# 내부점방법을 위한 초마디 열촐레스키 분해의 실험적 고찰*

설동렬** · 정호원*** · 박순달**

# Experimental Study on Supernodal Column Choleksy Factorization in Interior-Point Methods*

Tongryol Seol** · Howon Jung*** · Soondal Park**

## ABSTRACT

The computational speed of interior point method depends on the speed of Cholesky factorization. The supernodal column Cholesky factorization is a fast method that performs Cholesky factorization of sparse matrices with exploiting computer's characteristics. Three steps are necessary to perform the supernodal column Cholesky factorization: symbolic factorization, creation of the elimination tree, ordering by a post-order of the elimination tree and creation of supernodes. We study performing sequences of these three steps and efficient implementation of them.

## 1. 서 론

Karmarkar의 사영법을 비롯하여 아핀법, 장벽법 등의 모든 내부점방법은 대칭양정치 행렬의 선형방정식을 푸는 과정이 필요하다.[4] 일반적으로 대형 문제들은 희소한 특성을 가지는데, 내부점방법에서 나타나는 대칭양정치 행렬도 역시 희소행렬이다.[2] 내부점방법에서는 전체 해법 소요 시간의 많은 부분이 선형방정식을 푸는 데에 소요되어, 효율적으로 선형방정식을 풀어내는 것이 해법의 성능을 크게 좌우한다.

내부점방법에서의 선형방정식은 다음과 같다.

$$(A\Theta A^T)\boldsymbol{x} = \boldsymbol{b}$$

단, $\boldsymbol{x}$과 $\boldsymbol{b}$는 $m\times 1$ 벡터이다. 내부점방법에서는 선형방정식에서 보듯이 대칭양정치 행렬 $A\Theta A^T$가 나타난다. 단, 행렬 $A=(a_{ij})$는 $m\times n$ 행렬이고, $\Theta$는 모든 대각요소의 값은

* 본 연구는 한국과학재단의 목적기초연구과제(과제번호 95-0200-39-01-2)에 의해 지원되었음.
** 서울대학교 산업공학과
*** 고려대학교 경영학과

양인 $n \times n$ 대각행렬이다. 내부점방법의 종류에 따라서 $\Theta$가 다르게 계산된다. 대칭양정치 행렬 $A\Theta A^T$는 $m \times m$ 행렬이다.

촐레스키 분해는 대칭양정치 행렬의 선형방정식을 푸는 데에 사용되는 기법이다.[3] 선형방정식에 촐레스키 분해를 이용하면

$$(A\Theta A^T)\boldsymbol{x} = (LDL^T)\boldsymbol{x} = \boldsymbol{b}$$

이므로, 전방치환 및 후방치환 연산을 통해서 해를 얻을 수 있다. 단, 분해행렬 $L=(l_{ij})$은 대각요소가 1인 $m \times m$ 하삼각행렬이고, $D$는 $m \times m$ 대각행렬이다.

촐레스키 분해를 수행하는 알고리즘에는 비영요소를 보관하고 계산하는 순서에 따라서 열촐레스키 분해, 행촐레스키 분해, 부분행렬 촐레스키 분해가 있으며 이 가운데 열촐레스키 분해가 널리 사용되고 있다. 희소행렬 열촐레스키 분해는 희소행렬이 가지는 구조적인 특성을 이용하여 비영요소만을 보관하여 계산함으로 메모리와 시간을 절약하면서 열촐레스키 분해를 수행하는 방법이다. 촐레스키 분해의 효율적인 수행을 위해서 분해행렬에서 비영요소의 개수를 최소화하려는 순서화 방법들이 희소행렬 열촐레스키 분해에서 중요한 역할을 수행한다.[1,3] 열촐레스키 분해에서 초마디(supernode)는 분해행렬 $L$에서 대각블럭이 밀집행렬이고 비대각블럭의 열구조가 동일한 연속된 열의 집합으로 정의된다.[4] 순서화가 계산량을 줄이는 역할을 수행한다면, 초마디는 컴퓨터의 특성을 활용하도록 하는 역할을 수행한다. 초마디 열촐레스키 분해란 희소행렬 열촐레스키 분해를 수행할 때에 현대의 컴퓨터들이 가지고 있는 계층 메모리 구조의 특성을 활용하여 계산속도의 향상을 가져오는 기법이다.[8] 희소행렬 기법을 열촐레스키 분해에 적용할 경우에 분해행렬 $L$의 비영요소만을 압축하여 보관하므로 비영요소들의 행지수를 보관하는 배열이 필요하다. 초마디를 적용하면 초마디에 속한 열들에 대해서는 동일한 열구조를 가지기 때문에 모든 열들에 대해서 지수 배열을 보관할 필요가 없어서 페이지 폴트(page fault)를 줄이는 자료구조와 루프 전개(loop unrolling)과 같은 기법을 적용할 수 있게 된다.[5,8,9] Jung 등[4]에 의하면 초마디를 적용할 경우에 희소행렬 기법만을 적용한 경우보다 계산속도 향상을 가져온다.

초마디 열촐레스키 분해를 구현하기 위해서는 분해행렬의 자료구조를 파악하여 초마디를 찾아내는 과정과 분해행렬에 대응되는 삭제나무를 생성하여 초마디의 크기를 극대화하는 과정이 필요하다. 구현 방법에 따라서 분해행렬의 자료구조를 파악하는 심볼릭 분해, 초마디의 생성 그리고 삭제나무의 생성을 수행하는 순서를 달리할 수 있다. Liu[7]는 심볼릭 분해를 수행하기 전에 초마디 생성을 먼저 수행함으로써 심볼릭 분해에 걸리는 시간을 절약할 수 있음을 보였다.

본 논문에서는 내부점 방법에 초마디 열촐레스키 분해를 적용할 경우에 대해서 효과적인 구현 방법을 연구하였다. 내부점 방법에서는 심볼릭 분해 또는 삭제나무 생성을 수행할 때에 $A\Theta A^T$의 자료구조를 함께 생성해야 하기 때문에 $A\Theta A^T$의 자료구조를 생성하는 데에 걸리는 시간을 고려하였다. 초마디 열촐레스키 분해를 구현하는 방법은 심볼릭 분해를 언제 수행하는가에 따라서 세 가지로 결정된다. 본 논문에서는 초마디 열촐레스키 분해를 구현하기 위한 이들 세 가지 방법의 장단점을 비교하여 분석하였다. 즉, 심볼릭 분해, 삭제나무 생성 그리고 초마디 생성에 대해 기존에 연구되었던 방법

들을 정리하고, 이러한 방법들이 세 가지의 수행 순서에 따라서 조합되었을 때에 나타나는 결과를 실험적으로 분석하였다.

## 2. 심볼릭 분해와 초마디의 생성

Liu 등[7]은 두 가지의 초마디 생성 방법을 제시하였다. 첫 번째 방법은 심볼릭 분해를 먼저 수행해서 분해행렬 $L$의 자료구조를 구한 다음, 이것을 이용해서 초마디를 생성하는 방법이다. 두 번째 방법은 삭제나무를 이용하여 초마디를 먼저 생성한 다음, 초마디를 활용하면서 심볼릭 분해를 수행하는 방법이다. 어느 방법이나 초마디의 크기를 극대화하기 위해서 삭제나무의 후순서로 다시 순서화를 수행한다. 그런데, 삭제나무를 생성하는 데에는 $L$의 자료구조를 이용해서 생성하는 방법과 $AA^T$의 자료구조를 이용해서 생성하는 방법이 있다. 그리고, 두 번째 방법에서는 삭제나무를 이용해서 심볼릭 분해를 수행할 수 있다. 결국 심볼릭 분해와 초마디 생성 방법은 다음과 같은 세 가지 방법이 가능하다.

- 방법 1: 심볼릭 분해 → 삭제나무 생성 및 후순서화 → 초마디 생성
- 방법 2: 삭제나무 생성 및 후순서화 → 심볼릭 분해 → 초마디 생성
- 방법 3: 삭제나무 생성 및 후순서화 → 초마디 생성 → 심볼릭 분해

Liu 등[7]은 방법 3이 선형방정식을 푸는 데에 있어서 가장 효율적인 것으로 주장하였다. 그러나, 내부점방법에서는 Liu 등이 연구한 상황과 조금 다르다. Liu 등은 선형방정식이 이미 주어진 상황에 대해 고려했지만, 내부점방법에서는 행렬 $AA^T$에 대해서 촐레스키 분해를 수행하기 때문에 행렬 $A$로부터 행렬 $AA^T$의 구조를 계산하는 과정이 언제나 먼저 수행되어야 한다. 따라서, 삭제나무를 생성하거나 심볼릭 분해를 수행할 때에 행렬 $AA^T$의 구조를 계산하는 데에 소요되는 시간을 함께 고려하여 한다.

**삭제나무 생성**

삭제나무는 다음과 같이 정의된다.

정의 1. 삭제나무

$$\text{PARENT}(j) = Min\{\, i \mid l_{ij} \neq 0,\ i > j \},$$
$$j = 1, 2, \cdots, m$$

단, PARENT($j$)는 $j$번째 점의 부모점이고, 완벽성을 위해서 마지막 점 $m$의 부모점는 PARENT($m$) = 0으로 정의한다.

분해행렬 $L$의 비영요소 구조를 이미 알고 있을 경우에는 정의 1을 이용하여 간단하게 삭제나무를 생성할 수 있다. 특히 $L$의 구조가 열별로 저장되어 있을 경우 각 열의 첫번째 비영요소가 정의 1에 의한 부모점이기 때문에 유리하다. 다음은 정의 1을 이용한 삭제나무 생성 알고리즘이다. 벡터 **PARENT**는 부모점을 보관한다.

알고리즘 1. $L$ 구조로부터 삭제나무를 생성하는 알고리즘[1]

---

1:for $i$ := 1 to $n$ do

```
2:  PARENT[ i] := 0;
3:  for k < i and l_ik ≠ 0 do
4:  if PARENT[ k]=0 then PARENT[ k]:= i;
5:  end
6:end
```

심볼릭 분해와 초마디 생성을 위한 방법 1의 경우에는 알고리즘 1을 사용하면 되지만, 방법 2와 방법 3의 경우에는 아직 분해행렬 $L$의 구조를 모르는 상황이기 때문에 $AA^T$ 구조로부터 삭제나무를 생성해야 한다.

정리 1. $l_{ij} \neq 0$은 점 $j$가 삭제나무에서 $m_{ik} \neq 0$ 인 어떤 점 $k$의 조상(ancestor)이 될 필요충분조건이다. [6] 단, $m_{ik}$는 $AA^T$의 $(i, k)$요소이다.

다음의 알고리즘 2는 정리 1을 이용하여 $AA^T$ 구조로부터 삭제나무를 생성한다. $Adj(j)$는 점 $j$의 이웃한 점들의 집합이다.

알고리즘 2. $AA^T$구조로부터 삭제나무를 생성하는 알고리즘[5]

```
1:for i := 1 to n do
2:  PARENT[ i] := 0;
3:  for k∈Adj(i) and k< i do
4:     r := k;
5:    while PARENT[ r]≠0 and PARENT[ r]≠ i do
6:        r := PARENT[ r];
7:    if PARENT[ r]=0 then PARENT[ r]:= i;
8:  end
9:end
```

알고리즘 2는 삭제나무의 부분나무를 병합해 나가면서 결국에는 삭제나무를 완성한다. 따라서, 경로압축(path compression) 방법을 적용하여 알고리즘을 고속화할 수 있다. 다음의 알고리즘 3은 벡터 ANCESTOR를 이용하여 알고리즘 2에 경로압축을 적용한 알고리즘이다. PARENT는 삭제나무의 원래 구조를 보관하고, ANCESTOR는 경로압축에 의하여 변형된 나무를 보관한다.

알고리즘 3. 경로압축에 의한 $AA^T$구조로부터 삭제나무를 생성하는 알고리즘[5]

```
1:for i := 1 to n do
2:  PARENT[ i] := 0;
3:  ANCESTOR[ i] := 0;
4:  for k∈Adj(i) and k< i do
5:   r := k;
6:   while ANCESTOR[ r]≠0 and ANCESTOR[ r]≠ i do
7:       t := ANCESTOR[ r];
8:       ANCESTOR[ r] := i;
9:       r := t;
10:    end
11:    if ANCESTOR[ r] = 0 then
12:       ANCESTOR[ r] := i;
13:       PARENT[ r] := i;
14:    end
15:  end
16:end
```

심볼릭 분해

심볼릭 분해는 $AA^T$의 비영요소구조를 계산한 다음, 추가요소를 찾아서 분해행렬의 자료구조를 완성한다. 정리 2에 의하여 행렬 $AA^T$ 없이 행렬 $A$로부터 행렬 $AA^T$의 삭제그래프에서 점 $j$에 대한 이웃점들을 구할 수 있다.

정리 2. 점 $j$에 대해서

$\{(i,j) \mid m_{ij} \neq 0, i \neq j\} =$

$\{(i,j) \mid a_{ir} \neq 0, a_{jr} \neq 0 \text{ for some } r, i \neq j\}$

이다. 단, $m_{ij}$는 $AA^T$의 $(i,j)$요소이다.

알고리즘 4는 행렬 $A$를 이용해서 심볼릭 분해를 수행한다. 단, $Adj(j)$는 행렬 $AA^T$의 삭제그래프에서 점 $j$에 이웃한 점들의 집합이고 $Reach(j)$는 분해행렬 $L$의 $j$에서 비영요소 행지수의 집합이다.

알고리즘 4. 행렬 $A$를 이용한 심볼릭 분해 알고리즘[1]

```
1:for j := 1 to n do
2:   compute Adj(j) at AA^T;
3:   Reach(j) := φ;
4:   for each i in Adj(j) do
5:     if i>j then
         Reach(j) := Reach(j)∪{i};
6:   end
7:end
8:for j := 1 to n do
9:   if Reach(k)≠φ then
10:     m := min{j|j∈Reach(k)};
11:     Reach(m):=
        Reach(m)∪(Reach(k)-{m});
12:  end
13:end
```

심볼릭 분해와 초마디 생성의 방법 2에서는 심볼릭 분해 이전에 삭제나무가 생성되어 있으므로 알고리즘 5를 이용하여 심볼릭 분해를 수행한다. 알고리즘 5에서 $Child(j)$는 삭제나무에서 점 $j$의 자식점들의 집합이다.

알고리즘 5. 삭제나무와 행렬 $A$를 이용한 심볼릭 분해[6]

```
1:for j := 1 to n do
2:   Reach(j) := φ;
3:    for each column r of A with first
      nonzero in row j do
4:   Reach(j) := Reach(j)∪{i≠j|a_ir≠0};
5:    end
6:    for s∈Child(j) do
7:     Reach(j) :=
       Reach(j)∪(Reach(s)-{j});
8:    end
9:    if Reach(j)≠φ then
10:     p := min{i| i∈Reach(j)};
11:     Child(p) := Child(p)∪{j};
12:   end
13:   end
```

심볼릭 분해와 초마디 생성의 방법 3에서는 초마디를 생성해 놓은 상태이기 때문에 알고리즘 5에서 초마디의 대표점(master node)가 아닌 점들에 대해서는 심볼릭 분해를 수행할 필요가 없다.

**초마디 생성**

분해행렬 $L$이 존재할 경우에는 초마디의 정의에 따라서 쉽게 초마디를 생성해 낼 수 있다. 그러나, 방법 3의 경우에는 심볼릭 분해 이전에 초마디를 생성한다. 초마디의 정의에 의한 극대 초마디는 구하지 못하지만, 기본초마디를 구할 수 있다. 기본초마디는 초마디의 정의에서 형제점이 있는 점을 제외한 초마디이다.

정리 3. 삭제나무에서 점 $j$가 둘 이상의 자식점을 가지거나 삭제나무의 어떤 행부분나무에서 leaf node인 것은 분해행렬에서 $j$열이 기본

초마디(fundamental supernode)의 대표점일 필요충분조건이다.[5]

정리 3에 의하면 기본초마디의 대표점들을 구할 수 있고, 대표점들을 알면 초마디 구조를 생성할 수 있다. 기본초마디의 대표점일 필요충분조건 가운데 점 $j$가 둘 이상의 자식점을 가지는 경우는 삭제나무에서 직접 확인할 수 있다. 따라서, 효율적으로 행부분나무의 leaf node를 찾는 것이 중요하다. 행부분나무는 분해행렬 $L$에서 임의의 열 $j$를 뿌리점으로 하고, 열 $j$의 비영요소 행지수들로 이루어진 부분나무이다. 다음의 알고리즘 4는 모든 행부분나무에서 leaf node들을 찾아준다. 다음을 정의하자.

ISLEAF[ $j$ ] : 점 $j$가 어떤 행부분나무의 leaf node면 TRUE 아니면 FALSE

PREV_ROWNZ[ $i$ ] : 각 행에 대해서 가장 최근에 발견된 비영요소의 위치

|T[ $j$ ]| : 삭제나무에서 $j$를 root로 하는 부분나무의 크기

알고리즘 6. 행부분나무에서 leaf node를 찾는 알고리즘[7]

```
1:for j := 1 to n do
2:    ISLEAF[ j ] := FALSE;
3:    PREV_ROWNZ[ j ] := 0;
4:end
5:for j := 1 to n do
6:    compute |T[ j ]|
7:end
8:for j := 1 to n do
9:    for each a_ij≠0 and i>j do
10:       k := PREV_ROWNZ[ i ];
11:    if( k<j-|T[ j ]|+1) ISLEAF[ j ]:=TRUE;
12:     PREV_ROWNZ[ i ] := j;
13:   end
14:end
```

# 3. 실험 결과

<표 1>는 실험에 사용한 문제들에 대한 정보들이다. Netlib의 선형계획법 문제 가운데 비영요소수가 10000개 이상인 문제 24개를 선택하였다. C언어로 작성한 원쌍대 예측자-수정자 장벽법 프로그램으로 실험하였으며, 초마디에 대해서는 8-웨이 루프 전개(8-way loop unrolling)을 적용하였다. 실험에 사용한 컴퓨터는 Sun SPARC Ultra 170이다.

<표 1> 실험 문제

| 문제이름 | 행개수 | 열개수 | 비영요소수 |
|---|---|---|---|
| 25fv47 | 822 | 1571 | 10400 |
| 80bau3b | 2263 | 9799 | 29063 |
| bnl2 | 2325 | 3489 | 16124 |
| cycle | 1904 | 2857 | 21322 |
| czprob | 930 | 3523 | 14173 |
| d2q06c | 2172 | 5167 | 35674 |
| d6cube | 416 | 6185 | 43888 |
| fit1d | 25 | 1026 | 14430 |
| fit2d | 26 | 10500 | 138018 |
| greenbea | 2393 | 5405 | 31499 |
| greenbeb | 2393 | 5405 | 31499 |
| maros | 846 | 1408 | 9576 |
| nesm | 663 | 2923 | 13988 |
| pilot | 1442 | 3652 | 43220 |
| pilotja | 941 | 1988 | 14706 |
| pilotnov | 976 | 2172 | 13129 |
| scsd8 | 398 | 2750 | 11334 |
| sctap3 | 1481 | 2480 | 10734 |
| ship08l | 779 | 4283 | 17085 |
| ship12l | 152 | 5427 | 21597 |
| ship12s | 1152 | 2763 | 10941 |
| truss | 1001 | 8806 | 36642 |
| woodlp | 245 | 2594 | 70216 |
| woodw | 1099 | 8405 | 37478 |

<표 2>는 방법 1과 방법 2의 심볼릭 분해 및 삭제나무 생성에 소요된 시간을 비교한 것이다. 방법 1은 심볼릭 분해 후에 분해행렬 $L$을

<표 2> 방법 1과 방법 2의 심볼릭 분해 및 삭제나무 생성 시간

| 문제이름 | 방법 1 | | | 방법 2 | | |
|---|---|---|---|---|---|---|
| | 심볼릭 분해 | 삭제나무 | 합계 | 심볼릭 분해 | 삭제나무 | 합계 |
| 25fv47 | 0.02 | 0.09 | 0.11 | 0.11 | 0.04 | 0.15 |
| 80bau3b | 0.03 | 0.08 | 0.11 | 0.42 | 0.03 | 0.45 |
| bnl2 | 0.02 | 0.48 | 0.50 | 0.58 | 0.03 | 0.61 |
| cycle | 0.02 | 0.10 | 0.12 | 0.17 | 0.08 | 0.25 |
| czprob | 0.01 | 0.00 | 0.01 | 0.02 | 0.02 | 0.04 |
| d2q06c | 0.04 | 0.89 | 0.93 | 1.05 | 0.11 | 1.16 |
| d6cube | 0.04 | 0.38 | 0.42 | 0.42 | 0.11 | 0.53 |
| fit1d | 0.01 | 0.00 | 0.01 | 0.03 | 0.02 | 0.05 |
| fit2d | 0.07 | 0.00 | 0.07 | 0.29 | 0.16 | 0.45 |
| greenbea | 0.03 | 0.06 | 0.09 | 0.21 | 0.11 | 0.32 |
| greenbeb | 0.04 | 0.06 | 0.10 | 0.21 | 0.11 | 0.32 |
| maros | 0.01 | 0.01 | 0.02 | 0.03 | 0.04 | 0.07 |
| nesm | 0.01 | 0.04 | 0.05 | 0.08 | 0.01 | 0.09 |
| pilot | 0.01 | 1.72 | 1.73 | 1.87 | 0.62 | 2.49 |
| pilotja | 0.02 | 0.17 | 0.19 | 0.22 | 0.07 | 0.29 |
| pilotnov | 0.00 | 0.15 | 0.15 | 0.19 | 0.04 | 0.23 |
| scsd8 | 0.01 | 0.01 | 0.02 | 0.02 | 0.01 | 0.03 |
| sctap3 | 0.01 | 0.02 | 0.03 | 0.10 | 0.02 | 0.12 |
| ship08l | 0.01 | 0.00 | 0.01 | 0.04 | 0.02 | 0.06 |
| ship12l | 0.00 | 0.01 | 0.01 | 0.06 | 0.02 | 0.08 |
| ship12s | 0.00 | 0.01 | 0.01 | 0.02 | 0.01 | 0.03 |
| truss | 0.03 | 0.13 | 0.16 | 0.25 | 0.04 | 0.29 |
| woodlp | 0.02 | 0.05 | 0.07 | 0.21 | 0.18 | 0.39 |
| woodw | 0.02 | 0.07 | 0.09 | 0.14 | 0.08 | 0.22 |

<표 3> 방법 2와 방법 3의 심볼릭 분해 및 초마디 생성 시간

| 문제이름 | 방법 2 | | | 방법 3 | | |
|---|---|---|---|---|---|---|
| | 심볼릭 분해 | 초마디 생성 | 합계 | 심볼릭 분해 | 초마디 생성 | 합계 |
| 25fv47 | 0.11 | 0.01 | 0.12 | 0.02 | 0.00 | 0.02 |
| 80bau3b | 0.42 | 0.00 | 0.42 | 0.06 | 0.00 | 0.06 |
| bnl2 | 0.58 | 0.01 | 0.59 | 0.08 | 0.00 | 0.08 |
| cycle | 0.17 | 0.01 | 0.18 | 0.03 | 0.01 | 0.04 |
| czprob | 0.02 | 0.00 | 0.02 | 0.01 | 0.00 | 0.01 |
| d2q06c | 1.05 | 0.01 | 1.06 | 0.13 | 0.00 | 0.13 |
| d6cube | 0.42 | 0.00 | 0.42 | 0.03 | 0.01 | 0.04 |
| fit1d | 0.03 | 0.00 | 0.03 | 0.00 | 0.00 | 0.00 |
| fit2d | 0.29 | 0.00 | 0.29 | 0.02 | 0.00 | 0.02 |
| greenbea | 0.21 | 0.00 | 0.21 | 0.06 | 0.00 | 0.06 |
| greenbeb | 0.21 | 0.00 | 0.21 | 0.06 | 0.00 | 0.06 |
| maros | 0.03 | 0.00 | 0.03 | 0.01 | 0.00 | 0.01 |
| nesm | 0.08 | 0.00 | 0.08 | 0.03 | 0.00 | 0.03 |
| pilot | 1.87 | 0.01 | 1.88 | 0.19 | 0.02 | 0.21 |
| pilotja | 0.22 | 0.00 | 0.22 | 0.05 | 0.00 | 0.05 |
| pilotnov | 0.19 | 0.00 | 0.19 | 0.04 | 0.00 | 0.04 |
| scsd8 | 0.02 | 0.00 | 0.02 | 0.01 | 0.00 | 0.01 |
| sctap3 | 0.10 | 0.00 | 0.10 | 0.02 | 0.00 | 0.02 |
| ship08l | 0.04 | 0.00 | 0.04 | 0.01 | 0.00 | 0.01 |
| ship12l | 0.06 | 0.00 | 0.06 | 0.01 | 0.00 | 0.01 |
| ship12s | 0.02 | 0.00 | 0.02 | 0.00 | 0.00 | 0.00 |
| truss | 0.25 | 0.00 | 0.25 | 0.03 | 0.00 | 0.03 |
| woodlp | 0.21 | 0.00 | 0.21 | 0.01 | 0.00 | 0.01 |
| woodw | 0.14 | 0.00 | 0.14 | 0.03 | 0.00 | 0.03 |

이용하여 삭제나무를 생성하는 방법이고, 방법 2는 삭제나무를 먼저 생성한 다음 삭제나무를 이용하여 심볼릭 분해를 수행하는 방법이다. 전체적으로 방법 1이 방법 2에 비해서 빠른 계산 속도를 보여 준다. 그러나, 삭제나무 생성 시간만을 고려하면 방법 1과 방법 2가 서로 경쟁적인 방법임을 알 수 있다. 그리고 삭제나무를 이용한 심볼릭 분해 방법은 삭제나무를 이용하지 않고 행렬 $A$만을 이용하는 방법에 비해서 좋지 않은 것으로 나타났다. 내부점방법에서 방법 2가 적용되기 위해서는 $AA^T$의 인접구조 생성에 대한 고려가 필요하다. 방법 2에서 삭제나무 생성에 소요되는 시간의 상당한 부분은 행렬 $AA^T$의 인접구조를 구하는 데에 소요되기 때문이다. 따라서, 효율적으로 $AA^T$의 인접구조를 구하는 것이 행렬 $A$로부터 직접 삭제나무를 생성하는 데에 매우 중요하다.

<표 3>은 방법 2와 방법 3의 심볼릭 분해 및 초마디 생성에 소요된 시간을 비교한 것이다. 방법 2와 방법 3은 삭제나무 생성 방법이 동일하지만, 심볼릭 분해와 초마디 생성의 순서가 서로 다르다. 방법 2는 심볼릭 분해 결과를 이용해 직접 분해행렬 $L$ 자료구조로부터 초마디를 생성하지만, 방법 3은 삭제나무로부터 기

본초마디를 생성할 수 있는 성질을 이용해서, 기본초마디를 생성한 다음 이를 이용해서 심볼릭 분해를 수행하는 방법이다. 초마디 생성에 소요되는 시간은 방법 2와 방법 3에 있어서 거의 무시할만한 정도의 시간만이 소요된다. 그러나, 모든 열에 대해서 심볼릭 분해를 수행해야 하는 방법 2가 심볼릭 분해에 있어서는 초마디에 속한 점들에 대해서는 한 번만 심볼릭 분해를 수행하면 되는 방법 3에 비해서 약 7배 정도 많은 시간이 소요되어서 두 가지 방법 사이의 수행도 차이가 큰 것으로 나타났다.

<표 4>는 고려한 방법들에 대해서 내부점방법에 적용하였을 때에 전체 소요되는 시간을 서로 비교한 것이다. 방법 1과 방법 2는 심볼릭 분해와 삭제나무 생성에 소요되는 시간 이외에는 같은 과정을 수행하기 때문에 이 두 과정에서 나타나는 계산 시간만큼의 차이를 가져 온다. 방법 3의 경우에는 방법 1 또는 방법 2와는 달리 초마디가 아닌 기본초마디를 사용하기 때문에 수치를 사용하는 촐레스키 분해에서의 계산 속도가 영향 받는다. 기본초마디의 크기가 방법 1의 초마디보다 작기 때문에 1회 계산에 소요되는 시간이 많을 것이라는 기대와 달리 실험 결과는 방법 1보다 오히려 더 빨라서 방법 1이 방법 3보다 오히려 약 1.14배 정도 더 많은 시간이 소요되었다. 그리고, 전체 계산 시간에 있어서는 방법 3이 약 1.03배 정도 더 많은 시간이 소요되어서 그렇게 큰 차이를 보이지 않았다. 따라서, 방법 3이 기존의 방법 1에 대해서 상당한 경쟁력을 갖춘 방법임을 알 수 있다.

<표 4> 방법 1과 방법 3의 전체 시간

| 문제이름 | 방법 1 | | 방법 3 | |
|---|---|---|---|---|
| | 1회 계산 | 전체 계산 | 1회 계산 | 전체 계산 |
| 25fv47 | 0.29 | 6.46 | 0.25 | 6.56 |
| 80bau3b | 0.48 | 20.63 | 0.47 | 21.69 |
| bnl2 | 1.20 | 32.93 | 0.89 | 35.65 |
| cycle | 0.40 | 17.97 | 0.38 | 17.74 |
| czprob | 0.06 | 2.52 | 0.09 | 2.50 |
| d2q06c | 2.24 | 54.01 | 1.74 | 60.03 |
| d6cube | 1.00 | 23.35 | 0.72 | 23.20 |
| fit1d | 0.06 | 1.52 | 0.07 | 1.60 |
| fit2d | 0.66 | 21.22 | 0.78 | 21.78 |
| greenbea | 0.40 | 24.80 | 0.51 | 25.94 |
| greenbeb | 0.39 | 16.55 | 0.46 | 17.11 |
| maros | 0.09 | 2.51 | 0.11 | 2.54 |
| nesm | 0.18 | 7.95 | 0.18 | 8.22 |
| pilot | 4.30 | 134.91 | 3.31 | 134.38 |
| pilotja | 0.48 | 15.32 | 0.44 | 16.56 |
| pilotnov | 0.45 | 14.22 | 0.38 | 15.55 |
| scsd8 | 0.05 | 0.96 | 0.06 | 0.95 |
| sctap3 | 0.15 | 3.59 | 0.17 | 3.65 |
| ship08l | 0.07 | 2.84 | 0.10 | 3.01 |
| ship12l | 0.10 | 3.22 | 0.13 | 3.44 |
| ship12s | 0.04 | 1.19 | 0.05 | 1.31 |
| truss | 0.48 | 9.56 | 0.40 | 9.49 |
| woodlp | 0.39 | 12.83 | 0.50 | 12.81 |
| woodw | 0.29 | 9.50 | 0.30 | 9.50 |

## 4. 결 론

심볼릭 분해, 삭제나무 생성 그리고 초마디 생성에 관련하여 내부점방법에 적용 가능한 세 가지 방법을 구현하여 비교 실험하였다.

삭제나무의 생성은 $L$의 비영요소구조를 이용하는 것과 $A$의 비영요소구조를 이용하는 것 사이에 우열을 가리기 어렵다. 방법 2의 삭제나무 생성에서 $AA^T$ 자료구조를 생성하는 데에 많은 시간이 할애되는 점이 해결되면 방법 1에 비해서 우수한 방법이 될 가능성이 있다.

방법 2는 삭제나무를 이용해서 심볼릭 분해를 수행하도록 하였는데, 방법 1과 비교할 때에 시간이 많이 소요되었다. 방법 3은 삭제나무를 이용하지만 초마디 정보를 활용하여 계산량을 줄였기 때문에 상당히 빠른 계산이 가능하다.

따라서, 초마디 정보를 이용하는 것이 심볼릭 분해의 효율화에 중요하다는 것을 알 수 있다.

초마디 생성에 걸리는 시간은 내부점방법의 전체 계산 시간에 거의 영향을 주지 않는 적은 시간이 소요되기 때문에 초마디 생성에 소요되는 시간은 별로 중요한 부분이 아니다. 오히려 초마디를 미리 생성하여 이후의 과정에서 초마디의 성질을 활용할 수 있도록 하는 것이 해법에 효율화에 중요하다.

전체 계산 효율면에서 보면, 방법 1과 방법 3이 서로 경쟁적인 대안이 될 수 있다. 방법 2나 방법 3에서는 방법 1에서보다 삭제나무를 생성하는 데에 많은 시간이 소요됨을 알 수 있다. 삭제나무를 생성하는 데에 걸리는 시간을 제외하면 방법 3의 경우에 심볼릭 분해와 초마디 생성에 소요되는 시간이 가장 적기 때문에 앞으로 삭제나무를 보다 효율적으로 생성하기 위한 방법을 개발하는 것이 필요하다.

## 참 고 문 헌


[1] Alan George and J. W. H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice Hall, 1981

[2] Andersen, Erling D., Jacek Gondzio, Csaba Meszaros, Xiaojie Xu, "Implementation of interior point methods for large scale linear programming", Technical Report 1996. 3, Logilab, HEC Geneva, Section of Management Studies, University of Geneva

[3] I. S. Duff, A. M. Erisman and J. K. Reid, *Direct Methods for Sparse Matrices*, Oxford Univ. Press, 1986

[4] Jung, Ho-won et al, "Numerical factorization methods for interior point algorithms", *ORSA Journal on Computing*, Vol. 6, No. 1(1994), pp. 94-105

[5] Liu, J. W. H., "A compact row storage scheme for Cholesky factors using elimination tree", *ACM Transaction on Mathematical Software*, Vol. 12, No. 2(1986), pp. 127-148

[6] Liu, J. W. H., "The role of elimination trees in sparse factorization", *SIAM J. Mat. Anal. Appl.*, Vol. 11, No. 1(1990), pp. 134-172

[7] Liu, J. W. H. et al, "On finding supernodes for sparse matrix computations", *SIAM J. Matrix Anal., Appl.*, Vol. 14, No. 1(1993), pp. 242-252

[8] Rothberg, Edward and Anoop Gupta, "Efficient sparse matrix factorization on high-performance workstations-exploiting the memory heirarchy", Technical Report, Department of Computer Science, Standford Univ., Aug. 21, 1990