제21회 한국정보처리학회 춘계학술발표대회 논문집 제11권 제1호 (2004. 5)

D-클래스 계산 알고리즘에 관한 연구

신철규, 한재일
국민대학교 컴퓨터학부
e-mail: supply8@cs.kookmin.ac.kr

A Study on the D-Class Computing Algorithm

Chul-Gyu Shin, Jae-Il Han
School of Computer Science, Kookmin University
e-mail: supply8@cs.kookmin.ac.kr

요 약

D-클래스는 원소가 0과 1값을 가지는 $n \times n$ 불리언 행렬에서 특정 관계(relation)에 따라 동치(equivalent) 관계에 있는 $n \times n$ 행렬의 집합을 의미한다. D-클래스의 계산은 NP-완전 문제로서 보안에 응용될 수 있는 가능성을 가지고 있으나 계산 복잡도로 인해 현재 극히 제한된 크기의 행렬에 대한 D-클래스만이 알려져 있다. 본 논문은 이러한 D-클래스의 계산을 효율적으로 할 수 있는 알고리즘의 설계와 실행 결과에 대하여 논한다.

1. 서론

집합 F는 0과 1의 두 원소로 구성되며, $M_n(F)$는 행렬의 원소가 F의 원소를 값으로 갖는 모든 n × n 불리언 행렬식의 집합이다. $M_n(F)$에 속하는 임의의 두 행렬 A, B에 대하여 $AX=C, CY=A, UC=B, VB=C$를 만족시키는 C, X, Y, U, V가 $Mn(F)$에 존재할 때 A와 B는 R_D 관계가 성립한다. R_D는 반사성(reflexive), 대칭성(symmetric), 전이성(transitive)을 만족하는 동치(equivalent) 관계이며, D-클래스는 R_D에 의하여 동치 관계에 놓여 있는 $n \times n$ 불리언 행렬의 집합으로 정의한다. 따라서, $M_n(F)$에 속한 임의의 행렬 A에 대한 D -클래스는 다음과 같이 정의된다[1].

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

D-클래스의 계산은 NP-완전 문제로서 보안에 응용될 수 있는 가능성을 가지고 있다. 그러나 계산 복잡도로 인해 현재 극히 제한된 크기의 행렬에 대한 D-클래스만이 알려져 있다. 본 논문은 D-클래스의 계산을 효율적으로 할 수 있는 알고리즘의 설계와 구현, 실행결과에 대하여 논한다.

본 논문의 구성은 다음과 같다. 제 2장은 D-클래스 계산 알고리즘의 문제점 및 관련 연구를 살펴보고, 3장에서는 개선된 알고리즘의 설계 및 실행결과에 대하여 논하며, 4장은 결론 및 향후 과제에 대하여 기술한다.

```
initialize D_A  to an empty set

for each A in M_n(F)----------①
  for each X in M_n(F)) ----------②
    C = AX
    for each Y in M_n(F) ----------③
      if A = CY
        for each U in M_n(F)) ----------④
          B = UC
            for each V in M_n(F)----------⑤
              if C = VB
                insert B to D_A
```

[그림 1] 알고리즘1

2. 문제점 및 관련 연구

[그림 1]은 D-클래스 정의에 따라 D-클래스를 계산하는 순차 알고리즘을 보이고 있다. 각 단계에서 각각 한 개의 행렬이 선택되고, 각 순환문에서 2^m 개의 행렬이 선택된다. 이것은 순차적으로 ①-②에서 AX 행렬의 연산을 하여 C행렬을 얻는 과정이 $O(2^{2m})$이 된다. 이 결과에 ③번 $O(2^m)$의 Y와 CY 연산 과정에 의

initialize D_A to an empty set
initialize S_B to an empty set

for each A in $M_n(F)$ ----------①
  for each U in $M_n(F)$ ----------②
    $T = UA$
    for each X in $M_n(F)$ ----------③
      if TX in $M_n(F)$
        insert TX to S_B

  for each B in $M_n(F)$ ----------④
    if B is in S_B
    for each V in $M_n(F)$----------⑤
      $T_1 = VB$
      for each Y in $M_n(F)$----------⑥
        if A equal T_1Y
          insert B to D_A
          remove B from $M_n(F)$

[그림 2] 알고리즘 2

해 $O(2^{3m})$이 된다. ④, ⑤번 UC, VB를 연산하는 과정에서 각각 $O(2^m)$의 연산이 되어 총 연산은 $O(2^{5m})$의 결과로 $O(2^m)$이 되어 NP-완전문제가 된다[2].

일반 행렬이나 불리언 행렬 계산에 대한 연구가 많이 있으나[1,4,5,6] 적용할 수 있는 문제 범위가 한정되어 있어 D-클래스의 효율적인 계산을 위해서는 D-클래스 계산에 최적화된 알고리즘이 필요하다.

3. D-클래스 계산 알고리즘 및 실행결과

위에서 언급한 문제를 해결하기 위하여 먼저 위의 D-클래스 정의에서 $VB = C$ 식을 $CY = A$ 식에 대입하고 $AX = C$ 식을 $UC = B$의 식에 대입하여 $VBY = A$ 식과 $UAX = B$ 식을 얻을 수 있다. 따라서 D-클래스는 다음과 같이 재정의 될 수 있다.

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

[그림 2]는 이 정의를 기반으로 D-클래스를 계산하는 알고리즘이다.

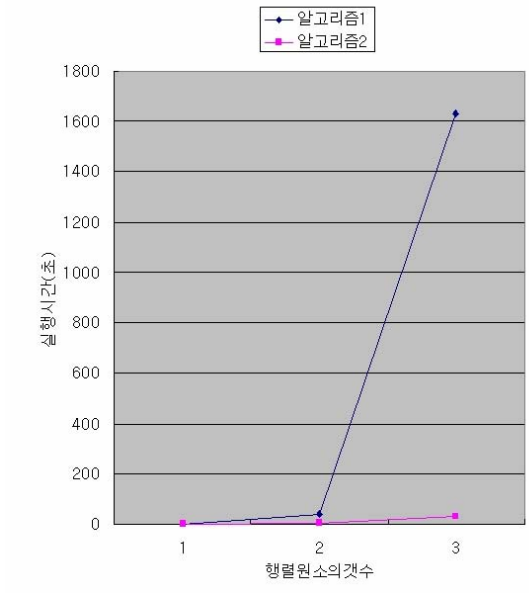
각 단계에서 각각 한 개의 행렬이 선택되고, 각 순환문에서 2^m 개의 행렬이 선택된다. 한 개의 행렬 A가 선택된 후 UAX 연산이 $O(2^{2m})$ 실행되며, 앞에서 연산된 결과를 이용하여 VBY 연산은 $O(2^{3m})$ 실행된다. A는 2^m 개가 있으므로 $O(2^m(2^{2m} + 2^{3m}))$이 되어 계산복잡도는 $O(2^m)$이 된다.

그러나 알고리즘2와 알고리즘1과의 차이점은 각 단계별로 알고리즘1처럼 하나의 행렬을 선택, 행렬을 선택하는 과정이 5번 실행된 결과인 $O(2^{5m})$이 되는 것이 아니라, ①-③의 과정에 의해 $UAX = B$ 의 연산결과로 나온 B값을 S_B에 저장해주고, ④-⑥의 과정에서 ④번의 루프문에서 선택된 B가 S_B에 저장된 것일 때 ⑤-⑥번 연산을 통해 D_A클래스인지 검사하여 D-클래스라면 해당 행렬 A, B 값으로 선택되지 않도록 한 것이다. 이것은 $O(2^{4m})$이 되어 알고리즘1 보다 실행시간이 단축 된다.

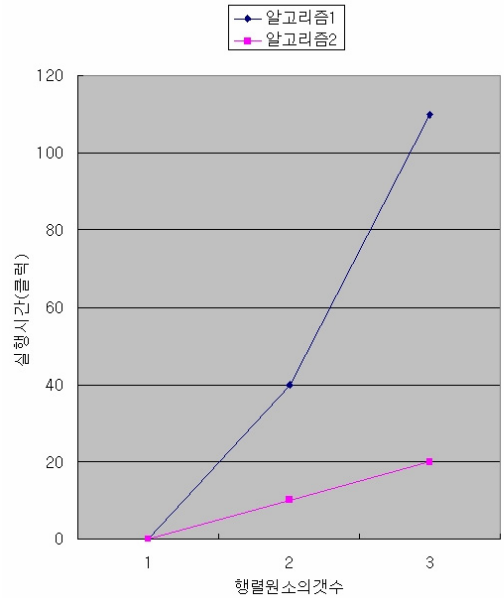
행렬의 원소 개수가 n일 때 알고리즘2와 알고리즘1의 계산복잡도는 모두 $O(2^{2n})$이 되지만 [그림3] 에서 보는 것처럼 실행 시간은 많은 차이를 보이고 있다. 알고리즘1의 일반적인 행렬 연산과 알고리즘 2의 비트연산의 비교는 [그림 4]에 보이고 있다.

[그림 3]과 [그림 4] 을 보면 $M_1(F)$, $M_2(F)$ 행렬에서는 알고리즘1, 알고리즘2를 실행 시켰을 때 실행시간의 차이가 많지 않다는 걸 알 수 있다. 하지만 $M_3(F)$ 행렬를 실행 시켰을 때에는 실행시간이 50배 이상 차이가 나고 있다. 이것은 $M_n(F)$ 이 4, 5, 6…등 숫자가 커짐에 따라 실행시간의 차이가 더욱 커질 것이다.

위에서의 결과를 보다 향상시키기 위해서 알고리



[그림 3] 실행결과 1



[그림 4] 실행결과 2

즘2를 쓰레드 3개를 이용하여 분산처리 한 것이 [그림 5] 이다. 이 알고리즘은 *UAX*의 연산 중에 *A* 값이 선택되고 *U* 값을 선택할 때 쓰레드 3개를 만들어 동시에 *U*값을 1/3씩 선택하여 *UAX*의 연산 결과를 S_B에 저장한다. 이렇게 *UAX* 연산 과정이 모두 끝난 후에 S_B 의 결과를 1/3로 *VBY*의 과정에서도 *B*를 1/3로 분산하여 쓰레드 3개를 만들어 처리 하게 해준다.

현재 작업을 1/3로 나눠주는 방법으로 첫번째 0부터 3씩 증가, 두번째 1부터 3씩 증가, 세번째 2부터 3씩 증가 하는 방식으로 되어 있다. 이런 방식은 *UAX* 연산시 연산할 데이터를 1/3씩 나눠주지만, *VBY* 연산시 S_B 중 연산 해야 될 값이 한쪽으로 할당 되어 질 수 있는데, 이것은 성능 향상에 문제가 된다. 이런 문제로 약 1/3정도의 성능향상이 안되고 약 40% 정도의 성능이 향상 되는 것을 [그림 6]에서 볼 수 있다.

행렬의 인덱스를 $i = \sum_{i,j=1}^{n} a_{ij} 2^{(i-1)n+(j-1)}$ 식에 의하여 주었을 때 $M_n(F)$는 다음과 같이 정의할 수 있다.

$$M_n(F) = \{M_i \mid 0 \le i \le 2^n - 1\}$$

따라서 다음과 같이 $M_n^k(F)$ 을 정의할 수 있다.

$$M_n^k(F) = \{ M_i \mid i = k \,(mod\, 3),\quad 0 \le i \le 2^n - 1 \}$$

```
initialize D_A  to an empty set
initialize S_B  to an empty set

for each A in M_n(F)  ----------①
 fork
  thread 1:
    for each U  in  M_n^0(F) ----------②
     T = UA
      for each X in M_n(F)  ----------③
        if TX in M_n(F)
          insert TX to S_B

  thread 2:
    for each U ∈ M_n^1(F) ----------②
      T = UA
      for each X in M_n(F)  ----------③
        if TX in M_n(F)
          insert TX to S_B

  thread 3:
    for each U ∈ M_n^2(F) ----------②
      T = UA
      for each X in M_n(F)  ----------③
        if TX in M_n(F)
          insert TX to S_B
 join

 fork
  thread 1:
    for each B in  M_n^0(F) ----------④
      if B is in S_B
      for each V in M_n(F)----------⑤
        T_1 = VB
        for each Y in M_n(F)----------⑥
          if A equal T_1Y
            insert B to D_A
            remove B from M_n(F)

  thread 2:
    for each B in  M_n^1(F) ----------④
      if B is in S_B
      for each V in M_n(F)----------⑤
        T_1 = VB
        for each Y in M_n(F)----------⑥
          if A equals T_1Y
            insert B to D_A
            remove B from M_n(F)

  thread 3:
    for each B in  M_n^2(F) ----------④
      if B is in S_B
      for each V in M_n(F)----------⑤
        T_1 = VB
        for each Y in M_n(F)----------⑥
          if A equals T_1Y
            insert B to D_A
            remove B from M_n(F)
 join
```

[그림 5] 알고리즘 3

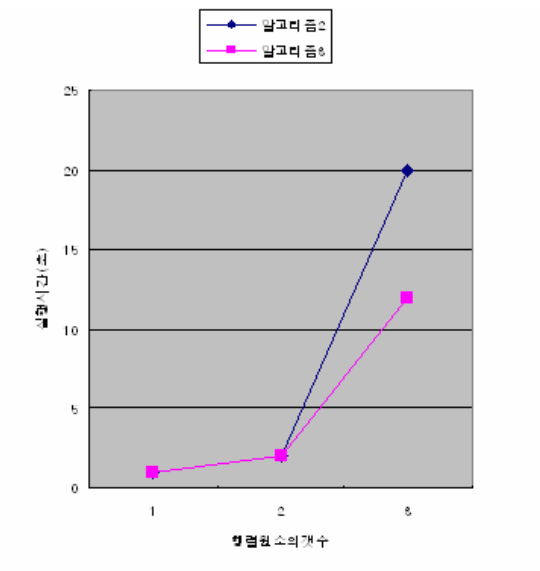
$M_1(F)$ 행렬는 2개의 D-클래스와 2개의 원소가 있고, $M_2(F)$ 행렬는 4 개의 D-클래스와 16개의 원소가 있으며, $M_3(F)$ 행렬는 11개의 D-클래스와 512개의 원소가 있다.

[그림 7]은 알고리즘2를 이용하여 실행시켜 얻어낸 $M_1(F)$, $M_2(F)$, $M_3(F)$ 행렬의 D-클래스 수와 D-클래스에 포함되는 행렬의 개수에 대한 결과를 보이고 있다.

4. 결론

D-클래스는 원소가 0과 1값을 가지는 $n \times n$ 불리언 행렬에서 특정 관계에 따라 동치 관계에 있는 $n \times n$ 행렬의 집합이다. D-클래스는 보안에 응용될 수 있는 가능성을 가지고 있으나 D-클래스 계산 복잡도는 NP-완전 문제로서 현재 극히 제한된 크기의 행렬에 대한 D-클래스만이 알려져 있는 상황이다. 본 논문은 NP-완전 문제인 D-클래스 계산을 효율적으로 할 수 있도록 비트연산을 이용하고 순환문 반복 회수를 줄이는 등 여러 방법을 적용하여 실행시간을 개선한 알고리즘과 실행결과를 제시하였다.

그러나 본 논문의 알고리즘은 아직 많은 개선이



[그림 6] 실행결과 3

요구되며 이를 위해 D-클래스를 정의한 수식 변형, 개선된 병렬 알고리즘 등에 대한 많은 연구가 필요하다. 또한 본 논문은 단일 프로세서를 이용한 순차 알고리즘과 멀티프로세서를 이용한 간단한 병렬 알고리즘을 설계하고 구현 하였으나 앞으로 병렬 컴퓨터나 클러스터 컴퓨팅 등의 환경에서 D-클래스 계산 병렬 알고리즘에 대한 연구가 요구된다[4,7,8].

참고문헌

[1] Dock Sang Rim, Jin Bai Kim, “Tables of D-Classes in the semigroup B_n of the binary relations on a set X wit n-elements,” Bull. Korea Math. Soc. Vol.20. No. 1, 1983, pp. 9-13

[2] Kyle Loudon, Algorithms with C, O’Reilly, 2000

[3] Gene H. Golub, Charles F. Van Loan. Matrix Computation, The Johns Hopkins’ University Press, 1983

[4] Barry Wilkinson, Michael Allen, Parallel Programming with MPI, Prentice Hall, 1999

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

[6] Ki Hang Kim, F. Roush, “Generalized fuzzy matrices,” Fuzzy sets and systems 4, 1980

[7] John Gunnels 외 3인, “A Flexible Class of Parallel Matrix Multiplication Algorithms,” Department of Computer Sciences The University of Texas at Austin

[8] F. Thomson Leighton, Parallel Algorithms And Architectures: Arrays · Trees · Hypercubes, Morgan Kaufmann, 1992

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

[10] Jong-Chuang Tsay, Pen-Yuang Chang, “Design of Efficient Regular Arrays for Matrix Multiplication by Two-Step Regularization,” IEEE Transctions on Parallel and Distributed Systems, Vol. 6, No. 2, February 1995

1. $M_1(F)$ has two D-클래스 (n : 원소)

1	2
0	1
(n = 1)	(n=1)

2. $M_2(F)$ has 4 D-클래스 (n : 원소)

1	2	3	4
0 0 0 0	1 0 0 0	1 0 0 1	1 0 1 1
(n=1)	(n=9)	(n=2)	(n=4)

3. $M_3(F)$ has 11 D-클래스 (n : 원소)

1	2	3	4
0 0 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0 0	1 0 0 0 1 0 0 0 0	1 0 0 1 1 0 0 0 0
(n=1)	(n=49)	(n=162)	(n=144)

5	6	7	8
1 0 0 0 1 0 0 0 1	1 0 0 0 1 0 1 0 1	1 0 0 0 1 0 1 1 1	1 0 0 1 1 0 1 0 1
(n=6)	(n=36)	(n=18)	(n=18)

9	10	11
1 0 0 1 1 0 1 1 1	1 0 0 1 1 0 0 1 1	1 1 0 1 0 1 0 1 1
(n=36)	(n=36)	(n=6)

[그림 7] $M_1(F), M_2(F), M_3(F)$ 행렬의 D-클래스