모든 n 차 정사각 불리언 행렬 쌍에 대한 벡터 기반의 곱셈 알고리즘

한재일*
*국민대학교 컴퓨터학부
e-mail : jhan@kookmin.ac.kr

An Algorithm for the Multiplication of all pairs of $n \times n$ Boolean Matrices using Vectors

Jae-Il Han*
*School of Computer Science, Kookmin University

요 약

일반 행렬이나 불리언 행렬의 연산에 대한 많은 연구가 있다. 대부분의 연구는 두 행렬의 효율적 곱셈을 다루고 있으며 하드웨어나 소프트웨어적 응용에 적합한 다양한 알고리즘을 제시하였다. 모든 행렬 쌍의 곱셈에 대한 연구는 NP-완전 계산 복잡도와 이러한 곱셈을 요구하는 응용의 희소성으로 인해 관심밖에 있었으며 최근에야 원소가 불리언 값을 가지는 n 차 정사각 불리언 행렬을 대상으로 기초적인 연구 결과를 보이고 있다. 본 논문은 모든 n 차 정사각 불리언 행렬 사이의 곱셈을 보다 효율적으로 할 수 있는 벡터 기반 불리언 행렬 곱셈 이론과 이를 바탕으로 설계한 알고리즘 그리고 실행 결과에 대하여 논한다.

1. 서론

행렬은 여러 분야에서 다양하게 사용되고 있으며 행렬의 연산에 대한 많은 연구가 수행되었다[1-15]. 대부분의 연구는 일반 행렬이나 특수한 구조를 갖는 두 행렬 사이의 곱셈에 초점을 두고 있으며, 이러한 응용에 적합한 다양한 알고리즘이 제시 되었다[6-15]. 모든 행렬 사이의 곱셈에 대한 연구는 NP-완전 계산 복잡도와 이러한 곱셈을 요구하는 응용의 희소성으로 인해 관심 밖에 있었으며, 최근에야 원소가 불리언 값을 가지는 n 차 정사각 불리언 행렬을 대상으로 한 기초적인 연구가 보이고 있다[17, 18].

불리언 행렬은 여러 분야에서 유용하게 사용되고 있다[11-15]. 불리언 행렬의 곱셈을 다루는 많은 연구가 수행 되었으며[6-10] 두개의 n 차 정사각 행렬 사이의 최적화된 곱셈을 위한 이론과 알고리즘은 [11]에 제시되어 있다. 두 불리언 행렬 사이의 곱셈을 초점으로 하는 다른 응용과 달리 D-클래스 계산 문제는 n 차 정사각 불리언 행렬의 전체 집합을 대상으로 이 집합에서 조합할 수 있는 모든 두 행렬 사이의 곱셈을 기본적으로 요구하며 이를 기반으로 모든 가능한 조합의 세 행렬 사이 곱셈을 요구한다. 이러한 D-클래스의 계산은 NP-완전 문제이며 이로 인해 현재 4 × 4 이하 크기의 불리언 행렬에 대한 결과만이 알려져 있다.

본 논문은 모든 가능한 조합의 두 n 차 정사각 불리언 행렬 사이의 곱셈을 벡터를 이용하여 보다 효율적으로 계산 할 수 있는 수학적 이론을 제시한다. 또한 제시한 이론을 바탕으로 설계한 개선된 알고리즘과 실행 결과에 대하여 논한다. 본 논문의 구성은 다음과 같다. 2 장은 관련연구에 대하여 기술하며 3 장은 모든 두 $n \times n$ 불리언 행렬 사이의 효율적인 곱셈을 위한 수학적 이론을 제시한다. 4 장은 수학적 이론을 바탕으로 설계한 알고리즘과 여러 실행결과를 기술하며 5 장은 결론 및 향후 연구방향에 대하여 논한다.

2. 관련 연구

일반 행렬이나 불리언 행렬의 연산에 대하여 많은 연구가 수행 되었으나, 대부분의 연구는 일반 행렬

이나 특수한 구조를 갖는 두 행렬 사이의 곱셈에 초점을 두고 하드웨어나 소프트웨어적으로 효율적 곱셈을 할 수 있는 문제를 다루고 있다[1-5]. n 차 정사각 불리언 행렬의 곱셈에 대한 연구도 대부분 단지 두 n 차 정사각 불리언 행렬의 곱셈에 대하여 수행 되었다[6-15]. [10]은 두 $n \times n$ 불리언 행렬의 최적 곱셈을 위한 이론과 알고리즘을 제시하고 있으며, 알고리즘은 두 $n \times n$ 불리언 행렬 A x B 연산에서 앞의 불리언 행렬 A 를 불리언 행렬 B 의 행에 대한 OR-연산 지시자로 사용하여 곱셈 결과를 얻는다.

모든 행렬 사이의 곱셈에 대한 연구는 일반 행렬이나 불리언 행렬에 상관없이 NP-완전 계산 복잡도와 이러한 곱셈을 요구하는 응용의 희소성으로 필요성이 시급하지 않아 관심 밖에 있다. D-클래스는 모든 $n \times n$ 불리언 행렬의 전체 집합에서 특정 관계(relation)에 따라 동등(equivalent)한 관계에 있는 행렬의 집합으로 구성되며 다음과 같이 정의된다[16]. F={0, 1}일때

$$M_n(F) = \{ A : A \text{ is an } n \times n \text{ matrix whose element is in } F \}$$
$$D_A = \{ B \in M_n(F) : \exists C, X, Y, U, V \in M_n(F) \text{ such that } AX = C,\ CY = A,\ UC = B,\ VB = C \}$$
$$D\text{-class} = \{ D_A : A \in M_n(F) \}$$

따라서 D-클래스 계산을 위한 핵심 문제는 다른 응용과 달리 $n \times n$ 불리언 행렬의 전체 집합을 대상으로 이 집합에서 조합할 수 있는 모든 두 행렬 사이의 곱셈을 효율적으로 할 수 있는 알고리즘 설계이다.

최근 D-클래스를 계산하기 위하여 $n \times n$ 불리언 행렬의 전체 집합을 대상으로 이 집합에서 조합할 수 있는 모든 두 행렬 사이의 곱셈에 대한 기초적인 연구가 수행 되었다[17. 18]. 그러나 [17, 18]에서 제시한 알고리즘은 여러 개선노력에도 불구하고 실행시간과 메모리 등의 문제가 크게 개선되지 못하고 있다. 이 연구 결과는 모든 $n \times n$ 불리언 행렬 쌍의 곱셈에서 실행시간과 메모리 사용량 등에 대한 개선이 단순한 불리언 행렬 곱셈의 산술계산이나 약간의 코드 개선으로 이루어지기 힘들며, 곱셈 연산의 근본적 문제를 다룰 수 있는 수학적 특성과 이론의 정립 그리고 이를 바탕으로 한 알고리즘 설계가 필요하다는 것을 보여 준다.

3. 벡터 기반의 불리언 행렬 곱셈

3 장은 모든 두 $n \times n$ 불리언 행렬 사이의 곱셈을 n 차원 벡터와 $n \times n$ 불리언 행렬의 곱셈으로 대체하여 효율적 계산을 가능하게 하는 수학적 이론을 논한다.

A 가 $n \times n$ 불리언 행렬일 때 A_i와 A^i는 각각 A 행렬의 i 번째 행과 열을 의미하며, 다음 정의를 사용한다.

$$v^c = \begin{pmatrix} e_0 \\ e_1 \\ \vdots \\ e_{n-1} \end{pmatrix} \quad where \quad v = (e_0\ e_1 \cdots e_{n-1})$$

$$V = \{ (e_0\ e_1 \cdots e_{n-1}) \mid e_i \in F \text{ for } 0 \le i \le n-1 \}$$
$$V^c = \{ (e_0\ e_1 \cdots e_{n-1})^c \mid e_i \in F \text{ for } 0 \le i \le n-1 \}$$
$$V^n = \{ [v_0^c\ v_1^c \cdots v_{n-1}^c] \mid v_i \in V \text{ for } 0 \le i \le n-1 \}$$
$$Av^c = (A_0 v^c\ A_1 v^c \cdots A_{n-1} v^c)^c \quad where\ v \in V$$

$M_n(F)$ 에 속한 모든 두 $n \times n$ 불리언 행렬의 곱셈은 행렬 사이의 직접 곱셈 대신 $n \times n$ 행렬과 n 차원 벡터의 곱셈으로 같은 결과를 얻을 수 있다. A 를 $M_n(F)$ 에 속한 임의의 $n \times n$ 불리언 행렬이라 하자. A 에 $M_n(F)$ 의 모든 불리언 행렬을 곱하여 얻는 불리언 행렬의 집합은 다음과 같다.

$$P_A = \{ AX \mid X \in M_n(F) \}$$

A 를 n 개의 행 벡터, X 를 n 개의 열 벡터로 나타내면

$$AX = \begin{bmatrix} A_0 \\ A_1 \\ \vdots \\ A_{n-1} \end{bmatrix} \begin{bmatrix} X^0\ X^1 \cdots X^{n-1} \end{bmatrix} = \begin{bmatrix} E^0\ E^1 \cdots E^{n-1} \end{bmatrix}$$

이 되고 임의의 $E^i (0 \le i \le n-1)$는 X 의 i 번째 열벡터 X^i 에 대해

$$AX^i = \begin{bmatrix} A_0 \\ A_1 \\ \vdots \\ A_{n-1} \end{bmatrix} X^i = \begin{pmatrix} A_0 X^i \\ A_1 X^i \\ \vdots \\ A_{n-1} X^i \end{pmatrix} = E^i \ \text{ for } 0 \le i \le n-1$$

인 관계를 얻는다. A 를 $M_n(F)$ 의 모든 불리언 행렬에 곱한다면 X 에 $M_n(F)$ 의 모든 불리언 행렬이 나타나게 된다. 따라서 X 의 각 n 차원 열벡터 X^i 에도 서로 독립적으로 2^n 개의 n 차원 벡터가 나타나게 되며 T_A 를

$$T_A = \{ AX^i \mid X^i \in V^c \text{ and } X^i \text{ is a column of } X \}$$

으로 정의하면 다음과 같은 관계를 얻을 수 있다.

$$\{ AX \mid X \in M_n(F) \} = \underbrace{T_A \times T_A \times \cdots \times T_A}_{n \text{ times}} = T_A^n$$

다음은 위 사실로부터 유도한 정리이다.

[정리 1] $M_n(F)$ 에 속한 임의의 $n \times n$ 불리언 행렬 A 에 대해 M_A 와 T_A 를 다음과 같이 정의할 때 $\mathrm{M}_A \subset T_A^n$ 이다.

$$\mathrm{M}_A = \{ AX \mid X \in M_n(F) \}$$
$$T_A = \{ (A_0 v^c\ A_1 v^c \cdots A_{n-1} v^c)^c \mid v \in V \}$$

(증명) $P \in \mathrm{M}_A$ 이라 하자. 이 경우 P 의 각 열 P^i 는 어떤 $v_k \in V(0 \le k \le 2^n - 1)$ 에 대해 $P^i = A(v_k)^c$ 가 된다. T_A 는 불리언 행렬 A 에 n 개의 원소를 가지는 모든 벡터 v^c 를 곱하여 얻은 열벡터의 전체 집합이므로 0 과 $n-1$ 사이의 모든 i 에 대해 $P^i \in T_A$ 이며, 따라서 $P \in T_A^n$ 이다.

[정리 2] A 를 $M_n(F)$ 에 속한 임의의 $n \times n$ 불리언 행렬이라 하고 M_A 와 T_A 를 다음과 같이 정의할 때 $T_A^n \subset \mathrm{M}_A$ 이다.

$$\mathrm{M}_A = \{\, AX \mid X \in M_n(F) \,\}$$
$$T_A = \{\, (A_0 v^c \;\; A_1 v^c \;\; \cdots \;\; A_{n-1} v^c)^c \mid v \in V \,\}$$

(증명) $P \in T_A^n$ 이라 하자. 만약 $P \notin \mathrm{M}_A$ 이라면 M_A 에 속하는 모든 불리언 행렬 D 에 대하여 $P \neq D$ 이고 따라서 P 의 어떤 벡터열 P^i 가 모든 행렬 D 의 i 번째 벡터열 D^i 와 같지 않다. $M_n(F)$ 는 모든 $n \times n$ 불리언 행렬의 집합이므로

$M_n(F) = \{\, (v_0^c \; v_1^c \cdots v_{n-1}^c) \mid v_i \in V \; for \; 0 \le i \le n-1 \,\}$ 이고

$X \in M_n(F)$ 이므로

$$\mathrm{M}_A = \{\, (A_0 v_0^c \;\; A_1 v_0^c \;\; \cdots \;\; A_{n-1} v_0^c)^c \cdots (A_0 v_{n-1}^c \;\; A_1 v_{n-1}^c \;\; \cdots \;\; A_{n-1} v_{n-1}^c)^c \mid v_i \in V \; for \; 0 \le i \le n-1 \,\}$$

이다. 가정에 의해 $P \in T_A^n$ 이므로 V 의 어떤 벡터 $v_k \; (0 \le k \le 2^n - 1)$ 에 대하여 $P^i = (A_0 v_k^c \;\; A_1 v_k^c \;\; \cdots \;\; A_{n-1} v_k^c)^c$ 가 되므로 P 가 M_A 에 속하게 되어 모순이 된다.

[정리 1]과 [정리 2]에 의해 $M_n(F)$ 에 속한 어떤 불리언 행렬이 주어졌을 때 A 에 $M_n(F)$ 의 2^{n^2} 개 불리언 행렬을 곱하는 대신 2^n 개의 열벡터를 곱하여 나오는 열벡터의 집합을 얻는다. 이 집합에 있는 열벡터를 n 번 조합하여 얻을 수 있는 모든 행렬의 집합은 A 에 $M_n(F)$ 의 모든 행렬을 곱하여 얻는 행렬의 집합과 동일하다.

위의 정의에서 보듯 D-클래스는 D_A 를 계산하여 얻으며, D_A 정의에서 $AX = C$ 와 $VB = C$ 를 $UC = B$ 와 $CY = A$ 에 각각 대입하면 다음 D_A 정의를 얻을 수 있다.

$$D_A = \{\, B \in M_n(F) : \; \exists X, Y, U, V \in M_n(F) \; such \; that \; UAX = B, VBY = A \,\}$$

이 정의에서 UAX 와 VBY 계산은 불리언 행렬 A 와 B 에 $M_n(F)$ 의 각 불리언 행렬을 곱하고 그 결과와 $M_n(F)$ 의 각 불리언 행렬에 대한 곱셈을 반복하여 요구한다. 이 경우에도 행렬 사이의 직접 곱셈 대신 위의 정리를 적용하여 행렬과 벡터 사이의 곱셈을 이용하여 다음 장에서 보듯 UAX 나 VBY 를 보다 효율적으로 계산할 수 있다

V : a set of all n-element vectors whose element belongs to F
SV_1, SV_2 : a set of vectors
CVS : a class of a set of vectors
UAXMap: a set of (A, A's CVS class) pairs where $A \in M_n(F)$

for each boolean matrix A in $M_n(F)$
  $SV_1 = \varnothing$
  for each n-element vector v_1 in V
    compute an Av_1 vector
    insert the vector into a set SV_1

  for each matrix M synthesized from SV_1
    $SV_2 = \varnothing$
    for each n-element vector v_2 in V
      compute an v_2M vector
      insert the vector into a set SV_2
    insert the set SV_2 into CVS
  insert a pair (A, CVS) into *UAXMap*

(그림 1) 알고리즘 개요

4. 알고리즘 및 실행 결과

(그림 1)은 3 장의 정리를 이용하여 작성된 UAX 계산 알고리즘의 개요를 보이고 있다. 알고리즘은 먼저 $M_n(F)$ 에 속한 각 불리언 행렬에 대하여 n 개의 원소를 가지는 모든 2^n 개의 벡터를 곱한다. 불리언 행렬과 각 벡터의 곱셈으로부터 얻은 벡터는 현 불리언 행렬에 대한 집합 SV_1 에 삽입한다. 모든 벡터와의 곱셈이 종료되면 AX 에 대한 열벡터 집합 SV_1 을 얻는다. SV_1 의 열벡터를 n 번 조합하여 얻는 각 행렬에 대해 모든 n 차원 행벡터를 곱하여 벡터집합 SV_2 를 계산하고 현재 선택된 바깥 순환문의 행렬 A 의 CVS 클래스에 SV_2 를 삽입한다. 모든 조합된 행렬에 대하여 벡터집합 계산이 완료되면 CVS 클래스는 행렬 A 에 대해 UAX 식으로부터 얻을 수 있는 모든 행렬의 집합을 나타내게 되며 (A, CVS) 짝을 *UAXMap* 에 삽입한다. 알고리즘이 종료하면 $M_n(F)$ 의 모든 행렬에 대한 UAX 계산 결과가 *UAXMap* 에 존재한다.

알고리즘은 행렬의 직접 곱셈에서 필요하지 않은 행렬 조합 단계가 추가 되지만 <표 1>은 알고리즘의 실행시간이 행렬의 직접 곱셈보다 대폭 개선된 결과를 보이고 있다. 알고리즘은 Java 언어로 구현하였으며 펜티엄 3.0GHz, 512MB RAM, Windows XP/Pro 환경에서 실행되었다. 불리언 행렬과 벡터의 곱셈은 비트 사이의 논리 연산으로 수행한다.

<표 1> 실행시간 비교

종류 / 행렬크기	기존 알고리즘	벡터기반 알고리즘
2 x 2	0.016 초	0.015 초
3 x 3	13.469 초	0.016 초
4 x 4	2294초	75.094 초
5 x 5	553650000초 이상 (6407일 이상)	21350.097 초

5. 결론 및 향후 연구방향

행렬의 연산에 대하여 많은 연구가 수행되었으나 대부분 일반 행렬이나 특수한 구조를 갖는 두 행렬 사이의 곱셈에 초점을 두고 있다. 모든 행렬 쌍의 곱셈에 대한 연구는 NP-완전 계산 복잡도와 응용의 희소성으로 인해 관심 밖에 있었으며, 최근에야 n 차 정사각 불리언 행렬을 대상으로 한 기초적인 연구가 보이고 있다[17-18]. 그러나 제시된 알고리즘은 실행시간이나 메모리 등의 문제가 크게 개선되지 못하고 있다. 본 논문은 모든 행렬 쌍에 대한 곱셈 연산의 효율성을 높이기 위해 벡터를 사용한 행렬 곱셈의 수학적 특성과 이론을 제시하였다. 또한 제시한 이론을 바탕으로 알고리즘을 설계 및 구현하였으며 실행시간이 상당 부분 개선되었음을 보였다.

모든 n 차 정사각 불리언 행렬 쌍에 대한 곱셈은 NP-완전 계산 복잡도를 가지므로 근본적인 해결방안은 찾기 쉽지 않다. 그러나 D-클래스 계산과 같은 미해결 문제에 대한 연구가 활성화 될 수 있도록 더 많은 정보를 주기 위해서 모든 n 차 정사각 불리언 행렬 쌍의 효율적 곱셈 알고리즘에 대한 연구가 필요하다. 이러한 알고리즘의 개선은 수학적 이론과 증명, 알고리즘 최적화 연구를 필요로 한다. 본 논문의 알고리즘은 논문에서 제시한 벡터 기반 곱셈 이론을 바탕으로 설계할 수 있는 하나의 알고리즘일 뿐이다. 앞으로 이 이론을 바탕으로 한 최적화된 알고리즘의 설계와 구현에 대한 연구가 필요하다. 또한 실행시간 개선을 위한 다른 수학적 이론, 병렬 및 분산 컴퓨팅의 적용 등에 대한 연구도 필요하다.

참고문헌

[1] Wilkinson, B., and Allen, M., Parallel Programming with MPI, Prentice Hall, 1999

[2] Golub, G. H., and Van Loan, C. F., Matrix Computation, The Johns Hopkins' University Press, 1983

[3] Butler, K. K., "On (0, 1)-matrix semigroups," Semigroup Forum 3, 1971, pp. 74-79

[4] Leighton, F. T., and Parallel Algorithms and Architectures: Arrays · Trees · Hypercubes, Morgan Kaufmann, 1992

[5] Fox, G., Otto, S., and Hey, A., "Matrix algorithms on a hypercube I: Matrix Multiplication," Parallel Computing 3, 1987, pp. 17-31

[6] Atkinson, D. M., Santoro, N., and Urrutia, J., "On the integer complexity of Boolean matrix multiplication," ACM SIGACT News, Vol. 18 No. 1, pp. 53, 1986

[7] Macii, E., "A Discussion of Explicit Methods for Transitive Closure Computation Based on Matrix Multiplication," The 29th Asilom Conference on Signals, Systems and Computers, Vol. 2, pp. 799-801, 1995

[8] Comstock, D. R., "A note on multiplying Boolean matrices II," CACM, Vol. 7 No. 1, pp. 13, 1964

[9] Booth, K. S., "Boolean matrix multiplication using only $O(n^{\log_2 7}\log n)$ bit operations," ACM SIGACT News, Vol. 9 No. 3, pp. 23, 1977

[10] Angluin, D., "The four Russians'algorithm for boolean matrix multiplication is optimal in its class," ACM SIGACT News, Vol. 8 No. 1, pp. 29-33, 1976

[11] Lee, L., "Fast context-free grammar parsing require fast Boolean matrix multiplication," JACM, Vol. 49 No. 1, pp. 1-15, 2002

[12] Yi, X., et al., "Fast Encryption for Multimedia," IEEE Transactions on Consumer Electronics, Vol. 47, No. 1, pp. 101-107, 2001

[13] Martin, D. F., "A Boolean matrix method for the computation of linear precedence functions," CACM, Vol. 15 No. 6, pp. 448-454,1972

[14] Nakamura, Y., and Yoshimura T., "A partitioning-based logic optimization method for large scale circuits with Boolean matrx," Proceedings of the 32nd ACM/IEEE conference on Design automation, pp. 653-657, 1995

[15] Pratt, V. R., "The Power of Negative Thinking in Multiplying Boolean matrices,"Proceedigs of the annual ACM symposium on Theory of computing, pp. 80-83, 1974

[16] Rim, D. S., and Kim, J. B., "Tables of D-Classes in the semigroup B_n of the binary relations on a set X with n-elements,"Bull. Korea Math. Soc., Vol. 20 No. 1, pp. 9-13, 1983

[17] 신철규, 한재일, "공유 메모리 기반의 고성능 D-클래스 계산 병렬 알고리즘", 한국컴퓨터종합학술대회 논문집, 제 32 권, 제 1 호, pp. 10-12, 2005

[18] 신철규, 한재일, "그리드 컴퓨팅 환경에서의 D-클래스 계산 병렬 알고리즘", 한국정보처리학회 춘계학술대회 논문집, 제 12 권, 제 1 호, pp. 929-932, 2005