Automatic Reproduction of a Genius Algorithm: Strassen's Algorithm Revisited by Genetic Search

Seunghyun Oh and Byung-Ro Moon, *Member, IEEE*

***Abstract*—In 1968, Volker Strassen, a young German mathematician, announced a clever algorithm to reduce the asymptotic complexity of $n \times n$ matrix multiplication from the order of n^3 to $n^{2.81}$. It soon became one of the most famous scientific discoveries in the 20th century and provoked numerous studies by other mathematicians to improve upon it. Although a number of improvements have been made, Strassen's algorithm is still optimal in his original framework, the bilinear systems of 2×2 matrix multiplication, and people are still curious how Strassen developed his algorithm. We examined it to see if we could automatically reproduce Strassen's discovery using a search algorithm and find other algorithms of the same quality. In total, we found 608 algorithms that have the same quality as Strassen's, including Strassen's original algorithm. We partitioned the algorithms into nine different groups based on the way they are constructed. This paper was made possible by the combination of genetic search and linear–algebraic techniques. To the best of our knowledge, this is the first work that automatically reproduced Strassen's algorithm, and furthermore, discovered new algorithms with equivalent asymptotic complexity using a search algorithm.**

***Index Terms*—Bilinear, Gaussian elimination, genetic algorithm, matrix multiplication, Sammon mapping, Strassen's algorithm.**

I. Introduction

MATRIX MULTIPLICATION is a fundamental algebraic operation used heavily in the fields of science, engineering, and economics. In particular, it is an important component in applications such as solving linear equations, image processing, control engineering, and graph problems. Multiplication of two $n \times n$ matrices of the form $C = A \cdot B$ generally requires n^3 scalar multiplications. This is because the n^2 elements of the resulting matrix C each needs n scalar multiplications. Before 1968, people never doubted that matrix multiplication requires at least n^3 scalar multiplications.

In 1968, Volker Strassen, a young German mathematician, announced a clever recursive algorithm for multiplying two $n \times n$ matrices that requires only $n^{2.81}$ scalar multiplications [1].

Manuscript received October 07, 2008; revised February 24, 2009 and May 29, 2009. This work was supported in part by Brain Korea 21 and the Engineering Research Center of Excellence Program of the Korea Ministry of Education, Science and Technology, and the Korea Science and Engineering Foundation, under Grant R11-2008-007-02002-0.

S. Oh is with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: austeban@gmail.com).

B.-R. Moon is with the Optimization and Investment Engineering Laboratory, School of Computer Science and Engineering, Seoul National University, Seoul 151-744, Korea (e-mail: moon@snu.ac.kr).

Digital Object Identifier 10.1109/TEVC.2009.2029568

Although improvements have been made since then, Strassen's work is still optimal under his original framework of dividing matrices into four $n/2 \times n/2$ matrices and finding bilinear combinations. Strassen's algorithm is one of the most famous scientific discoveries in the 20th century.

Two primary questions motivated this paper. First, how many algorithms other than Strassen's exist under the same framework? Second, can a search algorithm achieve the power of finding the same or equivalent solutions to Strassen's without enormous efforts of genius human beings? This paper contains a partial answer to these by using genetic search.

Suppose we wish to compute the product of two matrices, which takes the form $C = A \cdot B$, where each of A, B, and C is an $n \times n$ matrix. Assuming n is a power of 2 ($n = 2^k$ for some integer k), we divide each matrix into four $n/2 \times n/2$ matrices as follows:

$$\left(\begin{array}{c|c} C_1 & C_2 \\ \hline C_3 & C_4 \end{array}\right) = \left(\begin{array}{c|c} A_1 & A_2 \\ \hline A_3 & A_4 \end{array}\right) \left(\begin{array}{c|c} B_1 & B_3 \\ \hline B_2 & B_4 \end{array}\right).$$

This leads to the following four equations:

$$C_1 = A_1B_1 + A_2B_2, \quad C_2 = A_1B_3 + A_2B_4,$$
$$C_3 = A_3B_1 + A_4B_2, \quad C_4 = A_3B_3 + A_4B_4.$$

Each of the four equations requires two multiplications of $n/2 \times n/2$ matrices. Therefore, we need a total of eight multiplications of $n/2 \times n/2$ matrices. The total number of scalar multiplications, T_n, for the product of two $n \times n$ matrices is $T_n = 8T_{n/2}$ and $T_1 = 1$. This leads to an asymptotic complexity of $T_n = n^{\log_2 8} = n^3$. As a result, there is no advantage in asymptotic run-time by dividing matrices in this way, although it is a popular and useful way in linear algebra. Strassen's algorithm needs just *seven* multiplications of $n/2 \times n/2$ matrices instead of eight [1].

The core that constitutes his algorithm is in the following. He first found seven matrices, P_1 through P_7 each being able to be obtained by one multiplication of $n/2 \times n/2$ matrices

$$\begin{aligned} P_1 &= A_1(B_3 - B_4) \\ P_2 &= (A_1 + A_2)B_4 \\ P_3 &= (A_3 + A_4)B_1 \\ P_4 &= A_4(-B_1 + B_2) \\ P_5 &= (A_1 + A_4)(B_1 + B_4) \\ P_6 &= (A_2 - A_4)(B_2 + B_4) \\ P_7 &= (-A_1 + A_3)(B_1 + B_3). \end{aligned}$$

Each of the final matrices C_1 through C_4 then can be computed using a combination of P_i's with no multiplications

$$\begin{aligned} C_1 &= -P_2 + P_4 + P_5 + P_6 \\ C_2 &= P_1 + P_2 \\ C_3 &= P_3 + P_4 \\ C_4 &= P_1 - P_3 + P_5 + P_7. \end{aligned}$$

Therefore, the whole process consists of only seven multiplications of $n/2 \times n/2$ matrices. The recursive relation for the total number of scalar multiplications now becomes $T_n = 7T_{n/2}$, which results in $T_n = n^{\log_2 7} = n^{2.81}$. Strassen thus dropped the complexity of matrix multiplication from the order of n^3 to $n^{2.81}$. In total, Strassen's algorithm uses seven multiplications and 18 additions. Some years later Winograd found an algorithm that uses seven multiplications and only 15 additions [7], [2]. Winograd's algorithm reduced the number of additions to 15 by exploiting common subexpressions. However, the time required for the additions is negligible compared to the multiplications, and does not affect the asymptotic complexity of the algorithms when n is large enough.

After Strassen, a large number of mathematicians tried to find better algorithms. Currently, the best is $n^{2.376}$, which was obtained by using a basic trilinear form and the Salem-Spencer Theorem, which uses arithmetic progression [4]. However, it has been proven that when using bilinear combinations of $n/2 \times n/2$ matrices, it is not possible to obtain the product with only six multiplications of $n/2 \times n/2$ matrices [6], [7]; therefore, Strassen's algorithm using seven is optimal. If we divide an $n \times n$ matrix into nine $n/3 \times n/3$ matrices and try the same approach (bilinear combinations) as Strassen's, so far 23 multiplications of $n/3 \times n/3$ matrices is the best, which requires $n^{\log_3 23}$ scalar multiplications in total [8]. Since $n^{\log_3 23} > n^{2.81}$, it is asymptotically not as good as Strassen's original $n/2 \times n/2$ approach. An excellent survey about matrix multiplication was provided by Pan [9].

Strassen's method can be found by naively evaluating all the possible cases exhaustively. Unfortunately, if we attempt to find a comparable solution using an exhaustive search algorithm, it would take around 67 million years on a Pentium IV 2.4 GHz machine. Instead, we used a genetic algorithm for this, and succeeded to cut the run-time down to a few hours. In an extreme case, it took just 10 s to find a solution. To the best of our knowledge, this is the first work that automatically found algorithms comparable to Strassen's. We hereafter describe how we achieved the goal.

II. A Genetic Search

A. Problem Formulation

It is not clear exactly how Strassen discovered the set of P_i matrices, which are the crucial part of his algorithm. For this paper, we assumed each matrix product, P_i $(i = 1, 2, \ldots, 7)$, is written in the following form [10]:

$$\begin{aligned} P_i = {} & (\alpha_{i1}A_1 + \alpha_{i2}A_2 + \alpha_{i3}A_3 + \alpha_{i4}A_4) \\ & \cdot(\beta_{i1}B_1 + \beta_{i2}B_2 + \beta_{i3}B_3 + \beta_{i4}B_4), \ \alpha_{ij}, \beta_{ij} \in \{-1, 0, 1\}. \end{aligned}$$

These types of combinations for P_i are called bilinear combinations. If each of C_1, C_2, C_3, and C_4 is a linear combination of P_1, P_2, ..., P_7, then $n^{2.81}$ scalar multiplications are enough for an $n \times n$ matrix multiplication. Therefore, for $i = 1, \ldots, 7$ and $j = 1, \ldots, 4$, our goal is to find seven P_i's and associated δ_{ji}'s, satisfying

$$C_j = \sum_{i=1}^{7} \delta_{ji} P_i. \tag{1}$$

In this framework, the number of all possible P_i's is $3^4 \times 3^4$. Eliminating the zero matrix and symmetric pairs with reversed signs, the number of unique P_i's is $((3^4 - 1)/2)((3^4 - 1)/2) = 1600$. Therefore, the total number of the candidate solutions for seven P_i's is $\binom{1600}{7} = 5.2566 \times 10^{18}$. Verifying whether a solution satisfies (1) can be done by performing the Gaussian elimination of four 16×8 matrices as will be seen in Section II-D. On a Pentium IV 2.4GHz CPU, it takes 0.0001 s to compute the Gaussian elimination of a 16×8 matrix. Therefore, it would take $(5.2566 \times 10^{18} \times 0.0001 \times 4)/(60 \times 60 \times 24 \times 365) = 6.67 \times 10^7$ years to check all candidate solutions.

B. Genetic Algorithm

A genetic algorithm (GA) is a search method that mimics the process of natural selection in nature. Fig. 1 shows the flow of the GA we used. It is a typical steady-state hybrid GA. In the GA, we first create a fixed number of initial solutions at random, in which nearly all solutions have poor fitness; this set of solutions is called the population. Then we iterate the genetic main loop. Each loop runs as follows. We choose two parent solutions in the population based on their relative fitness. Then the chosen solutions are combined by partly mixing their characteristics to produce a new solution, called an offspring. The offspring is improved using a local optimization algorithm. Then the fitness of this offspring is evaluated. If the offspring satisfies some condition, it replaces one of the solutions in the population. The loop is repeated until the stopping condition is satisfied. For a good introduction to GAs, see [5].

Although GA has a wide space-search capability, it is usually not very powerful for greater than toy-sized problems. Particularly, it is weak in fine-tuning around the local optima. For practical competence, we often need to incorporate local optimization algorithms in the framework of GA. These types of GAs are called hybrid GAs or memetic GAs. Devising a synergetic local optimization algorithm is thus a crucial part in the design of hybrid GAs [3]. In the following sections, we describe each part of the GA in more detail.

C. Encoding

Two key parts of a Strassen-style algorithm are the set of P_i's and the combination of P_i's to produce C_i's. Once an arbitrary set of P_i's is given, it is not difficult to determine if they correctly produce the C_i's using matrix rank and the Gaussian elimination. Thus the GA here focuses on discovering a set of P_i's. In GAs, a solution is represented by a chromosome. Here, a chromosome is an 8×7 matrix; each of the seven columns is a set of eight α_{ij}'s/β_{ij}'s. That is, the ith column corresponds to P_i. As mentioned, the number of all possible

```
create initial population of fixed size;
do ¶
    choose parent1 and parent2 from population;
    offspring = crossover( parent1, parent2);
    local optimization( offspring);
    evaluation( offspring);
    replace(population, offspring);
◇until(stopping condition);
report the best answer;
```

Fig. 1. Steady-state genetic algorithm.

solutions is 5.2566×10^{18}. Due to the huge solution space, a search algorithm can examine only a relatively tiny fraction of the space.

D. Fitness

Our goal is to find a solution that satisfies (1). This means that C_1 through C_4 each can be represented by a linear combination of $\{P_1, P_2, \ldots, P_7\}$. We define fitness as the number of C_i's that can be represented by a combination of P_i's; thus, fitness is an integer value from 0 to 4. For example, if C_2 is a linear combination of P_i's and the others are not, then the fitness of this solution is 1. Therefore, in terms of fitness, our goal is to find a solution to fitness 4.

To evaluate a solution, we expand each P_i into 16 terms as

$$P_i = \sum_{j=1}^{16} \alpha_{i,\lceil j/4 \rceil} \beta_{i,j-1(\bmod 4)+1} A_{\lceil j/4 \rceil} B_{j-1(\bmod 4)+1}. \quad (2)$$

Let $\alpha_{ij}\beta_{ik} = \gamma_{i,(j-1)\times 4+k}$, $\Gamma = [\gamma_{ji}]$, and $\Delta_i = [\delta_{ij}]^T$. Then, P_i can be described as $P_i = \sum_{j=1}^{16} \gamma_{i,j} A_{\lceil j/4 \rceil} B_{j-1(\bmod 4)+1}$. For instance, the vector representation of $C_1 = A_1B_1 + A_2B_2$ becomes $C_1 = [1\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0]^T$. The ith element of the vector represents the term $A_{\lceil i/4 \rceil} B_{i-1(\bmod 4)+1}$. From the equation $C_1 = \sum_{i=1}^{7} \delta_{1i} P_i$, we obtain $\Gamma \cdot \Delta_1 = C_1$. The equation $\Gamma \cdot \Delta_1 = C_1$ is represented by a matrix expression as follows:

$$\begin{pmatrix}
\gamma_{1,1} & \gamma_{2,1} & \gamma_{3,1} & \gamma_{4,1} & \gamma_{5,1} & \gamma_{6,1} & \gamma_{7,1} \\
\gamma_{1,2} & \gamma_{2,2} & \gamma_{3,2} & \gamma_{4,2} & \gamma_{5,2} & \gamma_{6,2} & \gamma_{7,2} \\
\gamma_{1,3} & \gamma_{2,3} & \gamma_{3,3} & \gamma_{4,3} & \gamma_{5,3} & \gamma_{6,3} & \gamma_{7,3} \\
\gamma_{1,4} & \gamma_{2,4} & \gamma_{3,4} & \gamma_{4,4} & \gamma_{5,4} & \gamma_{6,4} & \gamma_{7,4} \\
\gamma_{1,5} & \gamma_{2,5} & \gamma_{3,5} & \gamma_{4,5} & \gamma_{5,5} & \gamma_{6,5} & \gamma_{7,5} \\
\gamma_{1,6} & \gamma_{2,6} & \gamma_{3,6} & \gamma_{4,6} & \gamma_{5,6} & \gamma_{6,6} & \gamma_{7,6} \\
\gamma_{1,7} & \gamma_{2,7} & \gamma_{3,7} & \gamma_{4,7} & \gamma_{5,7} & \gamma_{6,7} & \gamma_{7,7} \\
\gamma_{1,8} & \gamma_{2,8} & \gamma_{3,8} & \gamma_{4,8} & \gamma_{5,8} & \gamma_{6,8} & \gamma_{7,8} \\
\gamma_{1,9} & \gamma_{2,9} & \gamma_{3,9} & \gamma_{4,9} & \gamma_{5,9} & \gamma_{6,9} & \gamma_{7,9} \\
\gamma_{1,10} & \gamma_{2,10} & \gamma_{3,10} & \gamma_{4,10} & \gamma_{5,10} & \gamma_{6,10} & \gamma_{7,10} \\
\gamma_{1,11} & \gamma_{2,11} & \gamma_{3,11} & \gamma_{4,11} & \gamma_{5,11} & \gamma_{6,11} & \gamma_{7,11} \\
\gamma_{1,12} & \gamma_{2,12} & \gamma_{3,12} & \gamma_{4,12} & \gamma_{5,12} & \gamma_{6,12} & \gamma_{7,12} \\
\gamma_{1,13} & \gamma_{2,13} & \gamma_{3,13} & \gamma_{4,13} & \gamma_{5,13} & \gamma_{6,13} & \gamma_{7,13} \\
\gamma_{1,14} & \gamma_{2,14} & \gamma_{3,14} & \gamma_{4,14} & \gamma_{5,14} & \gamma_{6,14} & \gamma_{7,14} \\
\gamma_{1,15} & \gamma_{2,15} & \gamma_{3,15} & \gamma_{4,15} & \gamma_{5,15} & \gamma_{6,15} & \gamma_{7,15} \\
\gamma_{1,16} & \gamma_{2,16} & \gamma_{3,16} & \gamma_{4,16} & \gamma_{5,16} & \gamma_{6,16} & \gamma_{7,16}
\end{pmatrix}
\begin{pmatrix}
\delta_{11} \\ \delta_{12} \\ \delta_{13} \\ \delta_{14} \\ \delta_{15} \\ \delta_{16} \\ \delta_{17}
\end{pmatrix}
=
\begin{pmatrix}
1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0
\end{pmatrix}. \quad (3)$$

We apply the Gaussian elimination on the matrix $[\Gamma\ C_i]$. If the Gaussian elimination of the matrix $[\Gamma\ C_1]$ results in a matrix of the form $\left[\begin{array}{c|c} I_{7\times 7} & \delta_1 \\ O_{9\times 7} & O_{9\times 1} \end{array}\right]$, C_1 is a linear combination of P_i's. Similarly, we check C_2, C_3, and C_4 to see if they are a linear combination of P_i's.

E. Crossover

Crossover is a genetic operator for recombining two parent solutions into a new one. Each parent solution consists of seven vectors of P_1 through P_7; thus, there are 14 vectors in total. The offspring takes seven vectors out of the 14.

For $C_j = \sum_{i=1}^{7} \delta_{ji} P_i$, we define G_j to be $\{P_i | \delta_{ji} \neq 0\}$. G_j can be an empty set. Namely, G_j is the set of P_i's that contribute to the formation of C_j. For example, if $C_1 = P_1 + P_3 + P_4$, then $G_1 = \{P_1, P_3, P_4\}$, which implies that C_1 can be expressed with P_1, P_3 and P_4.

We choose one G_i at random among the nonempty G_i's of *parent1* and *parent2*, and we add a P_i that belongs to the chosen G_i to the offspring, unless the P_i already exists in the offspring. Then, we choose another G_i and repeat the same process until the offspring has seven P_i's. If no such P_i remains before we have seven P_i's, we randomly generate new P_i's and add them to the offspring until the offspring has seven P_i's.

F. Local Optimization

After a new offspring is generated by crossover, the GA improves the solution using a local optimization algorithm. Each solution has a position in the problem space. If we plot the fitness of each solution to its corresponding position, we can construct a fitness landscape for the problem. Some of the solutions are of high quality and play the role of attractors. When we have a moderate or low-quality solution, we can climb to one of the nearby attractors. Although this is not always beneficial, it saves considerable run-time in the GA. Considering the huge problem space and limited time budget, we cannot help but ignore the most seemingly unpromising areas of the space. The local optimization consists of the following two parts.

1) *Pursuing Linear Independence:* If P_i's are linearly dependent, the solution contains at least one unnecessary P_i. Removing unnecessary P_i's is the goal of the first local optimization. If P_i's are linearly independent, the rank of the matrix $\Gamma = [\gamma_{ji}]$ is 7. In the case of a rank value lower than 7, we find the nonessential P_i by means of the Gaussian elimination, and change an α_{ij} or β_{ij} of the P_i. We then compute the rank again and repeat this process until the rank of the matrix Γ becomes 7.

2) *Checking All the Cases:* To improve the fitness of a solution, we check the fitness after replacing one of the P_i's with each of the possible 1600 cases. First, we select a nonessential P_i which does not belong to G_1, G_2, G_3, or G_4. If such a P_i does not exist, an arbitrary P_i is selected. We replace it with another among the 1600 cases and compute the fitness. If a better case is found, P_i is replaced. Otherwise, P_i remains unchanged.

TABLE I
NUMBER OF SOLUTIONS

Group Index	Solutions Found	Distinct Solutions	Lower Bound of the Number of Solutions
Group 1	648	32	32
Group 2	891	128	128
Group 3	522	32	32
Group 4	61	35	64
Group 5	30	19	64
Group 6	3	3	32
Group 7	33	30	128
Group 8	149	57	64
Group 9	46	36	64

TABLE II
REPRESENTATIVE SOLUTIONS IN EACH GROUP

Group 1 (Strassen's Solution)	Group 2	Group 3
$P_1 = A_1(B_3 - B_4)$	$P_1 = A_1(B_3 - B_4)$	$P_1 = A_1(B_3 - B_4)$
$P_2 = (A_1 + A_2)B_4$	$P_2 = (A_1 + A_2)B_4$	$P_2 = (A_1 + A_2)B_4$
$P_3 = (A_3 + A_4)B_1$	$P_3 = A_4(B_2 + B_4)$	$P_3 = (A_3 - A_4)B_2$
$P_4 = A_4(-B_1 + B_2)$	$P_4 = A_3(B_1 + B_3)$	$P_4 = A_3(B_1 + B_2)$
$P_5 = (A_1 + A_4)(B_1 + B_4)$	$P_5 = (A_2 + A_4)(B_1 - B_2)$	$P_5 = (A_1 + A_2 + A_3 + A_4)(B_1 + B_2 + B_3 + B_4)$
$P_6 = (A_2 - A_4)(B_2 + B_4)$	$P_6 = (A_1 + A_2 + A_4)(B_1 + B_4)$	$P_6 = (A_1 + A_2 - A_3 + A_4)(B_1 + B_2 + B_3 - B_4)$
$P_7 = (-A_1 + A_3)(B_1 + B_3)$	$P_7 = (A_1 + A_2 + A_3 + A_4)B_1$	$P_7 = (A_1 - A_2 + A_3 - A_4)(B_1 - B_2 + B_3 - B_4)$
$C_1 = -P_2 + P_4 + P_5 + P_6$	$C_1 = -P_2 - P_3 - P_5 + P_6$	$C_1 = -P_1 - P_3 + 0.5P_6 + 0.5P_7$
$C_2 = P_1 + P_2$	$C_2 = P_1 + P_2$	$C_2 = P_1 + P_2$
$C_3 = P_3 + P_4$	$C_3 = P_2 + P_3 - P_6 + P_7$	$C_3 = -P_3 + P_4$
$C_4 = P_1 - P_3 + P_5 + P_7$	$C_4 = -P_2 + P_4 + P_6 - P_7$	$C_4 = -P_2 - P_4 + 0.5P_5 - 0.5P_6$
Group 4	Group 5	Group 6
$P_1 = A_4(-B_1 + B_2 - B_3 + B_4)$	$P_1 = (A_1 + A_2)(B_1 + B_3)$	$P_1 = (A_1 + A_2)(B_3 + B_4)$
$P_2 = A_1(B_1 - B_2 - B_3 + B_4)$	$P_2 = (A_1 + A_2 - A_3 + A_4)(B_2 - B_3)$	$P_2 = (A_1 - A_2)(B_3 - B_4)$
$P_3 = (A_1 + A_4)(B_1 - B_2 + B_3 + B_4)$	$P_3 = (-A_3 + A_4)(B_1 - B_3)$	$P_3 = (A_2 - A_4)(B_1 - B_2 - B_3 + B_4)$
$P_4 = (A_1 - A_3)B_3$	$P_4 = (A_1 + A_2 - A_3 - A_4)(B_1 + B_2)$	$P_4 = (A_2 + A_4)(B_1 + B_2 + B_3 + B_4)$
$P_5 = (A_3 + A_4)(B_1 + B_3)$	$P_5 = (A_1 - A_2 - A_3 + A_4)(B_1 - B_2)$	$P_5 = (A_1 - A_4)(B_1 + B_4)$
$P_6 = (A_1 + A_2)(B_2 - B_4)$	$P_6 = A_2(B_1 - B_2 + B_3 - B_4)$	$P_6 = (A_1 + A_2 - A_3 - A_4)(B_1 - B_3)$
$P_7 = (A_2 - A_4)B_4$	$P_7 = A_4(B_1 + B_2 - B_3 - B_4)$	$P_7 = (A_1 - A_2 + A_3 - A_4)(B_1 + B_3)$
$C_1 = 0.5P_1 + 0.5P_2 + 0.5P_3 + P_6 + P_7$	$C_1 = 0.5P_1 + 0.5P_2 - 0.5P_3 + 0.5P_5$	$C_1 = -0.5P_1 + 0.5P_2 - 0.5P_3 + 0.5P_4 + P_5$
$C_2 = 0.5P_1 - 0.5P_2 + 0.5P_3 + P_7$	$C_2 = 0.5P_1 - 0.5P_2 + 0.5P_3 - 0.5P_5 - P_6$	$C_2 = 0.5P_1 + 0.5P_2$
$C_3 = 0.5P_1 + 0.5P_2 - 0.5P_3 + P_4 + P_5$	$C_3 = 0.5P_1 + 0.5P_2 - 0.5P_3 - 0.5P_4$	$C_3 = -0.5P_1 - 0.5P_2 + 0.5P_3 + 0.5P_4 - 0.5P_6 + 0.5P_7$
$C_4 = 0.5P_1 - 0.5P_2 + 0.5P_3 - P_4$	$C_4 = 0.5P_1 + 0.5P_2 + 0.5P_3 - 0.5P_4 - P_7$	$C_4 = 0.5P_1 - 0.5P_2 - P_5 + 0.5P_6 + 0.5P_7$
Group 7	Group 8	Group 9 (Winograd's Solution)
$P_1 = A_1B_1$	$P_1 = A_1B_1$	$P_1 = A_1B_1$
$P_2 = A_2B_2$	$P_2 = A_2B_2$	$P_2 = A_2B_2$
$P_3 = A_3(B_1 + B_2 + B_3 + B_4)$	$P_3 = A_3(B_3 + B_4)$	$P_3 = A_3(B_1 + B_2 + B_3 + B_4)$
$P_4 = (A_2 + A_4)(B_1 + B_2 - B_3 - B_4)$	$P_4 = (A_2 + A_4)(B_1 + B_2 + B_3 + B_4)$	$P_4 = (A_2 + A_4)(B_3 + B_4)$
$P_5 = (A_3 - A_4)(B_2 + B_4)$	$P_5 = (A_3 - A_4)B_4$	$P_5 = (A_3 - A_4)(B_2 + B_4)$
$P_6 = (A_2 - A_3 + A_4)(B_1 - B_2 - B_3 - B_4)$	$P_6 = (A_2 - A_3 + A_4)(B_1 + B_3 + B_4)$	$P_6 = (A_2 - A_3 + A_4)(B_2 - B_2 - B_3 - B_4)$
$P_7 = (A_1 - A_2 + A_3 - A_4)(B_1 - B_4)$	$P_7 = (A_1 - A_2 + A_3 - A_4)(B_1 + B_3)$	$P_7 = (A_1 - A_2 + A_3 - A_4)B_3$
$C_1 = P_1 + P_2$	$C_1 = P_1 + P_2$	$C_1 = P_1 + P_2$
$C_2 = P_1 - P_2 + P_3 - P_6 - P_7$	$C_2 = -P_1 + P_5 + P_6 + P_7$	$C_2 = -P_2 + P_5 + P_6 + P_7$
$C_3 = -P_2 + 0.5P_3 + 0.5P_4 + 0.5P_6$	$C_3 = -P_2 - P_3 + P_4 - P_6$	$C_3 = -P_2 + P_3 - P_4 + P_6$
$C_4 = P_2 + 0.5P_3 - 0.5P_4 - P_5 + 0.5P_6$	$C_4 = P_3 - P_5$	$C_4 = P_2 + P_4 - P_5 - P_6$

III. EXPERIMENTAL RESULTS

In our GA, we set the population size to 15. The GA stops if a valid solution appears or 1000 iterations has passed. Many more solutions were found than we expected. So far, we have discovered 2383 solutions that have the same computational complexity as Strassen's. Removing duplicate solutions, 372 distinct ones remained. Among them, 32 solutions turned out to be similar to Strassen's in the way they are constructed. We divided the solutions into nine groups based on the numbers of nonzero entries α_{ij} and β_{ij} in the P_i's. For example, group 1 has [3 3 3 3 4 4 4] nonzero elements. It is notable that groups 3, 4, 5, 6, and 7 include 0.5 as the coefficients of P_i's, which we did not expect to encounter. (Although they introduce some scalar multiplications in C_i's, it does not affect the asymptotic complexity of the algorithm.)

Since there is no special order in C_i's, we can make different solutions with the same structure using the symmetry of C_i's and changing the signs of a_i's and b_i's in P_i's [10]. Using

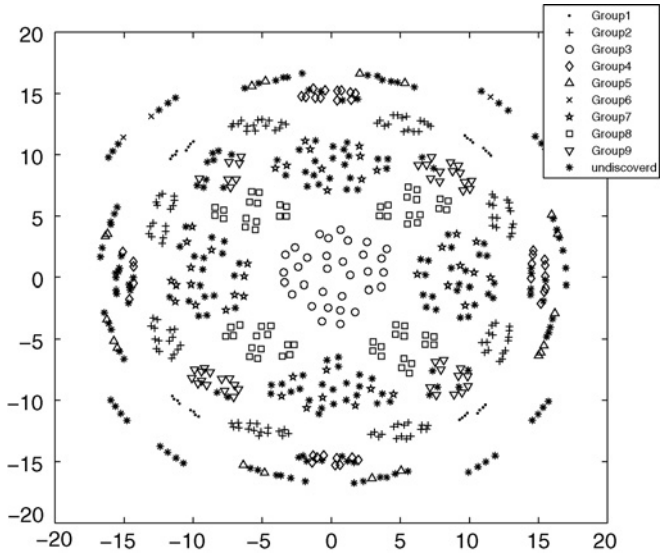

Fig. 2. Sammon mapping of the solutions.

this approach, we noticed that 236 solutions were undiscovered by the GA in groups 4, 5, 6, 7, 8, and 9. We thus have 608 distinct solutions in total. Table II-F shows the numbers of all the solutions found, distinct solutions, and a lower bound of the number of solutions, respectively, in each group. One can observe that there are many solutions that the GA did not discover. Table II-F shows a representative solution in each group.

Fig. 2 shows the relative distribution of all of the 608 solutions in the problem space, which is plotted by Sammon mapping [11]. In Sammon mapping, we define the distance between two solutions as follows. First, we compare the differences of all the pairs of P_i's and pick the pair with the minimum difference. Then we exclude those two P_i's and find the minimum difference pair among the remaining. We repeat this process until we match all the seven pairs. The distance between the two solutions is defined as the sum of the differences of the seven pairs. Based on the definition, the distances between all the pairs of the solutions are computed and they are mapped into a 2-D box so that the distortion of the distances are minimized. The solutions labeled "undiscovered" are those that the GA could not find but we can induce from the discovered solutions.

While their corresponding groups are not specified, most of them are clear because they are strongly clustered. Although they were mapped in two dimensions, the figure is informative. First, solutions of the same group formed clusters in the figure. For example, the solutions in group 7 can be found in four clusters in the figure, and those in group 1, the group of Strassen's, can be found in four other clusters. The clusters seem to provide good evidence that the grouping (into nine groups) is reasonable. Second, the solutions in each group show strong symmetry, which we suspect reflects the symmetry of the ways in which they are constructed.

IV. Discussion

According to our GA search, it turned out that the Strassen's algorithm is just one of many solutions with the same computational complexity. In fact, at least 608 distinct solutions exist either discovered by the GA or induced from the discovered solutions. We do not know whether there are more solutions that have the same complexity as Strassen's. Considering that group 6 has only 3 discovered solutions in which at least 64 solutions exist, more groups may be waiting to be discovered.

A bilinear system with $n/3 \times n/3$ matrices is another challenge. So far Laderman [8] found, from his mathematical insight, a solution requiring only 23 $n/3 \times n/3$ matrix multiplications, though still not as good as the 7 multiplications required in the $n/2 \times n/2$ system. If we want to find a better result than the $n/2 \times n/2$ case, we need 21 multiplications or fewer. The problem space of the $n/3 \times n/3$ system, however, is far larger than the $n/2 \times n/2$ system, and therefore, it would be a good problem to test the extreme performance of a search algorithm.

Acknowledgment

The authors would like to thank the Information and Communication Technology at Seoul National University for providing research facilities for this paper.

References

[1] V. Strassen, "Gaussian elimination is not optimal," *Numer. Math.*, vol. 13, no. 4, pp. 354–356, Sep. 1969.
[2] P. C. Fischer and R. L. Probert, "Efficient procedures for using matrix algorithms," in *Proc. 2nd Colloquium Automata, Languages Programming*, Jul. 1974, pp. 413–427.
[3] T. Bui and B. Moon, "Genetic algorithm and graph partitioning," *IEEE Trans. Comput.*, vol. 45, no. 7, pp. 841–855, Jul. 1996.
[4] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic programming," *J. Symbolic Comput.*, vol. 9, no. 3, pp. 251–280, Mar. 1990.
[5] D. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Reading, MA: Addison-Wesley, 1989.
[6] J. Hopcroft and L. Kerr, "Some techniques for proving certain simple programs optimal," in *Proc. IEEE 10th Ann. Symp. Switching Automata*, Oct. 1969, pp. 36–45.
[7] S. Winograd, "On multiplication of 2 × 2 matrices," *Linear Algebra Applicat.* vol. 4, pp. 381–388, 1971.
[8] 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, Jan. 1976.
[9] V. Pan, "How to multiply matrices faster," *Lecture Notes in Computer Science*, vol. 179. Berlin, Germany: Springer-Verlag, 1984, pp. 33–42.
[10] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA: MIT Press, ch. 28, 2001, pp. 735–741.
[11] J. Sammon, "A nonlinear mapping for data structure analysis," *IEEE Trans. Comput.*, vol. C-18, no. 5, pp. 401–409, May 1969.

Seunghyun Oh received the B.S. degree in electrical engineering from Seoul National University, Seoul, Korea, in 2007. He is currently pursuing the Ph.D. degree in electrical engineering at the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor.

His current research interests include integrated circuit design for adaptable wireless communication systems for both high-performance and energy constrained applications.

Mr. Oh received a fellowship under the Korea Foundation for Advanced Studies.

Byung-Ro Moon (M'09) received the B.S. degree in computer science from Seoul National University in 1985, the M.S. degree in computer science from the Korea Advanced Institute of Science and Technology in 1987, and the Ph.D. degree from Pennsylvania State University, University Park in 1994.

He joined Seoul National University in September 1997, where he is currently the Professor and Director of the Optimization and Investment Engineering Laboratory, the School of Computer Science and Engineering. From 1987 to 1991, he was an Associate Research Staff Member at the Central Laboratory, LG Electronics Co., Ltd., Seoul, Korea. From November 1994 to 1995, he was a Post-Doctoral Scholar in the VLSI Computer Aided Design Laboratory, University of California, Los Angeles. From 1996 to 1997, he was a Principal Research Staff Member at the DT Research Center, LG Semicon Co., Ltd., Seoul, Korea. He led the Korean International Olympiad on Informatics team as Principal Advisor. He has worked on investment optimization in the context of optimization techniques over the last ten years, and he is currently the Chief Executive Officer of Optus Inc., a company specializing in algorithm trading. He has published over 120 technical papers and four books. His major research interests include the theory and application of optimization methodologies, including evolutionary computation, algorithm design and analysis, and optimization modeling, among others. He has applied optimization methodologies to portfolio generation, investment strategy, data mining, personalized marketing, campaign optimization, search, context recognition, graph partitioning, scheduling, and combinatorial problems in general. He is a Research Member of the Institute of Finance and Economics, and he has provided over 50 invited talks on various topics, including computer algorithms, stock investment, genetic algorithms, optimization, customer relationship management, and education.

Dr. Moon received the Korean Government Overseas Scholarship from 1991 to 1994. He received the Best Teaching Award, the Research Excellency Award, and the Teaching Excellency Award from Seoul National University in 2001, 2007, and 2008, respectively. He is a Member of the Association for Computing Machinery, and the Council of Authors at the International Society for Genetic and Evolutionary Computation.