D-클래스 계산을 위한 $n \times n$ 불리언 행렬의 효율적 곱셈 알고리즘

한재일
국민대학교
jhan@kookmin.ac.kr

An Algorithm for Efficient Multiplication of $n \times n$ Boolean matrices for D-Class Computation

Jae-Il Han
School of Computer Science, Kookmin University

요 약

D-클래스는 $n \times n$ 불리언 행렬의 집합에서 특정 관계(relation)에 따라 동치(equivalent) 관계에 있는 불리언 행렬의 집합으로 구성된다. D-클래스 계산은 $n \times n$ 불리언 행렬의 전체 집합을 대상으로 이 집합에서 조합할 수 있는 모든 두 $n \times n$ 불리언 행렬 사이의 곱셈을 기본적으로 요구한다. 그러나 불리언 행렬에 대한 대부분의 연구는 두 개의 불리언 행렬에 대한 효율적인 곱셈에 집중되었으며 모든 $n \times n$ 불리언 행렬 사이의 곱셈에 대한 연구는 최근에야 소수가 보이고 있다. 두개의 $n \times n$ 불리언 행렬 곱셈에 대해 최적화된 알고리즘은 현재 알려져 있으나, 모든 $n \times n$ 불리언 행렬 사이의 곱셈에 대해 제시된 알고리즘은 아직 실행시간이 크게 향상되지 못하고 있으며 많은 개선과 연구가 필요하다. 본 논문은 개별적인 $n \times n$ 불리언 행렬 곱셈 대신 하나의 $n \times n$ 불리언 행렬과 불리언 행렬 집합과의 곱셈을 다루고 또한 이 곱셈에서 계산되는 모든 $n \times n$ 불리언 행렬을 집합으로 표현하는 방법을 통해 D-클래스 계산을 보다 효율적으로 할 수 있는 알고리즘에 대해 논한다.

1. 서 론

불리언 행렬은 파싱 알고리즘[1, 2], 선형 우선순위 함수 계산[3], 논리 최적화[4], 논리 회로[5] 등에서 다양하게 사용되고 있다. 이러한 대부분의 응용이 두 개의 $n \times n$ 불리언 행렬 곱셈에 기반을 두고 있는 반면 D-클래스[6] 계산과 같은 문제는 $n \times n$ 불리언 행렬의 전체 집합을 대상으로 모든 가능한 두 $n \times n$ 불리언 행렬 사이의 곱셈을 필요로 한다.

불리언 행렬에 대한 많은 연구가 수행 되었으나[7-10] 모두 두개의 $n \times n$ 불리언 행렬에 대한 효율적 곱셈과 여러 분야에서의 응용을 다루고 있다. 두 개의 $n \times n$ 불리언 행렬의 곱셈에 대한 최적의 알고리즘은 수년 전 개발 되었으나[11], 모든 두 $n \times n$ 불리언 행렬 사이의 곱셈은 NP-완전 계산 복잡도와 상대적으로 극히 적은 필요성으로 인해 최근에야 기초적인 연구 결과가 보이고 있다[12, 13]. 그러나 모든 $n \times n$ 불리언 행렬 사이의 곱셈에 대해 제시된 알고리즘은 아직 실행시간이 크게 향상되지 못하고 있어 효율적 알고리즘의 설계를 위하여 많은 개선과 연구가 필요하다.

본 논문은 D-클래스 계산을 위해 모든 가능한 두 $n \times n$ 불리언 행렬 사이의 곱셈을 보다 효율적으로 할 수 있는 개선된 알고리즘에 대해 논한다. 본 논문에서 제시한 알고리즘은 집합을 이용하여 많은 불리언 행렬 곱셈 결과를 개별 불리언 행렬 단위로 다루지 않고 집합처럼 처리함으로써 기존에 제시된 알고리즘과 비교하여 실행시간과 메모리 사용에 있어 상당한 개선을 이루었다. 본 논문의 구성은 다음과 같다. 2장은 관련연구에 대하여 기술하며 3장은 모든 가능한 두 $n \times n$ 불리언 행렬 사이의 곱셈 알고리즘을 집합 기반으로 설계하기 위한 이론을 논한다. 4장은 알고리즘과 실행결과를 기술하며 5장은 결론 및 향후 연구방향에 대하여 논한다.

2. 관련 연구

$n \times n$ 불리언 행렬의 곱셈에 대한 연구는 위에서 언급한 바와 같이 대부분 단지 두 $n \times n$ 불리언 행렬의 곱셈에 대하여 수행되었다[7-10]. [11]에서 제시한 두 $n \times n$ 불리언 행렬 곱셈 최적화 알고리즘은 A x B 연산에서 앞의 불리언 행렬 A를 불리언 행렬 B의 행을 OR-연산을 하는 지시자로 사용하여 곱셈 결과를 얻는다.

D-클래스는 다음과 같이 정의된다[6]. F={0, 1}일때

$$M_n(F) = \{ A : A \text{ is an } n \times n \text{ matrix whose element is in } F \}$$

$$D_A = \{ B \in M_n(F) : \exists C, X, Y, U, V \in M_n(F) \text{ such that } AX = C,\ CY = A,\ UC = B,\ VB = C \}$$

$$D\text{-class} = \{ D_A : A \in M_n(F) \}$$

위 정의를 분석하면 D-클래스를 얻기 위한 가장 기본적인 계산은 두 개의 $n \times n$ 불리언 행렬의 곱셈이나 D-클래스의 효율적인 계산에 중요한 것은 $M_n(F)$에 속한 모든 두 $n \times n$ 불리언 행렬 사이의 빠른 곱셈이다. [12, 13]는 D-클래스를 계산하기 위하여 모든 가능한 두 $n \times n$ 불리언 행렬 사이의 곱셈에 대한 알고리즘을 제시하였다. 그러나 [12, 13]에서 제시한 곱셈 알고리즘은 여러 개선 방안에도 불구하고 실행시간이 크게 개선되지 못하여 현재 알려지지 않은 D-클래스 계산에 사용할 수 없다.

3. 집합 기반의 $n \times n$ 불리언 행렬 곱셈

D-클래스의 원소 D_A는 $M_n(F)$의 불리언 행렬 A와 동등 관계에 있는 모든 $M_n(F)$에 속한 불리언 행렬의 집합을 의미한다. D-클래스 계산은 D_A 계산이 핵심 요소이며 위의 D_A 정의로부터 다음과 같은 변형된 D_A 정의를 얻을 수 있다.

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

이 정의에서 UAX 계산은 불리언 행렬 A에 $M_n(F)$의 각 불리언 행렬을 곱하고 그 결과와 $M_n(F)$의 각 불리언 행렬에 대한 곱셈을 반복하여 요구한다. UAX 계산 결과로 얻은 불리언 행렬 B에 대해서도 같은 방법으로 VBY 계산을 요구한다. 이 관계를 분석해 보면 이와 같이 개별 불리언 행렬을 반복 계산하는 것과, A에 $M_n(F)$의 모든 불리언 행렬 U와 X를 곱하여 얻을 수 있는 B의 집합을 구하고 이 집합에 속한 각 불리언 행렬 B에 $M_n(F)$의 모든 행렬 V와 Y를 곱하여 얻은 집합에 A가 속하는지를 보고 A와 동치(equivalent) 관계에 있는 불리언 행렬 B를 계산하는 것이 같은 결과를 주는 것을 쉽게 증명할 수 있다.

후자와 같이 집합을 이용하여 동치관계에 있는 불리언 행렬을 알아내기 위해서는 UAX 와 VBY 로부터 유도되는 집합에 대한 효율적 계산이 중요하다. UAX 와 VBY 에서 유도되는 집합은 먼저 A나 B에 $M_n(F)$의 모든 행렬을 곱하여 얻을 수 있는 집합을 계산한 후 그 집합에 있는 각 행렬에 $M_n(F)$의 모든 행렬을 곱하여 계산하여야 한다. 따라서 가장 기본적으로 요구되는 것은 하나의 불리언 행렬에 $M_n(F)$의 모든 행렬을 곱해서 집합을 계산할 때 효율적인 계산방법을 찾는 것이다.

다음 정리는 하나의 $n\times n$ 불리언 행렬에 $M_n(F)$의 모든 $n\times n$ 불리언 행렬을 곱하여 얻을 수 있는 $n\times n$ 불리언 행렬의 집합 계산이 행렬과 행렬 곱셈 대신 행렬과 벡터 곱셈으로 수행될 수 있는 근거를 제공한다. A가 $n\times n$ 불리언 행렬일 때 A_i와 A^i는 각각 A 행렬의 i 번째 행과 열을 의미하며, 본 논문은 다음 정의를 사용한다.

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

$$V = \{ (e_0\ e_1 \Lambda\ e_{n-1}) \mid e_i \in F \text{ for } 0 \le i \le n-1 \}$$

$$V^n = \{ [v_0^c\ v_1^c \Lambda\ v_{n-1}^c] \mid v_i \in V \text{ for } 0 \le i \le n-1 \}$$

$$Av = \{ (A_0 v^c\ A_1 v^c\ \Lambda\ A_{n-1} v^c)^c \mid v \in V \}$$

[정리] A는 $M_n(F)$에 속한 임의의 $n\times n$ 불리언 행렬이며, M과 T는 다음과 같이 정의한다.

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

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

이때, $\mathrm{M} = T^n$ 이다.

(증명) $P \in \mathrm{M}$ 이라 하자. T는 $n\times n$ 불리언 행렬 A에 n개의 원소를 가지는 모든 벡터 v를 곱한 집합이므로 0과 $n-1$ 사이의 모든 i에 대해 $P^i \in T$ 이며, 따라서 $P \in T^n$ 이다.

$P \in T^n$ 이라 하자. 만약 $P \notin \mathrm{M}$ 이라면 M에 속하는 모든 $n\times n$ 불리언 행렬 D에 대하여 $P \neq D$ 이고 따라서 P의 어떤 열 P^i가 모든 행렬 D의 i번째 열 D^i와 같지 않다. $M_n(F)$는 모든 $n\times n$ 불리언 행렬의 집합이므로

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

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

V : a set of all n-element vectors whose element belongs to F
SV: a set of vectors

for each boolean matrix A in $M_n(F)$
  $SV = \varnothing$
  for each n-element vector v in V
    compute an Av vector
    insert the vector into SV
  store the set SV for boolean matrix A

[그림 1] 알고리즘 개요

이므로 V 의 어떤 벡터 $v_k\ (0 \le k \le n-1)$ 에 대하여 $P^i = (A_0 v_k^c, A_1 v_k^c, \Lambda, A_{n-1} v_k^c)^c$ 가 되어 모순이 된다.

이 정리에 의해 $M_n(F)$에 속한 어떤 $n\times n$ 불리언 행렬이 주어졌을 때 $M_n(F)$의 2^{n^2} 개 불리언 행렬을 곱하여 집합을 계산하지 않고 2^n 개의 벡터를 곱하여 나오는 행이나 열 벡터의 집합을 다룬다. 이 집합에 있는 벡터를 n개의 열이나 행으로 모두 조합하면 2^{n^2} 개 불리언 행렬을 곱하여 얻은 불리언 행렬 집합과 동일한 결과를 얻는다.

4. 알고리즘 및 실행 결과

[그림 1]은 정리를 이용한 알고리즘의 개요를 보이고 있다. 알고리즘은 $M_n(F)$에 속한 각 $n\times n$ 불리언 행렬에 대하여 n개의 원소를 가지는 모든 2^n 개의 벡터를 곱한다. 불리언 행렬과 각 벡터의 곱셈으로부터 얻은 벡터는 현 불리언 행렬의 집합에 삽입한다. 모든 벡터와의 곱셈이 종료되면 현 불리언 행렬과 집합 원소를 저장한다.

[표 1] 위 알고리즘을 구현한 프로그램을 2개의 Zeon CPU, 1GB RAM, 250GB HDD, Fedora 9.0 환경에서 실행하여 얻은 결과를 보이고 있다. 알고리즘 Java 언어로 구현되었으며 불리언 행렬과 벡터의 곱셈은 비트 사이의 논리 연산으로 수행한다. 알고리즘의 계산 복잡도는 NP-완전 문제이나 [표 1]에 보는 것처럼 실제 실행시간은 기존의 알고리즘[12]과 비교하여 많은 개선을 보이고 있다.

5. 결론 및 향후 연구방향

불리언 행렬에 대한 기존의 연구는 두 개의 $n\times n$ 불리언 행렬의 효율적 곱셈과 그 응용에 중점을 두고 있다. 반면 D-클래스[6] 계산과 같은 문제는 $n\times n$ 불리언 행렬의 전체 집합을 대

[표 1] 실행시간 비교

종류 / 행렬크기	기존 알고리즘 (행렬 곱셈)	개선 알고리즘 (집합 곱셈)
2 x 2	0.003 초	0.003 초
3 x 3	0.095 초	0.013 초
4 x 4	1271 초	0.714 초
5 x 5	268435000 초 이상 (3106 일 이상)	17726 초 (4시간 56분)

상으로 모든 가능한 두 $n \times n$ 불리언 행렬 사이의 곱셈을 요구한다. 그러나 모든 두 $n \times n$ 불리언 행렬 사이의 곱셈은 NP-완전 계산 복잡도와 상대적으로 극히 적은 필요성으로 최근에야 소수의 연구 결과가 보이고 있다[12, 13].

본 논문은 D-클래스 계산을 위해 모든 두 $n \times n$ 불리언 행렬 사이의 곱셈을 보다 효율적으로 할 수 있는 개선된 알고리즘에 대하여 논하였다. 본 논문에서 제시한 알고리즘은 집합을 이용하여 많은 불리언 행렬 곱셈 결과를 개별 불리언 행렬 단위로 다루지 않고 집합처럼 처리함으로써 기존에 제시된 알고리즘과 비교하여 실행시간과 메모리 사용에 있어 상당한 개선을 이루었다. 그러나 실행시간을 개선시키기 위한 이론과 알고리즘 최적화 등에 대한 연구가 아직 많이 부족하다. 또한 D-클래스 연구를 활성화 할 수 있을 정도의 정보를 줄 수 있는 크기의 불리언 행렬을 다루기에는 아직 실행시간이 매우 느리거나 불가능한 상황으로서 앞으로 병렬 알고리즘 등에 대한 연구도 필요하다.

[참고문헌]

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

[2] Satta, G., " Tree-adjoining grammer parsing and Boolean matrix multiplication," Computational Linguistics, Vol. 20. No. 2, pp. 173-191,1994

[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] 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

[7] 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

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

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

[10] 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

[11] 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

[12] 신철규, 한재일, " 공유 메모리 기반의 고성능 D-클래스 계산 병렬 알고리즘" , 한국컴퓨터종합학술대회 논문집, Vol. 32 No. 1, pp. 10-12, 2005

[13] 신철규, 한재일, " 내부 순환문 개선을 통한 Linux 기반의 D-클래스 계산 고효율 순차 알고리즘" , 한국 SI 학회 춘계 학술대회 논문집, Vol. 1, pp. 526-531, 2005