최소차수순서화의 자료구조개선과 효율화에 관한 연구

모정훈 · 박순달*

Data structures and the performance inprovement of the minimum degree ordering method

Jeonghoon Mo, Soondal Park*

ABSTRACT

The ordering method is used to reduce the fill-ins in interior point methods. In ordering, the data structure plays an important role. In this paper, first, we compare the efficiency and the memory storage requirement of the quotient graph structure and the clique storage. Next, we propose a method of reducing the number of cliques and a data structure for clique storage. Finally, we apply a method of merging rows and absorbing cliques and show the experimental results.

1. 서 론

최근의 선형계획법의 해법에는 계산복잡도 뿐만 아니라 수행속도면에서도 효율적인 내부점기법(Interior Point Methods)이 크게 관심을 끌고 있다[1][5][8][12]. 내부점기법은 이론적인 측면과 구현적인 측면에서 연구되어 왔는데 본 논문은 후자 즉 내부점 기법의 효율적인 구현에 관한 것이다.

모든 내부점기법에서는 $(A\theta A^T)X=b$의 선형시스템을 풀어야 한다. 이 과정은 수행시간의 70-90%를 차지하는 가장 중요한 과정이며 내부점기법의 수행속도는 위의 시스템을 효과적으로 푸는 방법에 달려있다[8]. 이와 같은 풀이방법으로는 $A\theta A^T$의 대칭 양정치성(Symmetric Positive definite)을 이용하여 $A\theta A^T=LL^T$로 분해하는 촐레스키 분해를 사용하는데 여기서 하삼각행렬 L의 비영요소의 갯수를 감소시키는 것이 수

* 서울대학교 산업공학과

행속도에 결정적인 역할을 한다[9]. 이때 L의 비영요소의 갯수는 A의 행의 순서에 따라 달라지는데 이 논문에서는 이 하삼각 행렬의 비영요소를 줄여주는 순서화(Ordering)의 구현과 효율화를 다룬다.

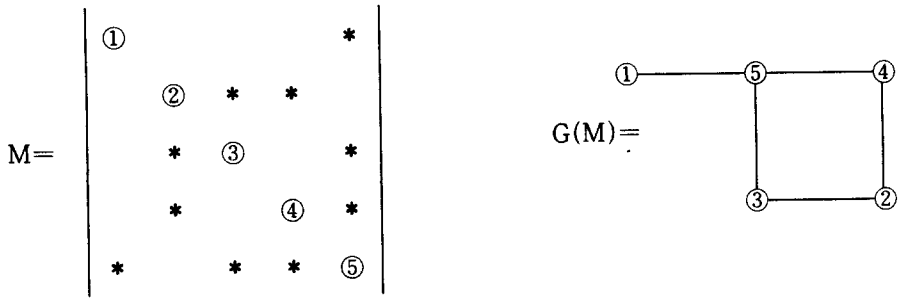
최소의 비영요소를 생성하는 순서를 찾는 문제는 NP-complete로 알려져 있다[7]. 휴리스틱방법중 가장 효과적인 방법중의 하나가 Tinney와 Walker에 의해 발표된 최소차수순서화(Minimum Degree Ordering)이다[3][7]. 최소차수순서화는 차수(인접노드의 수)가 가장 작은 노드를 선택하는 방법으로 그래프를 이용하는데 그래프를 어떻게 표현하는가에 크게 영향을 받는다[7]. George등은 Quotient 그래프를 이용한 자료구조와 방법을 제시하였고[6], Sleepening은 Generalized Element를 이용하여 그래프를 표현하였다[7]. George와 Liu은 최소차수순서화방법의 발전과 문제에 대하여 정리하였고 분별불가능 노드(Indistinguishable Nodes) 기법 등 순서화수행을 효율화하기 위한 여러가지 기법을 소개하였다[7][10].

구현의 측면에서는 Adler등[2], Lustig등[11]이 유한요소해석등에서 연구된 순서화를 내부점기법에 적용하여 효율적인 코드를 구현하였다. 그들은 최소차수순서화와 최소부족순서화(Minimum Deficiency Ordering)를 비교하였다.

이 연구에서는 그래프를 표현하는 방법인 Quotient 그래프 구조와 클릭 저장구조(Clique Storage)를 내부점기법에 적용하고 이를 비교하고자 한다. Quotient 그래프구조의 기존 연구를 이용하여 클릭저장구조를 위한 새로운 자료구조를 제시하고자 한다. 클릭저장구조에서 선형계획법의 A행렬을 이용하여 그래프를 효과적으로 표현할 수 있음을 보이고 클릭흡수기법과 분별불가능노드기법을 적용하여 이를 개선하려고 한다.

2. Quotient 그래프 모델과 클릭 저장 구조

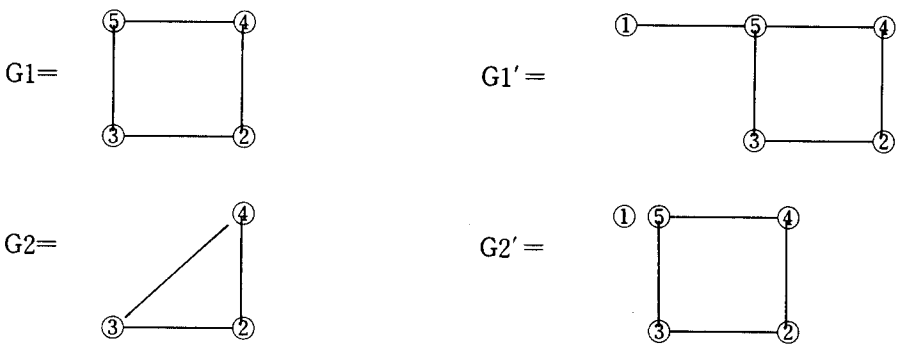
대칭 양정치 행렬의 순서화는 무방향그래프를 이용하여 수행된다[6][7][13]. 임의의 대칭행렬은 무방향 그래프로 표시할 수 있다[7]. 행렬의 행(열)은 그래프의 노드이고 행렬의 i행 j열 요소가 비영인 경우 이는 그래프에서 노드 i와 노드 j를 연결하는 호로 표시할 수 있다. 다음 [그림 1]은 대칭행렬과 그와 연관된 그래프이다.



[그림 1] 대칭 행렬 M과 연관된 그래프 G(M)

Quotient 그래프 모델

Quotient 그래프 모델은 George와 Liu에 의해서 제안된 방법이다[6]. 최소차수순서화방법은 그래프에서 노드를 하나씩 선택하여 삭제하면서 그래프를 변화시키면서 진행된다. 이 때 그 변화하는 그래프를 삭제그래프라고 하는 데, Quotient그래프는 삭제그래프를 표시하는 하나의 방법이다. [그림 1]의 그래프에서 노드 1을 삭제하는 경우 [그림 2]의 G1이 된다. Quotient 그래프 모델은 인접구조를 기본으로 삭제그래프를 표현하는 방법으로 G1을 표시하는 경우 G1'처럼 노드 1에 삭제표시만으로 그래프의 변형을 하지 않고 나타낸다. 두개의 인접노드가 삭제되는 경우 이것을 합하여 하나의 노드로 표시한다. 삭제그래프를 변형하는데 많은 비용이 들기 때문에 노드가 삭제될 때마다 변형을 하는 것이 아니라 삭제된노드를 삭제표시만 했다가 두개이상의 삭제노드가 인접한 경우에 이 삭제된 노드를 하나의 노드로 통합하는 방법이다. G1에서 노드 5가 삭제되는 경우 삭제그래프 G2가 되는 데 이 G2를 Quotient 그래프로는 노드 1과 노드 5를 하나로 합병한 G2'이다.



[그림 2] 삭제그래프와 Quotient 그래프 표현

Quotient 그래프 자료구조의 좋은 특징의 하나는 처음의 삭제그래프를 표시할때 필요한 공간 이상을 필요로 하지 않는 점이다. Quotient 그래프에서 노드를 삭제하게 되면 노드와 호의 수가 계속 감소하므로 처음의 인접호를 보관하였던 기억공간이상의 공간이 더이상 필요하지 않다.

George와 Liu는 위의 Quotient그래프 모델에 분별불가능 노드(Indistinguishable Nodes)를 이용하여 Quotient 그래프모델을 확장시켰다[6]. 분별불가능 노드는 인접노드와 자신을 포함한 집합이 서로 같은 노드로 [그림 2]의 G2의 노드 2, 3, 4가 그 예이다. 분별불가능 노드는 차수가 동일하고 동시에 삭제할 수 있어서 하나의 노드로 표시할 수 있어서 그래프를 축소표현하고 계산량을 감소시킬 수 있다.

위의 방법은 SPARSPAK에 구현되어 있다.

Quotient 그래프 모델을 표현하기 위하여 사용한 자료구조는 다음과 같다.

XADJ[i] : 인접노드 i의 ADJNCY배열의 시작첨자
ADJNCY[i] : 양수 : 인접노드의 번호
음수 : E : 끝표시
그외 : 인접노드의 block수가 2이상인 경우 다음번 block수
ELIM[i] : 노드 i의 삭제표시
0 : 삭제되지 않는 노드
1 : 삭제노드중 병합되지 않는 노드
2 : 삭제노드중 병합된 노드

AVLLIST[i] : 현재 사용할 수 있는 공간의 연결 리스트
BLKSIZE[i] : 노드 i의 인접노드의 block수
AVLHEAD : AVLLIST의 시작주소

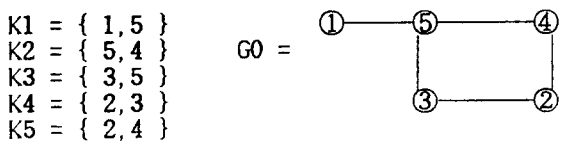
위의 배열에서 XADJ와 ADJNCY는 인접구조를 나타내는 배열이다. 노드 i의 인접노드는 XADJ[i]에서 XADJ[i+1]-1사이의 첨자에 해당하는 ADJNCY배열을 보면 알 수 있다. ELIM은 노드의 삭제표시이다. AVLLIST는 사용되지 않는 block의 번호를 연결 리스트구조로 가지고 있다. AVLHEAD는 AVLLIST의 가장 처음값이다. 이것은 새로운 공간이 필요하게 되는 경우 기억공간을 할당하는데 사용한다. BLKSIZE는 각 노드들의 인접노드가 몇개의 블럭으로 구성되어 있는지를 나타내 주는데 초기의 각 노드의 블럭크기는 1이다.

행렬 A가 M×N행렬이라고 할때 노드의 수는 M이 되고 호의 수는 L=$|AA^T|$이다. 위의 배열 중 XADJ, ELIM, AVLLIST, BLKSIZE는 크기가 노드의 수 M이고 XADJ는 호의 수인 L이 된다. 모두 정수 변수이므로 필요한 기억공간은 (L+4M+1) * sizeof(int)이다. 호의 수는 완전행렬일 경우 M^2이므로 기억공간의 상한은 (M^2+4M+1) * sizeof(int)이다.

클릭저장구조(clique storage)

Speelpenning은 유한요소방법에서 발행하는 문제를 고려하면서 일반화된 요소 접근(Generalized Element Approach)을 사용하였다[7]. Duff 등은 이를 클릭 저장구조라고 했다[4]. 이 관점에 의하면 삭제 그래프를 클릭의 집합으로 표시할 수 있다. 가장 간단한 방법은 주어진 그래프를 호의 수만큼의 두개의 노드로 구성된 클릭의 집합으로 표시될 수 있다.

[그림 3]은 주어진 그래프를 호의 갯수 만큼의 클릭으로 표시한 것이다.



[그림 3] G0의 클릭구조 표현

삭제그래프를 클릭의 집합으로 표시하면 이것은 삭제과정의 개념적으로 다른 관점을 제공할 뿐만 아니라 삭제그래프를 표시할 수 있는 효율적인 방법을 제시한다[7]. 삭제 그래프를 클릭의 집합으로 표시하면 삭제과정을 효과적으로 표시할 수 있다. 삭제과정에서 삭제되는 노드의 인접노드들이 클릭을 형성하기 때문이다.

클릭의 집합 {K_1, K_2, ···, K_q}이 현재의 그래프 G를 나타낸다고 하자. 노드 i가 삭제될 노드라고 할때 집합 {K_{s_1}, ···, K_{s_t}}를 노드 i가 속한 클릭들이라고 하자. 그러면 삭제 그래프를 변형하는 것은 다음과 같은 단계로 나타내 질 수 있다.

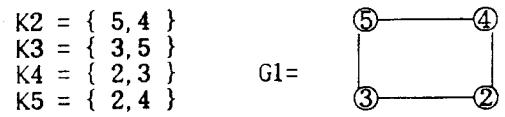
1) 집합 {K_1, K_2, ···, K_q}에서 K_{s_1}, ···, K_{s_t}를 삭제한다.

2) 새로운 클릭 K=(K_{s_1}∪···∪K_{s_t})-{ i }

즉 삭제노드 i가 있는 노드를 모두 삭제하고 이

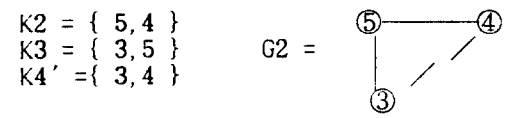
들 클릭의 합집합으로 이루어진 하나의 클릭을 추가하는 것이다.

[그림 3]의 그래프에서 노드 1을 삭제하는 과정은 1이 포함된 클릭 K1을 삭제하고 새로운 클릭 K1′를 만든다. K1′={ 5 }이다. 클릭의 요소수가 1인 경우는 삭제해도 상관없으므로 결과로 생기는 G1의 클릭구조는 다음과 같다.



[그림 4] G1의 클릭구조 표현

노드 2를 삭제하는 과정은 다음과 같다. K4와 K5를 삭제하고 새로운 클릭 K4′={3, 4}가 생긴다.



[그림 5] G2의 클릭구조 표현

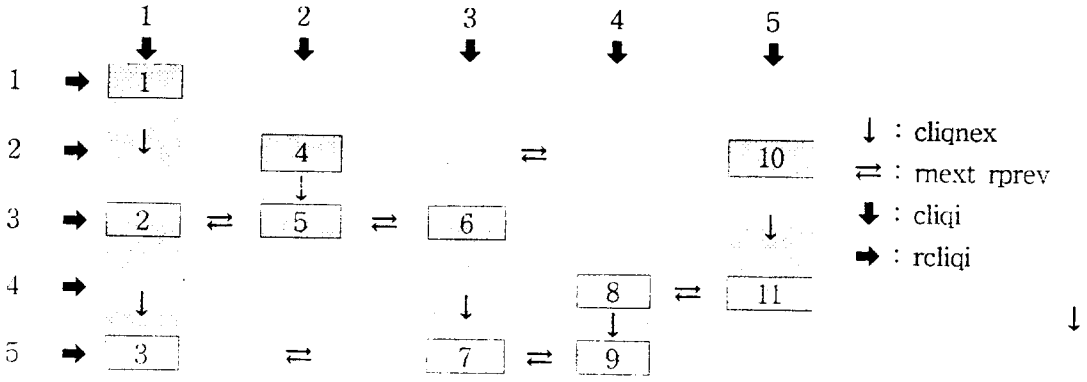
클릭 저장구조를 사용하는 경우 기억공간의 소요는 Quotient 그래프 모델과 같이 처음 필요한 기억공간이상을 필요로 하지 않는다. 새로 형성된 클릭의 집합은 기존 클릭의 집합보다 적은 양의 기억공간을 사용한다. 위의 삭제과정의 표시에서 삭제된 클릭에 의한 여유공간이 항상 새로 추가되는 클릭을 위한 공간보다 크기 때문이다.

클릭 그래프에서 가장 중심적인 연산은 클릭그래프의 합병이다. 클릭 그래프의 합병을 위해서는 임의의 노드 i가 포함되어 있는 클릭을 파악하는 연산과 이들 클릭을 합병해주는 연산이 필요하다.

클릭저장구조를 위해 사용한 자료구조는 다음과 같다.

rcliqi[i] : 노드 i의 시작 주소
rnext[MAXA], rprev[MAXA] : 노드 i들의 이중 연결 리스트
cliquelt[MAXA] : 클릭의 원소값
cliqset[MAXA] : cliqult[i]가 몇번째 클릭에 포함되었는가를 표시
cliqi[i] : 클릭 i의 cliqelt에서의 시작주소
cliqnex[MAXA] : 클릭집합의 다음 원소를 가리키는 포인터
cliqelm[i] : 클릭 i의 삭제표시 0이면 비삭제 1이면 삭제된 클릭

[그림 7]과 같이 주어진 A행렬이 있을때 만들어지는 클릭 집합과 자료구조는 다음과 같다.



[그림 6] 클릭구조의 개념적 표현

클릭집합의 원소들은 모두 cliqnex라는 포인터에 의하여 연결 리스트로 이어져 있다. cliqi는 클릭의 처음 요소를 파악하는 데 사용하며 cliqnex를 이용하여 다음 원소를 파악할 수 있다.

클릭의 요소의 값이 같은 것들은 rprev와 rnext의 이중 연결 리스트구조로 연결되어 있다. 이중 연결 리스트를 사용한 이유는 삭제를 용이하게 함을 위해서이다. 클릭의 집합을 합병하는 경우 동일한 원소가 2번이상 나오는 경우 삭제하여야 하는 데 이중 연결 리스트를 사용하면 간단한 연산으로 삭제할 수 있으므로 효율적이다.

예를 들면, 노드 3이 삭제되는 경우에 클릭 1, 2, 3이 합병이 된다. 클릭 1 = {1, 3, 5}이고 클릭 2={2, 3} 그리고 클릭 3={3, 5}이므로 형성되는 클릭은 {1, 2, 5}이다. 이 경우 중복되는 요소인 1과 5의 삭제필요가 발생한다. 이 경우 이중 연결 구조인 경우 포인터를 바꾸어 줌으로써 간단히 수행할 수 있다.

위의 배열중 cliqelt, cliqset, cliqnex, rnext, rprev의 배열의 크기는 A의 비영요소의 수이다. 그리고 배열 cliqi, cliqelm의 크기는 행렬 A의 열수 N이고 rcliqi는 A의 행수 M의 크기의 배열이다. 따라서 총기억공간 소요량은 (5 * Nonz(A)+M+N) * sizeof(int)이다. Nonz(A)는 행렬 A의 비영요소의 수이다.

Quotient 그래프 구조의 경우는 Nonz(AA^T)의 수에 비례하지만 클릭구조의 경우는 Nonz(A)의 수에 비례한다.

3. 클릭저장구조의 개선

임의의 그래프가 주어졌을때 이를 클릭의 집합으로 표시하는 가장 간단한 방법은 그래프의 호를 두개의 노드로 구성된 클릭으로 표시하는 것이다. 이 경우 그래프의 호의 수에 해당하는 클릭이 형성된다. [그림 3]에서 그래프 G0을 5개의 호로 형성된 클릭으로 표시하는 경우가 여기에 해당한다. 클릭의 수가 많으면 노드의 차수를 결정하고 또 클릭을 관리하는 데 드는 계산비용이 많이 들어서 바람직하지 못하다.

내부점 선형계획법에서는 행렬 AA^T에 연관된 그래프를 표현하는 클릭의 집합을 결정하여야 한다. 이 경우 행렬 A를 이용하여 AA^T에 해당하는 그래프를 적은 수의 클릭으로 표현할 수 있다.

주어진 행렬 A의 각 열들의 집합은 행렬 AA^T에 해당하는 그래프를 표현하는 클릭집합을 나타낸다. M=AA^T에서 행렬 M의 i행 j열을 m_{ij}라고 할 때 m_{ij}가 비영요소가 되는 경우(그래프에서 호 (i, j)가 존재하는 경우)는 $a_{ik}\neq 0$인 열 k가 존재하는 경우이다. 이것은 m_{ij}값이 A_i와 A_j의 내적으로 결정된다는 것에서 자명하다. 행렬 A의 임의의 열 k에서 비영요소가 행 I={i_1, i_2, ···, i_l}의 위치에 존재한다면 I의 원소로 이루어지는 모든 쌍 (p, q)에 대하여 $m_{pq}\neq 0$이다. 단 p, q∈I. 이것은 그래프의 관점에서 보면 I에 포함된 노드들은 모든 연결된 클릭을 형성한다는 것이다. 각 열들이 하나의 클릭을 표현하고 모든 열들의 집합은 AA^T를 표현하는 그래프를 나타내게 된다.

다음과 같은 비영요소를 가진 A행렬이 주어졌다고 하자. *는 비영요소이다.

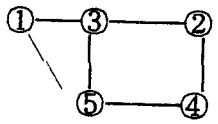
		1	2	3	4	5	6
	1	*					
	2		*				*
A =	3	*	*		*		
	4			*		*	*
	5	*			*	*	

[그림 7] 임의의 A행렬의 구조

이에 해당하는 AA^T의 구조와 그래프는 다음과 같다.

$AA^T=$

1		*		*
	2	*	*	
*	*	3		*
	*		4	*
*		*	*	5



[그림 8] AA^T의 구조와 해당하는 그래프

위의 그림을 보면 A의 각 행의 비영요소의 열은 각각 {1, 3, 5}, {2, 3}, {4}, {3, 5}, {4, 5}, {2, 4}이다. 위의 열의 집합은 각각 AA^T에 해당하는 그래프의 클릭집합을 형성함을 알 수 있다. {4}는 형성할 수 있는 호가 없으므로 의미가 없다. 그리고 {3, 5}⊂{1, 3, 5}이므로 필요가 없다. 따라서 위 그래프는 {1, 3, 5}, {2, 3}, {4, 5}, {2, 4}의 4개의 클릭으로 표시됨을 알 수 있다.

위의 관찰은 AA^T에 해당하는 그래프의 클릭집합을 보다 적은 수의 클릭으로 형성할 수 있는 방법을 제시해준다. 즉 각 행의 비영요소의 열의 첨자로 클릭의 집합을 표시하면 그것은 AA^T에 해당하는 그래프를 표현하는 클릭의 집합이다. 이 경우 AA^T를 표현하는 클릭의 수는 행렬 A의 열의 수를 초과하지 않는다. 이 때 비영요소의 수가 1인 열은 제외할 수 있다.

이 방법은 순서화를 수행하기 위해 AA^T를 계산할 필요가 없이 A의 자료구조를 이용하여 쉽게 구할 수 있게 해 준다. 대개의 선형계획법의 자료구조가 열위주로 들어오기 때문이다.

클릭의 흡수(clique absorption)

클릭 저장구조에서는 그래프를 형성하는 클릭의 수가 작을수록 순서화에 유리하다. 왜냐하면 클릭의 수가 작으면 차수를 계산할 때나 삭제변형을 수행할 때 비용이 감소하기 때문이다. 클릭의 흡수란 클릭수를 감소시키기 위해 중복되는 클릭을 제거하는 것이다.

클릭의 집합 $\{K_1, K_2, \cdots, K_q\}$이 현재의 그래프 G를 나타낸다고 하자. 만약 어떤 s와 t에 대하여 $K_s \subset K_t$이면 그래프 G는 $\{K_1, K_2, \cdots, K_q\}-K_s$로 표현될 수 있다. 이 경우 K_s는 그래프 G를 중복적으로 표현하고 있는 클릭이고 이는 K_t에 의해서 흡수되어 삭제될 수 있다. [그림 7]의 경우 클릭 {3, 5}는 클릭 {1, 3, 5}에 포함되므로 흡수할 수 있다.

행렬 A에서 각 열의 비영요소는 행렬 AA^T에 연관된 그래프를 나타내는 클릭의 집합으로 간주할 수 있다. A의 임의의 열 j의 비영요소의 행번호의 집합을 $Nonz(A_{\cdot j})$로 표현하겠다. $Nonz(A_{\cdot s}) \subset Nonz(A_{\cdot t})$이면 s번째 열에 의해서 형성되는 클릭은 삭제해도 무관하다. 특히 행렬 A의 구조가 블락구조인 경우에나 A의 열의 수가 행의 수에 비해 많은 경우 AA^T의 그래프를 표현하는 클릭이 중복적인 경우가 많을 수 있다.

본 연구에서는 순서화 알고리듬을 수행하기 전

단계에서 중복클릭을 제거하는 것과 삭제연산 직후 생성된 클릭에 포함되는 중복클릭을 제거하는 것을 고려했다. 순서화 직전의 클릭 흡수는 행렬 A를 이용하여 여기에서 중복되는 클릭을 제거하는 것이다. 이것은 모든 클릭에 대하여 수행하는 작업이다. 삭제연산 직후 중복클릭을 제거하는 것은 새로 형성된 클릭 K에 포함되는 클릭의 집합을 제거하는 것이다.

순서화직전의 중복클릭제거는 다음과 같은 방법으로 수행한다. 일단 클릭의 집합을 비영요소의 감소의 순서로 소팅한다. 이것은 뒤쪽에 있는 클릭에 대해서만 흡수여부를 판단하기 위해서이다. 이 때 타이가 생기는 경우는 클릭의 최소원소의 증가순으로 소팅한다. 이것도 같은 이유에서이다. 소팅이 끝나게 되면 처음 클릭부터 차례로 다음에 존재하는 클릭을 포함할지를 조사한다. 클릭의 포함여부를 판단하기 위해서는 하나의 표시(flag) 배열을 사용하는데 먼저 포함할 클릭에 해당하는 원소의 표시배열을 모두 셋팅하고 다음에 포함되는 클릭의 원소에 해당하는 표시배열이 모두 셋팅되어있는지를 조사한다. 이때 표시배열이 셋팅되어 있지 않으면 흡수되지 않는 경우다.

순서화도중에 클릭흡수는 삭제연산 후 형성되는 클릭 K에 대하여 이에 포함되는 클릭이 있는지를 조사하는 것이다. 이 경우도 하나의 표시배열을 사용한다. 이 때 형성된 클릭의 rnext배열을 이용하여 여기에 해당하는 클릭이 새로 형성된 클릭에 포함되는지를 조사하면 된다.

클릭흡수기법을 사용함으로 얻어지는 이득이 있고 중복클릭을 찾기 위한 비용도 있다. 이에 대한 실험결과는 뒤에서 제시하겠다.

분별불가능 노드

분별불가능 노드를 사용하는 경우 순서화의 수행속도가 감소되는 것은 잘 알려져 있다[7]. 분별불가능 노드기법은 여러개의 노드를 하나의 대표노드로 표시함으로 차수계산의 감소와 동시삭제를 통한 그래프 변환의 감소로 수행시간의 절감을 가져온다. 인접구조에 기반을 둔 Quotient구조의 경우는 인접노드를 쉽게 찾음으로 정의에 의하여 분별불가능 노드를 찾을 수 있다. 클릭저장구조는 인접노드를 찾는 것이 Quotient구조보다 복잡하다. [그림 7]에서 노드 3의 인접노드를 찾는 경우 노드 3이 포함된 3개의 클릭 {1, 3, 5}, {2, 3}, {3, 5}을 찾아 이들의 요소를 모두 조사하여야 한다. 즉, 인접노드를 찾기 위해서는 노드가 포함된 클릭을 모두 찾아야 한다.

Duff는 분별불가능 노드를 찾는 방법을 인접노드를 모두 찾는게 아니라 클릭을 이용하여 구했다[14]. 임의의 노드 i를 포함하고 있는 클릭들의 집합을 K(i)라고 할 때 K(i)=K(j)이면 노드 i와 노드 j는 분별불가능 노드이다. 인접노드의 집합은 K(i)의 원소들의 합집합이므로 K(i)=K(j)이면 두 노드가 분별불가능하다는 것은 자명하다. [그림 9]에서 노드 2가 포함된 클릭의 집합과 노드 5가 포함된 클릭의 집합은 클릭 1, 3, 4로 서로 같다. 잉 경우 노드 2, 5는 서로 분별불가능하다.

	클릭 1	클릭 2	클릭 3	클릭 4
노드 1		*		
노드 2	*		*	*
노드 3		*		*
노드 4	*		*	
노드 5	*		*	*

[그림 9] 클릭의 집합

이 방법은 인접노드를 찾지 않고도 분별불가능

노드를 찾을 수 있지만 분별불가능 노드를 모두 찾을 수 없다. 노드 5의 경우 K(5)는 K(2)와 다르지만 이들은 분별불가능 노드이다.

이웃집합(인접노드의 집합과 신을 포함하는 집합)을 이용하게 되면 노드 2, 4, 5는 분별불가능 노드를 형성하게 된다.

본 연구에서는 분별불가능노드를 찾는 방법으로 클릭을 이용하는 방법과 그리고 인접노드를 찾아 이웃집합을 비교하는 방법을 비교하였다. 첫번째 방법은 분별불가능 노드를 찾는 비용이 두번째 방법보다 적음을 알 수 있다. 그리고 두 번째 방법은 분별불가능 노드를 찾는 비용은 더 많이 들지만 많은 분별불가능 노드를 찾을 수 있다.

따라서 클릭을 이용하는 방법과 인접구노는 비교하는 방법은 서로 trade-off이다. 이에 대한 실험결과는 다음에 있다.

4. 실험 및 결과

순서화를 위한 자료구조인 Quotient 그래프 모델과 클릭 저장구조를 수행속도와 기억공간 소요면에서 분석하겠다.

본 논문의 실험결과는 다음의 표에 제시된 10개의 실험문제를 가지고 제시하겠다. 이 문제들은 Netlib Test Set에서 발췌된 모델이다. 다음의 문제들은 문제크기에 따라 정렬되었다. 본문의 실험은 HP 9000/730 워크스테이션에서 수행되었다.

[표 1] 실험문제

문제이름	문제크기		비영요소수	density(%)
agg2	517	302	4531	2.90
agg3	517	302	4515	2.90
fffff800	525	854	6235	1.39
fnll	644	1175	6129	0.81
scfxm2	661	914	5229	0.87
ship08s	779	2387	9501	0.51
25fv47	822	1571	11127	0.86
sctap2	1091	1880	8124	0.39
stocfor2	2158	2031	9492	0.22

[표 2] 수행속도 비교

(단위 : Sec)

	1	2	3	4	5 *	6 **
클릭 합병	×	×	○	×	○	○
분별불가능 노드	×	방법 1	방법 1	방법 2	방법 2	방법 2
agg2	2.9	1.0	1.0	**0.9**	**0.9**	**0.9**
agg3	2.9	0.9	0.9	**0.9**	**0.9**	**0.9**
fffff800	3.9	3.6	3.5	2.5	2.4	**2.1**
bnll	2.4	2.7	2.5	2.3	2.3	**1.6**
scfxm2	1.2	1.4	1.4	1.3	1.4	**1.1**
ship08s	**4.3**	6.4	6.1	5.9	6.2	10.3
25fv47	6.0	5.0	4.6	4.2	4.0	**3.4**
sctap2	2.4	2.5	2.5	2.7	2.7	**1.8**
stocfor2	5.2	6.3	6.4	6.2	6.3	**3.3**

* 5번 방법 : 초기클릭흡수 사용 ** 6번 방법 : 초기클릭흡수 사용안함

위의 표는 수행도중 클릭합병을 사용하는 것과 사용하지 않는 경우 그리고 분별불가능 노드를 찾는 방법에 따른 수행시간의 비교이다. 방법 1과 방법 2는 분별불가능 노드를 찾는 방법에 의한 구분이다. 방법 1은 노두가 포함된 클릭의 집합이 같은 경우로 찾는 것이고 방법 2는 인접노드를 모두 찾아서 인접노드가 같은 경우로 분별불가능 노드를 찾는 것이다.

방법 1과 방법 2를 비교하면 방법 2가 즉 인접노드를 찾아서 비교하는 경우가 수행속도면에서 유리함을 알 수 있다. 이것은 방법 2에서 찾은 분별불가능 노드의 수가 방법 1에서 찾은 분별불가능 노드보다 많은 데에서 연유한다. 방법 2가 방법 1보다는 분별불가능노드를 찾는데 걸리는 시간은 많지만 방법 2의 분별불가능 노드를 찾음으로써 얻어지는 이득이 있기 때문이다.

[표 3] 방법 1과 방법 2에서 찾아진 분별불가능 노드의 수

문제이름	방법 1	방법 2
agg2	372	377
agg3	370	377
fffff800	141	251
bnll	200	221
scfxm2	240	285
ship08s	134	184
25fv47	337	397
sctap2	175	224
stocfor2	506	618

방법 1은 방법 2보다 찾은 분별불가능 노드의 수가 적다. 이것이 표 2에서 나타난 방법 2의 수행시간이 적은 이유이다. 즉 분별불가능 노드의 수에서의 차이가 방법 2의 수행시간의 빠름을 설명해준다.

[표 2]에서 보면 클릭흡수기법은 방법 1과 방법 2에서 별다른 차이를 나타내지 못한다. 흡수된 클리의 수가 많지 않아서 클릭흡수로 인한 효과가 감소되기 때문인것 같다. 이것은 클릭합병의 비용과 합병후 차수 수정비용등의 trade-off때문이다.

ship08s의 경우는 클릭합병과 분별불가능 노드를 사용하지 않는 경우의 시간이 가장 빠르다. 위의 표에는 나타나 있지는 않지만 netlib문제중 scsd1, scsd6등 sc계열의 문제와 ship계열의 문제는 클릭합병과 분별불가능 노드를 사용하지 않는 경우에 우수한 성능을 나타냈다. sc계열이나 ship계열의 문제는 A의 비영요소의 형태가 열의 수가 행의 수에 비해 현저히 많고(scsd1의 경우 78 * 760) 계단 모양을 이루고 있으며 행의 비영요소의 수가 아주 많은 형태를 취하고 있다. 따라서 한개의 노드가 삭제되는 경우 이에 의하여 합병되는 클릭의 수가 아주 많고 또 클릭을 합병하는데 드는 비용이 많다. 또 분별불가능 노드를 찾기 위하여 모든 클릭을 검색해야 하는데 행보다 열의 수가 많기 때문에(노드에 비해 클릭의 수가 많음) 검색비용이 많이 든다.

위의 실험결과중 6번 열의 결과가 전반적으로 우수하게 나타났다. 이것은 초기클릭흡수를 사용하지 않고 방법 2로 분별불가능노드기법을 사용하고 중간클릭흡수를 사용한 것이다. 5번은 초기클릭흡수를 한것이고 6번은 초기클릭흡수를 사용하지 않은 것이다. 초기클릭흡수가 별 효과가 없는 것으로 나타났다.

Quotient 그래프 구조의 기억공간소요는 (L+4M+1)이고 클릭그래프 구조의 기억공간소요는 (5 * Nonz(A)+M+2N)이다. M과 N은 행렬 A의 행과 열의 수이다. 그리고 L=Nonz(AA^T)이다.

[표 4] 기억공간과 수행속도 비교

(단위 : *2 Byte, 초)

문제이름	AA^T의 크기	Quotient 그래프 구조		클릭 저장구조	
agg2	25754	27823	4/7	23776	0.9
agg3	25774	27843	4.7	23696	0.9
fffff800	20149	22250	6.6	33408	2.1
bnl1	8786	11363	5.4	33639	1.6
scfxm2	11650	14295	4.3	28634	1.1
ship08s	9453	12570	5.2	53508	10.3
25fv47	22145	25434	13.2	59599	3.4
sctap2	7007	11372	10.1	79471	1.8
stocfor2	25467	34100	25.73	53680	3.3

클릭저장구조의 수행속도는 분별불가능 노드기법과 클릭 흡수기법을 사용한 것이다. agg2, agg3문제를 제외하고는 클릭저장구조의 기억공간이 Quotient구조의 기억공간보다 많이 차지함을 알 수 있다. agg문제의 경우는 A의 비영요소의 수가 AA^T의 비영요소에 비해 상대적으로 많기 때문이다. 즉 AA^T의 비영요소수가 많은 문제에서는 클릭구조가 유리함을 알 수 있다.

수행속도의 면에서 보면 Quotient구조의 수행시간보다는 클릭 저장구조의 수행시간이 ship04s를 제외하고는 우수함을 알 수 있다. ship08에서는 Quotient구조가 우수했다. 위에서 이야기 했듯이 ship계열은 클릭저장구조에서 아무런 기법을 사용하지 않는 것이 우수하다. 전반적으로 보았을 때 수행속도 면에서 클릭구조가 약 70%−100% 이상이 빠르다.

5. 결론

본 연구에서는 내부점 선형계획법을 위한 순서화 방법에 대하여 다루었다. 순서화방법은 양정치 선형시스템 $(A\theta A^T)x=b$를 효과적으로 풀기 위하여 $(A\theta A^T)$의 비영요소의 수를 감소시키는 방법이다.

비영요소의 수를 감소시키는 순서화방법으로 본 논문에서 다룬 방법은 최소차수순서화방법이다. 내부점 기법에서 AA^T에 해당하는 그래프를 클릭저장구조를 이용하는 경우 AA^T를 계산하지 않고 A를 이용하여 효과적으로 수행할 수 있는 것을 보였다. 그리고 클릭 저장구조의 수행속도가 Quotient 그래프구조보다 좋았다. 기억공간소요면에서는 Quotient 그래프 구조가 적은 공간을 사용하였다.

그리고 클릭저장구조에서 클릭흡수기법과 분별불가능 노드를 찾는 방법등을 비교설명하였는데 분별불가능 노드를 클릭을 이용하여 결정는 것보다는 인접노드를 비교하여 찾는 방법이 우수하였다. 클릭흡수기법에서는 별다른 차이를 나타내지 못했다.

참 고 문 헌

[1] 박순달, 선형계획법, 제3판, 민음사, 1992

[2] Adler, I., N. Karmarkar, M. G. C. Resende and G. Veiga, "Data strucutures and programming techniques for the implementation of Karmarkar's algorithm", ORSA Jour. on comp. 1, pp. 84-106, 1989

[3] Berman, P., Georg Schintger, "On the performance of the minimum degree ordering for Gaussian elimination", SIAM J. Matrix Anal. Appl.,11,1,pp. 83-88, 1990

[4] Duff,I. S., A. M. Erisman and J. K. Reid, Direct methods for sparse matrices, Clarendon Press, Oxford, 1986

[5] Fang,S. C., S. Puthenpura, Linear optimization and extensions : theory and algorithms, Prentice-Hall, Inc. 1993

[6] George, J. A. and J. W-H Liu, Computer solution of large sparse positive definite systems, Prentice-Hall, Englewood Cliffs, NJ, 1981

[7] George, J. A. and J. W-H Liu, "The evolution of the minimum degree ordering algorithm", SIAM review 31,1, ppl-19,1989

[8] Jung, H. W., "Direct sparse matrix methods for interior point algorithm", Ph. D. Dissertation, Univ. of Arizona, 1990

[9] Jung, H. W., R. E Marsten, M. J. Saltzman, "Numerical factorization methods for interior point algorithms", ORSA Jour. on comp. 6,1 pp. 94-105, 1994

[10] Liu, J. W-H., "Modification of the minimum-degree algorithm by multiple elimination", ACM transactions on Math. Software.,11,2,pp. 141-153,1985

[11] Lustig, I. J., Roy E. Marsten, D. F. Shanno, "The interaction of algorithms and architectures for interior point methods", Rutcor Research Report, RRR # 36-91, July 1991

[12] Marsten, R., R. Subramanian, and M. Saltzman, "Interior point methods for linear programming: Just call Newton, Lagrange, and Fiacco & McCormick", Interfaces 20,4, pp. 105-116, 1990

[13] Rose, D, J. "A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations", Graph Theory and Computing, ed. R. Read, pp. 183-217, 1972

[14] Duff, I. S., J. K. Reid, "The multifrontal solution of indefinite sparse symmetric linear equations", ACM transactions on Math. Software., 9, 3, pp. 302-325,1983