Journal of the Korean Institute
of Industrial Engineers
Vol.24, N0.3, September, 1998

클릭저장구조에서 최소 부족수 순서화의 효율화*

Minimum Deficiency Ordering with the Clique Storage Structure*

설동렬** · 박찬규** · 박순달**

Tong-Ryeol Seol** · Chan-Kyoo Park** · Soon-Dal Park**

Abstract

For fast Cholesky factorization, it is most important to reduce the number of nonzero elements by ordering methods. Generally, the minimum deficiency ordering produces less nonzero elements, but it is very slow. We propose an efficient implementation method.

The minimum deficiency ordering requires much computations related to adjacent nodes. But, we reduce those computations by using indistinguishable nodes, the clique storage structures, and the explicit storage structures to compute deficiencies.

Keywords: Cholesky factorization, minimum deficiency ordering, adjacent nodes, indistinguishable nodes, clique storage structures, explicit storage structures

1. 서론

Karmarkar의 사영법을 바롯하여 아핀법, 장벽법 등의 내부점 선형계획법은 대칭양정치 행렬의 선형방정식을 푸는 과정이 필요하다.[9] 일반적으로 선형계획법 문제들은 희소한 특성을 가지는데, 내부점 선형계획법에서 나타나는 대칭양정치 행렬도 역시 희소행렬이다[2]. 내부점 선형계획법에서는 전체 해법 소요 시간의 많은 부분이 선형방정식을 푸는 데에 소요되어, 효율적으로 선형방정식을 풀어내는 것이 해법의 성능을 크게 좌우한다.

선형방정식을 푸는 데에는 주로 촐레스키 분해를 사용한다. 희소행렬에 대하여 촐레스키 분해를 수행할 때에는 촐레스키 분해행렬의 비영요소수를 줄이기 위하여 순서화 방법을 적용한다. 최적의 순서화를 찾는 것은 NP-Complete인 것으로 알려져 있다[14]. 따라서, 순서화에서는 최소 차수 순서화, 최소 부족수 순서화 등의 발견적 방법이 사용되는데, 특히 최소 차수 순서화가 널리 사용되고 있다. 최소 부족수 순서화는 대체로 최소 차수 순서화에 비해서 추가요소의 개수를 더 줄여준다. 그럼에도 불구하고 순서화에 소요되는 시간이 최소 차수 순서화에 비해 매우 많이 걸리는 문제점을 가지고 있다. Vanderbei [13]의 연구에 의하면 내부점 방법의 수행 회수가 많을 경우에 최소 차수 순서화보다 최소 부족수 순서화가 전체 수행 시간 면에서 유리할 수 있다. Lustig 등[9]은 예측자-수정자 방법과 같이 수행 회수가 적은 내부점 방법이 등장함으로 말미암아 최소 부족수 순서화보다는 최소 차수 순서화가 일반적인 경우에 더 효율적이라고 주장하였지만, Mészáros[4]는 최소 부족수 순서화의 개념을 이

* 본 연구는 한국과학재단 핵심전문연구과제(981-1016-024-2)에 의해 지원되었음.

** 서울대학교 산업공학과

용하여 최소 차수 순서화보다 비영요소를 줄임으로써 내부점 방법의 속도를 더 개선할 수 있음을 보였다.

최소 부족수 순서화 방법(minimum deficiency ordering)은 매 회마다 최소의 부족수를 가지는 점을 찾아서 삭제하는 방법으로 Tinney & Walker(1967)에 의해 제안된 순서화 방법 가운데 방법 3의 대칭행렬에 해당하는 방법이다. 최소 부족수 순서화 방법도 역시 최소 차수 순서화와 같이 최적 순서화 방법은 아니고, 발견적 기법이다.

최소 부족수 순서화라는 이름은 Rose(1972)에 의해서 사용되었다. Duff 등[5]은 현 단계에서 추가되는 비영요소의 개수를 가장 작게 하는 점을 찾아서 삭제하는 방법이라는 의미에서 국부 최소 추가요소 순서화(local minimum fill-in order)라고 이름하였다.

George와 Liu[7]는 순서화의 효율적인 수행을 위해서 삭제그래프 보관을 위한 자료구조로 쿼션트 그래프 저장구조를 제시하였고, 구별불능점을 이용한 구현 방법을 연구하였다. 삭제그래프를 보관하는 방법에는 Duff 등[5]이 소개한 클릭 저장구조와 Mészáros[4]에 의해 연구된 명시적 저장구조도 있다. Liu[11]는 최소 차수 순서화에서 복수삭제 기법을 적용하여 최소 차수를 가지는 점들을 한꺼번에 삭제하는 방법을 연구하였는데, 복수삭제는 최소 부족수 순서화에도 효과적으로 적용될 수 있다[3].

본 논문에서는 클릭 저장 구조를 사용하여 삭제그래프를 보관할 때에 최소 부족수 순서화를 효율화하기 위한 방법들을 연구하였다.

첫째, 구별불능점을 활용하여 부족수를 계산하는 방법을 연구하였다. 부족수는 선회요소로 선택되어 선회연산을 수행했을 때에 추가되는 비영요소의 수를 뜻한다. 행렬에 대응되는 삭제그래프의 입장에서는 선회요소에 해당하는 점을 삭제하였을 때에 추가되는 호의 개수가 부족수가 된다. 부족수를 계산할 때에는 삭제그래프에서 인접점 간에 이미 호가 존재하는지 여부를 검사해야 하기 때문에, 부족수의 인접점들과 부족수의 인접점들의 인접점에 관련한 계산량이 많게 된다. 순서화 방법을 구현할 때에는 인접점 집합이 같은 점들을 하나의 점으로 처리하여 부족수 또는 차수를 계산해야 하는 점의 수를 줄이는 구별불능점 기법을 많이 사용한다. 그런데, 구별불능점을 부족수를 계산할 때에도 적용하여 부족수 계산에서 고려해야 할 인접점의 수를 줄일 수 있다.

한편, 둘째로 삭제그래프를 관리하는 방법에 대하여 연구하였다. 선회연산에 의한 삭제그래프의 변형은 클릭 저장구조를 사용할 경우에 매우 간단하게 처리할 수 있다. 그러나, 클릭 저장구조를 사용하면, 인접점의 집합을 찾을 때에 여러 클릭을 탐색해야 하는 문제점이 있다. 클릭 저장구조 이외에 삭제그래프를 관리하는 데에 주로 사용되는 쿼션트 그래프 저장구조는 삭제그래프의 변형에 클릭 저장구조보다 복잡한 과정을 거쳐야 하며, 부족수를 계산할 때와 같이 인접점의 집합을 많이 다루어야 하는 경우에 적절한 자료구조는 아니다. 따라서, 본 연구에서는 클릭 저장구조를 기본적으로 채용하되 인접점 집합을 다루기에 유리한 명시적 자료구조를 공유하도록 하였다. 즉, 클릭 저장구조를 통해서 삭제그래프의 변형을 수행하고, 부족수 계산이 필요한 점들에 대해서만 임시로 명시적 자료구조를 유지하도록 하면 부족수 계산과 구별불능점 검사를 효과적으로 수행할 수 있다.

2. 부족수 계산 방법

대칭양정치 행렬의 비영요소 구조는 각 행/열 번호를 점으로 두고, 비영요소를 점과 점을 잇는 호라고 두면 무방향 그래프로 표현할 수 있다. 가우스소거 과정은 이러한 그래프에서 점을 하나씩 삭제하면서 삭제할 점의 인접집합을 클릭으로 만드는 과정으로 이해할 수 있다. 클릭을 형성하면서 새로 추가되는 호는 가우스소거를 수행할 때에 발생하는 추가요소(fill-in)에 대응된다. 이러한 용도로 사용되는 그래프를 특별히 삭제그래프라고 한다. 삭제그래프 $G(N,E)$에서 인접점 집합, 차수, 부족수는 다음과 같이 정의된다. N은 점의 집합, E는 호의 집합을 의미한다.

정의 1.

그래프 $G(N,E)$에서,

점 i의 인접점 집합은 $Adj(i)=\{j \mid (i,j)\in E\}$이다.

점 i의 차수는 $\deg(i)=|Adj(i)|$이다.

점 i의 부족수는 $\mathrm{def}(i) = |\{(j,k) \mid j,k\in Adj(i),\ j\neq k,\ j \notin Adj(k)\}|$이다.

점 i의 부족수는 점 i가 삭제될 때에 추가되는 비영요

소수의 수를 의미한다. 추가될 비영요소의 개수를 알아내기 위해서 선화연산을 수행할 수는 없으므로 다음의 정리 1과 같은 방법으로 부족수를 계산한다.

정리 1.
그래프 $G(N,E)$에서 점 i의 부족수는

$$def(i) = \left\lceil \frac{|Adj(i)|}{2} \right\rceil - |\{(j,k) \mid j,k \in Adj(i),\ j \neq k,\ k \in Adj(j)\}|$$

이다.

(증명)
그래프 G에서 점 i가 삭제될 때에 추가되는 호의 집합은 점 i의 이웃한 점들의 쌍 가운데에서 이미 존재하는 호를 뺀 주면 된다. 점 i의 이웃한 점들에서 가능한 쌍의 개수는

$$|\{(j,k) \mid j,k \in Adj(i),\ j \neq k\}| = \left\lceil \frac{|Adj(i)|}{2} \right\rceil$$

이다. 그리고, 점 i의 이웃한 점들에서 호가 존재하는 쌍의 개수는

$$|\{(j,k) \mid j,k \in Adj(i),\ (j,k) \in E\}| = |\{(j,k) \mid j,k \in Adj(i),\ j \neq k,\ k \in Adj(j)\}|$$

이다. 따라서, 그래프 G에서 점 i가 삭제될 때에 추가되는 호의 개수는

$$\left\lceil \frac{|Adj(i)|}{2} \right\rceil - |\{(j,k) \mid j \neq k,\ k \in Adj(j)\ \forall j,k \in Adj(i)\}|$$

$$= |\{(j,k) \mid j,k \in Adj(i),\ j \neq k\}| - |\{(j,k) \mid j,k \in Adj(i),\ j \neq k,\ k \in Adj(j)\}|$$
$$= |\{(j,k) \mid j,k \in Adj(i),\ j \neq k,\ k \notin Adj(j)\}| = def(i)$$

이므로 정리가 성립한다.

■

정리 1에 의하면, 점 i의 인접점들과 점 i의 인접점의 인접점들을 이용하여 부족수를 계산할 수 있다. 부족수를 계산하는 데에 인접점 집합을 알아내기 위한 계산이 많이 소요되기 때문에, 부족수 계산을 효율적으로 수행하기 위해서는 인접점 집합에 관련된 연산을 줄이는 것이 필요하다. 삭제그래프에서 인접점의 집합이 같은 점들은 구별불능점이라는 일종의 초마디로 처리될 수 있다. 구별불능점을 적용하면, 점의 개수가 줄어드는 효과를 얻을 수 있으므로 부족수 계산을 효율적으로 수행할 수 있다. 먼저 다음과 같이 구별불능점 집합, 점의 크기 그리고 구별불능점 인접점 집합을 정의한다.

정의 2.
그래프 $G(N,E)$에서,
점 i의 구별불능점 집합은 $Indis(i)=\{j \mid Adj(j)\cup\{j\} = Adj(i)\cup\{i\},\ j \neq i\}\cup\{i\}$이다.
점 i의 크기 ssize[i]는 점 i가 구별불능점의 대표점이면 ssize[i]=$|Indis(i)|$, 그렇지 않으면 ssize[i]=0이다.
점 i의 구별불능점 인접점 집합은 $Adj_S(i)=\{j \mid$ ssize[j]$>0,\ (i,j)\in E\}$이다.

구별불능점 집합에 속하는 점들 가운데 하나의 점을 선택하여 구별불능점을 대표하도록 한다. 예를 들어, 구별불능점 집합에서 가장 번호가 작은 점을 구별불능점의 대표점으로 정할 수 있다. 인접점 집합은 그림 1과 같이 삭제 그래프에서 호로 직접 연결되어 있는 점들의 집합을 의미하고, 구별불능점 인접점 집합은 그림 2와 같이 구별불능점을 하나의 점으로 처리한 구별불능점 삭제그래프에서의 인접점 집합을 의미한다.

구별불능점을 이용하여 다음의 정리 2와 같이 부족수를 계산할 수 있다.

정리 2.
그래프 $G(N,E)$에서,

$$|\{(j,k) \mid j,k \in Adj(i),\ j \neq k,\ j \in Adj(k)\}|$$
$$= \left\lceil \frac{\text{ssize}[i]-1}{2} \right\rceil + (\deg[i]-\text{ssize}[i]+1)\times(\text{ssize}[i]-1) + \sum_{j \in Adj(i)-Indis(i)} \left\lceil \frac{\text{ssize}[j]}{2} \right\rceil + \sum_{j<k;j,k \in Adj(i)-Indis(i)} \text{ssize}[j]\times\text{ssize}[k]$$

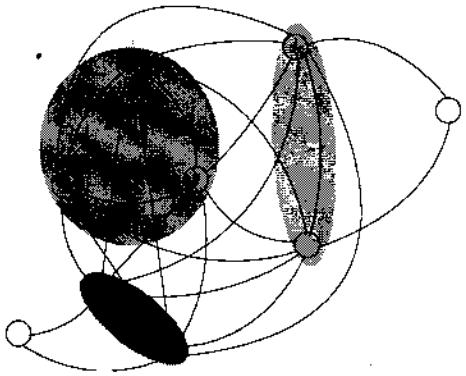

그림 1. 삭제그래프

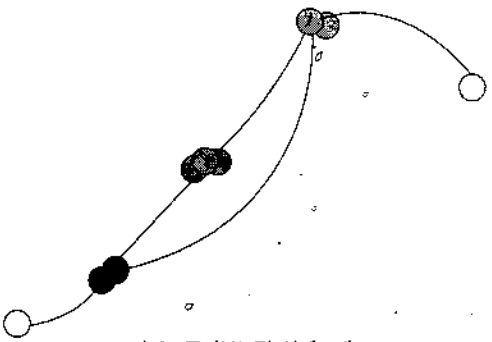

그림 2. 구별불능점 삭제그래프

이다.

(증명)

$\{(j,k) \mid j,k \in Adj(i),\ j \neq k,\ k \in Adj(j)\}$은 다음과 같은 공유원소가 없는 집합들로 분할된다.

$$
\begin{aligned}
&\{(j,k) \mid j,k \in Adj(i),\ j \neq k,\ k \in Adj(j)\} \\
&= \{(j,k) \mid j,k \in Indis(i)-\{i\},\ j \neq k,\ k \in Adj(j)\} \\
&\quad \cup \{(j,k) \mid j \in Indis(i)-\{i\}, k \in Adj(i)-Indis(i), \\
&\qquad\qquad j \neq k,\ k \in Adj(j)\} \\
&\quad \cup \{(j,k) \mid j,k \in Adj(i)-Indis(i),\ j \neq k,\ k \in Indis(j)\} \\
&\quad \cup \{(j,k) \mid j,k \in Adj(i)-Indis(i), \\
&\qquad\qquad j \neq k,\ k \in Adj(j)-Indis(j)\}
\end{aligned}
$$

점 j와 점 k가 모두 $Indis(i)$에 속하는 경우에는 $k=Adj(j)$가 성립하므로, 첫 번째의 분할된 집합에 속한 호의 개수는 다음과 같이 계산된다.

$$
\begin{aligned}
&|\{(j,k) \mid j,k \in Indis(i)-\{i\},\ j \neq k,\ k \in Adj(j)\}| \\
&= |\{(j,k) \mid j,k \in Indis(i)-\{i\},\ j \neq k\}| = \left\lfloor \frac{\text{ssize}[i]-1}{2} \right\rfloor
\end{aligned}
$$

점 k는 $Adj(i)$-$Indis(i)$에 속하는 경우에, 점 j가 점 i와 구별불능이므로 $k \in Adj(i)$이면 $k \in Adj(j)$이다. 따라서, 두 번째의 분할된 집합에 속한 호의 개수는 다음과 같이 계산된다.

$$
\begin{aligned}
&|\{(j,k) \mid j \in Indis(i)-\{i\}, k \in Adj(i)-Indis(i),\ j \neq k,\ k \in Adj(j)\}| \\
&= |\{(j,k) \mid j \in Indis(i)-\{i\}, k \in Adj(i)-Indis(i)\}| \\
&= (\deg[i] - \text{ssize}[i]+1) \times (\text{ssize}[i]-1)
\end{aligned}
$$

세 번째의 분할된 집합에 속한 호의 개수는 점 i에 이웃한 구별불능점 내부에 있는 호들의 개수이다.

$$
\begin{aligned}
&|\{(j,k) \mid j,k \in Adj(i)-Indis(i),\ j \neq k,\ k \in Indis(j)\}| \\
&= |\{(j,k) \mid j \in Adj(i)-Indis(i), k \in Indis(j)-\{j\}\}| \\
&= |\{(j,k) \mid j,k \in Indis(l), l \in Adj_S(i), j \neq k\}| \\
&= \sum_{j \in Adj(i)-Indis(i)} \left\lfloor \frac{\text{ssize}[j]}{2} \right\rfloor
\end{aligned}
$$

마지막 분할 집합에 속한 호의 개수는 점 i에 인접한 서로 다른 구별불능점 사이에 존재하는 호의 개수이다.

$$
\begin{aligned}
&|\{(j,k) \mid j,k \in Adj(i)-Indis(i),\ j \neq k,\ k \in Adj(j)-Indis(j)\}| \\
&= \sum_{j>k;j,k \in Adj(i)-Indis(i)} \text{ssize}[j] \times \text{ssize}[k]
\end{aligned}
$$

따라서, 정리 2가 성립한다.

■

다음의 그림 3, 그림 4, 그림 5, 그림 6은 정리 2의 분할된 집합을 보여주고 있다.

위의 그림에서 정리 2를 이용하면, 부족수를 구하려는 점 i의 인접점 가운데 구별불능점의 대표점인 점 j와 점 k의 경우만을 고려하여 부족수를 계산할 수 있다. 부족수를 구하려는 점 i의 인접점 사이에 존재하는 호는 점 i에 의해 대표되는 구별불능점 내에 호가 있는 경우(그림 3.), 점 i에 인접한 구별불능점과 점 i의 구별불능점 사이에 존재하는 호(그림 4.), 점 i에 인접한 구별불능점의 내부에 존재하는 호(그림 5.) 그리고 인접한 구별불능점 사이에 호가 존재하는 경우(그림 6.)를 모두 합하면 된

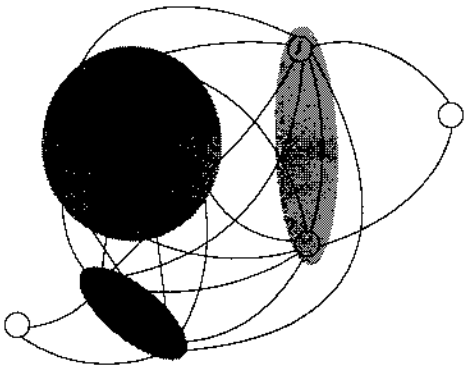

그림 3. 구별불능점 내부의 점들 간의 호

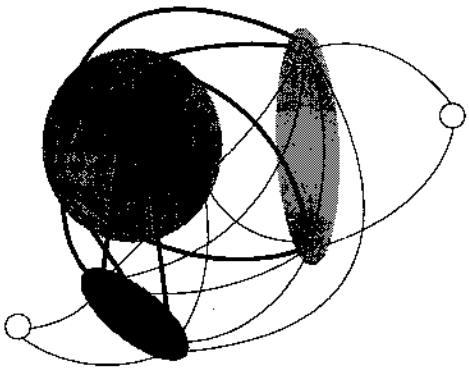

그림 4. 구별불능점에 속한 점과 인접점 간의 호

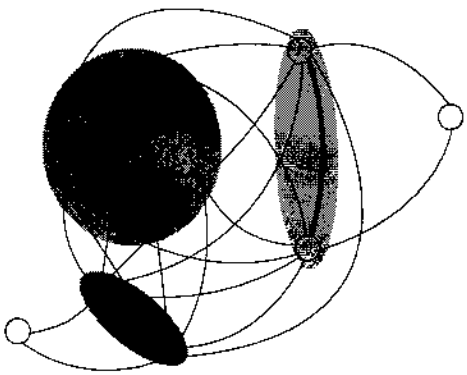

그림 5. 인접한 구별불능점 내부의 점들 간의 호

다. 이 때에 구별불능인 점이 없는 점에 대해서는 크기가 1인 구별불능점으로 처리하면 된다.

종정리 1.

그래프 $G(N,E)$에서 점 i의 부족수는

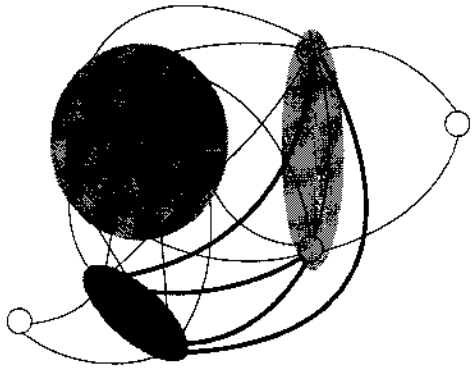

그림 6. 인접한 구별불능점 간의 호

$$\mathrm{def}(i) = \left\lfloor \frac{\deg[i]-\mathrm{ssize}[i]+1}{2} \right\rfloor - \sum_{j\in Adj(i)-Indis(i)} \left\lfloor \frac{\mathrm{ssize}[j]}{2} \right\rfloor - \sum_{j,k,j\neq k\in Adj(i)-Indis(i)} \mathrm{ssize}[j]\times\mathrm{ssize}[k]$$

이다.

(증명)

정리 1과 정리 2에 의하여 점 i의 부족수는 다음과 같이 계산된다.

$$\mathrm{def}(i) = \left\lfloor \frac{|Adj(i)|}{2} \right\rfloor - \left\lfloor \frac{\mathrm{ssize}[i]-1}{2} \right\rfloor - (\deg[i]-\mathrm{ssize}[i]+1)\times(\mathrm{ssize}[i]-1) - \sum_{j\in Adj(i)-Indis(i)} \left\lfloor \frac{\mathrm{ssize}[j]}{2} \right\rfloor - \sum_{j,k,j\neq k\in Adj(i)-Indis(i)} \mathrm{ssize}[j]\times\mathrm{ssize}[k]$$

그런데, 정의 1에 의해서 $|Adj(i)|=\deg(i)$이므로,

$$\left\lfloor \frac{|Adj(i)|}{2} \right\rfloor - \lfloor \mathrm{ssize}[i]-1 atot 2 \rfloor - (\deg[i]-\mathrm{ssize}[i]+1)\times(\mathrm{ssize}[i]-1)$$

$$= \frac{\deg(i)(\deg(i)-1)}{2} - \frac{(\mathrm{ssize}[i]-1)(\mathrm{ssize}[i]-2)}{2} - (\deg[i]-\mathrm{ssize}[i]+1)\times(\mathrm{ssize}[i]-1)$$

$$= \frac{(\deg[i]-\mathrm{ssize}[i])(\deg[i]-\mathrm{ssize}[i]+1)}{2}$$

$$= \left\lfloor \frac{\deg[i]-\mathrm{ssize}[i]+1}{2} \right\rfloor .$$

따라서, 정리가 성립한다.

■

종정리 1의 결과를 사용하면 구별불능점의 대표점들만을 고려하여 점 i의 부족수를 계산할 수 있다. 부족수를 계산하기 위하여 점 i의 인접점과 인접점들의 인접점에 더하여 인접집합을 일일이 계산할 필요 없이 구별불능점의 대표점인 경우만을 고려하면 되기 때문에 인접집합 계산이 줄어들게 된다. 결과적으로 부족수 계산에 소요되는 시간을 절약할 수 있다.

3. 삭제그래프 보관 방법

클릭 저장 방법과 삭제그래프 수정

클릭과 클릭커버는 다음과 같이 정의된다.

정의 3. 클릭, 클릭 커버(Clique Cover)

G의 지역그래프 $C(N')$가 모든 두 점 사이에 호가 있으면 $C(N')$을 클릭(clique)이라 한다.

G의 클릭 $C_1(N_1)=(N_1,E(N_1))$, $C_2(N_2)=(N_2,E(N_2))$, …, $C_k(N_k)=(N_k, E(N_k))$가 다음의 조건을 만족하면, $K=\{C_1(N_1), C_2(N_2), \cdots, C_k(N_k)\}$를 G의 클릭커버라 하고 k를 클릭커버 K의 크기라 한다.

(i) $N_1\cup N_2\cup\cdots\cup N_k=N$,
(ii) $E(N_1)\cup E(N_2)\cup\cdots\cup E(N_k)=E$

행렬 AA^T에 대해서 다음의 정리가 성립한다.

정리 3.

$A_{\cdot s}$의 비영요소들의 행번호 집합이 $N'=\{r_1,r_2,\cdots,r_l\}$이면 G의 지역그래프 $G(N')$은 클릭이다.

(증명)

$a_{ik}\neq 0$이고 $a_{jk}\neq 0$인 k가 존재하면 $m_{ij}\neq 0$이다. $A_{\cdot s}$의 비영요소들의 행번호 집합이 $N'=\{r_1, r_2, \cdots, r_l\}$이면 $a_{is}\neq 0, a_{js}\neq 0$ $(i,j\in N', i\neq j)$이므로 $m_{ij}\neq 0$이고 $(i, j)\in E$이다. 즉, 그래프 G에 $i,j\in N'(i\neq j)$인 호(i, j)가 존재하게 되고, 이는 $N'=\{r_1,r_2,\cdots,r_l\}$에 속하는 두 점 사이에 호가 있음을 의미한다. 따라서 G의 지역그래프 $G(N')$은 클릭이다.

■

정리 3에 의하면 행렬 A의 각 열들은 내부점 방법에서 촐레스키 분해를 적용해야 하는 행렬 $A\theta A^T$의 클릭커버가 이룬다. 따라서, 삭제그래프는 행렬 A의 각 열들로 표현될 수 있다.

삭제그래프에서 점을 삭제하고 수정된 삭제그래프가 만들어지는 과정을 클릭커버와 관련지어 생각해 보자. G의 클릭커버 $K=\{C_1,C_2,\cdots,C_m\}$가 있고, 여기에서 점 i가 포함된 클릭들을 $C_1^i,C_2^i,\cdots,C_p^i$라고 하자. 그러면 G에서 점 i가 삭제되고 난 후의 수정된 삭제 그래프 G'의 클릭커버 K'는 다음과 같이 구할 수 있다.

(1) $C = C_1^i\cup C_2^i\cup\cdots\cup C_p^i-\{i\}$
(2) $K' = K-\{C_1^i, C_2^i, \cdots, C_p^i\}$
(3) $K' = K'\cup\{C\}$

즉, 삭제점 i와 i를 포함하고 있는 클릭들을 K에서 삭제하고 i가 포함되어 있던 클릭들을 하나의 큰 클릭으로 만들어 클릭집합에 추가하는 것이다.

G의 클릭커버 K를 $K=\{C_1,C_2,\cdots,C_m\}$라고 하고 G에서 점 i를 삭제해서 만들어진 삭제그래프 G'의 클릭커버를 $K'=\{C_1',C_2',\cdots,C_k'\}$이라 하자. 그러면 다음의 정리가 성립한다.

정리 4.

$$\sum_{j=1}^{m}|C_j| > \sum_{j=1}^{m}|C_j'|$$

(증명)

$K' = K-\{C_1^i,C_2^i,\cdots,C_p^i\}\cup\{C\}$이고 $|C| < |C_1^i| + |C_2^i| + \cdots + |C_p^i|$이다. 따라서,

$$\sum_{j=1}^{k}|C_j'| = \sum_{j=1}^{m}|C_j|-\sum_{i=1}^{p}|C_j^i|+|C| < \sum_{j=1}^{m}|C_j|\text{이다.}$$

■

정리 2는 클릭커버 K'를 저장하는데 필요한 기억공간의 양이 K를 저장하는데 필요한 기억공간의 양보다 작음을 말한다. 즉, 삭제그래프의 수정에 의해 수정된 클릭커버를 저장하는데 필요한 기억공간의 양은 항상 감소함을 알 수 있다.

명시적 저장 방법과 인접집합 계산

C_i는 클릭 i에 속하는 점들의 집합이고, $|C_i|$는 클릭 i에 속하는 점의 개수이다. 클릭커버 $K=\{C_1, C_2, \cdots, C_m\}$가 있을 때, 클릭커버 K로부터 점 j의 인접점 집합 $Adj(j)$는 다음의 정리에 의해 구할 수 있다.

정리 5.

클릭커버 K에서 점 u가 포함된 클릭들을 $C_1^u, C_2^u, \cdots, C_k^u$이라고 하면,

$$Adj(j) = \bigcup_{i=1}^{k} C_i^j - \{j\}$$

이다.

(증명)

클릭과 클릭커버의 정의로부터 점 j의 인접점 집합 $Adj(j)$를 구할 때, 점 j를 포함하는 클릭만을 고려하면 된다. 그리고, 각 클릭에서 $j \in C_i^j$, $i=1,2,\cdots,k$ 이므로 점 j의 인접점의 집합은

$$Adj(j) = \bigcup_{i=1}^{k} C_i^j - \{j\}$$

이다.

■

최소 부족수 순서화를 수행할 때에 매 회 부족수를 새로 계산해야 하는 점은 삭제된 점의 인접점과 삭제된 점에 인접한 점의 인접점들이다. 그런데, 부족수를 계산하기 위해서는 정리 1과 정리 2에서처럼 부족수를 계산하려는 점의 인접점과 인접한 점의 인접점들이 필요하다. 서로 인접한 관계에 있는 점들이 연이어 계산에 이용되기 때문에 한번 계산한 인접점을 중복해서 계산하는 비율이 전체 인접점 계산량의 90%까지 이른다고 알려져 있다[3]. 따라서, $Adj(j)$를 임시로 보관하면 중복계산을 피할 수 있다.

인접점 집합은 삭제점 선택된 다음 부족수를 계산하면서 고려되는 인접점의 집합부터 보관하기 시작한다. 그리고, 다음 삭제점이 선택되면, 임시로 보관하던 인접점의 집합을 버리고 새로 인접점의 집합을 보관한다. 자료구조는 다음과 같다.

EADJSET[·] : 인접점의 번호를 보관하는 배열

SADJSET[i]: EADJSET에서 점 i의 인접점이 시작되는 위치를 보관하는 배열

NADJSET : EADJSET에 보관된 인접점의 전체 개수

그런데, 보관해야 할 인접점 집합의 양을 줄이기 위해서 그리고 정리 2의 방법으로 부족수를 계산할 때에 인접점 가운데 구별불능점의 대표점을 쉽게 가려낼 수 있도록 미리 인접점 가운데 구별불능점의 대표점만을 보관하는 것이 유리하다.

정리 3.

점 i의 인접점 j에 대해 ssize[j]=0이면, ssize[k]>0이고 점 j와 구별불능인 점 i의 인접점 k가 존재한다.

(증명)

ssize[j]=0이면 점 j가 속한 구별불능점 집합의 대표점 k가 존재한다. 점 k는 대표점이므로 ssize[k]=$|Indis(k)|$>0이다. 그런데, 점 j와 점 k가 구별불능이므로 $Adj(j) \cup \{j\} = Adj(k) \cup \{k\}$가 성립하기 때문에, $j \in Adj(i)$이면 $i \in Adj(j)$이므로 $i \in Adj(k)$가 성립한다. 따라서, $k \in Adj(i)$이다.

■

정리 3에 의하여 구별불능점의 경우에 대표점이 아닌 점들에 대해서는 반드시 대표점을 통해서 인접점 집합을 구할 수 있으므로 인접점 집합을 계산할 때에 따로 고려하지 않아도 대표점만을 보관하고 있으면 모든 점에 대한 정보를 가지게 됨을 알 수 있다.

정리 4.

ssize[i]>0이고 ssize[j]>0일 때에,

$Adj_s(i) \cup \{i\} = Adj_s(j) \cup \{j\}$이면, $Adj(i) \cup \{i\} = Adj(j) \cup \{j\}$이다.

(증명)

점 i와 점 j의 경우는 자명하므로, 점 i와 점 j가 아닌 $Adj_s(i) \cup \{i\} = Adj_s(j) \cup \{j\}$에 포함되는 점 k를 고려하자.

$k \in Adj_s(i) \cup \{i\}$인 k에 대해서 $l \in Indis(k)$인 점 l이 존재하고, $l \in Adj(i) \cup \{i\}$이다. 즉, 대표점 k가 점 i에 인접

하면 $l \in Indis(k)$인 모든 점 l이 점 i에 인접하다. 그런데, $k \in Adj_s(j) \cup \{j\}$이므로, 모든 점 l은 점 j에도 인접하다. ■

정리 4에 의하면 구별불능 관계를 검사할 때에도 모든 인접점을 비교할 필요 없이 구별불능 관계가 없는 점을 포함하여 구별불능점의 대표점만을 고려하면 된다.

용하지 않은 경우를 DEF1, 명시적 저장 구조만을 사용한 경우를 DEF2, 그리고 두 가지 방법을 함께 적용한 경우를 DEF3이라고 하였다. 어느 경우에나 삭제그래프는 클릭 저장 구조를 사용하여 관리되고 있다.

표 1은 DEF1, DEF2 그리고 DEF3의 순서화 수행 시간을 비교한 것이다. DEF1의 수행시간을 기준으로 하여 DEF2와 DEF3의 수행시간을 나누어 준 값을 서로 비교

표 1. 최소 부족수 순서화 수행 시간 비교

문제	DEF2/DEF1	DEF3/DEF1	문제	DEF2/DEF1	DEF3/DEF1
agg2	0.60	0.23	scrs8	0.80	0.29
agg3	0.60	0.24	scsd1	1.00	0.33
bnl1	0.94	0.26	scsd6	0.79	0.05
boeing1	1.02	0.54	sctap2	1.13	0.14
degen2	0.85	0.49	seba	0.24	0.07
fffff800	1.23	0.37	shell	0.75	0.29
forplan	0.81	0.39	ship04l	1.11	0.34
ganges	0.79	0.14	ship04s	0.77	0.27
gfrdpnc	0.80	0.13	ship08s	0.64	0.19
grow15	0.64	0.32	sierra	0.58	0.11
grow22	0.64	0.29	stair	0.65	0.25
perold	0.64	0.28	standata	0.82	0.21
pilot4	0.77	0.34	standgub	0.76	0.16
pilotwe	0.89	0.30	standmps	0.93	0.32
scfxm2	0.73	0.24	stocfor2	1.51	0.11
scfxm3	0.72	0.19	tuff	0.59	0.20

4. 실험결과

비가능 예측자-수정자 방법을 사용하는 내부점 프로그램에서 최소 부족수 순서화의 수행도를 실험하였다. 실험을 수행한 컴퓨터는 SunSparc Ultra 170이며 주기억장치의 용량은 64MB이다. 실험에 사용한 문제는 Netlib[6]의 가능문제들을 사용하였다. 본 논문에서 제시한 구별불능점을 활용한 부족수 계산 방법과 명시적 저장 구조를 사

하였다. DEF2의 경우에 DEF2/DEF1의 기하평균값이 0.77로써 전반적으로 수행시간이 23% 정도 감소되었다. 그러나, boeing1, fffff800, sctap2, ship04l, stocfor2의 5문제와 같이 오히려 순서화에 소요되는 시간이 증가하는 경우도 발생하였다. 명시적 저장 구조로 보관되어 있는 인접집합을 다시 사용하는 회수가 적을 경우에는 오히려 명시적 저장 구조를 유지하는 데에 드는 비용만큼이 더 추가되기 때문이다.

표 1에서 DEF3/DEF1에 대한 기하평균값은 0.23으로 DEF3을 사용할 경우에 DEF1의 수행시간에서 77%를 절약할 수 있었다. 구별불능점의 대표점만을 고려함으로 인접집합의 계산량을 많이 줄일 수 있었기 때문에 계산량이 증가되는 경우 없이 일반적으로 계산속도의 향상을 얻을 수 있었다. 특히, scsd6, seba와 같은 문제에 대해서는 크기가 큰 구별불능점들이 존재하여 계산량의 감소폭 또한 매우 큰 경우에 해당한다.

5. 결론

최소 부족수 순서화의 계산 과정에서 부족수의 계산과 삭제 그래프의 변형 계산이 중요하다. 부족수의 계산은 많은 인접집합 계산을 요구하는데, 구별불능점의 특성을 이용하여 인접집합의 계산량을 줄일 수 있다. 즉, 구별불능점에 대해서는 구별불능점의 대표점에 대해서만 인접집합을 계산하도록 하여 대표점이 아닌 나머지 점들에 대한 인접집합 계산을 피하도록 할 수 있다. 이 개념은 부족수의 계산뿐만 아니라 삭제 그래프 변형 이후에 추가로 생성된 구별불능점을 탐색하는 계산에서도 적용될 수 있다. 구별불능점의 크기가 클수록 절약할 수 있는 계산량이 늘어나므로 더욱 효과적이다.

삭제 그래프의 변형 계산은 삭제 그래프를 관리하는 자료구조에 그 계산의 효율이 좌우된다. 삭제 그래프를 표현하기 위한 자료구조로는 쿼션트 그래프 구조와 클릭 저장 구조 그리고 명시적 저장 구조가 있다. 이 가운데에 삭제 그래프의 변형을 처리하는 데에는 클릭 저장 구조가 가장 효율적이며, 인접집합에 관련된 계산을 수행하는 데에는 명시적 자료구조가 가장 효율적이다. 따라서, 기본적으로 클릭 저장 구조를 사용하되 인접집합 계산이 필요한 부족수 계산이나 구별불능점 탐색 등이 수행되어야 하는 점들에 대해서만 임시로 명시적 저장구조를 사용하는 것이 효과적이다. 함께 적용되는 명시적 저장구조는 모든 점들에 대해서 유지하는 것이 아니므로, 오히려 클릭 저장 구조에서 중복적인 인접집합의 계산을 피하는 효과를 준다. 즉, 클릭 저장 구조에서 일단 계산되어진 인접집합은 명시적 저장구조로 보관하여 다시 해당되는 인접집합이 요구되었을 때에 클릭 저장 구조로부터 인접집합을 계산해내는 것이 아니라 명시적 저장구조로부터 직접 사용할 수 있도록 하는 것이다.

본 연구에서 제시한 구별불능점을 활용한 부족수 계산 방법과 인접집합을 명시적 저장구조로 임시 보관하는 기법 들을 적용한 경우에 전반적으로 77%의 순서화 속도 향상을 얻을 수 있었다.

참고문헌

[1] 모정훈, 박순달, "최소차수의 자료구조개선과 효율화에 관한 연구", 경영과학, 12권, 2호, , pp. 31-42, 1995. 8.

[2] 박순달, 김병규, 성명기, "내부점기법에 있어서 효율적인 순서화와 자료구조", 한국경영과학회지, 제21권 3호, , pp. 63-74, 1996. 12

[3] 설동렬, 박순달, "내부점 방법에서 최소 부족수 순서화의 실험적 고찰", 97 춘계 한국경영과학회/대한산업공학회 공동학술대회 논문집, 포항공과대학교, pp. 410-413, 1997

[4] Csaba Mészáros, "The inexact minimum local fill-in ordering algorithm", Technical Report WP 95-7, Computer and Automation Institute, Hungarian Academy of Sciences, Budapest, 1995

[5] Duff, I., A. Erisman and J. Reid, *Direct Methods for Sparse Matrices*, Oxford University Press, New York, 1986

[6] Gay, D. M., "Electronic mail distribution of linear programming test problems", *Mathematical Programming Society Committee on Algorithms Newsletter*, No. 13, pp. 10-12, 1985

[7] George, Alan and Joseph W. A. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ., 1981

[8] George, Alan and Joseph W. A. Liu, "The evolution of the minimum degree ordering algorithm", *SIAM Review*, No. 31, pp. 1-19, 1989

[9] Irvin J. Lustig, Roy E. Marsten and David F. Shanno, "The Interaction of Algorithms and Architectures for Interior Point Methods", *Advances in Optimization and Parallel Computing*, P. M. Pardalos(Ed.), Elsevier

Science Publishers, pp. 190-205, 1992

[10] Kou, L. T., L. J. Stockmeyer, C. K. Wong, "Covering edges by cliques with regard to keyword conflicts and intersection graphs", *Comm. ACM*, No. 21, pp. 135-138

[11] Liu, Joseph W. A. "Modification of the minimum-degree algorithm by multiple elimination", *ACM Transaction of Mathematical Software*, Vol. 11, No. 2, pp.141-153, 1985

[12] Markowitz, H. M., "The elimination form of the inverse and its application to linear programming", *Management Science*, No. 3, pp. 255-269, 1957

[13] Robert J. Vanderbei, "A comparison between the minimum-local-fill and minimum-degree algorithm", Technical Report, Princeton University, Department of Civil Engineering and Operations Research, Princeton, NJ, 1990

[14] Yannakakis, M., "Computing the minimum fill-in is NP-complete", *SIAM Journal of Algebraic Discrete Methods*, No. 2, pp. 77-79, 1981

98년 4월 최초 접수, 98년 8월 최종 수정