Random Search Algorithm for 2x2 Matrices Multiplication Problem

Sheng-Jie Deng, Yu-Ren Zhou, Hua-Qing Min, and Jing-Hui Zhu

Abstract—**Since Volker Strassen proposed a recursive matrix multiplication algorithm reducing the time complexity to n2.81 in 1968, many scholars have done a lot of research on this basis. In recent years, researchers have proposed using computer algorithms to solve fast matrix multiplication problem. They have found Strassen's algorithm or other algorithms that have the same time complexity as Strassen algorithm by using genetic algorithm. In this paper, we used random search algorithm to find the matrix multiplication algorithms that require fewer multiplications. And we used combining Gaussian elimination for the first time to improve calculation speed; meanwhile we improved the local search technology to enhance the local search capability of the algorithm. In the numerical experiments of 2×2 matrices, the results verified the effectiveness of the algorithm. Compared with the existing genetic algorithm, the new method has obvious advantage of quick search, and found some of new matrix multiplication algorithms.**

I. Introduction

Matrix multiplication is one of the most basic arithmetic operations. As a mathematical tool, it has a wide range of applications in scientific research and engineering analysis. The time complexity of matrix multiplication has a direct impact on the time complexity of its application.

Suppose A, B are two n×n matrices, the product of matrix multiplication C=AB. General calculation method of matrix multiplication is $C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj} \ (1 \le i, j \le n)$. Calculating each element in matrix C needs n scalar multiplications and n-1 additions. So its running time is $T(n) = O(n^3)$.

In 1968, Volker Strassen, a German mathematician, proposed a recursive matrix multiplication algorithm [1] that reduces the time complexity of matrix multiplication from $O(n^3)$ to $O(n^{2.81})$. It greatly improves the efficiency of matrix multiplication. Matrix multiplication can be divided into the multiplication of partitioned matrices based on the nature. Supposing $n = 2^k$ (k is an integer), matrices A, B, C can be decomposed into four n/2×n/2 submatrices: $\begin{pmatrix} C_1 & C_2 \\ C_3 & C_4 \end{pmatrix} = \begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix} \begin{pmatrix} B_1 & B_3 \\ B_2 & B_4 \end{pmatrix}$ .Then the matrix multiplication corresponds to the following four equations:

$$C_1 = A_1B_1 + A_2B_2 \quad C_2 = A_1B_3 + A_2B_4$$
$$C_1 = A_3B_1 + A_4B_2 \quad C_1 = A_3B_3 + A_4B_4$$

After analyzing the four equations, we know that this method needs 8 multiplications of n/2×n/2 submatrices and 4 additions of n/2×n/2 submatrices. The time complexity is $T(n) = 8T(n/2) + O(n^2)$. We can get the solution $T(n) = O(n^3)$ by recursive solution. So it has the same time complexity as the general calculation of matrix multiplication. Strassen discovered a clever recursive approach that requires only 7 multiplications of submatrices. Then the time complexity can be reduced to $T(n) = n^{\log_2 7} = n^{2.81}$.

Strassen's algorithm is as follows. P_1 - P_7 can be produced by one multiplication of n/2×n/2 submatrices. C_1 - C_4 can be obtained by matrix addition and subtraction.

$$P_1 = A_1(B_3 - B_4) \qquad P_2 = (A_1 + A_2)B_4$$
$$P_3 = (A_3 + A_4)B_1 \qquad P_4 = A_4(-B_1 + B_2)$$
$$P_5 = (A_1 + A_4)(B_1 + B_4) \quad P_6 = (A_2 - A_4)(B_2 + B_4)$$
$$P_7 = (-A_1 + A_3)(B_1 + B_3)$$
$$C_1 = -P_2 + P_4 + P_5 + P_6 \qquad C_2 = P_1 + P_2$$
$$C_3 = P_3 + P_4 \qquad C_4 = P_1 - P_3 + P_5 + P_7$$

Since Strassen's algorithm was proposed, many scholars have done a lot of research on this basis. In the research of 2×2 matrix format, the fastest algorithm is Wingrad's algorithm [2]. Compared with Strassen's algorithm which needs 7 multiplications and 18 additions, Wingrad's algorithm only needs 7 multiplications and 15 additions. However, the time that addition requires is negligible compared with multiplication. So it doesn't reduce the time complexity of multiplication. It has been proved that 7 multiplications is optimal in multiplying 2×2 matrices problem[2]. And in [3], it proved that 15 additions are essential. However, when it comes to 3×3 matrices multiplication problem, the fastest multiplication algorithm people have found so far needs 23 multiplications. Its time complexity is $O(n^{\log_3 23})$ [4]. Since $O(n^{\log_3 23}) > O(n^{2.81})$, the calculation speed of this algorithm is not higher than Strassen's algorithm. Therefore, searching an algorithm that needs fewer multiplications in 3×3 matrix format is an interesting research direction. If we find an algorithm that needs fewer than 21 multiplications, it will be faster than Strassen's algorithm.

In addition to using matrix decomposition method for solving fast matrix multiplication problems, there are a number of mathematicians applying other mathematical methods to solve this problem. The fastest matrix multiplication algorithm with $O(n^{2.376})$ arithmetic operations [5] is obtained by using arithmetic progression. But due to its

Manuscript received April 9, 2010. This work was supported in part by the National Natural Science Foundation of China under Grant 60873078, in part by the Natural Science Foundation of Guangdong Province of China under Grant 9251064101000010, and in part by the Fundamental Research Funds for the Central Universities 2009ZM0297.

Sheng-Jie Deng, Yu-Ren Zhou,Hua-Qing Min, Jing-Hui Zhu are with School of Computer Science and Engineering, South China University of Technology, Guangzhou, 510006, China. (E-mail: dengshengjie@126.com(phone: 13570587821), yrzhou@scut.edu.cn (phone: 15302290362), hqmin@scut.edu.cn, csjhzhu@scut.edu.cn).

large multiplicative constants, it has no practical value. Henry Cohn proposed group-theoretic approach for fast matrix multiplication that the asymptotic complexity achieves $O(n^{2.41})$ [6][7], But it is only of theoretical significance so far.

Though many algorithms that have the same time complexity as Strassen's algorithm were proposed, they are obtained by mathematicians only through calculation and analysis. Reference [8][9] describe a simple way to construct the algorithm of fast matrix multiplication, but it is impossible to naively search all possible cases exhaustively. Therefore, some researchers started to propose using computer algorithm to solve fast matrix multiplication. In 2001, Kolean J F and Bruce P [9] applied evolutionary algorithm to search the fast matrix multiplication algorithm for the first time. But too many constraints were added in their search algorithm, the search efficiency is very low, and they could only find the algorithm that was exactly the same with Strassen's algorithm. O. Seunghyun and M. Byung-Ro[10] improved the genetic algorithm, and modified the fitness function, the way of the crossover, the local search technology in 2009. The efficiency has been significantly improved, and they found a large number of different types of matrix multiplication algorithms which have the same time complexity as Strassen's algorithm. But the efficiency of their search algorithm was still not satisfactory. They simply described that the average time of searching is about several hours.

In this paper, we used random search algorithm to solve 2×2 matrices multiplication problem. We used combining Gaussian elimination for the first time to improve calculation speed, and at the same time we improved the local search technology to enhance the local search capability of the search algorithm. The result is very well. The average time of search is about 2 minutes. Sometimes it can find a solution in 1 second. We found a more complete set of algorithm than [10].So it will be a possible solution for 3×3 matrices multiplication problem.

This paper is organized as follows. In Section **II**, we describe the fast matrix multiplication in mathematical language. Section **III** describes the detail of random search algorithm. Section **IV** shows the experimental result and comparison with genetic algorithm in [10]. Finally, this paper concludes in Section **V**.

II. 2×2 MATRICES MULTIPLICATION PROBLEM

Reference [8][9] describe a simple way to construct the solution of fast matrix multiplication problem. The problem can be described as follows:

Construct a set of $P_i (i = 1, 2, \ldots, 7)$ that can be described in the following form:

$$P_i = (\sum_{j=1}^{4} a_{ij} A_j)(\sum_{j=1}^{4} b_{ij} B_j) \quad a_{ij}, b_{ij} \in \{-1, 0, 1\} \tag{1}$$

If $C_j (j = 1, 2, 3, 4)$ can be represented by a linear combination of $P_i (i = 1, 2, \ldots, 7)$:

$$C_j = \sum_{i=1}^{7} \varphi_{ji} P_i \quad \varphi_{ji} \in R, \ \text{j} \in \{1, 2, 3, 4\} \quad . \tag{2}$$

$P_i (i = 1, 2, \ldots, 7)$ is a feasible 2×2 matrices multiplication algorithm (feasible solution).

In (1), each P_i has 8 variables, and $P_i (i = 1, 2, \ldots, 7)$ has 7×8=56 variables. Eliminating all zero and symmetric pairs, the total number of different P_i is $((3^4 - 1)/2)((3^4 - 1)/2) = 1600$. If we naively search all possible case exhaustively, we need to verify $\begin{pmatrix} 7 \\ 1600 \end{pmatrix} = 5.26 \times 10^{18}$ candidate solutions.

The problem can be described in matrix form. $P_i (i = 1, 2, \ldots, 7)$ can be turned into :

$$P_i = \sum_{j=1}^{16} (a_{i, \lceil j/4 \rceil})(b_{i,(j-1) \bmod 4 + 1}) A_{\lceil j/4 \rceil} B_{(j-1) \bmod 4 + 1} \quad . \tag{3}$$

Then it can be expressed in vector form:

$$\begin{aligned} \vec{P}_i = (&a_{i1}b_{i1} \ a_{i1}b_{i2} \ a_{i1}b_{i3} \ a_{i1}b_{i4} \ a_{i2}b_{i1} \ a_{i2}b_{i2} \ a_{i2}b_{i3} \ a_{i2}b_{i4} \\ &a_{i3}b_{i1} \ a_{i3}b_{i2} \ a_{i3}b_{i3} \ a_{i3}b_{i4} \ a_{i4}b_{i1} \ a_{i4}b_{i2} \ a_{i4}b_{i3} \ a_{i4}b_{i4})^T \end{aligned} \tag{4}$$

$a_{im}b_{im}$ denotes the coefficient of $A_m B_n$. Then $P_i (i = 1, 2, \ldots, 7)$ can be expressed as a 16×7 matrix:

$$P_{16\times7} = (\vec{P}_1 \ \vec{P}_2 \ \vec{P}_3 \ \vec{P}_4 \ \vec{P}_5 \ \vec{P}_6 \ \vec{P}_7) \, . \tag{5}$$

In the same way, $C_j (j = 1, 2, 3, 4)$ can be expressed as 16×4 matrix

$$C_{16\times4} = (\vec{C}_1 \ \vec{C}_2 \ \vec{C}_3 \ \vec{C}_4) \ . \tag{6}$$

The coefficients $\varphi_{ji} \ j \in \{1, 2, 3, 4\}, i \in \{1, 2, \ldots, 7\}$ can be reorganized as 7-dimensional vector $\vec{M}_j$ .Then coefficient matrix is

$$M_{7\times4} = (\vec{M}_1 \, \vec{M}_2 \, \vec{M}_3 \, \vec{M}_4) \, . \tag{7}$$

Then the problem can be converted to the following matrix equation.

$$\begin{aligned} &P_{16\times7}\vec{M}_1 = \vec{C}_1 \\ &P_{16\times7}\vec{M}_2 = \vec{C}_2 \qquad \rightarrow P_{16\times7} M_{7\times4} = C_{16\times4} \\ &P_{16\times7}\vec{M}_3 = \vec{C}_3 \\ &P_{16\times7} M_4 = C_4 \end{aligned} \tag{8}$$

$$\begin{array}{c} \begin{array}{ccccccc} \vec{P}_1 & \vec{P}_2 & \vec{P}_3 & \vec{P}_4 & \vec{P}_5 & \vec{P}_6 & \vec{P}_7 \end{array} \\ \begin{pmatrix} a_{11}b_{11} & a_{21}b_{21} & a_{31}b_{31} & a_{41}b_{41} & a_{51}b_{51} & a_{61}b_{61} & a_{71}b_{71} \\ a_{11}b_{12} & a_{21}b_{22} & a_{31}b_{32} & a_{41}b_{42} & a_{51}b_{52} & a_{61}b_{62} & a_{71}b_{72} \\ a_{11}b_{13} & a_{21}b_{23} & a_{31}b_{33} & a_{41}b_{43} & a_{51}b_{53} & a_{61}b_{63} & a_{71}b_{73} \\ a_{11}b_{14} & a_{21}b_{24} & a_{31}b_{34} & a_{41}b_{44} & a_{51}b_{54} & a_{61}b_{64} & a_{71}b_{74} \\ a_{12}b_{11} & a_{22}b_{21} & a_{32}b_{31} & a_{42}b_{41} & a_{52}b_{51} & a_{62}b_{61} & a_{72}b_{71} \\ a_{12}b_{12} & a_{22}b_{22} & a_{32}b_{32} & a_{42}b_{42} & a_{52}b_{52} & a_{62}b_{62} & a_{72}b_{72} \\ a_{12}b_{13} & a_{22}b_{23} & a_{32}b_{33} & a_{42}b_{43} & a_{52}b_{53} & a_{62}b_{63} & a_{72}b_{73} \\ a_{12}b_{14} & a_{22}b_{24} & a_{32}b_{34} & a_{42}b_{44} & a_{52}b_{54} & a_{62}b_{64} & a_{72}b_{74} \\ a_{13}b_{11} & a_{23}b_{21} & a_{33}b_{31} & a_{43}b_{41} & a_{53}b_{51} & a_{63}b_{61} & a_{73}b_{71} \\ a_{13}b_{12} & a_{23}b_{22} & a_{33}b_{32} & a_{43}b_{42} & a_{53}b_{52} & a_{63}b_{62} & a_{73}b_{72} \\ a_{13}b_{13} & a_{13}b_{13} & a_{13}b_{13} & a_{43}b_{43} & a_{53}b_{53} & a_{63}b_{63} & a_{73}b_{73} \\ a_{13}b_{14} & a_{13}b_{14} & a_{33}b_{34} & a_{43}b_{44} & a_{53}b_{54} & a_{63}b_{64} & a_{73}b_{74} \\ a_{14}b_{11} & a_{24}b_{21} & a_{34}b_{31} & a_{44}b_{41} & a_{54}b_{51} & a_{64}b_{61} & a_{74}b_{71} \\ a_{14}b_{12} & a_{24}b_{22} & a_{34}b_{32} & a_{44}b_{42} & a_{54}b_{52} & a_{64}b_{62} & a_{74}b_{72} \\ a_{14}b_{13} & a_{24}b_{23} & a_{34}b_{33} & a_{44}b_{43} & a_{54}b_{53} & a_{64}b_{63} & a_{74}b_{73} \\ a_{14}b_{14} & a_{24}b_{24} & a_{34}b_{34} & a_{44}b_{44} & a_{54}b_{54} & a_{64}b_{64} & a_{74}b_{74} \end{pmatrix} \end{array} M_{7\times4} = \begin{array}{c} \begin{array}{cccc} \vec{C}_1 & \vec{C}_2 & \vec{C}_3 & \vec{C}_4 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

So the problem can be described as: finding an assignment of 56 variables in $P_{16\times7}$ so that there exists a real matrix $M_{7\times4}$ to hold the equation (8).

III. Random search algorithm

We need to find a $P_{16\times7}$ that can represent C_1 - C_4. We can use fitness(P) to denote the number of $C_j\,(j=1,2,3,4)$ that can be represented. Then fitness(P)=4 means $C_j\,(j=1,2,3,4)$ can be all represented by $P_i(i=1,2,\ldots,7)$. So the fast matrix multiplication problem can be transformed to an optimization problem. Our aim is to find an assignment of 56 variables to maximize fitness(P).

Our random search algorithm is described as follows:

Algorithm 1:

Step 1: Select a value from {-1, 0, 1} uniformly at random and assign it to each of 56 variables in $P_{16\times7}$.

Step 2: If fitness(P) =4, then a feasible solution was found , quit . If time is out, then we fail to find the solution, quit.

Step 3: choose the column-vector $\vec{P}_i$ which is least used in the solvable equations in the matrix $P_{16\times7}$.

Step 4: Local search for $\vec{P}_i$: traverse all the possible value of $\vec{P}_i$ (a total of 1600 different assignments), and calculate the new fitness(P).If fitness(P) increase , replace it.

Step 5: If the fitness(P) increase, go to Step 2; else go to Step 1.

A. Combining Gaussian elimination

Given a matrix $P_{16\times7}$, we can use Gaussian elimination to solve the matrix equation. In general, when we calculate the fitness(P) ,we need to solve $P_{16\times7}\vec{M}_j=\vec{C}_j\ \ (j=1,2,3,4)$, if real vector $\vec{M}_j$ exists, $\vec{C}_j$ can be represented by $P_{16\times7}$. So we need to use Gaussian elimination four times.

We can find by observing that, in Gaussian elimination algorithm, we need to make the primary transformation of line in matrix $P_{16\times7}$ at fist. And this procedure has no effect on columns. So we can calculate fitness(P) by using Gaussian elimination only once. The detail is as follows:

We make the primary transformation of line in extended matrix $T_{16\times11}=P_{16\times7}\,|\,C_{16\times4}$, and obtain the matrix $\begin{pmatrix} D_{7*7} & H_{7*4} \\ F_{9*7} & G_{9*4} \end{pmatrix}$, such that the partitioned matrix D_{7*7} is a upper triangular matrix and F_{9*7} is a zero matrix. Then we can conclude:

(1) If the j th column of G_{9*4} is zero vector, then $\vec{C}_j$ can be presented by $P_{16\times7}$.

(2) If G_{9*4} is zero matrix, it means that $P_{16\times7}$ is a feasible solution.

(3) The number of zero vector in G_{9*4} is the fitness(P) we want to calculate.

When calculating the fitness(P) ,we only need to use Gaussian elimination once by using the above optimization algorithm. So the calculation time can reduce to 1/4.

IV. Experimental result and analysis

In the experiment, the timeout time is 20 minutes. That is, if the search algorithm can't find the feasible solution in 20 minutes, it will quit and restart the new search again. The search time is set to 50. The result was showed in Table 1.

In order to compare the result easily, we also conduct the experiment of genetic algorithm that [10] proposed. The following is a brief description of genetic algorithm.

Algorithm 2:

Step 1: Each chromosome represents a solution, and it is composed of 7 P_i . Randomly generate initial population. The size of Population is 15.

Step 2: Randomly select 7 pairs of parent chromosomes from population, and produce 14 offspring chromosomes. Offspring chromosomes and non-crossover chromosomes are put into the new group. The crossover procedure is as follows: uniformly to select P_i at random that is used at least once in the solvable equations and add it into the offspring until there are more than seven P_i in the offspring or no such P_i remains in parents. Then randomly produce new P_i and add it into the offspring until the offspring has seven P_i s.

Step 3: Local search for the all solutions of the new group.

(1) Pursue the solution to be linearly independent.

(2) Randomly select a P_i that is not used in any solvable equation (if all are used, arbitrary P_i is selected), then replace it with other 1600 cases and calculate the fitness, if fitness increases, replace P_i . Otherwise, P_i remains unchanged.

Step 4: Use the roulette selection methods according to their fitness, select 15 chromosomes into the new population. Replace the old population with the new population.

Step 5: If the best fitness is 4, it means we succeed to find the feasible solution, then quit. If the search exceeds the maximum number of iterations, we fail to find the solution and quit; else go to Step 2.

In the experiment of genetic algorithm, we found that the algorithm converged in fewer than 50 iterations. So we reduce the maximum number of iteration from 1000[10] to 50. The efficiency of the algorithm was improved obviously. We provided statistical result of contrast experiment in TABLE I.

The result in TABLE I. was obtained on an Intel Pentium 4 2.80GHz CPU. We can see that the efficiency of genetic algorithm [10] proposed is low. It took at least 4 hours to find a feasible solution in most of the cases. In the experiments, it can't find the feasible solution even in 15 hours sometimes. After we reduced the maximum number of iteration to 50, the efficiency of genetic algorithms largely increased. The result of random search algorithm we proposed was very good. It can find the feasible solution in 2 minutes most of the time. It can‘t find the feasible solution in 20 minutes only in a very few cases. Compared with the genetic algorithm, the efficiency of random search is much higher.

To compare with the feasible solutions that [10] had found, we run the random search algorithm 10000 times. TABLE II. showed the statistical result. We use the same method to analysis the feasible solution, classifying the solution according to the total number of a_{ij} and b_{ij} that are not zero in P_i . Strassen's algorithm belongs to group 3333444, and Wingrad's algorithm belongs to group 2244556. TABLE III. list a representative solution in each group.

TABLE II. showed that the random search algorithm can not only find more solutions in every group, but also find a new group of solutions (group 4: 226668). Only in the group 4 and group 9, the algorithm finds a few solutions.

By analyzing the solution of group 4 and group 9, we can find that in each solution of these groups, each P_i is at least used twice, compared with the solution of other groups in that there are 1-3 P_i had only been used once. So when searching the solution of these two groups, local search technology we proposed could not increase the fitness in greater probability. That is the reason why we only find very few solutions in these two groups.

TABLE I. STATISTICAL RESULT OF CONTRAST EXPERIMENT

	GA (1000 iterations)	**GA (50 iterations)**	**Random Search Algorithm**
Search times	50	50	50
Timeout time	5 h	20 m	20 m
the total time spent	9167 m	682 m	95m24s (5724s)
The total number of successfully finding the feasible solution	31	35	50
Success probability	62%	70%	100%
The average time of successfully finding the feasible solution	9167/31 =295.7m	682/35 =19.5m	5724/50 =114 s
The shortest time of successfully finding the feasible solution	369 s	46 s	1 s

TABLE II. SOLUTION COMPARISON

	Genetic algorithm[10]		**Random search algorithm**	
	The total number of solution	***The total number of different solution***	***The total number of solution***	***The total number of different solution***
Group 1: 2233666	149	57	1654	64
Group 2: 2244556	46	36	1591	64
Group 3: 2245667	33	30	1201	128
Group 4: 2266668	0	0	2	2
Group 5: 3333444	648	32	712	32
Group 6: 3333455	891	128	2753	128
Group 7: 3333888	522	32	697	32
Group 8: 3344556	61	35	645	64
Group 9: 4446666	3	3	6	6
Group 10: 4455666	30	19	712	64
Total	2383	372	9973	584

TABLE III. 10 DIFFERENT TYPES OF FAST MATRIX MULTIPLICATION ALGORITHM

Group 1:2233666	**Group 2:2244556**	**Group 3:2245667**
$P_1=A_1B_1$ $P_2=A_2B_2$ $P_3=(A_3+A_4)B_3$ $P_4=A_4(B_3-B_4)$ $P_5=(A_1+A_2+A_3+A_4)(B_2+B_4)$ $P_6=(A_1+A_3+A_4)(B_2-B_3+B_4)$ $P_7=(A_1+A_3)(B_1-B_2+B_3-B_4)$ $C_1=P_1+P_2$ $C_2=-P_2-P_3+P_5-P_6$ $C_3=-P_1+P_4+P_6+P_7$ $C_4=P_3-P_4$	$P_1=A_1B_1$ $P_2=A_2B_2$ $P_3=(A_1+A_3)(B_3+B_4)$ $P_4=(A_3-A_4)(B_1+B_3)$ $P_5=(A_1-A_2+A_3-A_4)B_4$ $P_6=A_4(B_1+B_2+B_3+B_4)$ $P_7=(A_1+A_3-A_4)(B_1+B_3+B_4)$ $C_1=P_1+P_2$ $C_2=-P_1-P_4-P_5+P_7$ $C_3=-P_1-P_3+P_6+P_7$ $C_4=P_1+P_3+P_4-P_7$	$P_1=A_1B_1$ $P_2=A_2B_2$ $P_3=(A_1+A_3)(B_3+B_4)$ $P_4=(A_1-A_2+A_3-A_4)B_4$ $P_5=(A_1-A_2-A_3+A_4)(B_1+B_3)$ $P_6=(A_2-A_4)(B_1+B_2+B_3+B_4)$ $P_7=(A_1+A_2+A_3-A_4)(B_1+B_3+B_4)$ $C_1=P_1+P_2$ $C_2=-P_1-0.5P_4+0.5P_5+0.5P_7$ $C_3=-P_1+P_2-P_3-P_6+P_7$ $C_4=P_1+P_3-0.5P_4-0.5P_5-0.5P_7$
Group 4:2266668（the new group）	**Group 5:3333444**	**Group 6:3333455**

$P_1 = A_3B_3$ $P_2 = A_4B_4$ $P_3 = (A_1+A_2+A_3+A_4)(B_1+B_3)$ $P_4 = (A_1+A_2-A_3-A_4)(B_2-B_4)$ $P_5 = (A_1-A_3)(B_1-B_2-B_3+B_4)$ $P_6 = (A_2+A_4)(B_1-B_2+B_3-B_4)$ $P_7 = (A_1+A_2-A_3+A_4)(B_1-B_2+B_3+B_4)$ $C_1 = P_1 - P_2 + 0.5P_3 + 0.5P_4 + 0.5P_5 - 0.5P_6$ $C_2 = P_1 - P_2 - 0.5P_5 - 0.5P_6 + 0.5P_7$ $C_3 = P_1 + P_2 + 0.5P_3 - 0.5P_4 - 0.5P_7$ $C_4 = P_1 + P_2$	$P_1 = (A_1+A_2)B_1$ $P_2 = A_2(B_1-B_2)$ $P_3 = (A_3+A_4)B_4$ $P_4 = A_3(B_3-B_4)$ $P_5 = (A_1+A_3)(B_1+B_3)$ $P_6 = (A_2-A_3)(B_1+B_4)$ $P_7 = (A_2+A_4)(B_2+B_4)$ $C_1 = P_1 - P_2$ $C_2 = -P_1 - P_4 + P_5 + P_6$ $C_3 = P_2 - P_3 - P_6 + P_7$ $C_4 = P_3 + P_4$	$P_1 = (A_1+A_2)B_1$ $P_2 = (A_1+A_3)B_3$ $P_3 = (A_2+A_4)B_4$ $P_4 = A_2(B_1-B_2)$ $P_5 = (A_3+A_4)(B_2+B_4)$ $P_6 = (A_2-A_3)(B_1-B_2-B_4)$ $P_7 = A_3(B_1-B_2+B_3-B_4)$ $C_1 = P_1 - P_4$ $C_2 = P_2 + P_4 - P_6 - P_7$ $C_3 = -P_3 + P_4 + P_5 - P_6$ $C_4 = P_3 - P_4 + P_6 + P_7$
Group 7:3333888	**Group 8:3344556**	
$P_1 = (A_1+A_2)B_1$ $P_2 = A_2(B_1-B_2)$ $P_3 = (A_3-A_4)B_3$ $P_4 = A_4(B_3+B_4)$ $P_5 = (A_1+A_2+A_3+A_4)(B_1+B_2+B_3+B_4)$ $P_6 = (A_1+A_2+A_3-A_4)(B_1-B_2-B_3-B_4)$ $P_7 = (A_1-A_2+A_3-A_4)(B_1-B_2+B_3-B_4)$ $C_1 = P_1 - P_2$ $C_2 = P_2 - P_3 - 0.5P_6 + 0.5P_7$ $C_3 = -P_1 - P_4 + 0.5P_5 + 0.5P_6$ $C_4 = P_3 + P_4$	$P_1 = (A_1+A_3)B_1$ $P_2 = (A_2+A_4)B_2$ $P_3 = (A_1+A_2)(B_1+B_3)$ $P_4 = (A_3+A_4)(B_2-B_4)$ $P_5 = A_2(B_1-B_2+B_3-B_4)$ $P_6 = A_3(B_1-B_2-B_3+B_4)$ $P_7 = (A_2-A_3)(B_1+B_2+B_3-B_4)$ $C_1 = P_1 - 0.5P_5 - 0.5P_6 + 0.5P_7$ $C_2 = -P_1 + P_3 - 0.5P_5 + 0.5P_6 - 0.5P_7$ $C_3 = P_2 + 0.5P_5 + 0.5P_6 - 0.5P_7$ $C_4 = P_2 - P_4 + 0.5P_5 - 0.5P_6 - 0.5P_7$	
Group 9:4446666	**Group 10:4455666**	
$P_1 = (A_1+A_2)(B_1+B_2)$ $P_2 = (A_1-A_2)(B_1-B_2)$ $P_3 = (A_1+A_4)(B_2+B_3)$ $P_4 = (A_1+A_2+A_3+A_4)(B_1-B_3)$ $P_5 = (A_1-A_2-A_3+A_4)(B_1+B_3)$ $P_6 = (A_2+A_4)(B_1-B_2-B_3+B_4)$ $P_7 = (A_2-A_4)(B_1+B_2+B_3+B_4)$ $C_1 = 0.5P_1 + 0.5P_2$ $C_2 = -0.5P_1 + 0.5P_2 + P_3 + 0.5P_6 + 0.5P_7$ $C_3 = -0.5P_1 + 0.5P_2 + P_3 + 0.5P_4 - 0.5P_5$ $C_4 = 0.5P_1 + 0.5P_2 - 0.5P_4 - 0.5P_5 + 0.5P_6 - 0.5P_7$	$P_1 = (A_1+A_2)(B_1+B_3)$ $P_2 = (A_3-A_4)(B_1-B_3)$ $P_3 = A_2(B_1-B_2+B_3-B_4)$ $P_4 = A_4(B_1+B_2-B_3-B_4)$ $P_5 = (A_1+A_2+A_3+A_4)(B_1+B_2)$ $P_6 = (A_1-A_2+A_3-A_4)(B_1-B_2)$ $P_7 = (A_1+A_2+A_3-A_4)(B_2-B_3)$ $C_1 = 0.5P_1 - 0.5P_2 + 0.5P_6 + 0.5P_7$ $C_2 = 0.5P_1 + 0.5P_2 - P_3 - 0.5P_6 - 0.5P_7$ $C_3 = -0.5P_1 + 0.5P_2 + 0.5P_5 - 0.5P_7$ $C_4 = -0.5P_1 - 0.5P_2 - P_4 + 0.5P_5 - 0.5P_7$	

V. Conclusion

We can conclude from the experimental results that the random search algorithm we proposed for 2×2 matrices multiplication problem is effective. And its efficiency is much higher than genetic algorithm [10]. It also provides a possible solution for 3×3 matrices multiplication problem.

References

[1] V. Strassen. Gaussian elimination is not optimal. *Numer. Math* vol.13, no.4, pp. 354–356, 1969.

[2] Winograd S. On multiplication of 2×2 matrices. *Linear Algebra and its Applications*, vol.4, no.4, pp. 381-388, 1971.

[3] Bshouty N H. On the additive complexity of 2×2 matrix multiplication. *Information Processing Letters*, vol.56, no. 6, pp. 329-335, 1995.

[4] J. Laderman. A non-commutative algorithm for multiplying 3×3 matrices using 23 multiplications. Bull. *Amer. Math. Soc*, vol.82, no.1, pp. 126–128, 1976.

[5] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progression. *Symbolic Computation*, vol.9, no.3, pp. 250-280, 1990.

[6] H. Cohn, R. Kleinberg, B. Szegedy, and C. Umans. Group-theoretic algorithms for matrix multiplication. *Foundations of Computer Science. 46th Annual IEEE Symposium*, pp. 23-25, 2005.

[7] H. Cohn and C. Umans. A group-theoretic approach to fast matrix multiplication. *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium*, pp. 438-449, 2003.

[8] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. Second Edition. Cambridge, MIT Press, pp.735–741, 2001.

[9] Kolen J F, Bruce P. Evolutionary Search for Matrix Multiplication Algorithms. *Proceedings of the Fourteenth International Florida Artificial Intelligence Research Society Conference, 2001. AAAI Press*, pp. 161-165, 2001.

[10] O. Seunghyun and M. Byung-Ro, Automatic Reproduction of a Genius Algorithm: Strassen's Algorithm Revisited by Genetic Search, *IEEE Transactions on Evolutionary Computation*, vol.14, pp. 246-251, 2010.