 0 & 2 & 7 \\ 2 & 2 & 3 \end{pmatrix}$, then W is filled with the following values:

$$W[0] = 1,\ W[1] = 0 \text{ and } W[2] = 3.$$

2.3.1. Computation of W

We use the following algorithm to compute W:

Input: B a square matrix of size n with m_2 non–zero elements.
Output: W an array of length n initialized with zero.
Process:

1 $u \leftarrow B(1,0)$
2 $W_u \leftarrow 1$
3 **while** $2 \leq i \leq m_2$ **do**
4 **if** $u \neq B(i,0)$ **then**
5 $u \leftarrow B(i,0)$
6 $W_u \leftarrow i$

In the following we give an algorithm to compute AB:

Input: A and B two sparse matrices of size n and with m_1 and m_2 non-zero elements where A is stored in Storage Model 2 as $\bar{\bar{A}}$ and B is stored in Storage Model 1 as $\bar{B}$.
Output: The product $C := AB$ (which is a square matrix of size n) in its ordinary storage model.
Process:
//Z: a matrix with $k+1$ rows and 3 columns,
//i : index for non-zero elements of A,
//j : index for non-zero elements of B,
//u: index for non-zero elements of tiny matrices,
//t: counter of non-zero elements of tiny matrices.

1 $i \leftarrow t \leftarrow u \leftarrow 1$
2 $(Z(0,0)\,,\, Z(0,1)) \leftarrow (n\,,\, n)$
3 **while** $i \leq m_1$ **do**
4 $m \leftarrow W(\bar{\bar{A}}(i,0))$
5 **if** $m \neq 0$ **then**
6 **while** $m \leq j \leq m_2$
7 **if** $\bar{\bar{A}}(i,0) = \bar{B}(j,0)$ **then**
8 $(Z(t,0)\,,\, Z(t,1)\,,\, Z(t,2)\,,\, t) \leftarrow (\bar{\bar{A}}(i,1)\,,\, \bar{B}(j,1)\,,\, \bar{\bar{A}}(i,2)\bar{B}(j,2)\,,\, t+1)$
9 $k \leftarrow t-1$
10 $Z(0,2) \leftarrow k$
11 **while** $u \leq k$ **do**
12 $C(Z(u,0)\,,\, Z(u,1)) \leftarrow C(Z(u,0)\,,\, Z(u,1)) + Z(u,2)$

Remark 1.

The multiplication of two sparse matrices is not necessarily sparse. Thus the output of product $C := AB$ is a square matrix of size n and so we have to store it in its ordinary storage model.

Remark 2.

One can apply the Partial Algorithm 1 when $m_2 > n$. But the role of Partial Algorithm 2 is to reduce the dependence of the main algorithm from m_2. This will improve the functionality of our algorithm as n grows.

3. Complexity analysis

In the following for each $1 \leq t \leq n$ let $\bar{a}_t$ and $\bar{b}_t$ denote the number of non–zero elements of the t-th column of A and t-th row of B respectively.

3.1. Complexity analysis of multiplications

The number of multiplications in this algorithm is $\sum_{t=1}^{n} \bar{a}_t \bar{b}_t$; see [5] for more information. On the other hand, the loops will run in $O(m_1 m_2)$. Thus the total complexity of multiplications is $O(\min\{m_1 n, m_2 n, m_1 m_2\})$.

Corollary 3.1.

The particular case of matrix-vector product has cost $O(m_1)$. This is because $m_2 \leq n$ and for each i, $\bar{b}_i = 0$ or $\bar{b}_i = 1$. Thus the number of multiplications is $\sum_{i=1}^{n} \bar{a}_i \bar{b}_i \leq \sum_{i=1}^{n} \bar{a}_i = m_1 = O(n^{1.37})$. A similar proof shows that the vector-matrix product has cost $O(m_2) = O(n^{1.37})$. Recall that for a dense matrix A of size $n \times n$, its product Ax with an arbitrary input vector x has cost $O(n^2)$. Furthermore in [7] the authors presented an $O(nlogn)$ algorithm for computing the matrix-vector product of a Pascal matrix and a vector. Note that a fast solution for the matrix-vector product has many applications in solving a system of equations $Ax = b$ where its solution is given by $x = A^{-1}b$.

3.1.1. Complexity of multiplications in a practical case

Let A and B be two sparse matrices of size n with m_1 and m_2 non-zero elements respectively. Assume that non-zero elements of A (resp. B) are distributed independently with uniform distribution among columns (resp. rows) of them. For constructing A (resp. B) we run the following Bernoulli random test m_1 (resp. m_2) times.
For each $1 \leq i \leq n$ define two random variable $\bar{a}_i$, $\bar{b}_i$ that count the number of non-zero elements of the i th column of A and the i th row of B respectively. For $\bar{a}_i$, put the first non-zero element randomly with uniform distribution in one of columns in A. Then with probability $1/n$ this elements locates in the column i and with probability $(n-1)/n$ this element does not locate in the column i of A. A similar random test can be applied for $\bar{b}_i$. Thus the random distribution of $\bar{a}_i$, P_A, and of $\bar{b}_i$, P_B, are binomial with the following probability functions for which we have $0 \leq \bar{a}_i \leq \bar{m}_1 := \min\{m_1, n\}, 0 \leq \bar{b}_i \leq \bar{m}_2 := \min\{m_2, n\}$:

$$P_A(j) = P(\bar{a}_i = j) = \binom{m_1}{j} \left(\frac{1}{n}\right)^j \left(\frac{n-1}{n}\right)^{m_1 - j},$$
$$P_B(j) = P(\bar{b}_i = j) = \binom{m_2}{j} \left(\frac{1}{n}\right)^j \left(\frac{n-1}{n}\right)^{m_2 - j}.$$

Since $\bar{a}_i$, $\bar{b}_i$ are independent we have

$$\begin{aligned}
E[\text{The number of multiplications}] &= E[\sum_{i=1}^{n} \bar{a}_i \bar{b}_i] = \sum_{i=1}^{n} E[\bar{a}_i] E[\bar{b}_i] \\
&= \sum_{i=1}^{n} \sum_{j=0}^{\bar{m}_1} j.P_A(j). \sum_{j=0}^{\bar{m}_2} j.P_B(j) \\
&\leq \sum_{i=1}^{n} \sum_{j=0}^{m_1} j.P_A(j). \sum_{j=0}^{m_2} j.P_B(j) \\
&= \sum_{i=1}^{n} \sum_{j=0}^{m_1} j. \binom{m_1}{j} \left(\frac{1}{n}\right)^j \left(\frac{n-1}{n}\right)^{m_1 - j} . \sum_{j=0}^{m_2} j \binom{m_2}{j} \left(\frac{1}{n}\right)^j \left(\frac{n-1}{n}\right)^{m_2 - j} \\
&= \sum_{i=1}^{n} \sum_{j=0}^{m_1} j. \binom{m_1}{j} \frac{(n-1)^{m_1 - j}}{n^{m_1}} . \sum_{j=0}^{m_2} j \binom{m_2}{j} \frac{(n-1)^{m_2 - j}}{n^{m_2}} \\
&\leq \frac{m_1 m_2}{n} = O(n^{1.74}).
\end{aligned}$$

The second inequality is a simple simplification by Mathematica for example. We summarize this result in the following corollary.

Corollary 3.2.
The expected number of multiplications in the product of two sparse matrices A and B where non-zero elements of A (resp. B) are distributed independently with uniform distribution among columns (resp. rows) of them is $O(n^{1.74})$.

In Section 5 we compare the required number of multiplications in our algorithm with those of the naïve and Strassen's methods.
The complexity of additions in both Partial Algorithms 1 and 2 is $O(k)$. A precise analysis of this issue is done in the following.

3.2. Complexity analysis of additions

Note that as we mentioned k is the complexity of additions. We can give upper and lower bounds for k as in the following:

Proposition 3.3.
Let A and B be two matrices of size n and with m_1 and m_2 non–zero elements respectively. Then upper and lower bounds for k are:

$$k \leq \begin{cases} m_1 m_2, & m_1, m_2 \leq n \\ m_1 n, & m_1 \leq n, m_2 > n \\ m_2 n, & m_1 > n, m_2 \leq n \\ n^2 + (m_1 - n)(m_2 - n), & m_1 > n, m_2 > n \end{cases} \quad \text{and} \quad k \geq \begin{cases} -n^3 + (m_1 + m_2)n, & m_1 + m_2 > n^2 \\ 0, & m_1 + m_2 \leq n^2 \end{cases}$$

Proof. Let $A = (a_{ij})$ and $B = (b_{ij})$. Then multiplying A and B by columns times rows method we have:

$$AB = \begin{bmatrix} a_{11} \\ \vdots \\ a_{n1} \end{bmatrix} \begin{bmatrix} b_{11} & \cdots & b_{1n} \end{bmatrix} + \cdots + \begin{bmatrix} a_{1t} \\ \vdots \\ a_{nt} \end{bmatrix} \begin{bmatrix} b_{t1} & \cdots & b_{tn} \end{bmatrix} + \cdots + \begin{bmatrix} a_{1n} \\ \vdots \\ a_{nn} \end{bmatrix} \begin{bmatrix} b_{n1} & \cdots & b_{nn} \end{bmatrix}.$$

Thus

$$AB = \begin{bmatrix} a_{11}b_{11} & \cdots & a_{11}b_{1n} \\ \vdots & & \vdots \\ a_{n1}b_{11} & \cdots & a_{n1}b_{1n} \end{bmatrix} + \cdots + \begin{bmatrix} a_{1t}b_{t1} & \cdots & a_{1t}b_{tn} \\ \vdots & & \vdots \\ a_{nt}b_{t1} & \cdots & a_{nt}b_{tn} \end{bmatrix} + \cdots + \begin{bmatrix} a_{1n}b_{n1} & \cdots & a_{1n}b_{nn} \\ \vdots & & \vdots \\ a_{nn}b_{n1} & \cdots & a_{nn}b_{nn} \end{bmatrix}.$$

Thus we obtained a sum of n tiny matrices. Note that k is in fact the number of non–zero elements of these tiny matrices (rows of Z).

- Lower bounds:
  Let $m = n^2 - m_1$ and $p = n^2 - m_2$ be the number of zero elements of A and B respectively. Since each zero in A will produce n zero in one of tiny matrices, we have mn zero elements in all tiny matrices. For example, if $a_{23} = 0$, then the second row in the third tiny matrix is zero. Now if for each j, $1 \leq j \leq n$, a_{ij} and b_{jk} are not zero simultaneously, then
  $$k = n^3 - (m + p)n = (m_1 + m_2)n - n^3.$$
  Thus $k \geq (m_1 + m_2)n - n^3$ when $m_1 + m_2 > n^2$ and $k \geq 0$ if $m_1 + m_2 \leq n^2$.
- Upper bounds:
  The maximum for k, the maximum number of non–zero elements in tiny matrices, is obtained if whenever some column of A is non–zero (say t), then the t-th row of B is non–zero too. This is because then the t-th tiny matrix is non–zero. We have the following upper bounds for k in different cases:

(a) If $m_1, m_2 \leq n$, then $k \leq m_1 m_2$.

(b) If $m_1 \leq n$ and $m_2 > n$, then $k \leq n m_1$.

(c) If $m_1, m_2 > n$, then $k \leq n^2 + (m_1 - n)(m_2 - n)$.

In order to see this when $m_1, m_2 \leq n$, let $m_1 = n - i$ and $m_2 = n - j$ for some $i, j \geq 0$. Then as it was mentioned the maximum for k is obtained if non–zero elements of A (respectively B) are located in the t-th column of A (respectively row of B). Hence all tiny matrices except the t-th one become zero and in this tiny matrix, ni elements vanish (because of zeros in the t-th column of A) and $(n - i)j$ elements vanish (because of zeros in the t-th row of B). Hence we have $n^2 - ni - (n - i)j = (n - i)(n - j) = m_1 m_2$ non–zero elements in Z totally.

If $m_1 = n - i$ for some $0 \leq i \leq n$ and $m_2 > n$, then one can argue similarly to see that $k \leq n^2 - ni = n(n - i) = n m_1$.

Finally if $m_1, m_2 > n$, then let $m_1 = n + i$ and $m_2 = n + j$ for some $i, j > 0$. Assume that n non–zero elements of A and B are in A_{*t} and B_{t*} respectively (in the worth case). This will produce n^2 non–zero elements in the t-th tiny matrix. Then each of other i elements, will produce at most j non–zero elements. Hence $k \leq n^2 + ij = n^2 + (m_1 - n)(m_2 - n)$.

□

Remark 3.

Note that the upper bound for k whenever $m_1, m_2 > n$ is still large. For example let $m_1 = 3n$ and $m_2 = 2n$ (i.e., $i = 2n$ and $j = n$). Now assume that all elements of A_{*i}, A_{*j}, A_{*t} and B_{i*}, B_{j*} be non–zero. Thus by Proposition 3.3, $k \leq 3n^2$, whereas we have only $2n^2$ non–zero elements in the i-th and the j-th tiny matrices, i.e., $k = 2n^2$. It is obvious that if non–zero elements of A and B are located in different columns and rows, then $k \leq 2n^2$. In Proposition 3.4 we overcome this problem.

Proposition 3.4.

Let A and B be two matrices of size n and with $m_1 > n$ and $m_2 > n$ non–zero elements respectively. Let $\alpha = \lfloor m_1/n \rfloor$. Then the upper bound for k is:

$$k \leq \begin{cases} m_2 n, & m_2 \leq \alpha n, \\ \alpha n^2 + (m_1 - \alpha n).\min\{m_2 - \alpha n, n\}, & m_2 > \alpha n \end{cases}$$

In particular,

$$k \leq \begin{cases} m_2 n, & m_2 \leq \alpha n, \\ n.\min\{m_1, m_2\}, & m_2 > \alpha n. \end{cases}$$

That is, in the worth case, $k \leq n m_2$.

Proof. Let $m_1 = n + i$ and $m_2 = n + j$ for some $i, j > 0$. As it was mentioned in the proof of Proposition 3.3, in the worth case, one can assume that n non–zero elements of A and B are in A_{*t} and B_{t*} respectively (for some t). This will produce n^2 non–zero elements in the t-th tiny matrix. In addition,

- If $j \leq (\alpha - 1)n$, then in fact $(\alpha - \beta - 1)n \leq j \leq (\alpha - \beta)n$ for some $1 \leq \beta \leq \alpha - 1$. Thus we have $(\alpha - \beta - 1)n^2$ further non–zero elements. Finally $0 \leq j - (\alpha - \beta - 1)n \leq n$ and $i - (\alpha - \beta - 1)n \geq n$ which yields at most $[j - (\alpha - \beta - 1)n]n$ further non–zero elements. Hence for $j \leq (\alpha - 1)n$ we have totally

$$n^2 + (\alpha - \beta - 1)n^2 + [j - (\alpha - \beta - 1)n]n = n^2 + jn = m_2 n$$

non–zero elements.

- If $j > (\alpha-1)n$, then we have $(\alpha-1)n^2$ further non–zero elements. Finally $j-(\alpha-1)n > 0$ and $0 \leq i-(\alpha-1)n \leq n$ which gives at most $[i-(\alpha-1)n].\min\{j-(\alpha-1)n, n\}$ further non–zero elements. Hence for $j > (\alpha-1)n$ we have totally
$$n^2 + (\alpha-1)n^2 + [i-(\alpha-1)n].\min\{j-(\alpha-1)n, n\} = \alpha n^2 + (m_1 - \alpha n).\min\{m_2 - \alpha n, n\}$$
non–zero elements.

Thus,
$$k \leq \begin{cases} m_2 n, & m_2 \leq \alpha n, \\ \alpha n^2 + (m_1 - \alpha n).\min\{m_2 - \alpha n, n\}, & m_2 > \alpha n \end{cases}$$

Now let $j > (\alpha-1)n$. If $\min\{m_2 - \alpha n, n\} = n$, then $\alpha n^2 + (m_1 - \alpha n).\min\{m_2 - \alpha n, n\} = m_1 n$, and if $\min\{m_2 - \alpha n, n\} = m_2 - \alpha n$, then

$$\alpha n^2 + (m_1 - \alpha n).\min\{m_2 - \alpha n, n\} = \alpha n^2 + (m_1 - \alpha n).(m_2 - \alpha n) < \alpha n^2 + n.(m_2 - \alpha n) = m_2 n,$$

where the inequality is due to the fact that $(\alpha-1)n \leq m_1 - n < \alpha n$ and so $0 \leq m_1 - \alpha n < n$. Hence for $j > (\alpha-1)n$, we have $k \leq n.\min\{m_1, m_2\}$.

□

Example 4.

Let $m_1 = 3n$ and $m_2 = 6n$ (i.e., $i = 2n$ and $j = 5n$) and assume that, in the worth case, all elements of A_{*i_1}, A_{*i_2}, A_{*i_3} and $B_{i_1*}, \cdots, B_{i_6*}$ are non–zero. Then $k = 3n^2$ and it is obvious that if non–zero elements of A and B are located in different columns and rows, then $k \leq 3n^2$. This result is also admitted by Proposition 3.4 applied for $\alpha = 3$.

As a corollary of Proposition 3.3 and Proposition 3.4 we have the following result:

Corollary 3.5.

In the worst case, $k \leq m_2 n$. In fact this upper bound is reachable only in the following three cases:

(i) $m_1 > m_2$, $m_2 > \alpha n$ and $m_1, m_2 > n$.

(ii) $m_2 \leq \alpha n$ and $m_1, m_2 > n$.

(iii) $m_1 > n$ and $m_2 \leq n$.

Example 5.

Let $A = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$ and $B = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$. Then $n = 3, m_1 = 3$ and $m_2 = 6$. By Proposition 3.3, $k \leq 9$. One can note that in this situation in fact $k = 0$. But if one moves each of 1's to the first row, it will produce three non–zero in one of tiny matrices. As a result, if all elements of the first row of B become non–zero (eg., by commuting the first two rows of B), then $k = 9$. Now in order to apply Partial Algorithm 2, note that $\bar{\bar{A}} = \begin{pmatrix} 3 & 3 & 3 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 2 & 1 \end{pmatrix}$, $\bar{B} = \begin{pmatrix} 3 & 3 & 6 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 2 & 1 \\ 2 & 0 & 1 \\ 2 & 1 & 1 \\ 2 & 2 & 1 \end{pmatrix}$ and W on $\bar{B}$ is $W = \begin{pmatrix} 0 & 1 & 4 \end{pmatrix}$. Hence $Z = \begin{pmatrix} 3 & 3 & 0 \end{pmatrix}$ which confirms the fact that $AB = 0$ and no multiplications or additions are applied.

Example 6.
Let $A = \begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix}$ *and* $B = A$. *Then* $n = 4, m_1 = m_2 = 10$. *Then the precise number of multiplications and additions that our algorithm does is* 26 *and the reported numbers are* 40 *and* 36 *respectively. Furthermore,* $C := AB = \begin{pmatrix} 5 & -4 & 1 & 0 \\ -4 & 6 & -4 & 1 \\ 1 & -4 & 6 & -4 \\ 0 & 1 & -4 & 5 \end{pmatrix}$

3.3. Space complexity

Both storage models 1 and 2 use $3(m_1 + 1) + 3(m_2 + 1) = 3(m_1 + m_2 + 2) = O(\max\{m_1, m_2\}) = O(n^{1.37})$ space. Since the product of two sparse matrices is not necessarily sparse, we have to use a further $O(n^2)$ space for storing the output. Furthermore, we use $3k$ memory for computing the number of non-zero elements of tiny matrices, since $k < m_2 n$ we deduce that $k = O(n^{2.37})$. As a result the total space complexity is $O(n^{2.37})$.

4. Conclusions and future work

In this paper we improved the complexity of the algorithm for multiplication of two sparse matrices posed in [3]. In fact, when we use the sparse storage model for storing input matrices, the required time for multiplication, for example via the algorithm in [3, Section 2.4.4], exceeds the time presented by the naïve algorithm. In this paper we present an algorithm that stores the input matrices in the sparse storage model and the time for multiplication is less than the naïve and Strassen's algorithms. Furthermore, tight upper bounds for the complexity of additions is presented. Studying our algorithm for other alternatives to store sparse matrices (for example [8]) is the subject of future work.

5. Comparision of our algorithm with the naïve and Strassen's methods

The aim of this section is to compare the required number of multiplications in our algorithm, the naïve and Strassen's methods. In Tables 1 and 2 the columns N, S, O represent the required number of multiplications in the naïve, Strassen's and our algorithm, respectively. We have generated 100 pairs of sparse random matrices (each pair of different size) in the following two situations:

1. When each pair of the sparse matrices are uniformly distributed random matrices:

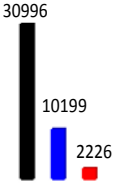

Figure 1. The average number of multiplications in case (1) in 100 tests

Using the functions sprand (or sprandn) of MATLAB, one can generate 100 pairs of sparse uniformly (or normally) distributed random matrices. We have written a Java applet for computing the average number of multiplications that each of the three algorithms (naïve, Strassen's and our algorithm) are doing. The average number of multiplications for the naïve matrix multiplications is 30996, for Strassen is 10199 and for our algorithm is 2226 as it is shown in Figure 1. This observation also shows that our algorithm is doing the least number of multiplications

Table 1. The number of multiplications for 20 pairs of sparse uniform random matrices

No.	Input			Output			No.	Input			Output		
	size	m_1	m_2	N	S	O		size	m_1	m_2	N	S	O
1	40	452	216	64000	19208	2446	11	25	178	148	15625	9261	1036
2	45	167	553	91125	19208	2024	12	7	17	3	343	189	10
3	44	592	225	85184	19208	3056	13	46	815	64	97336	19208	1149
4	5	4	12	125	56	10	14	11	26	29	1331	392	73
5	9	8	26	729	448	22	15	48	778	915	110592	64827	14930
6	32	454	308	32768	21952	4337	16	3	1	2	27	27	2
7	11	57	52	1331	392	267	17	8	22	3	512	448	8
8	39	490	457	59319	21952	5721	18	17	44	20	4913	3136	39
9	13	60	68	2197	1323	313	19	23	236	122	12167	2744	1211
10	40	259	125	64000	19208	792	20	4	4	5	64	64	5

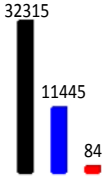

Figure 2. The average number of multiplications in case (2) for 100 tests

and has the best performance. Furthermore a benchmark about multiplication of 20 pairs of such matrices with different sizes is given in Table 1. This table can be read off as follows. For example when No. = 15, the required number of multiplications in three algorithms for two sparse matrices A and B of size 48 with uniform distribution (among columns of A and rows of B) with 778 and 915 non-zero elements are 110592, 64827 and 14930 respectively.

Table 2. The number of multiplications for 20 pairs of sparse random matrices

No.	Input			Output			No.	Input			Output		
	size	m_1	m_2	N	S	O		size	m_1	m_2	N	S	O
1	8	6	12	512	448	10	11	12	9	27	1728	1323	22
2	19	32	10	6859	3136	14	12	2	2	2	8	8	2
3	22	36	21	10648	2744	38	13	10	7	21	1000	392	9
4	42	103	98	74088	19208	227	14	20	34	52	8000	2744	99
5	11	3	18	1331	392	4	15	30	81	15	27000	9261	42
6	21	49	57	9261	2744	121	16	10	8	15	1000	392	5
7	34	38	60	39304	21952	67	17	25	9	15	15625	9261	6
8	6	2	3	216	189	0	18	41	7	72	68921	19208	11
9	31	106	1	29791	9261	5	19	12	10	3	1728	1323	1
10	50	158	35	125000	64827	104	20	30	23	104	27000	9261	64

2. When there is no limit on the distribution of sparse matrices:

   In this case we generate 100 pairs of sparse random matrices and compute the average number of multiplications that each algorithm is doing. Similar to case (1) the results are presented in Figure 2 and Table ??. For example when No. = 4, the required number of multiplications in three algorithms for two random sparse matrices of size

42 with 103 and 98 non-zero elements are 74088, 19208 and 227 respectively.

Acknowledgements

The paper benefited from the helpful comments and encouragements of Professor Gilbert Strang. The authors would like to thank him very much. The first author is also thankful to the National Elite Foundation of Iran for partial financial support.

References

[1] G. Strang, Introduction to Linear Algebra (Wellesley-Cambridge Press, Wellesley, USA, 2003)
[2] V. Strassen, Gaussian elimination is not optimal, Numer. Math. 13, 354–356, 1969
[3] A. Horrowithz, J. Sahny, Fundamentals of Data Structures (Computer Science Press, New York, 1983)
[4] D. Coppersmith, S. Winograd, Matrix multiplication via arithmetic progression, J. Symb. Comput. 9, 251–280, 1990
[5] R. Yuster, U. Zwick, Fast sparse matrix multiplication, ACM T. Alg. 1, 2–13, 2005
[6] D. Coppersmith, Rectangular matrix multiplication revisited, J. Complexity 13, 42–49, 1997
[7] Z. Tang, R. Duraiswami, N. Gummerov, Fast algorithms to compute matrix-vector products for pascal matrices, Technical Reports from UMIACS UMIACS-TR-2004-08, 2004
[8] A. Björck, Block bidiagonal decomposition and least square problems, Perspectives in numerical Analysis, Helsinki, May 27–29, 2008