# D-클래스 계산을 위한 불리언 행렬의 효율적 곱셈 및 알고리즘

# (Efficient Multiplication of Boolean Matrices and Algorithm for D-Class Computation)

**한재일*, 신범주****
(Jae-Il Han, Bum-Joo Shin)

**요 약** D-클래스는 주어진 동치관계(equivalence relation)에 있는 $n \times n$ 불리언 행렬의 집합으로 정의된다. D-클래스 계산은 $n \times n$ 불리언 행렬의 전체 집합을 대상으로 이 집합에서 조합할 수 있는 모든 세 불리언 행렬 사이의 곱셈을 요구한다. 그러나 불리언 행렬에 대한 대부분의 연구는 단지 두 개의 불리언 행렬에 대한 효율적인 곱셈에 집중되었으며 모든 불리언 행렬 사이의 곱셈에 대한 연구는 최근에야 소수가 보이고 있다. 본 논문은 모든 세 개의 불리언 행렬 곱셈과 모든 D-클래스를 보다 효율적으로 계산할 수 있는 이론을 제시하고 이를 적용한 알고리즘과 실행결과에 대하여 논한다.

**핵심주제어** : D-클래스, 불리언 행렬, 행렬 곱셈, NP-완전, 계산복잡도

**Abstract** D-class is defined as a set of equivalent $n \times n$ boolean matrices according to a given equivalence relation. The D-class computation requires the multiplication of three boolean matrices for each of all possible triples of $n \times n$ boolean matrices. However, almost all the researches on boolean matrices focused on the efficient multiplication of only two boolean matrices and a few researches have recently been shown to deal with the multiplication of all boolean matrices. The paper suggests a mathematical theory that enables the efficient multiplication for all possible boolean matrix triples and the efficient computation of all D-classes, and discusses algorithms designed with the theory and their execution results.

**Key Words** : D-class, Boolean Matrix, Matrix Multiplication, NP-Complete, Computational Complexity

## 1. 서 론

불리언 행렬은 원소가 0(거짓)이나 1(참) 값을 갖는 행렬로서, 구문분석, 선형 우선순위 함수 계산, 논리 최적화 등 여러 분야에서 유용하게 사용되고 있다[1-5]. 이러한 대부분의 응용은 두 불리언 행렬의 효율적인 곱셈에 초점을 두고 있으며 많은 연구를 통해 여러 응용에 적합한 다양한 알고리즘이 제시되었다[6-11].

$F=\{0,1\}$, $M_n(F)$를 원소가 $F$에 속하는 모든 $n \times n$ 행렬의 집합, $A,B$를 $M_n(F)$에 속하는 임의의 $n \times n$ 행렬이라 할 때 동치관계(equivalence

* 국민대학교 컴퓨터학부
** 부산대학교 바이오시스템공학부, 교신저자

relation) 와 D-클래스는 다음과 같이 정의된다 [12].

$$A \sim_R B \text{ if } \exists\, X, Y, U, V \in M_n(F) \text{ such that } UAX = B,\ VBY = A.$$

$$D_A = \{B \in M_n(F) \mid A \sim_R B$$

이러한 D-클래스 계산은 기존 응용과 달리 조합가능한 모든 세 불리언 행렬 사이의 곱셈을 기본적으로 요구한다. 그러나 모든 불리언 행렬 사이의 곱셈에 대한 연구는 NP-완전 계산복잡도와 효율적 곱셈의 어려움, 상대적으로 적은 응용 분야 등으로 인해 관심 밖에 있으며, 최근에야 기초적인 연구결과가 보이고 있다[13-15]. D-클래스는 이러한 이유로 현재 4×4 이하 크기의 불리언 행렬에 대한 결과만이 알려져 있으며 D-클래스의 특성에 대한 기본적인 연구가 활성화되지 못하고 있다.

본 논문은 D-클래스 계산 시간을 개선하기 위해 조합가능한 모든 세 불리언 행렬 사이의 효율적 곱셈이론과 이를 바탕으로 한 D-클래스의 계산 이론을 제시하고, 이론을 적용하여 설계한 알고리즘과 실행결과에 대하여 논한다.

본 논문의 구성은 다음과 같다. 2장은 기존의 연구 및 문제점에 대하여 논한다. 3장은 본 논문에서 사용할 용어 및 기호를 정의하고, 4장은 모든 $l \times n$, $n \times m$, $m \times k$ 불리언 행렬사이의 연속된 이중곱셈을 효율적으로 할 수 있는 이론, 이론을 적용한 불리언 행렬 곱셈의 공간 및 시간 복잡도 분석, 그리고 알고리즘 실행 결과를 기술한다. 5장은 제시한 불리언 행렬 곱셈 이론을 바탕으로 D-클래스를 효율적으로 계산할 수 있는 이론과 이를 적용한 알고리즘의 실행결과를 논하며, 6장은 결론 및 향후 연구방향에 대하여 기술한다.

## 2. 관련연구 및 문제점

앞 장에서 기술한 바와 같이 불리언 행렬에 대한 연구는 대부분 두 불리언 행렬의 효율적인 곱셈에 초점을 두고 있으며[1-11], 소수의 연구[13-15]만이 모든 불리언 행렬 사이의 곱셈을 다루고 있다. 두 불리언 행렬의 곱셈에 대한 연구를 살펴보면 행을 이용한 곱셈 알고리즘 중에서는 $O(n^2/\log n)$ 번의 행 OR-연산 알고리즘[10]이, 비트 기반 연산을 이용한 알고리즘 중에서는 $O(n^{\log_2^7}\log n)$ 번의 비트 연산 알고리즘[11]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 행렬에 대해 모든 $n \times n$ 불리언 행렬을 곱하는 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 [10]은 최악의 경우 $O(2^n)$, [11]은 $O(n^2 \log n)$의 계산 시간이 추가적으로 필요하다. 따라서 [10, 11]의 알고리즘은 모든 불리언 행렬을 대상으로 한 곱셈에 사용하기 어렵다.

모든 불리언 행렬의 곱셈에 대한 연구 중 [13, 14]는 순환문 개선에 초점을 두었으나 메모리 공간이나 실행시간 등의 개선정도가 매우 미약하다. 반면 [15]는 행렬 연산을 벡터 기반으로 계산할 것을 제안하였으며 동시에 메모리 공간과 실행시간을 개선하였다. 본 논문은 [15]에서 제시한 벡터기반 곱셈 이론을 바탕으로 연속곱셈 및 D-클래스 계산에 필요한 실행시간을 개선할 수 있는 이론을 제시한다.

## 3. 용어 및 기호 정의

본 논문의 설명을 위해 다음과 같이 용어와 기호를 정의한다. 임의의 $n\times m$ 불리언 행렬 $A$가 주어지고 $F=\{0,1\}$이라 할 때 $M_n^m(F)$는 모든 $n\times m$ 불리언 행렬의 집합을 정의하며 $A_i$와 $A^i$는 각각 $A$ 행렬의 $i$ 행과 $i$ 열을 의미한다. $A^T$는 $A$의 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 부른다. $v^T$는 $v$ 의 열과 행 번호를 바꾼 $m\times 1$ 불리언 행렬로서

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

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

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

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

$$V^T(m)=\{v^T|v\in V(m)\}$$

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

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

$$vB=(vB^0 vB^1 \cdots vB^{n-1});\ \text{단}\ v\in V(m)$$

$$Av^T=\begin{pmatrix} A_0v^T \\ A_1v^T \\ \vdots \\ A_{n-1}v^T \end{pmatrix},\ \text{단}\ v\in V(m)$$

## 4. 불리언 행렬 곱셈

본 절은 모든 $l\times n$, $n\times m$ $m\times k$ 불리언 행렬의 이중 연속곱셈을 불리언 행렬 사이의 곱셈 대신 $n$ 차원 벡터와 $m$ 차원 벡터의 곱셈을 이용하여 보다 효율적으로 같은 결과를 얻을 수 있음을 보이는 정리, 공간 및 시간 복잡도, 알고리즘 및 실행결과에 대하여 기술한다.

### 4.1 벡터 기반의 불리언 행렬 연속곱셈

[정리 1] $M_n^m(F)$에 속한 임의의 $n\times m$ 불리언 행렬 $A$에 대해 $R_A$, $C_A$를

$$R_A=\{AB|B\in M_m^k(F)\},$$
$$C_A=\{(A_0v^T A_1v^T\cdots A_{n-1}v^T)^T|v\in V(m)\}$$

으로 정의할 때 $R_A=(C_A)^k$이다.

(증명) $H$를 $(C_A)^k$에 속한 임의의 $n\times k$불리언 행렬이라 하고 $H\notin R_A$라고 가정하자. $H\notin R_A$이므로 $R_A$에 속한 모든 $n\times k$ 불리언 행렬 $G$에 대하여 $H\neq G$이다. 따라서 임의의 행렬 $G$에 대한 어떤 $i$열은 $H$의 $i$열과 달라야 한다. $M_n^m(F)$는 모든 $n\times m$ 불리언 행렬의 집합이므로

$$M_m^k(F)=\left\{\left(v_0^T,v_1^T,\cdots,v_{k-1}^T\right)\middle|\begin{array}{r}v_i\in V(m),\\ 0\le i\le k-1\end{array}\right\}$$

$$R_A=\left\{\left(\begin{array}{c}\left(A_0u_0^T,A_1u_0^T,\cdots,A_{n-1}u_0^T\right)^T\\ \left(A_0u_1^T,A_1u_1^T,\cdots,A_{n-1}u_1^T\right)^T\\ \cdots\\ \left(A_0u_{k-1}^T,A_1u_{k-1}^T,\cdots,A_{n-1}u_{k-1}^T\right)\end{array}\right)\middle|\begin{array}{r}v_i\in V(m),0\le i\le k-1\end{array}\right\}$$

이다. $H$는 $(C_A)^k$에 속한 $n\times k$ 불리언 행렬이므로 $H$의 각 열 $H^i$는 $V(m)$에 속한 어떤 벡터 $u$에 대하여 $H^i=\left(A_0u^TA_1u^T\cdots A_{n-1}u^T\right)$이 되므로 $H$가 $R_A$에 속하게 되어 모순이다.

$D$를 $R_A$에 속한 임의의 $n\times k$ 불리언 행렬이라 하자. $C_A$는 $n\times m$ 불리언 행렬 $A$의 각 행에 $V^T(m)$에 속한 각 $m$ 차원 불리언 벡터 $v^T$를 곱하여 얻은 $m$ 차원 벡터의 전체 집합이다. 따라서 0과 $k-1$ 사이의 모든 $i$에 대해 $D^i$가 $C_A$에 속하며, $D\in(C_A)^k$ 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. 다음 정리는 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.

[정리 2] $M_n^m(F)$에 속한 임의의 $n\times m$ 불리언 행렬 $A$에 대해 $L_A, T_A$를 각각

$$L_A=\left\{BA\middle|B\in M_l^n(F)\right\},$$
$$T_A=\left\{\left(uA^0uA^1\cdots uA^{m-1}\right)\middle|v\in V(n)\right\}$$

로 정의할 때 $L_A=(T_A)_l$ 이다.

(증명) $H$를 $(T_A)_l$에 속한 임의의 $l\times m$ 불리언 행렬이라 하고 $H\notin L_A$라고 가정하자. $H\notin L_A$ 이므로 $L_A$에 속한 모든 $l\times m$ 불리언 행렬 $G$에 대하여 $H\neq G$이다. 따라서 임의의 행렬 $G$에 대한 어떤 $i$ 행은 $H$의 $i$ 행과 달라야 한다. $M_l^n(F)$는 모든 $l\times n$ 불리언 행렬의 집합이므로

$$M_l^n(F)=\left\{\begin{pmatrix}v_0\\ v_1\\ \vdots\\ v_{l-1}\end{pmatrix}\middle|v_i\in V(n),0\le i\le l-1\right\}$$

이고 $B\in M_l^n(F)$이므로

$$L_A=\left\{\left(\begin{array}{c}\left(v_0A^0,v_0A^1,\cdots,v_0A^{m-1}\right)\\ \left(v_1A^0,v_1A^1,\cdots,v_1A^{m-1}\right)\\ \cdots\\ \left(v_{l-1}A^0,v_{l-1}A^1,\cdots,v_{l-1}A^{m-1}\right)\end{array}\right)\middle|\begin{array}{r}v_i\in V(m),0\le i\le l-1\end{array}\right\}$$

이다. $H$는 $(T_A)_l$에 속한 $l\times m$ 불리언 행렬이므로 $H$의 각 행 $H_i$는 $V(n)$에 속한 어떤 불리언 벡터 $w$에 대하여 $H_i=\left(wA^0wA^1\cdots wA^{m-1}\right)$이 되므로 $H$가 $L_A$에 속하게 되어 모순이다.

$D$를 $L_A$에 속한 임의의 $l\times m$ 불리언 행렬이라 하면. $T_A$는 $n\times m$ 불리언 행렬 $A$의 각 열에 $V(n)$에 속한 각 불리언 벡터 $v$를 곱하여 얻은 $m$ 차원 벡터의 전체 집합이다. 따라서 0과 $l-1$ 사이의 모든 $i$에 대해 $D_i$가 $T_A$에 속하며, $D\in(T_A)$이다.

다음 정리는 $n\times m$ 불리언 행렬에 모든 $l\times n, n\times m$ 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.

[정리 3] $A$를 $M_n^m(F)$에 속한 임의의 $n\times m$ 불리언 행렬, $Z$를 $M_n^k(F)$에 속한 임의의 $n\times k$ 불리언 행렬이라 할 때 $R_A, Q_A, T_Z$를

$R_A = \{AX \mid X \in M_m^k(F)\}$,
$Q_A = \{UAX \mid U \in M_l^n(F), X \in M_m^k(F)\}$,
$T_Z = \{(vZ^0\ vZ^1 \cdots vZ^{k-1}) \mid v \in V(n)\}$

로 정의하면 $Q_A = \bigcup_{Z \in R_A} (T_Z)_l$ 이다.

(증명)

$$\begin{aligned} Q_A &= U(AX) \mid U \in M_l^n(F), X \in M_m^k \\ &= UZ \mid U \in M_l^n(F), Z \in R_A \\ &= \bigcup_{Z \in R_A} UZ \mid U \in M_l^n(F) \end{aligned}$$

따라서 [정리 2]에 의해 $Q_A = \bigcup_{Z \in R_A} (T_Z)_l$ 이다.

[정리 3]에 의하여 주어진 $n \times m$ 불리언 행렬과 모든 $l \times n$, $m \times k$ 불리언 행렬 사이의 연속곱셈을 다음과 같이 수행할 수 있다. [정리 1]을 적용하여 주어진 $n \times m$ 불리언 행렬에 모든 $m \times k$ 불리언 행렬을 곱하는 대신 모든 $m$ 차원 벡터를 곱하여 $n$ 차원 열벡터 집합을 얻고, 이 집합의 벡터를 $k$번 조합하여 $n \times m$ 불리언 행렬과 모든 $m \times k$ 불리언 행렬을 곱할 때와 같은 $n \times k$ 불리언 행렬의 집합을 얻는다. 다음에 [정리 2]를 적용하여 모든 $n$ 차원 벡터와 이 $n \times k$ 불리언 행렬 집합의 각 불리언 행렬을 곱하여 $k$ 차원 행벡터 집합을 얻고 이 집합의 벡터를 $l$ 번 조합하여 $l \times k$ 불리언 행렬 집합을 얻으면 주어진 $n \times m$ 불리언 행렬과 모든 $l \times n, m \times k$ 불리언 행렬을 중첩하여 곱할 때와 같은 결과를 얻을 수 있다. 모든 $l \times n, n \times m, m \times k$ 불리언 행렬 사이의 연속곱셈 결과는 $M_n^m(F)$ 의 각 $n \times m$ 불리언 행렬에 대해 위 연속곱셈을 수행하여 얻을 수 있다.

그러나 이 경우 실제 알고리즘을 설계할 때 각 불리언 행렬에 대하여 모든 벡터와의 첫 번째 곱셈에서 얻은 벡터 집합으로부터 불리언 행렬을 하나씩 생성하여야 하는 오버헤드를 포함하며 많은 계산이 중복된다. [정리 4]는 임의의 두 $n \times n$ 불리언 행렬 $A, B$에 모든 $n \times n$ 불리언 행렬을 곱하여 얻는 행렬 집합이 같으면 $A, B$에 모든 $n \times n$ 불리언 행렬을 뒤와 앞 순서로 연속곱하여 얻는 행렬 집합도 같다는 것을 보인다.

따라서 $M_n^m(F)$ 의 어떤 불리언 행렬에 모든 $m \times k$ 불리언 행렬을 곱하여 얻는 행렬 집합이 이미 존재하는 경우 모든 $l \times n$ 불리언 행렬에 대한 다음 곱셈을 수행하여 행렬 집합을 계산할 필요 없이 이전의 연속곱셈 결과를 사용하면 되므로 앞의 정리만 적용하였을 경우 나타나는 오버헤드와 많은 중복계산을 상당부분 개선할 수 있다.

[정리 4]

$M_n^m(F)$ 에 속한 임의의 두 불리언 행렬 $A, B$ 에 대해 $R_A, R_B, Q_A, Q_B$ 를

$R_A = \{AX \mid X \in M_m^k(F)\}$,
$R_B = \{BY \mid Y \in M_m^k(F)\}$,
$Q_A = \{UAX \mid U \in M_l^n(F), X \in M_m^k(F)\}$,
$Q_B = \{VBY \mid V \in M_l^n(F), Y \in M_m^k(F)\}$

로 정의할 때 $R_A = R_B$이면 $Q_A = Q_B$ 이다.

(증명)

$$\begin{aligned} Q_A &= \{UAX \mid U \in M_l^n(F), X \in M_m^k(F)\} \\ &= \{UW \mid U \in M_l^n(F), W \in S_A\} \\ &= \{UW \mid U \in M_l^n(F), W \in S_B\} \\ &= \{UBY \mid U \in M_l^n(F), Y \in M_m^k(F)\} = Q_B \end{aligned}$$

### 4.2 공간 및 시간 복잡도 분석

모든 불리언 행렬에 대한 곱셈은 NP-완전문제이다. 본 절의 공간 및 시간 복잡도에 대한 분석은 곱셈 알고리즘의 개선정도를 알기 위한 노력으로써 <표 1>은 불리언 행렬을 직접 곱한 경우와 [정리 1-3]을 적용한 경우의 공간 및 시간 복잡도를 보이고 있다. $n \times m$ 불리언 행렬에 모든 $m \times k$ 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 실행시간에 상당한 성능 개선을 가져올 수 있다. 또한 곱셈 결과로 $n$ 차원 벡터의 집합을 생성하여 이 집합에 속한 벡터들을 모든 가능한 방법으로 $k$ 번 조합하면 결과로 얻을 모든 $n \times k$ 불리언 행렬을 얻을 수 있으므로 최악의 경우에도 $n2^n$에 비례한 메모리 공간이 요구되므로 불리언 행렬의 집

합을 직접 나타낼 때 최악의 경우 요구되는 메모리 공간이 $nk^{nk}$에 비례하는 것과 비교하면 중간결과 저장을 위한 메모리 공간을 고려하지 않더라도 많은 메모리 공간이 절약되는 것을 알 수 있다.

$n\times m$ 불리언 행렬에 모든 $l\times n, m\times k$ 불리언 행렬을 연속 곱하는 경우는 불리언 행렬을 직접 곱하면 $2^{\ln}2^{mk}$에 비례하는 시간 복잡도를 가지나 위 정리를 적용하면 최악의 경우, 즉$m\times k$ 불리언 행렬을 곱했을 때 모든 $n\times k$ 불리언 행렬이 나오는 경우 $2^m2^{nk}2^n$에 비례하는 시간 복잡도를 가진다. 최악의 경우에 대한 공간복잡도는 불리언 행렬의 직접 곱셈은 $lk2^{lk}$에 비례하며 정리를 적용하는 경우 $k2^{(n+1)k}$에 비례한다.

<표 1> 공간 및 시간 복잡도 비교

| 구분 | | 행렬곱셈 알고리즘 | 정리적용 알고리즘 |
|---|---|---|---|
| 시간 복잡도 | 일반곱셈 | $2^{mk}$ | $2^m$ |
| | 연속곱셈 | $2^{\ln+mk}$ | $2^{n(k+1)+m}$ |
| 공간 복잡도 | 일반곱셈 | $nk2^{nk}$ | $n2^n$ |
| | 연속곱셈 | $lk2^{lk}$ | $k2^{(n+1)k}$ |

### 4.3 연속곱셈 알고리즘 및 실행결과

이론을 적용한 연속곱셈의 실제 실행시간 개선정도를 알아보기 위하여 행렬끼리의 연속곱셈 알고리즘, [정리 1-3]을 적용한 연속곱셈 알고리즘, 모든 정리를 적용한 연속곱셈 알고리즘을 실행하였다. 행렬을 직접 곱하여 수행되는 연속곱셈 알고리즘은 5개의 중첩된 for 순환문으로 간단히 설계할 수 있으므로 알고리즘 및 설명을 생략하며, 정리를 적용한 알고리즘에 대해서만 기술한다.

[그림 1]의 (a)는 [정리 1-3]을 적용하여 모든 $l\times n, n\times m, m\times k$ 불리언 행렬 사이의 연속곱셈을 수행하는 알고리즘 개요이다. 알고리즘은 먼저 $M_n^m(F)$에 속한 각 불리언 행렬에 대하여 $m$개의 원소를 가지는 모든 $2^m$개의 벡터를 곱한다. 불리언 행렬과 각 벡터의 곱셈으로부터 얻은 $m$ 차원 열벡터는 현 불리언 행렬에 대한 집합 $S_c$에 삽입한다. 모든 $m$차원 벡터와의 곱셈이 종료되면 $AX$에 대한 열벡터 집합 $S_c$를

```
𝕄 = ∅
for each boolean matrix A in M_n^m(F)
    S_c = ∅
    for each row vector v in V(m)
        compute an Av^T vector
        insert the vector into S_c
    for each matrix M synthesized from S_c
        S_r = ∅
        for each row vector v in V(n)
            compute an vM vector
            insert the vector into S_r
        insert S_r into ℂ_A
    insert a pair (A, ℂ_A) into 𝕄
```

(a) 개선 알고리즘 1

```
𝕄 = ∅
for each boolean matrix A in M_n^m(F)
    S_c = ∅
    for each row vector v in V(m)
        compute an Av^T vector
        insert the vector into S_c
    if (S_c ∉ 𝕄[0])
        S_M = ∅
        insert A into S_M
        for each matrix M synthesized from
      S_c
            S_r = ∅
            for each row vector v in V(n)
                compute an vM vector
                insert the vector into S_r
            insert S_r into ℂ_A
        insert a tripple (S_c, S_M, ℂ_A) into 𝕄
    else
        insert A into 𝕄(S_c)[1]
```

(b) 개선 알고리즘 2

* $S_c$ : a set of $n$-dimensional column vectors
* $S_r$ : a set of $k$-dimensional row vectors
* $S_M$ : a set of $n\times m$ boolean matrices
* $\mathbb{C}_A$ : a class of $S_r$'s for $S_c$
* $\mathbb{M}$ : a set of ($S_c$, $S_M$, $\mathbb{C}_A$) triples
* $\mathbb{M}[i]$ : a set of the $i$'th elements in all triples
* $\mathbb{M}(S_c)$ : a triple whose first element matches $S_c$
* $\mathbb{M}(S_c)[i]$ : the $i$'th element in $\mathbb{M}(S_c)$

<그림 1> 정리적용 연속곱셈 알고리즘

얻는다. $S_c$의 열벡터를 $k$번 조합하여 얻는 각 행렬에 대해 모든 $n$차원 행벡터를 곱하여 $k$차원 행벡터의 집합 $S_c$를 계산하고 현재 선택된 바깥 순환문 행렬 $A$에 대한 $c_A$ 클래스에 $S_c$를 삽입한다. 모든 조합된 행렬에 대하여 벡터집합 계산이 완료되면 $c_A$ 클래스는 행렬 $A$에 대한 연속곱셈으로부터 얻을 수 있는 모든 행렬의 집합을 나타내게 되며 집합 M에 $(A, c_A)$ 짝을 삽입한다. 알고리즘이 종료하면 $M_n^n(F)$의 모든 행렬에 대한 연속곱셈 계산결과가 M에 존재한다.

[그림 1]의 (b)는 모든 정리를 적용한 알고리즘이며 (a)의 알고리즘과 다른 점은 [정리 4]의 적용을 위해 첫 번째 곱셈의 결과로 얻은 벡터집합이 이미 계산되었는지를 검사하는 것이다. 만약 벡터집합이 이전에 계산되었으면 같은 벡터집합을 생성하는 불리언 행렬의 집합에 현재의 불리언 행렬 $A$를 추가한 후 나머지 계산을 생략하고 다음 불리언 행렬 $A$에 대한 연속곱셈을 수행한다. 이전에 계산되지 않은 경우는 $S_M$을 처음 벡터집합을 생성한 불리언 행렬로 초기화하고, (a)의 알고리즘과 같이 $S_c, S_r$를 계산해 나가며, 차기 순환문에서 벡터집합의 동일여부를 검사하고 같은 벡터집합을 생성하는 불리언 행렬을 저장하기 위해 세 개의 원소로 구성되는 $(S_c, S_M, C_A)$ 트리플을 M에 저장한다.

알고리즘은 Java 언어로 구현하였으며 2개의 2.8GHz Zeon CPU, 1GB RAM, 250GB HDD, Fedora 9.0 환경에서 실행하였다. 불리언 행렬과 벡터의 곱셈은 행 OR-연산 기반 불리언 행렬 곱셈[10]을 비트사이의 논리연산에 적합하도록 변형하여 수행하였다.

<표 2>는 행렬끼리의 곱셈을 수행하는 연속곱셈 알고리즘과 정리적용 연속곱셈 알고리즘을 실행하여 얻은 결과를 보이고 있다. 행렬곱셈 알고리즘의 $n \times m$ 행렬에 대한 $4 \times 4$ 이상 크기의 실행 결과는 가장 내부에 있는 루프의 평균 실행시간을 필요한 외부 루프 수만큼 곱하여 얻은 최소 예상 실행시간이다. 알고리즘의 계산복잡도는 NP-완전문제이나 <표 2>에서 보는 것처럼 실제 실행시간은 행렬곱셈 알고리즘과 비교하여 상당한 개선을 보이고 있다.

<표 2> 실행시간 비교

| 행렬 크기 | | | 행렬곱셈 알고리즘 (초) | 개선 알고리즘1 (초) | 개선 알고리즘2 (초) |
|---|---|---|---|---|---|
| $n \times m$ | $n \times m$ | $n \times m$ | | | |
| $2 \times 2$ | $2 \times 2$ | $2 \times 2$ | 0.006 | 0.002 | 0.002 |
| $2 \times 2$ | $2 \times 3$ | $3 \times 3$ | 0.073 | 0.009 | 0.003 |
| $3 \times 3$ | $3 \times 3$ | $3 \times 3$ | 9.682 | 0.041 | 0.021 |
| $3 \times 3$ | $3 \times 4$ | $4 \times 4$ | 11778 | 0.064 | 0.045 |
| $4 \times 4$ | $4 \times 4$ | $4 \times 4$ | 29312169 | 54.056 | 1.586 |
| $4 \times 4$ | $4 \times 5$ | $5 \times 5$ | $3.660 \times 10^{11}$ 이상 | 15657 | 44.696 |
| $5 \times 5$ | $5 \times 5$ | $5 \times 5$ | $5.126 \times 10^{15}$ 이상 | 2728587 | 15596 |

## 5. D-클래스 계산

직접적인 행렬 곱셈을 통하여 D-클래스를 계산하려면 모든 $n \times n$ 불리언 행렬에 대한 사중 연속곱셈을 수행하여야 한다. 그러나 직접적인 행렬곱셈으로 D-클래스를 계산하면 4절의 이중 연속곱셈 실행결과로부터 $4 \times 4$ 크기의 행렬에 대해서도 많은 시간이 소요되며 $5 \times 5$ 크기의 행렬에 대해서는 실질적으로 계산이 불가능하다는 것을 알 수 있다. 본 절은 벡터기반의 불리언 행렬 곱셈을 통해 D-클래스를 효율적으로 계산할 수 있는 이론과 알고리즘 및 실행결과를 기술한다.

[정리 5] $M_n^n(F)$에 속한 임의의 두 불리언 행렬 $A, B$에 대해 $Q_A, Q_B$를

$$Q_A = \{UAX \mid U, X \in M_n^n(F)\},$$
$$Q_B = \{VBY \mid V, Y \in M_n^n(F)\}$$

로 정의하면, $A \sim_R B \ iff \ Q_A = Q_B$ 이다.

(증명) $I$가 $n \times n$ 단위행렬(identity matrix)이라고 하면, $A = IAI, B = IBI$ 이므로

$A \in Q_A$, $B \in Q_B$ 이다. $Q_A = Q_B$ 라면 $B \in Q_A$, $A \in Q_B$ 이므로, $UAX = B$ 이고 $VBY = A$ 를 만족하는 $U, V, X, Y$가 $M_n^n(F)$에 존재한다. 따라서 $A \sim_R B$이다.

$A \sim_R B$라고 가정하자. 그러면 정의에 의해 $UAX = B$ 이고 $VBY = A$ 인 $U, V, X, Y$가 $M_n^n(F)$에 존재한다. $C$를 $Q_A$에 속한 임의의 불리언 행렬이라고 하면 $C = UAX$을 만족하는 $U, X$이 항상 $M_n^n(F)$에 있으므로 $A$대신 $VBY$를 대입하여 $C = UVBYX$ 을 얻는다. 이 때 $UV$와 $YX$은 모두 $M_n^n(F)$에 속하므로 $C \in Q_B$이며 $Q_A \subseteq Q_B$이다. $Q_B \subseteq Q_A$ 도 같은 논리로 증명되며 따라서 $Q_A = Q_B$ 이다.

[정리 6] $A, B$를 $M_n^n(F)$에 속한 임의의 두 불리언 행렬이라 하고 $Q_A, Q_B, R_A, R_B, M_A, M_B$ 를

$$Q_A = \{UAX \mid U, X \in M_n^n(F)\},$$
$$Q_B = \{VBY \mid V, Y \in M_n^n(F)\},$$
$$R_A = \{AX \mid X \in M_n^n(F)\},$$
$$R_B = \{BY \mid Y \in M_n^n(F)\},$$
$$M_A = \{C \in M_n^n(F) \mid R_C = R_A\},$$
$$M_B = \{C \in M_n^n(F) \mid R_C = R_B\}$$

으로 정의할 때, $Q_A = Q_B$ 라면 $M_A \cup M_B$에 속한 임의의 두 불리언 행렬 $K, L$, $L$에 대해 $K \sim_R L$이다.

(증명) $K, L \in M_A$ 이거나 $K, L \in M_B$인 경우 [정리 4]와 [정리 5]에 의해 $K \sim_R L$이다. $K \in M_A$ 이고 $L \in M_B$이라 가정하자. $R_A = R_B$ 라면 $M_A = M_B$가 되어 앞의 경우와 같이 [정리 4]와 [정리 5]에 의해 $K \sim_R L$ 이다. $R_A \neq R_B$라고 가정하자. 이 경우 $M_A \cap M_B = \varnothing$ 가 된다. 주어진 조건이 $Q_A = Q_B$ 이므로 [정리 5]에 의해 $A \sim_R B$이고, $\sim_R$은 동치관계이므로 $K \sim_R L$이다.

[정리 5-6]은 $M_n^n(F)$의 임의의 두 행렬 $A, B$에 대해 $\{UAX \mid U, X \in M_n^n(F)\} = \{VBY \mid V, Y \in M_n^n(F)\}$ 이면 $\{AX \mid X \in M_n^n(F)\}$ 나 $\{BY \mid Y \in M_n^n(F)\}$ 집합을 생성하는 모든 불리언 행렬이 동치관계, 즉 같은 D-클래스에 속한다는 것을 의미한다. [그림 2]는 이러한 사실을 이용하여 설계한 D-클래스 계산 알고리즘의 개요를 보이고 있다. 알고리즘은 먼저 [그림 1]의 알고리즘에 의해 트리플 집합 $\mathbb{M}$을 생성한 후, $\mathbb{M}$에서 임의의 $(S_c, S_M, \mathbb{C}_A)$ 트리플을 선택하여 세 번째 원소

```
M = ∅
for each boolean matrix A in M_n^n(F)
   S_c = ∅
   for each row vector v in V(n)
      compute an Av^T vector
      insert the vector into S_c
   if (S_c ∉ M[0])
      S_M = ∅
      insert A into S_M
      for each matrix M synthesized from S_c
         S_r = ∅
         for each row vector v in V(n)
            compute an vM vector
            insert the vector into S_r
         insert S_r into C_A
      insert a triple (S_c, S_M, C_A) into M
   else
      insert A into M(S_c)[1]

while (M ≠ ∅)
  take one S_c from M[0]
  remove M(S_c) from M
  insert M(S_c) into D_Sc
  for each triple (S'_c, S'_M, C'_A) in M
    if (C'_A = M(S_c)[2])
      remove M(S'_c) from M
      insert M(S'_c) into D_Sc
  insert D_Sc into DClassList
```

* $S_c$ : a set of $n$-dimensional column vectors
* $S_r$ : a set of $n$-dimensional row vectors
* $S_M$ : a set of $n \times n$ boolean matrices
* $\mathbb{C}_A$ : a class of $S_r$'s for $S_c$
* $\mathbb{M}$ : a set of $(S_c, S_M, \mathbb{C}_A)$ triples
* $\mathbb{M}[i]$ : a set of the $i$'th elements in all triples
* $\mathbb{M}(S_c)$ : a triple whose first element matches $S_c$
* $\mathbb{M}(S_c)[i]$ : the $i$'th element in $\mathbb{M}(S_c)$
* $\mathbb{D}_{S_c}$ : a set of $(S_c, S_M, \mathbb{C}_A)$ triples whose $S_M$ belong to the same D-class

<그림 2> D-클래스 계산 알고리즘

가 $\mathbb{C}_A$와 같은 트리플을 모두 찾아 같은 그룹으로 묶는다. 이 그룹이 하나의 D-클래스를 나타내며, 그룹의 각 트리플에 있는 $S_M$을 유니온하여 D-클래스를 얻는다. 그룹에 속한 트리플은 M에서 제거한다. 이 과정을 M에 트리플이 남아 있는 동안 반복하며 루프가 종료되면 모든 D-클래스를 찾게 된다.

알고리즘은 위와 동일한 환경에서 실행하였으며 <표 3>은 실행시간과 해당 크기의 불리언 행렬 집합에 존재하는 D-클래스 수를 보이고 있다. $M_5^5(F)$에 대한 D-클래스 수는 새로운 결과로서 이론상 600개 이상일 것으로 추정되고 있었으며 본 논문의 결과는 이 예측과 부합한다.

<표 3> 실행시간 비교

| 행렬크기 | D-클래스 계산 알고리즘 | D-클래스 수 |
|---|---|---|
| 2×2 | 0.02 초 | 4 |
| 3×3 | 0.049 초 | 11 |
| 4×4 | 3.013 초 | 60 |
| 5×5 | 168877 초 | 877 |

## 6. 결론 및 향후 연구방향

불리언 행렬과 관련된 대부분의 연구와 응용은 두 불리언 행렬의 최적화된 곱셈에 중점을 두고 있으나, D-클래스 계산은 모든 불리언 행렬 사이의 이중 연속곱셈을 요구한다. 이러한 곱셈은 NP-완전 문제로서 공간 및 시간 복잡도 개선에 근본적인 어려움이 내포되어 있다.

본 논문은 기존에 제시된 두 불리언 행렬의 최적 곱셈 알고리즘이 모든 불리언 행렬에 대한 곱셈을 해야 하는 경우 부적합함을 보이고, 모든 불리언 행렬 사이의 곱셈과 이중 연속곱셈을 보다 효율적으로 개선할 수 있는 이론을 제시하였다. 또한 이론을 적용한 불리언 행렬 연속곱셈의 공간 및 시간 복잡도를 분석하고, 알고리즘 실행결과를 통해 이론을 적용할 경우 실제 실행결과에 많은 개선이 있음을 논하였다. 본 연구에서 제시한 벡터 기반 행렬 곱셈 이론은 분산 및 병렬 컴퓨팅 등의 실행환경에 상관없이 D-클래스 계산처럼 모든 행렬에 대한 곱셈이 필요한 경우 성능개선 등을 위해 적용가능 하다. 또한 원소가 세 개 이상의 값을 갖는 일반 행렬에도 이론을 확대 적용할 수 있다.

벡터 기반 행렬 곱셈 알고리즘의 개선을 위해 중요한 문제는 각 행렬에 모든 벡터를 곱하여 얻는 벡터집합의 집합을 효율적으로 계산할 수 있는 방법과 연속곱셈을 할 때 벡터집합으로부터 불리언 행렬을 조합하여 다시 벡터 곱셈을 하지 않고 벡터집합으로부터 바로 연속곱셈 결과를 알아 낼 수 있는 방법을 찾는 것이다. 행렬의 크기가 커짐에 따라 각 행렬에 모든 벡터를 곱하여 얻는 벡터집합의 수는 <표 4>에서 보듯이 가능한 벡터집합의 전체 개수와 비교하여 극히 작아진다. 이 사실은 모든 불리언 행렬에 대한 곱셈이 다차원 함수의 계산복잡도로 수행될 수도 있는 가능성을 제시하며, 각 행렬에 모든 벡터를 곱하지 않고도 다른 방법을 통해 벡터집합의 집합을 효율적으로 찾아낼 수 있을 경우 이러한 가능성은 보다 구체화 될 것이다. 연속곱셈을 위해 벡터집합으로부터 불리언 행렬을 조합하는 오버헤드는 실행결과에서 보듯 실행시간에 큰 영향을 미치지 않으며, 보다 중요한 것은 조합되어 나오는 행렬의 수로서 이 수에 비례하여 실행시간이 늘어난다. 본문에 기술한 연속곱셈 알고리즘도 많은 중복곱셈을 회피하도록

<표 4> 벡터집합 수 비교

| 행렬 크기 | | 생성 벡터집합 수 ($a$) | 총 벡터집합 수 ($b$) | $a/b \times 100$ (%) |
|---|---|---|---|---|
| $n \times m$ | $m \times k$ | | | |
| 2×2 | 2×2 | 7 | 16 | 43.75 |
| 2×3 | 3×3 | 7 | 16 | 43.75 |
| 3×3 | 3×3 | 55 | 256 | 21.48 |
| 3×4 | 4×4 | 61 | 256 | 23.83 |
| 4×4 | 4×4 | 1324 | 65536 | 2.02 |
| 4×5 | 5×5 | 2106 | 65536 | 3.21 |
| 5×5 | 5×5 | 120633 | 4294967296 | 0.0028 |

설계 하였지만, 각각의 조합된 행렬에 다시 모든 벡터를 곱할 때 나타나는 중복 곱셈은 근본적으로 다른 방법을 통해 연속곱셈 결과를 벡터집합으로부터 얻을 수 있어야 방지할 수 있을 것이다.

본 논문은 이론을 적용한 순차 알고리즘을 설계하여 아직 알려지지 않았던 5×5 불리언 행렬에 대한 D-클래스를 계산하였다. 앞으로 6×6 이상 크기의 불리언 행렬에 대한 D-클래스를 찾기 위해 기존 알고리즘의 최적화, 이론을 적용한 분산 또는 병렬 알고리즘의 설계 및 구현, 실행환경에 맞는 이론 정립 등에 대한 연구가 필요하다.

## 참 고 문 헌


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

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

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

[4] 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.

[5] 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.

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

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

[9] Macii, E., "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.

[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] 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.

[12] Rim, D. S. and Kim, J. B., "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.

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

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

[15] 한재일, "D-클래스 계산을 위한 n x n 불리언 행렬의 효율적 곱셈 알고리즘", 한국정보과학회 추계학술대회 논문집, 제32권, 제2호, pp. 952-954, 2005.

**한 재 일** (Jae-Il Han)

- 1983년 2월 연세대학교 수학과, 학사
- 1986년 5월 미국 Syracuse 대학교 전산학과, 석사
- 1992년 5월 미국 Syracuse 대학교 전산학과, 박사.
- 1993년~1995년 한국전자통신연구원 선임연구원
- 1995년~현재 국민대학교 컴퓨터학부 교수.
- 관심분야 : 분산처리, 객체지향 시스템, 컴퓨터 이론, 컴퓨터 및 네트워크 보안, RFID/USN 미들웨어 등

**신 범 주** (Bum-Joo Shin)

- 1998년 8월 경북대학교 컴퓨터공학과, 박사
- 1987년~2002년 한국전자통신연구원 연구원.
- 2002년~2006년 밀양대학교 컴퓨터공학부, 교수
- 2006년~현재 부산대학교 바이오정보전자공학과, 교수
- 관심분야 : 분산처리, 시스템소프트웨어 등