Evolutionary Search for Matrix Multiplication Algorithms

John F. Kolen & Phillip Bruce
Institute of Human and Machine Cognition
University of West Florida
Pensacola, Florida 32501
{jkolen,pbruce}@ai.uwf.edu

Abstract

This paper addresses the problem of algorithm discovery, via evolutionary search, in the context of matrix multiplication. The traditional multiplication algorithm requires $O(n^3)$ multiplications for square matrices of order n. Strassen (Strassen 1969) discovered a recursive matrix multiplication algorithm requiring only seven multiplications at each level, resulting in a runtime of $O(n^{lg7})$, or $O(n^{2.81})$. We have been able to replicate this discovery using evolutionary search (Fogel 1995). The paper presents the representational schema, evaluation criteria, and evolution mechanisms employed during search. The most crucial decision was removing the determination of coefficients used to combine the product terms in the final addition steps from the search space and calculating them directly from the specified multiplications. Extending this methodology from 2×2 submatrices to algorithms using 3×3 decompositions is also discussed.

Introduction

Matrix multiplication is an operation that underlies many computer application areas such as simulation, data analysis, and signal processing. The traditional algorithm ($z_{ij} = \sum x_{ik} y_{kj}$) for multiplying two square matrices of order n requires $O(n^3)$ multiplications. The runtime complexity of the standard algorithm is not minimal. By casting the problem recursively, Strassen (Strassen 1969) has shown that it is possible to reduce the number of multiplications to $O(n^{lg7})$, or $O(n^{2.81})$. This feat was accomplished by performing only seven, instead of the requisite eight, multiplications on each recursion.

An asymptotically optimal matrix multiplication algorithm has yet to be discovered. The best non-commutative algorithm to date requires $O(n^{2.376})$ running time (Coppersmith & Winograd 1987). Even the lower bounds for this problem's complexity has yet to extend past $O(n^2)$ (Winograd 1971). We have embarked on a research project that will employ AI techniques to discover efficient matrix multiplication algorithms. Recall that Strassen's algorithm to multiply two $n \times n$ matrices, $Z = XY$, is a collections of seven multiplications (q_i) that are combined in four weighted summations (z_l). The multiplications are of the form ($q_i = (\sum a_{ij} x_j \times \sum b_{ik} y_k)$ where x_j and y_k are the submatrices of the original matrices, and a_{ik} and b_{ik} are their inclusion coefficients. These inclusion coefficients were either negative one, zero, or one. The summations are merely weighted sums of the q_i terms ($\sum c_{li} q_i$). Thus, our search for algorithms takes us to the space of the inclusion and combining coefficients. The size of the search space was reduced when we discovered that the combining coefficients could be calculated from a given set of inclusion coefficients. We selected an evolutionary approach (Fogel 1995) to explore the space of 2×2 algorithms. The current experiments will validate our approach, as we later look for completely novel algorithms involving larger decomposition matrices.

The remainder of the paper describes our efforts to date. First, we address representational issues that impact our search. The specifics of the evolutionary algorithm that explores the representational space are then described. The results of our experiments on the 2×2 case are presented. Finally, we address the implications of this work and what steps are necessary to progress to effectively searching 3×3 decompositions for novel algorithms.

Algorithm Representation

Given a dimension, δ, and two $\delta \times \delta$ matrices, X and Y, we wish to find an algorithm to compute $Z \equiv X \times Y$ using σ multiplication steps. This algorithm can be applied to square matrices with dimension δ^k via recursive decomposition. Each matrix will be decomposed into δ^2 submatrices and the matrix operations will manipulate these submatrices instead of scalars. For example, one can construct a recursive algorithm for $2^k \times 2^k$ matrices from the formula for multiplying two 2×2 matrices that requires eight multiplications.

$$\begin{aligned} Z_{11} &= X_{11}Y_{11} + X_{12}Y_{21} \\ Z_{12} &= X_{11}Y_{12} + X_{12}Y_{22} \\ Z_{21} &= X_{21}Y_{21} + X_{22}Y_{21} \\ Z_{22} &= X_{21}Y_{22} + X_{22}Y_{22} \end{aligned} \quad (1)$$

In Equation 1, the $X_{..}$ terms are the square submatrices of X. These equations can be recursively applied on the multiplication of the submatrices as well. The recursion will bottom out upon reaching 2×2 matrices. At this point, the

$$\begin{matrix} & X_{11}Y_{11} & X_{11}Y_{12} & X_{11}Y_{21} & X_{11}Y_{22} & X_{12}Y_{11} & \dots & X_{21}Y_{11} & \dots & X_{11}Y_{11} & X_{11}Y_{12} & X_{11}Y_{21} & X_{22}Y_{22} & \\ [& 1 & 0 & 0 & 0 & 0 & \dots & -1 & \dots & 0 & 0 & 0 & 0 &] \end{matrix}$$

Figure 1: The vector representation for $X_{11}Y_{11} - X_{21}Y_{11}$ is [1 0 0 0 0 0 0 0 −1 0 0 0 0 0 0 0].

equations will be operating on scalars instead of submatrices. The runtime, $T(n)$, for multiplying two $n \times n$ matrices is $T(n) = 8T(n/2) + O(n^2)$. Analysis of the recurrence equation shows that this algorithm will run in $O(n^3)$ time.[1]

In this paper, we shall be concerned with the special case of $\delta = 2$ and $\sigma = 7$. Strassen (Strassen 1969) addressed this case and was able to rewrite these equations in a sum of products form (Equation 2).

$$\begin{aligned} q_1 &= (X_{11} + X_{22}) * (Y_{11} + Y_{22}) \\ q_2 &= (X_{21} + X_{22}) * Y_{11} \\ q_3 &= X_{11} * (Y_{12} - Y_{22}) \\ q_4 &= X_{22} * (Y_{21} - Y_{11}) \\ q_5 &= (X_{11} + X_{12}) * Y_{22} \\ q_6 &= (X_{21} - X_{11}) * (Y_{11} + Y_{12}) \\ q_7 &= (X_{12} - X_{22}) * (Y_{21} + Y_{22}) \\ Z_{11} &= q_1 + q_4 - q_5 + q_7 \\ Z_{12} &= q_3 + q_5 \\ Z_{21} &= q_2 + q_4 \\ Z_{22} &= q_1 + q_3 - q_2 + q_6 \end{aligned} \quad (2)$$

Note that in all cases, the multiplication operators take a subset of values from X and multiply them with a subset from Y. This constraint arises from the observation that no Z submatrix formula contains quadratic terms solely from one multiplicand or the other (i.e. no $X_{11}X_{22}$ terms appear). In addition, each term present in the product of sums (POS), $q_.$, and sum of products (SOP), $Z_{..}$, is multiplied by a coefficient of either -1 or 1. The representation scheme described below will preserve these two constraints.

We can define the search space that satisfies these constraints mathematically. Consider the $\delta \times \delta$ matrices A^r, B^r and C^r for $r \varepsilon \{1, \dots, \sigma\}$ such that the following equation holds for all $n, m \in \{1, \dots, \delta\}$:

$$\sum_{p=1}^{\delta} X_{np}Y_{pm} = \sum_{r=1}^{\sigma} \left(\sum_{i,j=1}^{\delta} A_{ij}^r X_{ij} \right) \left(\sum_{k,l=1}^{\delta} B_{kl}^r Y_{kl} \right) C_{mn}^r \quad (3)$$

The A^r and B^r matrices identify which terms of X and Y to include in the calculation of the left and right side of the product operation. The C^r matrix controls the summation of the products. In practice, the elements of these matrices are taken from the same ring defining X and Y. Strassen's algorithm, for example, has A^r and B^r with elements taken from the set $\{-1, 0, 1\}$. Multiplications by scalars is asymptotically equivalent to adding two matrices and thus are ignored in the multiplication count. Originally, we limited the elements of A^r, B^r, and C^r to the set $\{-1, 0, 1\}$.

Initial experiments within the search space over A^r, B^r, and C^r, as defined above, yielded dismal results. We found that maintaining the relationship between the inclusion and combination coefficients was difficult, if not impossible. This difficulty arises from the coupling of these parameters. In the explanation that follows, we show that it is possible to eliminate the C^r matrices from the search space by directly calculating it from the contents of A^r and B^r. As we shall see below, the same calculation turns out to yield an excellent objective function. Consider then an alternate formulation of Equation 3:

$$Z_{nm} = \sum_{p=1}^{\delta} X_{np}Y_{pm} = \sum_{r=1}^{\sigma} \left(\sum_{i,j,k,l=1}^{\delta} D_{ijkl}^r X_{ij} Y_{kl} \right) C_{mn}^r \quad (4)$$

The new term, D_{ijkl}^r, the outer product of A^r and B^r, identifies the contribution of a *pair* of terms drawn from X and Y. This selection process can be abstracted by considering the group of elements $\{X_{ij}Y_{kl}\}_{i,j,k,l \varepsilon \{1,\dots,\delta\}}$ with the addition operation to be a vector space over the field of rational coefficients. Under this interpretation Equation 4 states that for any n and m a certain vector $\vec{s}_{mn}$, whose elements are either zero or one, is equal to a linear combination of σ vectors. Figure illustrates the vector representation schema with the representation for the product term $X_{11}Y_{11} - X_{21}Y_{11}$. Defining the matrix D to be the $\delta^4 \times \sigma$ matrix whose columns correspond to the vectors A_{ijkl}^r and defining $\vec{c}_{mn}$ to be the σ−dimensional vector whose elements are C_{mn}^r, we may then express Equation 5 as the following linear system:

$$\vec{s}_{mn} = A\vec{c}_{mn} \quad (5)$$

Or expressed for all $m, n \in \{1, \dots, \delta\}$:

$$S = AC \quad (6)$$

where S is a matrix depending only on δ and whose elements are each members of the set $\{0, 1\}$. If we then specify values for A we may attempt to solve Equation 6 for C. Figure illustrates the matrices constructed for the products found in Strassen's original algorithm (Equation 2). If a solution is found, then we have we have A_{ij}^r and B_{kl}^r satisfying Equation 3 and a matrix multiplication algorithm with the desired properties.

We placed additional restrictions on the representations to contract the search space. For instance, it is clear that the matrices A^r and B^r must have at least one non-zero entry in order to yield a solution to Equation 3. It is also true that in Strassen's algorithm, these matrices never have more than two entries so we might benefit from placing an upper bound on the number of non-zero elements. While we considered utilizing this constraint early in our work, we found that it was unnecessary to constrain the search space in this manner. Furthermore, multiplying either of these matrices by -1 has no effect in that this negation can be instantiated during

[1] For simplicity of analysis, we assume that dimension of the matrices is a power of two.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} C$$

Figure 2: The system of equations corresponding to Strassen's original algorithm (Equation 2). The sixteen rows correspond to the sixteen possible pairings of submatrices from X and Y. The four columns of the matrix on the right hand side of the equal sign correspond to the four submatrices of the resulting matrix. The seven columns of the other matrix correspond to the seven products. C is a 7×4 matrix of unknown combination coefficients.

the calculation C^r. Restricting the first non-zero entry to one further reduces the search space. Our experiments below reflect the application of these last two reduction techniques.

Searching for Algorithms

Given the algorithmic space described above, we now address the problem of search. We selected an evolutionary approach as our search tool (Fogel 1995). The justification for this choice is two–fold. First, we felt that a population–based approach would work well for this task. Second, we wanted to operate directly upon the values defining our algorithms. This latter decision allows us to avoid problems associated with bit encodings used in genetic algorithms.

An individual is consists of the sets containing all the A^r and B^r matrices. The population contains a large number of such individuals. A reproductive generation consists of the following steps. First, pairs of individuals from the population are selected for reproduction according to their fitness. The newborn individuals undergo mutation and are then ranked with the older population by the fitness function and those with high fitness (see below) scores are removed from the population. Generations are repeated until either a satisfactory algorithm is found, or the number of generations reaches a predefined stopping value.

The fitness function must somehow estimate how close the individual is to actually computing the matrix multiplication. Using the vector space model described in the previous section, we define an objective function based on the nearness of the matrix S of Equation 6 to the range of the transformation A. If the system in Equation 6 is solvable, then the goal has been reached. In the likely event that the matrix is unsolvable, we turn to an objective function f that measures how close it is to a solution. A norm is then estimated for the matrix $AC - S$. More precisely, we define it to be the function

$$f(A) = \min_X \|AX - S\| \tag{7}$$

where the norm is taken to be the Euclidean norm. Note that such a function is 0 if and only if A corresponds to a solution to the problem. The lower the score its value, the closer the individual is to being a solution.

If we assume that A is not the zero matrix, the degenerate case, then because f is strictly convex, bounded below, and not bounded above, any critical point is a minimizer. Because the gradient of f is $A^TAX - A^TS$, we must therefore solve the system:

$$A^TAX = A^TS \tag{8}$$

for X in order to find a minimum of f. Thus the evaluation of f (Equation 9) for any given X consists of a few basic matrix calculations and is reasonably fast.

$$f(A) = \|A(A^TA)^{-1}A^TS - S\| \tag{9}$$

The objective function, f, described above captures the main features of our goal states. We found, however, that the search mechanism needed additional information in order to discover solutions to this problem. The fitness function is the objective function adjusted in two ways. First, if an individual's matrix is sparse, then the score is increased. Individuals with zero rows in their A and B matrices will also incur a similar penalty. Both of these augmentations help the evolutionary mechanism quickly escape the flat regions of the objective function that are often found with randomly generated individuals. Note that fit individuals have low fitness scores, zero indicating a working algorithm.

While the fitness function described above provides score for a given individual, we allow the actual fitness score for the individual changes over time. That is, each generation an individual survives, its fitness is increased by a constant amount. This serves to keep potentially good starting points in the population, but eventually removes them, as fitness increases, to make room for younger individuals.

Having defined a fitness function, we then define operations for the creation of new individuals. The simplest operation is the creation of an individual whose chromosomes have A^r and B^r matrices with randomly chosen entries. We call this operation *immigration*. It serves to increase the diversity of the population and is used to initialize the population.

The reproduction of two individuals, or cross-breeding, consists of the creation of two new individuals whose chromosomes are derived from the parent using uniform crossover (Ackley 1987; Syswerda 1989). Consider the chromosome to be the serialization of A^r's and B^r's. The children are constructed by walking down the two chromosomes of the parents and copy the current genome, an A^r or a B^r, from one parent to either child and then copying the corresponding genome from the other parent to the other child. Probability of a child receiving a genome from the

first parent is 0.5. Thus, the children receive a complementary pairing of their parent's genetic material while maintaining integrity of the the low-level sums (the subsets from selected from matrices A and B).

For any given individual P, *mutation* is the creation of a new individual O whose chromosomes are identical to those of P, except for the probabilistic changing of each element in the A^r and B^r matrices to one of the two other possible values. The probability of any element changing is such that the expected number of changed elements is fairly small, though it increases with the age of P in order to promote diversity in the population. The restrictions on the structure of A^r and B^r, first non-zero element is positive and at least one non-zero element, described above are maintained during this process. The second type of mutation focuses the attention of this mechanism on problematic areas of the chromosome.

Thus, our evolutionary algorithm for searching the space matrix multiplication methods consists of the following steps. First, a population of immigrants is created. New members of the population are created using cross-over and mutation. Cross-over parents are selected according to the fitness score. The less fit individuals are then removed from the population.

Results

Our evolutionary search was performed under the following conditions. The base population size was three hundred individuals. Cross-breeding yielded three hundred new members from 150 pairings. An additional three hundred individuals were created from simple mutation of existing individuals. On each generation, ten new immigrants were added to increase genetic diversity of the population. The base mutation probability, that is, the probability of a single gene mutating was 0.07. This probability increased by 0.07 for every generation an individual remained in the population. A penalty of 0.2 was incurred to the fitness score for each generation during the life span of the individual. The maximum non-zero coefficients in the sum of POS was four. The number of non-zero coefficients in the POS was not limited.

We employed a restart method. After 550 iterations, the search algorithm empties the population and starts with a new set of randomly generated individuals. After 24 restarts, a solution was found after 91 generations. It consisted of seven multiplications and eighteen additions and is of the same complexity as Strassen's decomposition (Equation 2).

$$\begin{aligned}
q_1 &= X_{21} * (Y_{12} + Y_{22}) \\
q_2 &= (X_{11} - X_{12}) * Y_{11} \\
q_3 &= (X_{12} - X_{21}) * (Y_{11} - Y_{22}) \\
q_4 &= (X_{21} - X_{22}) * Y_{22} \\
q_5 &= (X_{12} - X_{22}) * (Y_{21} + Y_{22}) \\
q_6 &= (X_{11} - X_{21}) * (Y_{11} + Y_{12}) \\
q_7 &= X_{12} * (Y_{11} + Y_{21}) \\
Z_{11} &= q_2 + q_7 \\
Z_{12} &= -q_1 - q_2 - q_3 + q_6 \\
Z_{21} &= -q_3 + q_4 - q_5 + q_7 \\
Z_{22} &= q_1 - q_4
\end{aligned} \tag{10}$$

Conclusion

The discovery of a recursive matrix multiplication algorithm was described above. Since the original discovery, other schemes have presented themselves, all within similar time frame and solution complexity. Our automated search method differs from that used by Strassen. Computer search, however, is not novel in this area. Brent (Brent 1970) used a least-squares minimization technique on a function whose minima correspond with matrix-multiplication algorithms to rediscover the $\delta = 2$ solution. Our search, however, differs from that of Brent in that the function to be minimized is simpler, and the optimization is done using evolutionary search.

Since Winograd (Winograd 1971) demonstrated that for the 2×2 case, seven multiplications are necessary, we will have to turn to larger decompositions if we are to discover better algorithms. We currently are examining the 3×3 case with twenty three multiplications. We have been able to discover solutions when our initial populations contain mutations of the a target solution. Success has been achieved with populations with 120 mutations of a working solution, however, random initial populations have yet to discover a working solution. The main difficulty here is that the number of possible term one side or the other of a product has increased from four to nine. Without restricting the relationships between coefficients, the number of possible elements in a multiplicand (the subset of X or Y submatrices) has increased from 3^4 to 3^9. Given that the search space for the entire tableau contains 7^{2k} items for the 2×2 case, the m^{2k}, where $7 < m < 3^3 = 27$, algorithms in the new space demands attention, otherwise we are drawn into an exponential quagmire. Our current efforts, thus, are focused on restricting the search space. In addition, we are examining additional population management techniques, such as having the age of an individual mediate mutation. Currently, the best 3×3 algorithm requires 23 multiplications (Laderman 1976). It is unknown at this time whether or not 23 multiplications is necessary for this case. It is our hope that we will be able to find a 3×3 algorithm requiring fewer multiplications.

Even though we have only matched the complexity of currently known algorithms, the results of the search described above can be used in other ways. We are currently exploring the idea that one can dynamically reduce the number of multiplications for recursive matrix multiplication by selecting an decomposition appropriate for the given matrices. Consider the case when the submatrices of X are equal. Then q_2 through q_6 of Strassen's algorithm need not be calculated as the left hand side is the zero matrix. Yet, if $X_{11} = X_{21} = -X_{12} = -X_{22}$, five of the seven multiplications of the second algorithm disappear. The count of elements equal, and negated, between submatrices of X and Y can be used as a heuristic for selecting an algorithm with the fewest multiplications.

Acknowledgments

Dr. Kolen received support from the Office of Naval Research, grant number N000014–990983, National Aeronautical and Space Administration, grant number NCC2–1026

Prime, and the South Florida Water Management District, contract C–10805 during the course of this research. Mr. Bruce received support from the Office of Naval Research, grant number N000014–990983.

References

Ackley, D. H. 1987. *A Connectionist Machine for Genetic Hillclimbing*. Boston, MA: Kluwer.

Brent, R. P. 1970. Algorithms for matrix multiplication. Technical Report CS 157, Computer Science Department, Stanford University, Stanford, CA.

Coppersmith, D., and Winograd, S. 1987. Matrix multiplication via arithmetic progressions. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, 1–6.

Fogel, D. B. 1995. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. Piscataway, NJ: IEEE Press.

Laderman, J. D. 1976. A noncommutative algorithm for multiplying 3×3 matrices using 23 multiplications. *Bulletin of the American Mathematial Society* 82(1):126–128.

Strassen, V. 1969. Gaussian elimination is not optimal. *Numberische Mathematik* 14(3):354–356.

Syswerda, G. 1989. Uniform crossover in genetic algorithms. In Schaffer, J. D., ed., *Proceedings of the Third International Conference on Genetic Algorithms (Fairfax, VA, June 1989)*, 2–9. San Matao, CA: Morgan Kaufmann.

Winograd, S. 1971. On multiplication of 2×2 matrices. *Linear Algebra and Applications* 4:381–388.