

Empirical $O(n^2)$ Complexity is Convincingly Gettable with Two Dense Matrices in $n \times n$ Matrix Multiplication!

Suman Kumar Sourabh
University Department of Statistics
And Computer Applications
T. M. Bhagalpur University
Bhagalpur-812007
India
Email:sourabh.suman@rediffmail.com

*Soubhik Chakraborty**
Department of Statistics
Marwari College
T. M. Bhagalpur University
Bhagalpur-812007
India
*email ID of communicating
author: soubhikc@yahoo.co.in

Abstract: The present paper shows that Empirical $O(n^2)$ Complexity is convincingly gettable with two dense matrices in $n \times n$ matrix multiplication.

Key words: Amir Schoor's Algorithm, Sparse matrices, Dense matrices, Average Case Complexity

Introduction: We already know that for square matrices of order n , the average case complexity of Amir Schoor's algorithm is $O(d_1 d_2 n^3)$ where d_1 and d_2 are the densities (fraction of non-zero elements) of the pre factor and the post factor matrices respectively. For a formal proof, see ref [1]. If the product of these densities is kept at $1/n$, we automatically have an $O(n^2)$ complexity trivially. For example, we may keep the pre factor matrix a sparse matrix with density $1/n$ and the post factor matrix fully dense with density unity. **Our aim is to get similar complexity under more robust input conditions.** Therefore, in the present paper, we keep the pre factor matrix approximately as dense as triangular [see ref(2)] except that the zeroes are randomly allocated and the post factor matrix is fully dense. Using "smart statistics" we observe that fits to quadratic and cubic are equally good. Section I gives the code we used, section II gives the observations followed by the statistical analysis. Section III gives conclusion and suggestions for future work.

Amir Schoor's Algorithm: Let A , B , and C be pre-factor, post-factor and product matrices respectively. Amir Schoor's algorithm states that for every non-zero $a(i, k)$, multiply the k^{th} row of B by $a(i, k)$ and add it to the i^{th} row of C . See also ref.[1]
The pseudocode for the computational version only is as follows:-

```
for i = 1 to n
  for k = 1 to n
    while( a(i, k) <> 0)
      r = a(i, k)
      for j = 1 to n
        b(k, j) = b(k, j) * r
      endfor
      for j = 1 to n
        c(i, j) = c(i, j) + b(k, j)
      endfor
    endwhile
  endfor
endfor
```

Remark: We would build the product matrix with all zero entries before starting the algorithm. Also, we would be using the code for dense matrices only rather than sparse and hence Schoor's original data structure (the "row-column-value" structure) normally used for sparse matrices need not be adhered to. Recall that a triangular matrix is dense (see ref.[2]) with density $(n+1)/(2n)$. Since the borderline between sparse and dense matrices is not well defined, we would agree to call the pre-factor matrix dense in which the fraction of zeroes is approximately equal to that in a triangular matrix.

SECTION – I

C++ code prepared by Suman Kumar Sourabh depicting Amir Schoor's algorithm implemented with a pre-factor matrix approximately as dense as a triangular matrix (zeroes randomly allocated) and post factor matrix fully dense, executed in Pentium4 :-

```
#include<iostream.h>
#include<conio.h>
#include<time.h>
#include<stdlib.h>
#include<iomanip.h>

int main()
{
    clrscr();
    clock_t start, end;

    randomize(); //Initialize random number
    int n;
    cout<<"Enter n ";cin>>n; //Order of square matrices
    float d=(n+1)/(2*n); //Density of pre factor matrix (d1)

    int **a, **b, **c; //Dynamic memory creation
    a=new int *[n];
    b=new int *[n];
    c=new int *[n];

    for(int i=0;i<n;i++)
    {
        *(a+i)=new int [n];
        *(b+i)=new int [n];
        *(c+i)=new int [n];
        if (!(a+i) && !(b+i) && !(c+i)) //Check for memory
allocation
        {
            cout<<i<<" Insufficient memory...";
            exit (1);
        }
    }

    //pre-factor matrix generation
    //with random numbers
    for(int i=0;i<n;i++)
```

```

    {
    for(int j=0;j<n;j++)
        {
        int ran=rand();
        int maxr=RAND_MAX;
        float r=ran/float(maxr);
        if(r < d)
            *(*a+i)+j)=ran;
        else
            *(*a+i)+j)=0;
        }
    }

for(int i=0;i<n;i++)                //post-factor matrix generation
    {                                //with random numbers
    for(int j=0;j<n;j++)
        {
        int ran=rand();
        *(*b+i)+j)=ran;
        }
    }

for(int i=0;i<n;i++)                //Initialization of elements of
    {                                //resultant matrix with zero elements
    for(int j=0;j<n;j++)
        {
        *(*c+i)+j)=0;
        }
    }

                                        //Amir Schoor's algorithm begins
start = clock();                    //Start timer
for(int i=0;i<n;i++)
    {
    for(int k=0;k<n;k++)
        {
        if(*(*a+i)+k)!=0
            {
            for(int j=0;j<n;j++)
                {
                *(*c+i)+j)= *(*c+i)+j)+(*(*a+i)+k)**(*b+k)+j);
                }
            }
        }
    }
end = clock();                        //End timer

cout.precision(6);
cout.setf(ios::showpoint);
cout<<"Elapsed Time: "<<(end - start)/float(CLK_TCK)<<" Seconds"<<endl;
                                        //Count time in seconds

delete a;                             //Delete dynamic memory
delete b;
delete c;

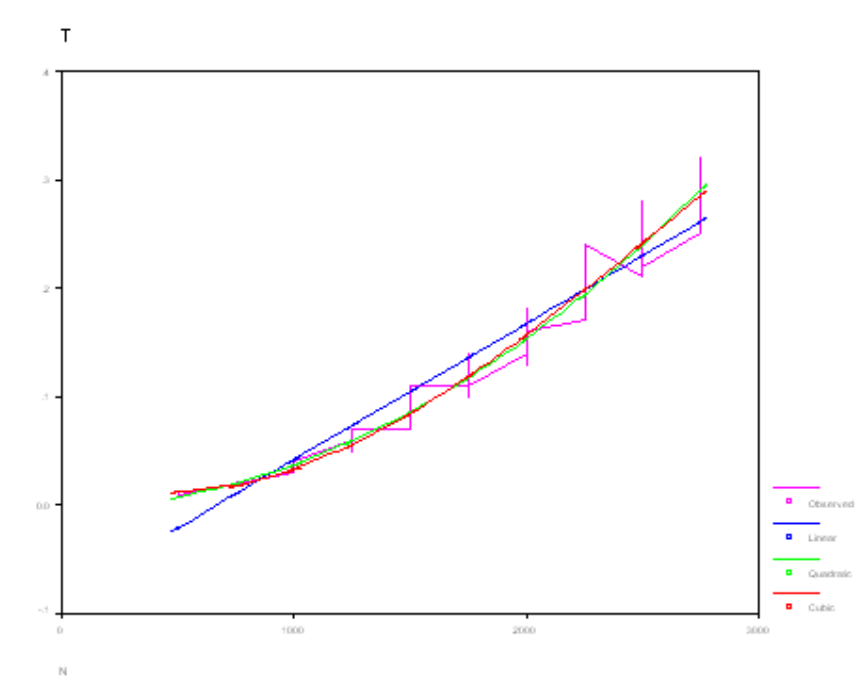
return 0;

```

SECTION -II**Table 1**Table of run times (in seconds) for $d_1 = (n+1)/(2n)$ $d_2 = 1$

n	500	750	1000	1250	1500	1750	2000	2250	2500	2750
1	.010	.020	.030	.060	.070	.110	.140	.170	.210	.250
2	.010	.020	.030	.050	.080	.111	.130	.180	.220	.260
3	.010	.020	.040	.060	.070	.100	.140	.190	.220	.291
4	.010	.020	.030	.060	.080	.100	.160	.180	.280	.321
5	.010	.020	.040	.050	.080	.120	.181	.231	.231	.310
6	.011	.021	.040	.050	.081	.120	.160	.210	.260	.320
7	.010	.020	.030	.070	.070	.110	.151	.211	.260	.291
8	.010	.020	.030	.050	.080	.140	.171	.220	.260	.260
9	.010	.020	.040	.060	.091	.121	.150	.220	.270	.260
10	.010	.020	.040	.071	.110	.110	.160	.240	.220	.260

Note: Columns give the results for 10 trials corresponding to each n



SECTION – III

Conclusions and suggestions for future work:

It is easy to see that an empirical $O(n^2)$ complexity is certainly gettable with both matrices dense, the pre-factor matrix approximately as dense as triangular and the post factor matrix fully dense. Future work includes studying whether the same can be had if the density of the pre-factor matrix is increased even further (this increases robustness of input conditions), the post factor matrix remaining fully dense. Perhaps the quadratic fit will have to be “Kusmaul-protected” (use of D-optimality design is proposed). This means that *we would be fitting a quadratic fit with a data set where observations are taken at those points which are optimal points for a cubic fit*. The quadratic fit is then said to be “Kusmaul-protected” against the error for not fitting a cubic fit. (see **K. Kusmaul, Protection against assuming the wrong degree in polynomial regression, Technometrics, vol.11, 1966**). The protection is suggested because we already know that when both matrices are fully dense, the complexity becomes $O(n^3)$ theoretically. It follows therefore that, with one density full and the other approaching fullness, *there will be a certain theoretically proven drift towards a cubic pattern*. As such, any adventure with a quadratic fit demands some kind of protection. Note further that to get an empirical $O(n^2)$ complexity for any two matrices convincingly, Coppersmith-Winograd’s algorithm should be used. But as this algorithm is not very practicable, we used Winograd’s algorithm instead for two arbitrary $n \times n$ matrices in ref [5] which shows how close we got!

To the question why we are able to work with a quadratic fit without sacrificing predictive power in the present case, two arguments come to our minds. First, Schoor has clearly claimed that his algorithm is **not only fast for sparse matrices but also fast for dense matrices because it is faster to work with rows only than both rows and columns (see p. 89 of ref [1])**. But this interesting claim demanded an empirical verification and hence our study. Second, when we say $T_{\text{avg}}(n) = O(f(n))$ we merely mean there exist two positive constants c and N such that $T_{\text{avg}}(n) < cf(n)$ whenever $n > N$. It is to be remembered that the constant c , which depends on implementation, can get very close to zero. If that happens, it is quite possible to work with an empirical complexity lower than the theoretical counterpart without sacrificing predictive power. The concept can be useful in some situations like chain matrix multiplication with a large number of matrices in the chain (see ref [2]), preventing ill-conditioning brought about by unnecessary complicated modeling in a *computer experiment* in general (ref [3]) and in tracking down empirically the complexity of an arbitrary algorithm for which determining a precise upper bound is a “deep intellectual challenge” (see ref [6] and [4]).

[CONCLUDED]

Acknowledgement: The second author wishes to thank **Prof. Donald Knuth** (Stanford University) for his note of criticism on the author’s paper related to this theme published in *Appl. Math. Lett.*, 12(7), 1999. That paper was on classical matrix multiplication which is amenable to Schoor’s modification if there are some zeroes to be exploited.

References:-

1. A. Schoor, *Information Processing Letters*, vol. 15, 1982, p. 87-89
2. Sartaj Sahni, *Data Structure and Algorithms in C++*, Tata McGrawHill, 2000
3. Jerome Sacks *et. al.* *Design and Analysis of Computer Experiments*, *Statistical Science*, vol 4, no. 4, 1989
4. Soubhik Chakraborty *et. al.* *On How Statistics Can Provide a Reliable and Valid Measure for an Algorithm’s Complexity*, *Journal InterStat*, Dec2004#2 (www.InterStat.StatJournals.net)
5. Soubhik Chakraborty and Kiran Kumar Sundararajan, *A Statistical Adventure Towards Getting an Empirical $O(n^2)$ Complexity in $n \times n$ Matrix Multiplication* *InterStat* March2006#5(website as in ref[4])
6. Aho, Hopcroft, Ullman, *Data Structures and Algorithms*, Pearson Edu. Reprint, 2000, chapter one

