모든 m x k 불리언 행렬과의 효율적 곱셈에 관한 연구

A Study on the Efficient Multiplication with All $m \times k$ Boolean Matrices

한재일
국민대학교 컴퓨터학부
Jae-Il Han(jhan@kookmin.ac.kr)

요약

불리언 행렬은 다양한 분야에 응용되어 유용하게 사용되고 있으며 불리언 행렬에 대한 많은 연구가 수행되었다. 대부분의 연구에서는 불리언 행렬의 곱셈을 다루고 있으나 모두 두 불리언 행렬 사이의 곱셈에 관심을 두고 있으며 다수의 $n \times m$ 불리언 행렬과 모든 $m \times k$ 불리언 행렬 사이의 곱셈은 극히 소수의 연구에서 보이고 있다. 본 논문은 기존에 제시된 두 불리언 행렬의 최적 곱셈 알고리즘이 모든 불리언 행렬에 대한 곱셈을 해야 하는 경우 부적합함을 보이고 $n \times m$ 불리언 행렬과 모든 $m \times k$ 불리언 행렬의 곱셈을 효율적으로 계산할 수 있는 이론을 정립한 후 이를 적용한 불리언 행렬 곱셈의 실행결과에 대하여 논한다.

■ 중심어 : | 불리언 행렬 | 행렬 곱셈 | 알고리즘 | NP-완전 | 계산복잡도 |

Abstract

Boolean matrices are applied to a variety of areas and used successfully in many applications, and there are many researches on boolean matrices. Most researches deal with the multiplication of boolean matrices, but all of them focus on the multiplication of two boolean matrices and very few researches deal with the multiplication between many $n \times m$ boolean matrices and all $m \times k$ boolean matrices. The paper discusses the existing optimal algorithms for the multiplication of two boolean matrices are not suitable for the multiplication between a $n \times m$ boolean matrix and all $m \times k$ boolean matrices, establishes a theory that enables the efficient multiplication of a $n \times m$ boolean matrix and all $m \times k$ boolean matrices, and shows the execution results of a multiplication algorithm designed with this theory.

■ keyword : | Boolean Matrix | Matrix Multiplication | Algorithm | NP-Complete | Computational Complexity |

I. 서 론

불리언 행렬은 원소가 0(거짓)이나 1(참) 값을 갖는 행렬로 정의되며 단순하고 논리적인 특성으로 인해 여러 분야에 응용되어 유용하게 사용되고 있다. 다양한 분야에서의 응용을 위해 불리언 행렬에 대하여 많은 연구가 수행되었으며[4-9] 대부분의 연구에서 불리언 행렬의 곱셈을 다루고 있다. 그러나 이 연구들은 모두 두 불리언 행렬의 곱셈에 관심을 두고 있으며 극히 소수의 연구[1-3]만이 주어진 크기의 모든 불리언 행렬을 대상으로

접수번호 : #051226-002
접수일자 : 2005년 12월 26일
심사완료일 : 2006년 01월 03일
교신저자 : 한재일, e-mail : jhan@kookmin.ac.kr

한 곱셈을 다루고 있다. 이러한 곱셈은 D-클래스[10] 계산 등에서 요구되나 NP-완전 계산복잡도와 효율적 곱셈에 대한 연구 부재로 인해 극히 제한된 결과만이 알려져 있다.

본 논문은 기존에 제시된 두 불리언 행렬의 최적(optimal) 곱셈 알고리즘이 주어진 크기의 모든 불리언 행렬을 대상으로 곱셈을 해야 하는 경우 부적합함을 보이고, 하나의 $n \times m$ 불리언 행렬과 모든 $m \times k$ 불리언 행렬의 곱셈을 보다 효율적으로 할 수 있는 수학적 이론을 정립한 후 이론을 적용할 경우 메모리 공간과 실행 시간에 상당한 개선이 있음을 논하며 제시된 이론을 바탕으로 설계한 성능시험 알고리즘과 실행결과에 대하여 기술한다. 본 논문의 구성은 다음과 같다. 2장은 기존에 제시된 두 불리언 행렬의 최적 곱셈 알고리즘을 살펴보고 하나의 $n \times m$ 불리언 행렬과 모든 $m \times k$ 불리언 행렬의 곱셈에 적용할 경우 나타나는 문제점에 대하여 기술한다. 3장은 하나의 $n \times m$ 불리언 행렬과 모든 $m \times k$ 불리언 행렬의 곱셈을 보다 효율적으로 할 수 있는 수학적 이론을 기술하고, 메모리 공간과 실행시간에 대하여 분석한다. 4장은 성능시험 알고리즘과 실행결과에 대하여 기술하며 5장은 결론 및 향후 연구방향에 대하여 논한다.

II. 관련 연구 및 문제점

두 불리언 행렬의 최적 곱셈에 대한 연구가 많이 수행되었으나 행을 이용한 곱셈 알고리즘 중에서 $O(n^2/\log n)$ 번의 행 OR-연산을 보이는 알고리즘[8]이, 비트 기반 연산을 이용한 알고리즘 중에서는 $O(n^{\log_2 7}\log n)$ 번의 비트 연산을 요구하는 알고리즘[9]이 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 행렬과 모든 $n \times n$ 불리언 행렬의 곱셈이 요구되는 경우 위의 두 알고리즘은 두 불리언 행렬 곱셈이 수행되기 전에 요구되는 정보를 생성하기 위하여 각 불리언 행렬마다 [8]은 최악의 경우 $O(2^n)$, [9]는 $O(n^2\log n)$의 계산 시간이 추가적으로 필요하다.

위에 언급한 바와 같이 [8, 9]의 알고리즘은 불리언 행렬에 여러 불리언 행렬을 곱할 때 바로 사용되기 어렵다. 그러나 [8]의 경우 알고리즘의 핵심 아이디어가 불리언 행렬을 여러 불리언 행렬에 곱할 때 부분적으로 사용될 수 있으므로 간단히 [8]의 핵심내용을 기술한다. $n \times n$ 불리언 행렬 A, B가 주어졌을 때 두 행렬의 곱 $C = AB$는

$$C_{i,j} = \bigcup_{k=1}^{n} (A_{ik} \cap B_{kj}) \text{ for } 1 \le i, j \le n$$

으로 정의된다. $M_{k\cdot}$를 M의 k 행에서 1 값을 가지는 열 번호의 집합 즉,

$$M_{k\cdot} = \{ j \mid A_{kj} = 1 \}$$

으로 정의하였을 때

$$C_{k\cdot} = \bigcup_{j \in A_{k\cdot}} B_{j\cdot}$$

와 같은 결과를 얻으며, 이는 두 불리언 행렬 곱셈에서 앞의 불리언 행렬 A를 불리언 행렬 B의 행에 대한 OR-연산 지시자로 사용하여 곱셈 결과를 얻을 수 있음을 보인다.

III. 모든 $m \times k$ 불리언 행렬에 대한 곱셈

본 장은 하나의 $n \times m$ 불리언 행렬과 모든 $m \times k$ 불리언 행렬의 곱셈을 $n \times m$ 불리언 행렬과 m 차원 벡터의 곱셈으로 대체하여 효율적 계산을 가능하게 하는 수학적 이론을 기술하고 벡터기반 곱셈과 행렬기반 곱셈에 요구되는 메모리 공간과 실행시간에 대하여 논한다.

1. 용어 및 기호 정의

본 장의 설명을 위해 다음과 같이 용어와 기호를 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 번째 행과 열을 의미한다. A^T는 A의 전치(transpose) 행렬로서

$$A^T = (a_{ji}) \; where \; A = (a_{ij}), 0 \le i, j \le n-1$$

으로 정의되며, 원소가 F 에 속하는 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 동등하게 다룬다. v^T는 v 의 열과 행 번호를 바꾼 $m \times 1$ 불리언 행렬과 같으며 $m \times n$ 불리언 행렬의 열에 대응하여 열벡터로 부른다.

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

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 열벡터의 집합이며 $(V^T)^k$ 는 열벡터를 k 번 조합하여 생성한 모든 $m \times k$ 불리언 행렬의 집합이다.

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

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

$$V^T = \{ v^T | v \in V \}$$

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

$n \times m$ 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.

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

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

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

2. 벡터 기반의 불리언 행렬 곱셈 이론

본 절은 하나의 $n \times m$ 불리언 행렬과 모든 $m \times k$ 불리언 행렬의 곱셈을 할 때 불리언 행렬 사이의 곱셈대신 불리언 행렬과 벡터 사이의 곱셈으로 같은 결과를 얻을 수 있음을 보이는 정리에 대하여 기술한다.

[정리 1]

$M_n^m(F)$의 임의의 $n \times m$ 불리언 행렬 A에 대해

V를 m 차원 불리언 벡터의 전체 집합,

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

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

로 정의할 때 $(C_A)^k \subset R_A$ 이다.

[증명] H를 $(C_A)^k$에 속한 임의의 $n \times k$ 불리언 행렬이라 하고 $H \notin R_A$라고 가정하자. $H \notin R_A$이므로 R_A에 속한 모든 $n \times k$ 불리언 행렬 G에 대하여 $H \neq G$ 이다. 따라서 H의 임의의 i 열(H^i)은 R_A에 속한 모든 $n \times k$

불리언 행렬 G의 i 열(G^i)과 다르다. $M_n^m(F)$는 모든 $n \times m$ 불리언 행렬의 집합이므로 V가 m 차원 불리언 벡터의 전체 집합일 때

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

이고 $B \in M_m^k(F)$이므로

$$\begin{aligned} R_A = \{ & ((A_0 u_0^T A_1 u_0^T \cdots A_{n-1} u_0^T)^T \\ & (A_0 u_1^T A_1 u_1^T \cdots A_{n-1} u_1^T)^T \\ & \cdots \\ & (A_0 u_{k-1}^T A_1 u_{k-1}^T \cdots A_{n-1} u_{k-1}^T)) \\ & | u_i \in V \text{ for } 0 \le i \le k-1 \} \end{aligned}$$

이 된다. H는 $(C_A)^k$에 속한 $n \times k$ 불리언 행렬이므로 H의 각 열 H^i는 V에 속한 어떤 m 차원 불리언 벡터 u 에 대하여 $H^i = (A_0 u^T A_1 u^T \cdots A_{n-1} u^T)$이 되므로 H가 R_A에 속하게 되어 모순이다. ■

[정리 2]

V를 m 차원 불리언 벡터의 전체 집합이라 하자. $M_n^m(F)$의 임의의 불리언 행렬 A에 대해

$$\begin{aligned} R_A &= \{ AB | B \in M_m^k(F) \}, \\ C_A &= \{ (A_0 v^T A_1 v^T \cdots A_{n-1} v^T)^T | v \in V \} \end{aligned}$$

로 정의하면 $R_A \subset (C_A)^k$ 이다.

[증명] D를 R_A에 속한 임의의 $n \times k$ 불리언 행렬이라 하자. C_A는 $n \times m$ 불리언 행렬 A의 각 행에 V^T에 속한 각 m 차원 불리언 벡터 v^T를 곱하여 얻은 n 차원 벡터(또는 $n \times 1$ 행렬)의 전체 집합이다. 따라서 0과 $k-1$ 사이의 모든 i 에 대해 D^i가 C_A에 속하며, $D \in C_A^k$ 이다. ■

위의 두 정리로부터 다음 부속정리를 얻을 수 있다.

[정리 3]

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

$$\begin{aligned} V &= \{ (b_0 b_1 \cdots b_{m-1}) | b_i \in F \text{ for } 0 \le i \le m-1 \} \\ R_A &= \{ AB | B \in M_m^k(F) \}, \\ C_A &= \{ (A_0 v^T A_1 v^T \cdots A_{n-1} v^T)^T | v \in V \} \end{aligned}$$

로 정의하면 $R_A = (C_A)^k$이다.

[정리 3]은 $n \times m$ 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. 다음은 위 정리에 대한 예이다.

$$\begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

(a) 주어진 2×3 불리언 행렬

$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$
$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}$$

(b) 행렬 곱셈 결과 생성되는 2×2 불리언 행렬

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

(c) 행렬과 벡터 곱셈으로 생성되는 열벡터

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

(d) 주어진 3×4 불리언 행렬

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(e) 주어진 5×5 불리언 행렬

그림 1. 주어진 불리언 행렬과 곱셈 결과

[그림 1]의 (a)에 주어진 2×3 불리언 행렬과 모든

3×2 불리언 행렬(2^6개)을 곱하면 (b)에 나타난 9개의 2×2 불리언 행렬을 얻는다. 정리에 따라 주어진 행렬 (a)에 모든 3차원 열벡터(8개)를 곱하면 (c)에 보이는 세 개의 2차원 열벡터를 얻는다. 이 세 개의 2차원 열벡터를 2개씩 모든 가능한 조합으로 묶으면 (b)의 9개 행렬을 모두 얻는다.

[그림 1]의 (c)에 주어진 3×4 불리언 행렬에 4차원 열벡터를 모두(2^4개) 곱하면 3차원 열벡터 전체 집합(8개)을 얻으며 이들을 가능한 3개의 조합으로 묶으면 2^9개의 3×3 불리언 행렬을 모두 얻으며, 이는 (c)에 주어진 불리언 행렬과 모든 4×3 불리언 행렬(2^{12}개)을 곱하여 얻는 3×3 불리언 행렬의 전체 집합(2^9개)과 같은 결과이다.

다음 5×5 불리언 행렬에 모든 5×5 불리언 행렬(2^{25}개)을 곱하면 2^{25}개의 5×5 불리언 행렬을 전부 얻는다. 주어진 행렬에 5차원 열벡터를 모두 곱하면 32개의 5차원 열벡터를 얻으며 이들을 가능한 5개의 조합으로 묶으면 2^{25}개의 5×5 불리언 행렬을 모두 얻는다.

3. 메모리 공간 및 실행시간 분석

위의 정리에 의해 하나의 $n\times m$ 불리언 행렬과 모든 $m\times k$ 불리언 행렬의 곱셈을 $n\times m$ 불리언 행렬과 모든 m 차원 벡터의 곱셈으로 수행할 수 있으므로 곱셈 시간이 개선된다. 또한 계산 도중이나 곱셈 결과 생성되는 불리언 행렬의 집합을 많은 불리언 행렬의 집합대신 훨씬 적은 수의 벡터 집합으로 나타내어 중간 결과와 최종 곱셈 결과를 저장할 메모리 공간을 줄일 수 있다.

구체적으로 $n\times m$ 불리언 행렬에 $m\times k$ 불리언 행렬을 곱하는 경우 $2^{m\times k}$개의 $m\times k$ 불리언 행렬을 곱하는 대신 2^m 개의 m 차원 벡터를 곱하여 결과를 얻을 수 있어 실제 실행 시간에 상당한 성능 개선을 가져올 수 있다. 또한 곱셈 결과로 n 차원 벡터의 집합을 생성하여 이 집합에 속한 벡터들을 모든 가능한 방법으로 k번 조합하면 결과로 얻을 모든 $n\times k$ 불리언 행렬을 얻을 수 있으므로 최악의 경우에도 $2^n\cdot n$ 에 비례한 메모리 공간이 요구되므로 불리언 행렬의 집합을 직접 나타낼 때 최악의 경우 요구되는 메모리 공간이 $2^{n\cdot k}\cdot n\cdot k$ 에 비례하는 것과 비교하면 중간결과 저장을 위한 메모리 공간을 고려하지 않더라고 많은 메모리 공간이 절약되는 것을 알 수 있다.

IV. 알고리즘 및 실행 결과

D-클래스 계산은 기본적으로 모든 n 차 정사각 불리언 행렬사이의 곱셈을 요구한다. 이러한 응용을 염두에 두고 [정리 3]을 적용하였을 경우 실제 실행시간의 개선 정도를 알아보기 위하여 모든 $n\times m$ 불리언 행렬과 $m\times k$ 불리언 행렬 사이의 곱셈을 수행하는 간단한 알고리즘을 작성하였다[그림 2]. 알고리즘은 각 $n\times m$ 불리언 행렬에 대하여 모든 m 차원 열벡터를 곱한다. 불리언 행렬과 각 벡터의 곱셈으로부터 얻은 벡터는 현 불리언 행렬에 대한 벡터 집합에 삽입한다. 모든 벡터와의 곱셈이 종료되면 현 불리언 행렬에 대한 벡터 집합과 벡터로부터 조합된 모든 불리언 행렬의 집합을 저장한다.

```
for each boolean matrix A in M_n^m(F)
    S = ∅
    for each row vector v in V
        compute an Av^T vector
        insert the vector into S
    store the set S and synthesized
        matrices for boolean matrix A
```

* V : a set of m-dimensional row vectors
* S : a set of n-dimensional column vectors

그림 2. 성능시험 알고리즘

알고리즘은 Java 언어로 구현하였으며 2개의 Zeon CPU, 1GB RAM, 250GB HDD, Fedora 9.0 환경에서 실행하였다. 불리언 행렬과 벡터의 곱셈은 2절에서 언급한 행 OR-연산 기반 불리언 행렬 곱셈의 핵심 아이디어를 비트사이의 논리연산에 적합하도록 변형하여 수행하였다.

표 1. 실행시간 비교

행렬 크기		기존 알고리즘 (행렬 곱셈)	개선 알고리즘 (벡터 곱셈)
$n \times m$	$m \times k$		
2×2	2×2	0.002 초	0.002 초
3×3	3×3	0.095 초	0.011 초
3×4	4×3	3.721 초	0.05 초
3×4	4×4	65.385 초	0.056 초
4×4	4×4	1225 초	0.705 초
4×5	5×4	337027 초	19.926 초
4×5	5×5	12384124 초	195.38 초
5×5	5×5	377954566 초 (4374일 이상)	17653 초 (4시간 55분)

[표 1]은 기존 알고리즘[12]과 위 알고리즘을 실행하여 얻은 결과를 보이고 있다. 기존 알고리즘의 $m \times k$ 행렬에 대한 5×4 이상 크기의 실행 결과는 10000개의 $n \times m$ 불리언 행렬에 대한 평균 곱셈 시간을 $2^{n \times m}$에 곱하여 얻은 예상 실행시간이다. 알고리즘의 계산복잡도는 NP-완전문제이나 [표 1]에서 보는 것처럼 실제 실행시간은 불리언 행렬의 원소개수가 증가함에 따라 기존의 알고리즘[12]과 비교하여 상당한 개선을 보이고 있다.

V. 결론 및 향후 연구방향

다양한 분야에서의 응용을 위해 불리언 행렬에 대하여 많은 연구가 수행되었으며 대부분의 연구에서 불리언 행렬의 곱셈을 다루고 있다. 그러나 이 연구들은 모두 두개의 불리언 행렬 곱셈에 관심을 두고 있으며 다수의 $n \times m$ 불리언 행렬과 모든 $m \times k$ 불리언 행렬 사이의 곱셈에 대한 연구는 극히 소수가 보이고 있다.

본 논문은 기존에 제시된 두 불리언 행렬의 최적 곱셈 알고리즘이 많은 불리언 행렬 쌍에 대한 곱셈을 해야 하는 경우 실행시간과 메모리 공간의 문제점으로 인해 부적합함을 보이고 하나의 $n \times m$ 불리언 행렬과 모든 $m \times k$ 불리언 행렬의 곱셈을 보다 효율적으로 할 수 있는 수학적 이론을 제시하였다. 또한 이론을 적용할 경우 직접적인 행렬 곱셈 방법과 비교하여 메모리 공간과 실행 시간에 상당한 개선이 있음을 논하였으며, 제시된 이론을 바탕으로 설계한 알고리즘과 실행결과에 대하여 기술하였다.

모든 불리언 행렬에 대한 곱셈은 NP-완전 계산 복잡도를 가지므로 해결 방안을 찾기 어려우나, 경우에 따라 적용할 수 있는 수학적 이론을 정립할 수 있다면 적은 메모리 공간으로 빠른 실행이 가능할 수도 있음을 본 논문의 결과는 보이고 있다. 이러한 관점에서 본 논문의 결과는 단지 시작일 뿐이며 앞으로 특수한 경우의 불리언 행렬 곱셈을 효율적으로 할 수 있는 알고리즘을 개발하기 위해 이론뿐 아니라 알고리즘 최적화, 병렬 컴퓨팅 적용 등에 대한 많은 연구가 필요하다.

참 고 문 헌

[1] 신철규, 한재일, "그리드 컴퓨팅 환경에서의 D-클래스 계산 병렬 알고리즘", 한국정보처리학회 춘계학술대회 논문집, 제12권, 제1호, pp.929-932, 2005.

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

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

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

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

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

[7] 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,

pp.799-801, 1995.

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

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

[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, pp.9-13, 1983.

저 자 소 개

한 재 일(Jae-Il Han) 정회원

- 1980년 2월 : 연세대학교 수학과(이학사)
- 1986년 5월 : 미국 Syracuse 대학교 전산학과(전산학석사)
- 1992년 5월 : 미국 Syracuse 대학교 전산학과(전산학박사)
- 1995년 5월~현재 : 국민대학교 컴퓨터학부 교수

<관심분야> : 분산처리, 객체지향 시스템, 컴퓨터 및 네트워크 보안, RFID/USN, 지능형 시스템