A Statistical Adventure Towards Getting an Empirical $O(n^2)$ Complexity in *nxn* matrix multiplication

Soubhik Chakraborty
Lecturer in Statistics
Marwari College
T.M. Bhagalpur University
Bhagalpur-812007
India
Email:soubhikc@yahoo.co.in

Kiran Kumar Sundararajan
Statistician
Infotech Enterprises Limited
Infocity, Hyderabad-500033
India
Email:kiranks@infotechsw.com

Abstract: Given that the statistical approach "weighs" rather than counts the computing operations which arguably makes it more realistic(see ref[1] and ref[2]), we revisit Winograd's algorithm statistically with the objective of getting an empirical $O(n^2)$ complexity in two nxn matrix multiplication(n even). Next we briefly analyze our findings.

Key Words: Winograd's algorithm, empirical $O(n^2)$ complexity

Section I gives the introduction, section II the code on Winograd's algorithm and some interesting run time results and regression plots in MINITAB which clearly shows that our adventure is not at all senseless! Finally section III gives the conclusions and future work.

Section I

Introduction: Given that the statistical approach "weighs" rather than counts the computing operations which arguably makes it more realistic(see ref[1] and ref[2]), we revisit Winograd's algorithm statistically with the objective of getting an empirical $O(n^2)$ complexity in two nxn matrix multiplication(n even).*

Winograd's algorithm is simply obtained by re-arranging the terms of classical matrix multiplication and **pre-processing** all calculations that involve terms within a particular row or within a particular column that cleverly reduces the number of multiplications but increases those of additions. The strategy is acceptable since addition is faster than multiplication. For mathematical details on Winograd's algorithm, see ref[3] on which our code is based.

*Details of theoretical attempts towards getting an $O(n^2)$ complexity in nxn matrix multiplication can be found in Don Knuth's "bible" on algorithms vol. 2[The Art of Computer Programming(Semi-Numerical Algorithms), Pearson Education reprint, 2000)]

Section II

C++ Program for Matrix Multiplication using Winograd's Algorithm

```
#include<iostream.h>
#include<new.h>
#include<stdlib.h>
#include <Time.h>
#include<math.h>

void main()
{
      int **a, **b, **c, *ra, *cb;
      int n,i,j,k,l,k1,k2,d;

      clock_t start, finish;
      double duration;

      cout<<"\n   Enter size of matrix: ";
      cin>>n;

//allocating memory for the array a

      a = new int*[n];
      for(i=1;i<=n;i++)
            a[i] = new int[n];

      for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
            a[i][j]=rand();

//allocating memory for the array b

      b = new int*[n];
      for(i=1;i<=n;i++)
            b[i] = new int[n];

      for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
                  b[i][j]=rand();

//allocating memory for A and B

      ra = new int[n];
      cb = new int[n];
      c = new int*[n];

      for(i=1;i<=n;i++)
            c[i] = new int[n];

// the multiplication starts
```

```
      start=clock();

      if (fmod(n,2) == 0)
            d=n/2;
      else
            d=(n+1)/2;

      for (l=1;l<=n;l++)
      {
            ra[l] = 0;
            cb[l] = 0;
            for (k=1; k<=d; k++)
            {
                  k1= 2*k-1;
                  k2= 2*k;
                  ra[l] = ra[l] + a[l][k1]*a[l][k2];

                  cb[l] = cb[l] + b[k1][l]*b[k2][l];
            }
      }

      for (i=1;i<=n;i++)
      {
        for (j=1; j<=n;j++)
        {
            c[i][j] = 0;
            for (k=1;k<=d;k++)
            {
             k1= 2*k-1;
             k2= 2*k;
             c[i][j] = c[i][j]+(a[i][k1]+b[k2][j])*(a[i][k2]+b[k1][j]);
            }
            c[i][j] = c[i][j] - ra[i] - cb[j];
        }
      }

      finish = clock();

// multiplication ends

      duration = (double) (finish-start) / CLOCKS_PER_SEC;

      cout<<"\n\n Elapsed time : "<<duration<<" seconds \n";


}
```

System Specifications:

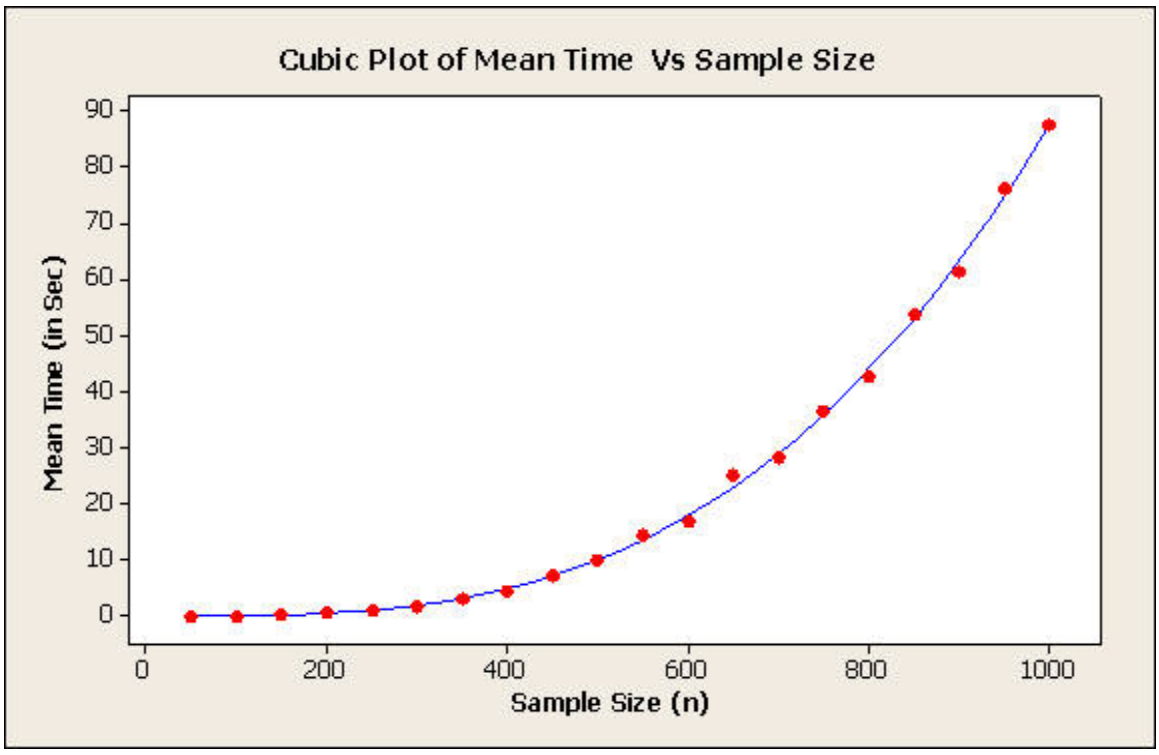
128 MB RAM, Pentium III processor with 550 MHz, 12 GB HDD

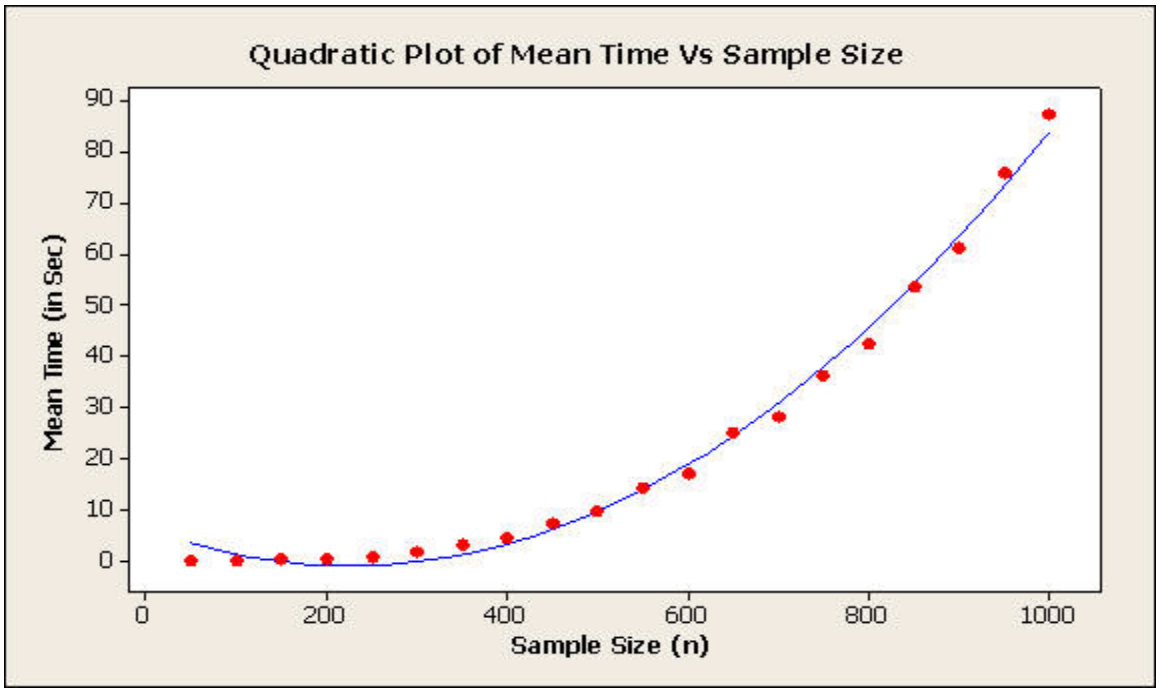
Run time results:

n	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Mean
50	0.01	0.009	0.01	0.01	0.009	0.0096
100	0.06	0.05	0.05	0.06	0.05	0.054
150	0.19	0.18	0.18	0.19	0.18	0.184
200	0.431	0.451	0.421	0.431	0.441	0.435
250	0.831	0.851	0.821	0.811	0.851	0.833
300	1.662	1.683	1.652	1.662	1.671	1.666
350	3.004	2.984	3.014	2.994	2.984	2.996
400	4.517	4.517	4.537	4.527	4.517	4.523
450	7.27	7.29	7.221	7.25	7.29	7.2642
500	9.835	9.824	9.845	9.835	9.845	9.8368
550	14.391	14.391	14.361	14.391	14.381	14.383
600	16.896	16.865	16.875	16.855	16.875	16.8732
650	25.067	25.067	25.046	25.086	25.057	25.0646
700	28.143	28.162	28.102	28.112	28.132	28.1302
750	36.224	36.234	36.214	36.234	36.234	36.228
800	42.514	42.553	42.503	42.524	42.523	42.5234
850	53.69	53.651	53.681	53.691	53.661	53.6748
900	61.03	61.071	61.051	61.061	61.071	61.0568
950	75.903	76.013	76.004	75.904	76.013	75.9674
1000	87.22	87.201	87.22	87.201	87.212	87.2108

In the MINITAB plots the order of the square matrices n is referred to as “sample size”.

Minitab Plot:





For QUADRATIC FIT

Polynomial Regression Analysis: mean versus n

The regression equation is
mean = 6.198 - 0.06382 n + 0.000141 n**2

S = 2.11494 R-Sq = 99.5% R-Sq(adj) = 99.4%

Analysis of Variance

Source	DF	SS	MS	F	P
Regression	2	14130.1	7065.06	1579.50	0.000
Error	17	76.0	4.47		
Total	19	14206.2			

Sequential Analysis of Variance

Source	DF	SS	F	P
Linear	1	11934.1	94.54	0.000
Quadratic	1	2196.1	490.96	0.000

For CUBIC FIT

Polynomial Regression Analysis: mean versus n

The regression equation is
mean = - 0.017 - 0.000390 n - 0.000006 n**2 + 0.000000 n**3

S = 0.988899 R-Sq = 99.9% R-Sq(adj) = 99.9%

Analysis of Variance

Source	DF	SS	MS	F	P
Regression	3	14190.5	4730.17	4836.96	0.000
Error	16	15.6	0.98		
Total	19	14206.2			

Sequential Analysis of Variance

Source	DF	SS	F	P
Linear	1	11934.1	94.54	0.000
Quadratic	1	2196.1	490.96	0.000
Cubic	1	60.4	61.76	0.000

Section III

Conclusions and future work:

Evidently the best fit is expectedly cubic but surprisingly the quadratic approximation is not a bad one! If the observations were taken at optimally selected points of n(use of D-optimality is proposed), it might be of interest to compare the predictive powers of the two fits which we reserve as a rewarding future work. In chain matrix multiplication, it might make sense to work with a quadratic fit and *design our computer experiment in an optimal sense(*which means efficient choice of inputs; see Jerome Sacks et. al., Design and Analysis of Computer Experiments, Statistical Science, 4(4), 1989) so that we can allow more matrices in the chain without inviting the serious problem of ill-conditioning (see G. A. F. Seber, Linear Regression Analysis, John Wiley). For polynomial fits, ill conditioning is severe when the degree of the polynomial is at about five or higher. Therefore it is certainly not wise to fit a polynomial of a very high degree unless there is a corresponding enhancement in the predictive power. My point is proved. The link between algorithmic complexity and computer experiments has been elaborated in ref[4].

References:

[1] Soubhik Chakraborty and Pabitra Pal Choudhury, A Statistical Analysis of an Algorithm's Complexity, Applied Mathematics Letters, 13(5), 2000

[2] Soubhik Chakraborty *et. al.* , On How Statistics Provides a Reliable and Valid Measure for an Algorithm's Complexity,
InterStat Dec2004(www.interstat.statjournals.net)

[3] Udi Manber, Introduction to Algorithms: a Creative Approach, Addison Wesley, 1989

[4] Soubhik Chakraborty *et. al*., On Why Parameters of Input Distribution Need be Taken Into Account for a More Precise Evaluation of Complexity for Certain Algorithms, InterStat Oct2005(www.interstat.statjournals.net)

The mathematics of Winograd's algorithm:-

The (i, j)-th element in the product of two nxn matrices A and B is defined as
$C(i, j) = \sum_{k=1 \text{ to } n} a(i, k)\, b(k, j)$; where i, j = 1, 2, …n ……..(1)
This is how we multiply two nxn matrices by the classical method.

To describe Winograd's algorithm assume for simplicity that n is even.
Denote $A_i = \sum_{k=1 \text{ to } n/2} a(i, 2k-1)\, a(i, 2k)$, i=1, 2…n ……(2)
And $B_j = \sum_{k=1 \text{ to } n/2} b(2k-1, j)\, b(2k, j)$, j=1, 2…n …… (3)

After re-arranging the terms of (1), we get

$$C(i, j) = \sum_{k=1 \text{ to } n/2} [a(i, 2k-1) + b(2k, j)][a(i, 2k) + b(2k-1, j)] - A_i - B_j$$

$$= Z - A_i - B_j \text{ (say)} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(4)$$

$$\text{where } Z = \sum_{k=1 \text{ to } n/2} [a(i, 2k-1) + b(2k, j)][a(i, 2k) + b(2k-1, j)]$$

But the A_i's and the B_j's need to be computed once only for each row or column and therefore can be pre-processed. To compute all the A_i's and the B_j's requires only n^2 multiplications (nxn/2 + nxn/2 from (2) and (3)). The expression for Z for all the C(i, j) elements (there are n^2 of them) will require n^2 x n/2 multiplications from (4)so that the total number of multiplications for Winograd's algorithm reduces to $n^3/2 + n^2$ while the number of additions increases by about $n^3/2$ from ordinary (classical) matrix multiplication described in (1). Winograd's algorithm is specially recommended in those computer systems which perform additions considerably faster than multiplications. Thus an insight into Winograd's algorithm provides an eye-opener to the fact that even for less complicated expressions such as in matrix multiplication, re-arranging the order of the computating operations (and pre-processing some) can certainly make a difference in the computational and hence time complexity.

Winograd Runtime Results

Observations taken at optimal points of a cubic fit (Kussmaul protection)

Table – 1
Run Time (of 6 trials in seconds) output on 'n'.

n / Trials	100	650	1500	4000
1	0.015	12.578	157.111	374.958
2	0.016	12.625	157.424	374.817
3	0.016	12.626	157.127	375.106
4	0.015	12.563	157.312	374.918
5	0.015	12.547	157.158	374.924
6	0.016	12.547	157.172	374.994

System Specifications:

128 MB RAM, Pentium III processor with 550 MHz, 12 GB HDD

The regression equation for a quadratic fit is
run-time = 15.56 - 0.1190 n1 + 0.000148 n1**2

S = 10.1453 R-Sq = 99.6% R-Sq(adj) = 99.6%

Analysis of Variance

Source	DF	SS	MS	F	P
Regression	2	545474	272737	2649.82	0.000
Error	21	2161	103		
Total	23	547635			

Sequential Analysis of Variance

Source	DF	SS	F	P
Linear	1	477106	148.82	0.000
Quadratic	1	68368	664.24	0.000