모든 $l \times n$, $n \times m$, $m \times k$ 불리언 행렬의 곱셈 시간 개선에 관한 연구

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

A Study on the Improvement of Execution Time for the Multiplication of All $l \times n$, $n \times m$ and $m \times k$ Boolean Matrices

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

요 약

대부분의 불리언 행렬에 대한 연구는 두 불리언 행렬의 곱셈에 초점을 두고 있으며 모든 불리언 행렬을 대상으로 한 곱셈에 대한 연구는 최근에야 극히 소수의 연구결과가 보이고 있다. 이 연구들은 모든 불리언 행렬 사이의 곱셈 실행시간을 개선시켰으나 연속된 세 개의 모든 $l \times n$, $n \times m$, $m \times k$ 불리언 행렬에 대한 곱셈은 아직 많은 개선이 필요하다. 본 논문은 모든 $l \times n$, $n \times m$, $m \times k$ 불리언 행렬의 곱셈 실행시간을 보다 개선할 수 있는 이론을 제시하고 이를 적용한 불리언 행렬 연속곱셈의 실행결과에 대하여 논한다.

1. 서론

불리언 행렬은 원소가 0(거짓)이나 1(참) 값을 갖는 행렬이다. 불리언 행렬에 대한 대부분의 응용은 두 불리언 행렬의 효율적인 곱셈에 초점을 두고 있으며 많은 연구를 통해 여러 응용에 적합한 다양한 알고리즘이 제시되었다[1-6]. 모든 불리언 행렬 사이의 곱셈은 D-클래스[10] 계산 등에서 요구되고 있으나 이에 대한 연구는 NP-완전 계산복잡도와 효율적 곱셈의 어려움, 상대적으로 적은 응용 분야 등으로 인해 관심 밖에 있다. D-클래스 계산은 모든 두 $n \times n$ 불리언 행렬 사이의 곱셈에 더해 모든 세 $n \times n$ 불리언 행렬 사이의 연속곱셈까지 요구하는 계산의 어려움으로 인해 D-클래스에 대한 극히 제한된 결과만이 알려져 있다.

최근 모든 불리언 행렬 사이의 곱셈에 대한 소수의 연구결과[11-15]가 보이고 있으나 실행시간 등에서 아직 많은 개선이 필요하다. 본 논문은 연속된 세 개의 모든 $l \times n$, $n \times m$, $m \times k$ 불리언 행렬에 대한 곱셈 실행시간을 보다 개선할 수 있는 이론을 정립한 후 이를 적용한 불리언 행렬 연속곱셈의 실행결과에 대하여 기술한다. 본 논문의 구성은 다음과 같다. 2장은 기존에 제시된 불리언 행렬 곱셈에 대한 연구와 문제점을 논하고, 본 논문에서 사용할 용어 및 기호를 정의한다. 3장은 모든 $l \times n$, $n \times m$, $m \times k$ 불리언 행렬사이의 연속 곱셈을 보다 개선할 수 있는 이론을 기술하며, 4장은 이를 적용한 불리언 행렬 연속곱셈의 실행결과에 대하여 논한다. 5장은 결론 및 향후 연구방향에 대하여 기술한다.

2. 관련연구 및 문제점

위에서 언급하였듯이 불리언 행렬에 대한 연구는 대부분 두 불리언 행렬의 효율적인 곱셈에 초점을 두고 있으며[1-9], 극히 소수의 연구[11-15]만이 모든 불리언 행렬 사이의 곱셈을 다루고 있다. 두 불리언 행

렬의 곱셈에 대한 연구를 살펴보면 행을 이용한 곱셈 알고리즘 중에서는 [5]이, 비트 기반 연산을 이용한 알고리즘 중에서는 [6]가 현재 최적의 알고리즘으로 나타나고 있다. 그러나 이 알고리즘들은 $n \times n$ 불리언 행렬을 대상으로 단지 두개의 불리언 행렬 곱셈만을 다루고 있으며, 행렬 곱셈을 수행하기 전에 주어진 특정 행렬에 대해서 행 OR-연산이나 비트 연산에 필요한 정보를 미리 계산하여 저장할 것을 요구한다.

하나의 $n \times n$ 불리언 행렬과 모든 $n \times n$ 불리언 행렬의 곱셈을 하는 경우 위의 두 알고리즘 중 어떤 알고리즘이 사용되는가에 상관없이 2^{n^2}번의 두 불리언 행렬 곱셈이 요구되며 곱셈 결과로 나오는 불리언 행렬을 저장할 때 최악의 경우 2^{n^2}에 비례하는 메모리 공간이 필요하다. 따라서 하나의 불리언 행렬과 모든 불리언 행렬의 곱셈에 두 불리언 행렬의 최적 곱셈 알고리즘을 그대로 적용하여 각각의 두 불리언 행렬 곱셈을 하는 경우 효율적인 곱셈이 어렵다. 또한 D-클래스 계산과 같이 모든 $n \times n$ 불리언 행렬 사이의 곱셈을 수행해야 하는 경우 위의 두 알고리즘은 두 불리언 행렬 곱셈이 수행되기 전에 요구되는 정보를 생성하기 위하여 각 불리언 행렬마다 [5]은 최악의 경우 $O(2^n)$, [6]는 $O(n^2 \log n)$의 계산 시간이 추가적으로 필요하다. 따라서 [5, 6]의 알고리즘은 모든 불리언 행렬을 대상으로 한 곱셈에 사용하기 어렵다.

모든 불리언 행렬의 곱셈에 대한 연구 중 [14, 15]는 순환문 개선에 초점을 두었으나 메모리 공간이나 실행시간 등의 개선정도가 매우 미약하다. 반면 행렬 연산을 벡터 기반으로 계산할 것을 제안한 [11-13]은 동시에 메모리 공간과 실행시간을 개선하였으나, 모든 불리언 행렬의 곱셈은 근본적으로 NP-완전 계산복잡도를 가지므로 보다 큰 크기의 행렬에 적용하려면 더 많은 개선이 필요하다.

본 논문은 용어와 기호를 다음과 같이 정의한다. 임의의 $n \times m$ 불리언 행렬 A가 주어지고 $F = \{0, 1\}$ 이라 할 때, A_i 와 A^i 는 각각 A 행렬의 i 행과 열을 의미한다. A^T는 A의 전치(transpose)행렬이며, m 차원 벡터 v 는

$v = (b_0 b_1 \cdots b_{m-1}),\ b_i \in F,\ 0 \le i \le m-1$

으로 정의하고 $n \times m$ 불리언 행렬의 행에 대응하여 행벡터로 부르며 $1 \times m$ 불리언 행렬과 같다. v^T는 v 의 열과 행 번호를 바꾼 $m \times 1$ 불리언 행렬로서

$$v^T = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{m-1} \end{pmatrix} \quad where \ \ v = (b_0 b_1 \cdots b_{m-1})$$

으로 정의되며 $m \times n$ 불리언 행렬의 열에 대응하여 열벡터로 부른다. V 는 모든 m 차원 행벡터의 집합이며 $(V)_k$ 는 행벡터를 k 개 조합하여 생성한 모든 $k \times m$ 불리언 행렬의 집합이다. V^T 는 모든 m 차원 열벡터의 집합이며 $(V^T)^k$ 는 열벡터를 k 번 조합하여 생성한 모든 $m \times k$ 불리언 행렬의 집합이다.

$V = \{\ (b_0 b_1 \cdots b_{m-1}) \mid b_i \in F \text{for } 0 \le i \le m-1\ \}$

$$(V)_k = \{ \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_k \end{pmatrix} \mid v_i \in V \text{for } 0 \le i \le k-1 \}$$

$V^T = \{\, v^T \mid v \in V \}$

$(V^T)^k = \{\ (v_0^T v_1^T \cdots v_{k-1}^T) \mid v_i \in V, 0 \le i \le k-1\ \}$

$n \times m$ 불리언 행렬 A 와 m 차원 열벡터 v^T 의 곱셈은 n 차원 열벡터를, m 차원 행벡터 v 와 $m \times n$ 불리언 행렬 B 의 곱셈은 n 차원 행벡터를 생성한다.

$$Av^T = \begin{pmatrix} A_0 v^T \\ A_1 v^T \\ \vdots \\ A_{n-1} v^T \end{pmatrix} \quad where \ v \in V$$

$vB = (vB^0\, vB^1 \cdots vB^{n-1})\ where\ v \in V$

$M_n^m(F)$는 모든 $n \times m$ 불리언 행렬의 집합을 정의한다.

3. 연속된 세 개의 모든 불리언 행렬 곱셈

본 장은 연속된 세 개의 모든 $l \times n$, $n \times m$, $m \times k$ 불리언 행렬 곱셈을 중간 결과로 나오는 벡터 집합을 이용하여 보다 효율적으로 계산할 수 있는 이론을 기술한다.

[정리 1] $M_n^m(F)$에 속한 임의의 $n \times m$ 불리언 행렬 A에 대해 V, R_A, C_A를

$V = \{\ (b_0 b_1 \cdots b_{m-1}) \mid b_i \in F \ \text{ for } 0 \le i \le m-1\ \}$

$R_A = \{\ AB \mid B \in M_m^k(F)\ \}$,

$C_A = \{\ (A_0 v^T\ A_1 v^T \cdots A_{n-1} v^T)^T \mid v \in V\ \}$

으로 정의할 때 $R_A = (C_A)^k$ 이다.

[정리 1]은 주어진 $n \times m$ 불리언 행렬에 모든 $m \times k$ 불리언 행렬을 곱하여 얻는 불리언 행렬의 집합이 $n \times m$ 불리언 행렬에 모든 m 차원 열벡터를 곱하여 얻는 열벡터들을 모든 가능한 k 번의 조합으로 얻는 불리언 행렬 집합과 같다는 것을 보인다[12]. [정리 2]는 모든 $l \times n$ 불리언 행렬을 하나의 $n \times m$ 불리언 행렬에 곱하여 얻는 불리언 행렬의 집합이 모든 n 차원 행벡터를 $n \times m$ 불리언 행렬에 곱하여 얻는 행벡터들을 모든 가능한 k 번의 조합으로 얻는 불리언 행렬 집합과 같다는 것을 보인다[13].

[정리 2] $M_n^m(F)$에 속한 임의의 $n \times m$ 불리언 행렬 A에 대해 V, L_A, T_A를

$V = \{ (b_0 b_1 \cdots b_{n-1}) \mid b_i \in F \text{ for } 0 \le i \le n-1 \}$

$L_A = \{ BA \mid B \in M_l^n(F) \}$,

$T_A = \{ (vA^0\ vA^1 \cdots vA^{m-1}) \mid v \in V \}$

로 정의할 때 $L_A = (T_A)_l$ 이다.

[정리 3]은 $n \times m$ 불리언 행렬에 모든 $l \times n$, $m \times k$ 불리언 행렬을 직접 곱하여 얻은 결과가 $n \times m$ 불리언 행렬에 모든 m 차원 벡터와 n 차원 벡터를 차례로 곱하여 얻은 결과와 같다는 것을 보인다[11].

[정리 3] A를 $M_n^m(F)$에 속한 임의의 $n \times m$ 불리언 행렬. Z를 $M_n^k(F)$에 속한 임의의 $n \times k$ 불리언 행렬이라 할 때 S_{AX}, S_{UAX}, V, T_Z를

$S_{AX} = \{ AX \mid X \in M_m^k(F) \}$

$S_{UAX} = \{ UAX \mid U \in M_l^n(F), X \in M_m^k(F) \}$,

$V = \{ (b_0 b_1 \cdots b_{n-1}) \mid b_i \in F \text{ for } 0 \le i \le n-1 \}$

$T_Z = \{ (vZ^0\ vZ^1 \cdots vZ^{k-1}) \mid v \in V \}$

로 정의하면 $S_{UAX} = \bigcup_{Z \in S_{AX}} (T_Z)_l$ 이다.

[정리 1-3]에 의하여 주어진 $n \times m$ 불리언 행렬과 모든 $l \times n$, $m \times k$ 불리언 행렬 사이의 곱셈을 행렬 사이의 연산대신 행렬과 벡터의 연산으로 수행할 수 있다[11].

[정리 4] A, B를 $M_n^m(F)$에 속한 임의의 두 불리언 행렬에 대해

$S_A = \{ AX \mid X \in M_m^k(F) \}$,

$S_B = \{ BY \mid Y \in M_m^k(F) \}$,

$T_A = \{ UAX \mid U \in M_l^n(F), X \in M_m^k(F) \}$,

$T_B = \{ VBY \mid V \in M_l^n(F), Y \in M_m^k(F) \}$

로 정의할 때 $S_A = S_B$이면 $T_A = T_B$이다.

(증명) $T_A = \{ UAX \mid U \in M_l^n(F), X \in M_m^k(F) \}$

$= \{ UW \mid U \in M_l^n(F), W \in S_A \}$

$S_A = S_B$ 이므로

$T_A = \{ UW \mid U \in M_l^n(F), W \in S_B \}$

$= \{ UBY \mid U \in M_l^n(F), Y \in M_m^k(F) \}$

$= T_B$ ■

[정리 4]는 임의의 두 $n \times m$ 불리언 행렬 A, B에 모든 $l \times n$ 불리언 행렬을 곱하여 얻는 행렬 집합이 같으면, A, B에 모든 $l \times n$, $m \times k$ 불리언 행렬을 곱하여 얻는 행렬 집합도 같다는 것을 보인다. 따라서 $M_n^m(F)$의 불리언 행렬에 모든 $l \times n$ 불리언 행렬을 곱하여 얻는 행렬 집합이 이미 존재하는 경우 모든 $l \times n$, $m \times k$ 불리언 행렬을 곱한 행렬 집합을 계산할 필요없이 이전 결과를 사용하면 되므로 상당한 중복계산을 피할 수 있다.

4. 실행결과

<표 1>은 [정리 4]를 [11]에 있는 알고리즘에 적용하여 얻은 실행 결과로서, 알고리즘은 Java 언어로 구현하였으며 Pentium 4 3.0G CPU, 512MB RAM, 150GB HDD, Windows XP Professional 2002 환경에서 실행하였다. 불리언 행렬과 벡터의 곱셈은 2절에서 언급한 행 OR-연산 기반 불리언 행렬 곱셈의 핵심 아이디어를 비트사이의 논리연산에 적합하도록 변형하여 수행하였다. (그림 1)에서 보듯이 이론을 적용할 경우 행렬의 크기가 커질수록 더 좋은 실행시간 개선 결과를 보이고 있다.

5. 결론 및 향후 연구방향

불리언 행렬의 곱셈에 대하여 많은 연구가 수행

<표 1> 실행시간 비교

행렬 크기			기존 알고리즘	정리적용 알고리즘
$l \times n$	$n \times m$	$m \times k$		
2×2	2×2	2×2	0.015 초	0.015 초
2×2	2×3	3×3	0.015 초	0.015 초
3×3	3×3	3×3	0.031 초	0.016 초
3×3	3×4	4×4	0.953 초	0.031 초
4×4	4×4	4×4	74.375 초	2.125 초
4×4	4×5	5×5	24279 초	65.9 초
5×5	5×5	5×5	4825249 초	26267 초

되었으나 모든 불리언 행렬 사이의 곱셈에 대한 연구는 극히 소수가 있을 뿐이다. 이러한 연구 중 메모리 공간과 실행시간을 상당부분 개선시킨 연구결과도 보이고 있으나, 모든 $l \times n$, $n \times m$, $m \times k$ 불리언 행렬 사이의 곱셈이 가지는 NP-완전 계산복잡도로 인해 계속적인 실행시간 개선에 대한 연구가 필요하다.

본 논문은 기존에 제시된 연구의 결과에 더해 세 개의 연속된 모든 $n \times m$, $l \times n$, $m \times k$ 불리언 행렬의 곱셈을 보다 개선할 수 있는 이론을 제시하였다. 또한 이론을 적용한 불리언 행렬 중첩곱셈의 실행결과를 기존의 행렬곱셈 방법과 비교하여 실행시간이 더욱 개선되었음을 논하였다.

모든 불리언 행렬에 대한 곱셈은 NP-완전 계산복잡도를 가지므로 다차원 함수의 계산복잡도를 가지는 해결책을 찾기 어려우나, 특정 경우에 적용할 수 있는 수학적 이론을 정립함으로써 보다 개선된 공간 및 시간 복잡도를 얻을 수 있다. 따라서 특수한 경우의 불리언 행렬 곱셈을 효율적으로 계산할 수 있는 이론에 대한 연구가 필요하며, 이와 더불어 이론을 적용한 알고리즘의 설계 및 최적화, 병렬 컴퓨팅 적용 등에 대한 연구도 필요하다.

참고문헌

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

[2] L. Yelowitz, "A Note on the Transitive Closure of a Boolean Matrix", *ACM SIGMOD Record*, Vol. 25 No. 2, 1978, pp. 30

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

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

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

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

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

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

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

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

[11] 한재일, "모든 l x n, n x m, m x k 불리언 행렬 사이의 중첩곱셈에 대한 연구", 한국IT서비스학회 논문지, 제5권, 제1호, 2006, (게재 예정)

[12] 한재일, "모든 m x k 불리언 행렬과의 효율적 곱셈에 관한 연구", 한국콘텐츠학회 논문지, 제6권, 제2호, 2006, pp. 27-33

[13] 한재일, "n x m 불리언 행렬과 모든 l x n, m x k 불리언 행렬과의 중첩곱셈에 요구되는 공간 및 시간 복잡도 개선에 대한 연구", 국민대학교 기초과학연구소 논문집, 제25권, 2006, pp. -

[14] 신철규, 한재일, "내부 순환문 개선을 통한 Linux 기반의 D-클래스 계산 고효율 순차 알고리즘", 한국SI학회 춘계학술대회 논문집, 2005, pp. 526-531

[15] 신철규, 한재일, "공유 메모리 기반의 고성능 D-클래스 계산 병렬 알고리즘", 한국컴퓨터종합학술대회 논문집, 제32권, 제1호, 2005, pp. 10-12