韓國經營科學會誌
第24卷 第1號
1999年 3月

내부점 방법을 위한 사전처리의 구현*

성명기** · 임성묵** · 박순달**

An Implementation of Preprocessing for Interior Point Methods for Linear Programming*

Myeongki Seong** · Sungmook Lim** · Soondal Park**

■ Abstract ■

We classified preprocessing methods into (1) analytic methods, (2) methods for removing implied free variables, (3) methods using pivot or elementary row operations, (4) methods for removing linearly dependent rows and columns and (5) methods for dense columns. We noted some considerations to which should be paid attention when preprocessing methods are applied to interior point methods for linear programming. We proposed an efficient order of preprocessing methods and data structures. We also noted the recovery process for dual solutions.

We implemented the proposed preprocessing methods, and tested it with 28 large scale problems of NETLIB. We compared the results of it with those of preprocessing routines of HOPDM, BPDPM and CPLEX.

1. 서 론

선형계획문제의 해법으로는 정점해를 찾아가면서 최적해를 구하는 단체법(simplex method)과 가능해 영역의 내부에서 해를 이동하여 최적해를 구하는 내부점 방법(interior point method for linear programming) 등이 있다[1]. 이들 방법을 구현한 프로그램에서 문제를 풀려면 우선 문제를 입력받아야 하는데, 대형 문제의 경우에는 입력할 자료가 많고, 또 행렬 A의 대부분의 요소가 0이기 때문에 MPS 형태[1]와 같이 A의 비영요소만을 입력하게 된다. 이러한 경우 공백행(empty row)과 공

* 본 연구는 한국과학재단의 특정기초연구과제(과제번호 98 0200 07-01-2)에 의해 지원되었음.
** 서울대학교 산업공학과

백열(empty column)이 생길 수 있게 된다. 또 문제가 크기 때문에 중복적이거나 비가능인 제약식이 생길 수 있고, 또 변수의 상한과 하한에 오류가 생길 수 있다. 그래서 선형계획 프로그램에서는 문제를 입력받은 후, 해법 수행 전에 필요 없는 제약식이나 변수를 제거해야 할 필요가 있는데, 이러한 일을 사전처리(preprocessing)라고 한다[2][7].

사전처리로 얻을 수 있는 이익으로는, 해법을 수행하기 전에 문제의 비가능성을 판정할 수 있고, 중복제약식의 제거 및 변수 값의 고정 등으로 문제 크기를 작게 만들어 보다 빨리 최적해에 도달할 수 있다는 것 등이다[2][6]. 내부점 방법의 경우에 중복제약식의 제거는 촐레스키 분해(Cholesky factorization)를 보다 안정적으로 수행하게 해 준다는 점에서도 중요하다[9].

내부점 방법을 위한 사전처리를 할 때 단체법의 경우와 다르게 주의해야 할 것으로는 다음과 같다.

- 문제의 크기를 줄이거나 비영요소의 수를 줄일 때 촐레스키 분해의 결과 생성되는 촐레스키 요소(Cholesky factor)의 희소도(sparsity)가 나빠지지 않게 해야 한다[6].
- 변수의 상한과 하한의 간격을 지나치게 강화하면 수치적으로 불안정해질 수 있다[6].
- 자유변수의 수를 최대한 줄이는 것이 유리하다[8].

본 논문에서는 사전처리 방법들 중 내부점 방법을 적용하는데 있어서 필요한 방법들과 이의 효율적인 구현하는 방법에 대해 연구하였고, 논문에서 제안한 방법을 구현한 결과를 다른 내부점 방법 프로그램들과 비교해 보았다.

2. 사전처리 방법

일반적인 선형계획문제는 (P)와 같은 일반한계문제이다. (D)는 (P)의 쌍대문제이다.

$$\text{(P)}\quad \begin{aligned} &Max && \boldsymbol{c}^T\boldsymbol{x} \\ &s.t. && A\boldsymbol{x} \le \boldsymbol{b} \\ & && \boldsymbol{l} \le \boldsymbol{x} \le \boldsymbol{u} \end{aligned}$$

$$\text{(D)}\quad \begin{aligned} &Min && \boldsymbol{b}^T\boldsymbol{y} + \boldsymbol{u}^T\boldsymbol{w} - \boldsymbol{l}^T\boldsymbol{z} \\ &s.t. && A^T\boldsymbol{y} + \boldsymbol{w} - \boldsymbol{z} = \boldsymbol{c} \\ & && \boldsymbol{z},\ \boldsymbol{w} \ge 0 \end{aligned}$$

여기서 A는 $m \times n$ 행렬이고, $\boldsymbol{x}$, $\boldsymbol{l}$, $\boldsymbol{u}$, $\boldsymbol{c}$는 n차 벡터, 그리고 $\boldsymbol{b}$, $\boldsymbol{y}$는 m차 벡터이다.

내부점 방법을 적용하기 위해서는 문제 (P)를 단순상한문제로 변형하여야 하는데, 본 논문에서는 일반한계문제에 대해서 사전처리를 한 후 단순상한문제로 변형하는 방법을 사용하였다. 왜냐하면 변수 중에 상·하한이 모두 무한 값을 가지는 자유변수가 있을 수 있는데, 단순상한문제로 변형하면 이들이 두 개의 변수로 나누어지게 되어서 사전처리 방법들을 구현하기 복잡해지고, 또 특정 사전처리 방법을 적용하기 힘들어지기 때문이다.

본 논문에서 구현한 사전처리 방법들은 다음과 같이 분류될 수 있다.

- 분석적 방법
- 암시적 자유변수(implied free variable) 제거 방법
- 선회연산(pivot operation) 또는 기본행연산(elementary row operation)을 이용한 방법
- 선형종속인 행과 열의 제거
- 밀집열(dense column) 처리 방법

각 방법에 대해서 설명하면 다음과 같다.[2][4][5][6][7]

2.1 분석적 방법

명백한 중복 제약식과 필요 없는 변수의 제거, 변수의 한계를 강화시키는 방법 등이 있다. 이 중에서 변수의 한계(upper and lower bound)를 강화하는 방법은 제약식과 변수를 추가적으로 제거할

수 있도록 해 준다.

변수가 고정되면 목적함수의 상수 값이 변하게 된다. 따라서 사전처리된 문제는 원래 문제와 목적함수 값이 다를 수 있다. 각 변수와 목적함수의 원래의 값은 해법을 종료한 다음 해의 복원 과정을 거쳐 구해진다.

2.1.1 공백행 및 공백열 제거

공백행에 대해서는 제약식의 형태와 우변상수 값에 따라 그 제약식이 중복인가 비가능인지를 판정한다. 공백열에 대해서는 목적함수 계수의 부호 및 유한성에 따라 해당 변수의 값을 고정할 것인지 무한해를 가지는지 판정한다.

이 과정들은 처음에 한 번만 수행하는 것이 아니라 다른 기법들의 중간 과정에 나타날 수도 있다.

2.1.2 단일요소행 제거

단일요소행(singleton row)이란 제약식에 변수가 하나만 있는 제약식이다. 등식 제약식의 경우에는 변수의 값이 유일하게 결정되기 때문에 변수의 값을 고정하거나 비가능으로 판정할 수 있다. 다른 형태의 제약식들에서는 제약식의 형태와 우변상수 값에 따라 변수 값의 고정, 한계의 변경, 그리고 비가능 판정 등이 일어나게 된다.

2.1.3 단일요소열의 제거

단일요소열(singleton column)이란 하나의 제약식에만 영향을 미치는 변수를 의미한다. 만일 단일요소열에 해당되는 목적함수 계수가 영이라면 그 변수는 어떠한 값을 갖더라도 목적함수에 직접적인 영향을 주지 않고, 단지 하나의 제약식에만 영향을 주게 된다. 따라서, 목적함수 계수가 영인 단일요소열은 해당 제약식에 그 영향을 반영한 후 제거되어질 수 있다. 즉, 해당 제약식의 우변상수가 변경된 후 단일요소열은 제거될 수 있다.

2.1.4 제약식의 최대·최소값을 이용한 중복 제약식의 제거

각 변수의 상한과 하한 값을 이용하여 다음과 같이 i번째 제약식이 가질 수 있는 최대값과 최소값을 구할 수 있다.

$$\underline{b}_i = \sum_{j\in P_i} a_{ij}l_j + \sum_{j\in N_i} a_{ij}u_j$$
$$\overline{b}_i = \sum_{j\in P_i} a_{ij}u_j + \sum_{j\in N_i} a_{ij}l_j$$

단, $P_i = \{j \mid a_{ij} > 0\}$, $N_i = \{j \mid a_{ij} < 0\}$

그러면 다음의 관계가 성립한다.

$$\underline{b}_i \le \sum_j a_{ij}x_j \le \overline{b}_i$$

i번째 제약식이 '$\le$'인 경우에 우변상수 값에 따라 다음의 결과가 성립한다.

- $b_i < \underline{b}_i$: 비가능
- $b_i = \underline{b}_i$: $\sum_j a_{ij}x_j$의 값이 $\underline{b}_i$가 되도록 각 변수의 값을 고정하고 제약식을 제거한다.
- $\overline{b}_i \le b_i$: 중복제약식이므로 제거한다.
- $\underline{b}_i < b_i < \overline{b}_i$: '제거불능'이다.

'$\ge$' 형태의 제약식에 대해서는 이와 비슷한 관계가 성립하며, '$=$' 제약식에 대해서는 '$\le$, $\ge$'에 대한 결과가 모두 성립한다.

2.1.5 변수의 한계 강화

앞에서 설명한 방법에서 제약식을 제거할 수 없는 경우에 대해서 $\underline{b}_i$, $\overline{b}_i$의 값을 이용하여 변수들의 한계를 변경시키는 방법이다. 이 방법은 다른 사전처리 방법을 적용할 때 변수와 제약식을 더 많이 제거할 수 있도록 도와주는 기능도 있다.

앞에서와 마찬가지로 '$\le$' 형태의 제약식에 대해서만 설명하겠다.

'$\le$' 형태의 제약식에는 다음이 성립한다.

$$\begin{aligned}&\text{if } \forall k\in P_i,\\ &\underline{b}_i + a_{ik}(x_k - l_k) \le \sum_j a_{ij}x_j \le b_i\\ &\text{if } \forall k\in N_i,\\ &\underline{b}_i + a_{ik}(x_k - u_k) \le \sum_j a_{ij}x_j \le b_i\end{aligned} \tag{2.1}$$

(2.1)에서 다음과 같이 암시적 한계를 유도할 수 있다.

$$\begin{aligned}x_k &\le u_k' = l_k + (b_i - \underline{b}_i)/a_{ik} \quad \forall k\in P_i\\ x_k &\ge l_k' = u_k + (b_i - \underline{b}_i)/a_{ik} \quad \forall k\in N_i\end{aligned}$$

새로 구해진 한계 (u_k', l_k')와 기존의 한계 (u_k, l_k)를 비교해서 한계를 강화하거나 비가능을 판정할 수 있다.

한편, 제약식에서 한 개의 변수만 무한일 때, 즉 $l_k = -\infty$, $\exists k\in P_i$ 또는 $u_k = +\infty$, $\exists k\in N_i$ 인 경우에 x_k에 대한 암시적 한계를 유도할 수 있다. 이러한 경우에 (2.1)은 (2.2)로 교체된다.

$$\begin{aligned}&a_{ik}x_k + \sum_{j\in P_i - \{k\}} a_{ij}l_j + \sum_{j\in N_i} a_{ij}u_j\\ &\quad \le \sum_j a_{ij}x_j \le b_i, \quad \text{if } k\in P_i\\ &a_{ik}x_k + \sum_{j\in P_i} a_{ij}l_j + \sum_{j\in N_i - \{k\}} a_{ij}u_j\\ &\quad \le \sum_j a_{ij}x_j \le b_i, \quad \text{if } k\in N_i\end{aligned} \tag{2.2}$$

(2.2)는 다음과 같은 암시적 한계를 유도한다.

$$\begin{aligned}&\text{if } k\in P_i, \quad x_k \le u_k'\\ &u_k' = (b_i - \sum_{j\in P_i - \{k\}} a_{ij}l_j - \sum_{j\in N_i} a_{ij}u_j)/a_{ik}\\ &\text{if } k\in N_i, \quad x_k \ge l_k'\\ &l_k' = (b_i - \sum_{j\in P_i} a_{ij}l_j - \sum_{j\in N_i - \{k\}} a_{ij}u_j)/a_{ik}\end{aligned}$$

그런데 상한과 하한의 간격이 너무 좁은 경우에는 해법을 진행할 때 수치적으로 불안정해질 수 있기 때문에, 실제로 구현할 때는 변경되는 상한이나 하한이 이 간격을 너무 좁게 만들 경우에는 이 방법을 적용하지 않게 하였다.

2.2 암시적 자유변수 제거 방법

x_j가 포함된 모든 제약식에 대해서 앞에서 설명한 방법으로 구한 암시적 한계가 x_j의 한계 안에 있게 되면, x_j의 한계를 제거하더라도 같은 최적해를 가지므로 x_j는 암시적으로 자유변수가 된다. 단체법의 경우는 자유변수가 많을수록 바람직하지만 내부점 방법에서는 그렇지 않다. 그러므로 자유변수를 일부로 만들 필요는 없다. 그러나 암시적 자유변수가 단일요소열(singleton column)이면 다른 제약식에 영향을 주지 않기 때문에 이 제약식을 제거할 수 있다. 이 때 제거되는 제약식은 x_j의 값을 정하는데 사용되므로 보관해야 한다.

단일요소 암시적 자유변수 x_j에 대해서는 유일한 쌍대변수 값을 정할 수 있다. 즉, 제약식 i가 자유행이면 $y_i = c_j/a_{ij}$와 같이 쌍대변수 y_i의 값을 정할 수 있다. 이 때 x_j가 목적함수에 주는 영향을 반영하기 위해 목적함수 계수를 다음과 같이 수정한다.

$$\boldsymbol{c}' = \boldsymbol{c} - a_{i*}^T y_i$$

2.3 선회연산 또는 기본행연산을 이용한 방법

2.3.1 치환에 의한 방법

변수를 두 개만 가지는 등식제약식은 치환에 의해 제거할 수 있다. 다음과 같은 제약식이 있다고 하자.

$$a_{ij_1}x_{j_1} + a_{ij_2}x_{j_2} = b_i$$

이 때 x_{j_1}은 다음과 같이 x_{j_2}의 식으로 표현될 수 있다.

$$x_{j_1} = (b_i - a_{ij_2}x_{j_2})/a_{ij_1}$$

이 식을 x_{j_1}이 있는 모든 제약식과 목적함수에

대입하게 되면 이 행을 제거할 수 있다. 이 때 x_{j_2}의 한계도 변하게 된다. x_{j_1}을 x_{j_2}의 식으로 치환한다는 뜻은 a_{ij_1}을 선회요소로 하여 선회연산을 하는 것을 의미한다. 이 때 선회연산에 의해 비영요소가 추가될 수 있는데, 너무 많은 비영요소가 추가되면 j_2열이 밀집열이 되어 촐레스키 요소를 밀집하게 만들므로 해법의 효율성을 떨어뜨릴 수 있다. 선회연산 후 j_2열의 비영요소 수를 정확하게 알기 위해서는 많은 비교연산을 수행해야 하기 때문에 실제로 구현할 때는 휴리스틱 방법을 사용하였다. 즉 j_2열의 비영요소의 최대값은 $n(j_1)+n(j_2)-1$이 되므로, 이 값이 일정한 값 이상이 되면 치환을 하지 않도록 하였다. 여기서 $n(j)$는 j열의 비영요소의 수를 의미한다.

x_{j_1}의 값은 해법이 종료된 다음에 $x^*_{j_2}$의 값을 이용하여 해를 복원한다.

한편, 등식제약식에서 비영요소가 세 개 이상일 때도 치환을 통해 제약식을 제거할 수 있는 경우가 있다. 다음과 같은 제약식이 있다 하자.

$$a_{i1}x_1+\ldots+a_{in}x_n=b_i$$

이 때 x_n의 한계가 다음 식이 가질 수 있는 최대값과 최소값 사이에 있게 되면 x_n을 제거할 수 있다.

$$(b_i-a_{i1}x_1-\ldots-a_{i,n-1}x_{n-1})/a_{in}$$

이러한 과정을 두 개의 비영요소가 남을 때까지 반복할 수 있으면 앞에서 설명한 방법대로 치환을 통해 제약식을 제거할 수 있다.

2.3.2 내포 제약식을 이용한 비영요소의 제거

a_{i*}를 i번째 제약식의 비영요소들이라고 할 때, 각 등식제약식 a_{i*}에 대해서 이 행의 비영요소 패턴을 포함하는 제약식을 찾아서 기본행연산을 통해 A의 희소도를 향상시키는 방법이다. 즉, a_{k*}가 그러한 제약식이면 어떤 실수 α에 대해

$$a_{k*}{}' = a_{k*}+\alpha a_{i*}$$

는 a_{k*}와 같은 비영요소 패턴을 가지게 된다. 이 때 a_{k*}의 한 비영요소를 소거할 수 있는 α를 선택함으로써 적어도 한 개의 비영요소를 제거할 수 있다.

같은 비영요소 패턴을 가지는 행들을 찾기 위해서는 많은 시간이 소요되므로 이를 효율화할 필요가 있다. 즉 제약식의 비영요소의 수에 따라 리스트를 만들어 비영요소의 수가 작은 순서로 같은 패턴을 가지는 행을 찾도록 하였다. 이 때 A의 비영요소의 개수를 τ라고 하면 리스트의 시작점을 구하는데 $O(\tau)$, 리스트에 열을 채우는데 $O(m)$이 소요된다. 그리고 비영요소의 수가 일정 수 이상이 되면 이 방법을 적용하지 않도록 하였다.

2.3.3 암시적 자유변수 제거

앞에서 단일요소열 암시적 자유변수가 있으면 이를 제거할 수 있다고 하였다. 그런데 단일요소열이 아닌 암시적 자유변수의 경우 선회연산을 통해 단일요소열로 만들 수 있다.

선회연산의 단점은 시간이 많이 소요되는 것과 비영요소의 증가에 있다. 따라서 모든 암시적 자유변수에 대해서 이 방법을 적용하지 않고 열의 비영요소 수가 적고 또 그러한 열에 대해서도 선회행의 비영요소 수가 일정한 수보다 적은 경우에만 선회연산을 하도록 하였다.

2.4 선형종속행 및 열의 제거

2.4.1 선형종속행

모든 선형종속행을 제거하는 것은 어렵기 때문에[3] 비교적 빨리 판별할 수 있는 이중행(duplicate row)을 제거하는 방법을 사용했다. 이중행이란 두 개의 행, a_{i*}, a_{k*}에 대해 다음을 만족하는 실수 α가 존재하는 것이다.

$$a_{k*} = \alpha a_{i*}$$

그런데 이러한 행들은 '내포 제약식을 이용한 비영요소의 제거' 방법에 의해서 모두 제거된다.

2.4.2 선형종속열

두 개의 열 a_{*j}, a_{*k}에 대해 다음을 만족하는 실수 α가 존재하는 이중열(duplicate column)을 제거한다.

$$a_{*k} = \alpha a_{*j}$$

이중열을 처리하는 방법은 α의 부호, c_j와 c_k의 관계, 그리고 x_j와 x_k의 유한한 한계의 존재 여부에 따라 다르다.

만약 $c_k = \alpha c_j$이면 x_j와 x_k를 다음과 같이 합친다.

$$x_k' = x_k + \alpha x_j$$

새로운 변수의 한계는 다음과 같이 α의 부호에 따라 다르다.

$$\begin{aligned} l_k' &= l_k + \alpha l_j, \quad u_k' = u_k + \alpha u_j, \quad \text{if } \alpha > 0 \\ l_k' &= l_k + \alpha u_j, \quad u_k' = u_k + \alpha l_j, \quad \text{if } \alpha < 0 \end{aligned}$$

만약 $c_k \neq \alpha c_j$이면 추가되는 가정에 의해 한 열이 다른 열에 우세하게[dominate] 된다. 여기에서는 $c_k \geq \alpha c_j$인 경우에 대해서만 설명하기로 한다.

쌍대제약식에서 다음을 유도할 수 있다.

$$\begin{aligned} -z_k^* + w_k^* &= c_k - a_{*k}^T \boldsymbol{y}^* \\ &> \alpha c_j - a_{*j}^T \boldsymbol{y}^* \\ &= \alpha(-z_j^* + w_j^*) \end{aligned}$$

이 때 $u_j = +\infty$(즉, $w_j^* = 0$), $\alpha > 0$이거나, $l_j = -\infty$(즉, $z_j^* = 0$), $\alpha < 0$ 이면 $-z_k^* + w_k^* > 0$이 성립한다. 따라서 x_k는 하한 값으로 고정될 수 있다.

2.5 밀집열 처리 방법

밀집열이 있으면 촐레스키 팩터의 비영요소가 많아지기 때문에 이를 방지하는 기법이다.

n_d개의 비영요소를 가지는 A의 밀집열 d는 다음과 같이 $d_1, d_2, \ldots, d_k$로 분할된다.

$$d = \sum_{i=1}^{k} d_i$$

이 때 다음 제약식을 추가함으로써 동일성을 얻는다.

$$x_d^i - x_d^{i+1} = 0, \quad i = 1, 2, \ldots, k-1$$

3. 사전처리 순서 및 자료구조

사전처리에서 제약식과 변수를 많이 제거하는 것도 중요하지만 이에 소요되는 시간이 전체 계산 시간에 비해서 너무 많은 것은 바람직하지 않다. 사전처리를 효율적으로 수행하기 위해서는 위에서 설명한 방법들을 효과적으로 배치하는 것과 알맞은 자료구조를 사용하는 것이 필요하다.

앞에서 설명한 사전처리 방법은 다음과 같다.

(a) 분석적 방법
(b) 암시적 자유변수 제거 방법
(c) 선회연산 또는 기본행연산을 이용한 방법
(d) 선형종속행과 열의 제거
(e) 밀집열 처리 방법

이 중에서 (a)와 (b)는 비교적 빨리 수행할 수 있는 것이고 (c)와 (d)는 상대적으로 많은 시간이 소요되는 방법이다. 따라서 본 논문에서는 다음과 같은 순서로 사전처리를 구현하였다.

(a) ➡ (b) ➡ (c) ➡ (d) ➡ (a) ➡ (b) ➡ (d) ➡ (e)

여기서 (a)와 (b)는 제거되는 행이나 열이 없을

때까지 반복적으로 수행하게 하였다.

선형계획문제를 처음 읽어서 보관하는 방식은 대부분 열위주압축형태(columnwise packed form)이다. 사전처리에서는 열뿐만 아니라 행으로 따라가는 연산이 많고 또 선회연산에 의해 비영요소가 추가될 수 있기 때문에 행-열 연결구조(row-column linked list)를 사용하는 것이 효율적인 것으로 판단된다. 또 사전처리 과정의 편의상 순서가 있는 연결구조를 사용하였다.

4. 해의 복원

원문제 (P)의 해 $\boldsymbol{x}$에 대한 복원은 사전처리 과정의 역순으로 행해지는데 자세한 과정은 [6]을 참조하기 바란다. 본 논문에서는 기존의 문헌에서 잘 언급되지 않은 쌍대해의 복원에 대해 설명하겠다.

사전처리 방법 중에서 변수의 한계를 변경시키는 방법들이 있는데 이러한 경우에는 쌍대해의 복원이 복잡해진다. 변수의 한계가 변하지 않았을 경우와 변했을 경우에 대해 각각 쌍대해의 복원에 대해 설명한다.

4.1 변수의 한계가 변하지 않은 경우

내부점 방법은 단순상한문제에 대해서 적용되기 때문에 일반한계문제 (P)를 단순상한문제로 변형하였을 때의 쌍대해에 대해서 알아볼 필요가 있다. 일반한계문제 (P)의 단순상한문제는 (P1)이다.

$$\text{(P1)}\quad \begin{aligned} &Max\ \ \boldsymbol{c}^T\boldsymbol{x}' + \boldsymbol{c}^T\boldsymbol{l} \\ &s.t.\ \ A\boldsymbol{x}' = \boldsymbol{b} - A\boldsymbol{l} \\ &\quad\ \ \boldsymbol{0} \le \boldsymbol{x}' \le \boldsymbol{u} - \boldsymbol{l} \end{aligned}$$

(P1)의 쌍대문제는 (D1)이 된다.

$$\text{(D1)}\quad \begin{aligned} &Min\ \ (\boldsymbol{b} - A\boldsymbol{l})^T\boldsymbol{y}' + (\boldsymbol{u} - \boldsymbol{l})^T\boldsymbol{w}' + \boldsymbol{c}^T\boldsymbol{l} \\ &s.t.\ \ A^T\boldsymbol{y}' + \boldsymbol{w}' - \boldsymbol{z}' = \boldsymbol{c} \\ &\quad\ \ \boldsymbol{w}' \ge \boldsymbol{0},\ \boldsymbol{z}' \ge \boldsymbol{0} \end{aligned}$$

(D1)의 최적해 $(\boldsymbol{y}', \boldsymbol{w}', \boldsymbol{z}')$는 (D)의 제약식을 만족하는 해이고, (D1)의 최적 목적함수 값은 아래에서 보는 바와 같이 (P)의 최적 목적함수 값과 같다. 따라서 (D1)의 최적해는 (D)의 최적해가 된다.

$$\begin{aligned} \boldsymbol{c}^T\boldsymbol{x} &= \boldsymbol{c}^T(\boldsymbol{x}' + \boldsymbol{l}) \\ &= (\boldsymbol{b} - A\boldsymbol{l})^T\boldsymbol{y}' + (\boldsymbol{u} - \boldsymbol{l})^T\boldsymbol{w}' + \boldsymbol{c}^T\boldsymbol{l} \\ &= \boldsymbol{b}^T\boldsymbol{y}' + \boldsymbol{u}^T\boldsymbol{w}' - \boldsymbol{l}^TA^T\boldsymbol{y} - \boldsymbol{l}^T\boldsymbol{w} + \boldsymbol{l}^T\boldsymbol{c} \\ &= \boldsymbol{b}^T\boldsymbol{y}' + \boldsymbol{u}^T\boldsymbol{w}' - \boldsymbol{l}^T(A^T\boldsymbol{y}' - \boldsymbol{w}' + \boldsymbol{c}) \\ &= \boldsymbol{b}^T\boldsymbol{y}' + \boldsymbol{u}^T\boldsymbol{w}' - \boldsymbol{l}^T\boldsymbol{z}' \end{aligned}$$

즉, (D1)의 해는 (P)의 쌍대해가 된다.

(P)에 사전처리를 적용하면 제약식들과 변수들이 제거되게 된다. 제약식 i가 제거되면 그에 해당하는 쌍대변수 y_i가 제거되고 변수 x_j가 제거되면 쌍대변수 w_j, z_j가 제거된다. 따라서 쌍대해를 복원한다는 것은 제거된 제약식에 대해서 쌍대변수 y_i를 복원하고, 제거된 변수에 대해서 쌍대변수 w_j, z_j의 값을 복원하는 것이다.

먼저 w_j, z_j의 복원에 대해서 알아보자.

변수가 제거된다는 것은 그 값이 고정되게 되는데 x_j가 어떤 값으로 고정되느냐에 따라 w_j, z_j의 값을 다음과 같이 정할 수 있다.

- if $x_j = l_j$, $z_j = 0$, $w_j = c_j - (A^T\boldsymbol{y})_j$.
- if $x_j = u_j$, $w_j = 0$, $z_j = (A^T\boldsymbol{y})_j - c_j$.
- if $l_j < x_j < u_j$, $w_j = 0$, $z_j = 0$.

이 때 $\boldsymbol{y}$의 값은 완전히 복원되어 있어야 한다. $\boldsymbol{y}$의 복원은 다음과 같다.

- 중복제약식인 경우 : $y_i = 0$
  (공백행도 중복제약식의 일종이다.)
- 단일요소행에서 이 요소가 속한 열이 제거되지 않는 경우 : 다음을 만족하는 y_i
  $(A^T\boldsymbol{y})_j + w_j - z_j - c_j = 0$
- 행이 제거될 때 각 변수의 값을 고정시키는 경우 : 다음을 만족하는 y_i

$$(A^T\boldsymbol{y})_j - c_j \begin{cases} \geq 0, & \text{if } x_j = l_j \\ \leq 0, & \text{if } x_j = u_j \\ = 0, & \text{if } l_j < x_j < u_j \end{cases} \quad (4.1)$$

$$y_j \begin{cases} \geq 0, & \text{if } (A\boldsymbol{x})_i = b_i, \text{ 제약식: '}\leq\text{'} \\ \leq 0, & \text{if } (A\boldsymbol{x})_i = b_i, \text{ 제약식: '}\geq\text{'} \\ = 0, & \text{if } (A\boldsymbol{x})_i - b_i \neq 0 \end{cases} \quad (4.2)$$

사전처리 과정 중에는 선회연산이 필요한 경우가 있다. 이러한 경우에는 위와 같이 원래 문제 행렬 A에 대해서가 아니라 그 당시의 A 행렬에 대해서 (4.1)과 (4.2)를 만족하는 y_i를 구해야 한다. 이렇게 하기 위해서는 사전처리 후의 A 행렬을 보관해야 할 필요가 있다. 그리고 이 보관된 행렬에 대해 선회연산을 역으로 수행하면서 $\boldsymbol{y}$의 값을 구해야 한다.

한편, 선회연산을 하게 되면 선회행에 해당하는 쌍대변수 값이 변하게 되므로 이 값도 복원시켜야 한다. 즉 i행을 선회행으로 하여 v를 곱하여 k행에 더하는 선회연산을 하였다고 하자. 그러면 다음 관계에 의해서 y_i의 값을 구한다.

$$\begin{aligned} b_i y_i + b_k y_k &= b_i y_i' + (b_k + v b_i) y_k' \\ &= b_i(y_i' + v y_k') + b_k y_k' \end{aligned}$$

$$\therefore y_i = y_i' + v y_k', \quad y_k = y_k'$$

단, y_i, y_k는 선회연산 전의 해이고,

y_i', y_k'는 선회연산 후의 해이다.

4.2 변수의 한계가 변하였을 경우

변수의 상 · 하한이 변경되었을 때는 보조정리 4.1과 같이 쌍대해가 변경될 수 있다.

보조정리 4.1 변수의 한계가 변경되었을 경우에 쌍대해는 변경될 수 있다.

(증명) l_j와 u_j를 변수 x_j의 하한과 상한이라고 하고, l_j'와 u_j'를 변경된 하한과 상한이라고 하자. 일반성을 잃지 않고도 (P)가 비퇴화이고 사전처리에서 변수 x_j만 한계가 변경되었다고 가정할 수 있다. B를 (P)의 기저, 또 B′을 사전처리된 문제인 (P′)의 기저라고 하자. ξ를 (P′)에서 x_j의 최적값이고, l_j'와 u_j' 중 하나가 ξ와 같다고 하자. 그러면 (P′)은 퇴화가 되므로 B′은 B와 같지 않을 수 있다. 따라서 (P)의 쌍대해는 (P′)의 쌍대해와 같지 않을 수 있다. ■

쌍대변수 중 한 두 개만 변경되는 경우에는 이들의 정확한 값 또는 범위를 (4.1)과 (4.2)에 의해서 쉽게 구할 수 있지만, 최악의 경우 정리 4.2와 같이 m개의 변수와 n개의 등식 또는 부등식으로 이루어진 선형부등식을 풀어야 하는 것과 같다.

정리 4.2 변수의 한계가 변경되었을 때, (4.1)과 (4.2)를 이용하여 쌍대해를 복원하는 것은 최악의 경우 m개의 변수와 n개의 등식 또는 부등식으로 이루어진 선형부등식을 푸는 것과 같다.

(증명) (P)에 m개의 제약식이 있고 사전처리에서 제거되는 제약식이 없다고 하다. 또 $k(\leq n)$개의 변수의 한계가 변경되어서 m개의 쌍대변수의 값이 변경되었다고 하자. 이 때 원최적해를 알고 있기 때문에 (4.1)과 (4.2)에 의해서 n개의 식을 만들 수 있다. 그렇지만 쌍대변수의 값을 하나도 모르기 때문에 m개의 변수와 n개의 등식 또는 부등식으로 만들어진 선형부등식을 풀어야 모든 값을 알 수 있다. ■

정리 4.2에서 보는 바와 같이 변수의 한계가 변경된 경우에는 (4.1)과 (4.2)를 이용하여 쌍대해를 복원하는 것이 매우 어렵다. 따라서 다른 방법을 사용하여야 하는데 이에는 최적기저를 복원하여 거기에 해당하는 쌍대해를 구하는 방법이 있을 수 있다.

5. 실험결과

Netlib 문제 중 제약식의 개수가 1000개 이상인 대형문제 중 scsd8, truss와 같이 사전처리에서 제거되는 행과 열이 없는 문제를 제외한 28개의 문제에 대해서, 사전처리를 통해 제거되는 제약식과 행

의 개수 및 사전처리 후 A의 비영요소 개수 등을 다른 프로그램들과 비교하였다. 비교 대상은 HOPDM v2.13, BPMPD v2.1, 그리고 CPLEX v3.0 등이다. 또 사전처리에 소요되는 시간도 비교하였다. 본 논문의 방법은 LPABO v5.5에 구현되었다.

각 프로그램은 64MB의 메인메모리를 가지는 SUN SPARC ULTRA 170 워크스테이션에서 실험하였고 컴파일 옵션은 다음과 같다.

- LPABO : gcc -O3
- HOPDM : f77 -fast -O3 -libmil -native
- BPMPD : f77 -fast -O3 -libmil -native -N150
- CPLEX : 알려지지 않음.

실험문제의 정보는 <표 5.1>과 같다.

<표 5.1>의 문제를 실험한 결과는 <표 5.2>, <표 5.3>과 같다.

<표 5.2>를 보면, 제거되는 제약식과 변수의 개수에 있어서 본 논문에서 제시한 방법은, HOPDM과 BPMPD에 비해 대부분의 문제에 대해서 우수하다는 것을 알 수 있다. LPABO의 사전처리가 HOPDM 및 BPMPD의 사전처리에 비해서 뒤지는 문제인 80bau3b, d6cube, stocfor3을 보게 되면, 제거되는 행의 수는 뒤지지 않으나 제거되는 열의 개수가 뒤짐을 알 수 있다. 이는 LPABO의 사전처리에 열을 제거하는 방법에 대한 고려가 부족하다는 것을 보여 준다. 따라서 이에 대해서는 추가적인 연구가 필요하다.

한편, CPLEX와 비교해 보면, 일부 문제들(fit1p, fit2p, nesm, pilotja, pilotnov)에서는 결과가 우수하지만 전반적으로 뒤지는 것을 알 수 있다.

<표 5.2>는 사전처리 후 행렬 A의 비영요소의 수도 나타내고 있는데 제거되는 행과 열의 수의 차이에 비해서는 비영요소의 수에는 큰 차이가 없는 것을 알 수 있다. 이는 각 프로그램의 사전처리 과정에서 추가로 제거되는 행과 열에 비영요소의 수가 많지 않다는 것을 의미한다.

<표 5.3>은 사전처리를 수행하는 데 소요되는 시간을 나타내고 있다. 여기에 나타난 수치는, 비록 HOPDM과 BPMPD를 제외하고는 컴파일러가 서로 다르기 때문에 직접적인 비교를 하기에는 무리가 있지만, 각각의 프로그램을 작성한 컴퓨터 언어에 대해 테스트 기계에 있는 해당 컴파일러로 최대로 빠르게 문제를 풀 수 있는 옵션으로 컴파일한 후 실험한 결과이다.

LPABO의 사전처리 수행 시간을 HOPDM, BPDPM의 그것과 비교하면, LPABO가 행과 열을 더 제거함에도 불구하고 수행 시간이 더 적음을 알 수 있다. 이를 통해 본 논문의 방법이 효율적으로

〈표 5.1〉 실험문제의 정보

문제이름	제약식	변 수	비영요소	문제이름	제약식	변 수	비영요소
25fv47	821	1571	10400	greenbeb	2392	5405	30877
80bau3b	2262	9799	21002	maros	846	1443	9614
bnl2	2324	3489	13999	nesm	662	2923	13288
cycle	1903	2857	20720	pilot	1441	3652	43167
czprob	929	3523	10669	pilot87	2030	4883	73152
d2q06c	2171	5167	32417	pilotja	940	1988	14698
d6cube	415	6184	37704	pilotnov	975	2172	13057
degen3	1503	1818	24646	sctap3	1480	2480	8874
dfl001	6071	12230	35632	ship08l	778	4283	12802
fit1d	24	1026	13404	ship12l	1151	5427	16170
fit1p	627	1677	9868	ship12s	1151	2763	8178
fit2d	25	10500	129018	stocfor3	16675	15695	64875
fit2p	3000	13525	50284	woodlp	244	2594	70215
greenbea	2392	5405	30877	woodw	1098	8405	37474

〈표 5.2〉 사전처리 결과 비교

문제이름	LPABO			HOPDM			BPMPD			CPLEX		
	(1)	(2)	(3)	(1)	(2)	(3)	(1)	(2)	(3)	(1)	(2)	(3)
25fv47	136	122	9986	52	36	9959	48	33	10123	137	122	9903
80bau3b	277	765	19636	297	1063	19048	298	870	19472	297	1119	18981
bnl2	1370	1341	10288	476	482	12458	765	762	10963	1381	1394	10252
cycle	827	928	13416	503	454	14111	493	445	14347	974	1066	12993
czprob	454	968	5172	268	835	5376	268	835	5376	465	1032	4982
d2q06c	291	485	30831	159	203	30263	109	88	30964	296	550	30600
d6cube	13	2	37693	12	741	32523	12	1	36576	13	724	34298
degen3	93	93	24460	0	0	24363	0	0	24568	96	96	24427
dfl001	2066	2790	32138	164	165	35021	87	87	35284		*	
fit1d	0	0	13404	0	2	13358	0	0	13404	0	2	13386
fit1p	0	627	9241	0	22	9846	0	22	9846	0	250	9618
fit2d	0	0	129018	0	137	127760	0	0	129018	0	50	128564
fit2p	0	3000	47284	0	0	50284	0	0	50284	0	0	50284
greenbea	1183	2041	23624	520	1494	23161	461	1309	23855	1372	2347	23028
greenbeb	1185	2051	23477	527	1510	23052	468	1332	23737	1373	2356	22927
maros	291	541	6279	220	491	5743	221	483	6119	307	600	5788
nesm	56	273	12941	16	247	12926	16	183	13054	40	216	12933
pilot	142	370	40677	91	323	40506	79	320	40598	166	409	40467
pilot87	116	347	70412	62	288	70361	63	290	70377	140	372	70370
pilotja	210	585	10979	145	442	10931	131	427	11080	195	568	10985
pilotnov	202	449	11523	145	311	11466	128	296	11593	190	435	11528
sctap3	134	713	7640	134	124	8229	134	124	8229	136	713	7630
ship08l	308	1184	7100	308	1184	7100	308	1184	7100	308	1184	7100
ship12l	541	1280	9230	541	1280	9230	541	1280	9230	541	1280	9222
ship12s	883	916	4129	811	844	4273	811	844	4273	884	916	4121
stocfor3	6096	4957	45212	1339	9159	55088	1339	9159	55088	5935	13755	52492
woodlp	74	794	46828	74	874	43657	74	793	45747	74	866	44884
woodw	395	3211	19574	395	3058	19727	391	3054	19735	543	4395	14536

(1) 제거된 행의 수 (2) 제거된 열의 수 (3) 사전처리 후 A의 비영요소 수
* 메모리 문제로 읽지 못함

〈표 5.3〉 사전처리 시간 비교

(단위 : 초)

문제이름	LPABO	HOPDM	BPMPD	CPLEX	문제이름	LPABO	HOPDM	BPMPD	CPLEX
25fv47	0.08	0.34	0.10	0.10	greenbeb	0.47	0.92	0.41	0.41
80bau3b	0.22	0.83	0.55	0.45	maros	0.07	0.25	0.09	0.09
bnl2	0.23	0.62	0.26	0.23	nesm	0.07	0.32	0.10	0.12
cycle	0.24	0.60	0.20	0.21	pilot	0.40	1.79	0.34	0.35
czprob	0.07	0.16	0.06	0.10	pilot87	0.47	2.32	0.53	0.67
d2q06c	0.36	1.30	0.52	0.33	pilotja	0.13	0.53	0.14	0.12
d6cube	0.27	2.12	1.16	0.31	pilotnov	0.12	0.61	0.15	0.13
degen3	0.12	0.25	0.30	0.15	sctap3	0.08	0.24	0.14	0.16
dfl001	1.16	2.37	0.85	*	ship08l	0.07	0.20	0.07	0.12
fit1d	0.04	0.39	0.05	0.09	ship12l	0.10	0.26	0.10	0.16
fit1p	0.03	0.34	0.06	0.10	ship12s	0.07	0.13	0.05	0.08
fit2d	0.55	10.69	2.61	0.88	stocfor3	0.86	1.35	0.61	1.19
fit2p	0.29	8.07	0.36	0.46	woodlp	0.41	2.65	0.64	0.36
greenbea	0.47	0.93	0.41	0.42	woodw	0.34	0.87	0.73	0.25

* 메모리 문제로 읽지 못함

구현한 것이라고 볼 수 있다.

한편, CPLEX의 수행 시간과 비교해 보면, LPABO의 수행 시간이 비슷하거나 더 빠른 것을 알 수 있다. CLPEX는 매우 효율적인 코드라고 알려져 있기 때문에, LPABO가 CPLEX에 비해 제거하는 행과 열의 개수가 더 적기 때문에 사전처리에 소요되는 시간이 더 적게 걸리는 것이라고 판단할 수도 있지만, 사전처리에서 제거되는 행과 열의 차이가 크지 않은 점을 감안한다면 본 논문의 방법이 효율적이라는 것을 간접적으로 알려주고 있다.

6. 결론 및 추후 연구과제

본 논문에서 사용한 사천처리 방법들은 아래와 같이 분류된다.

- 분석적 방법
- 암시적 자유변수 제거 방법
- 선회연산 또는 기본행연산을 이용한 방법
- 선형종속행과 열의 제거
- 밀집열 처리 방법

이러한 방법을 구현하여 HOPDM, BPMPD 보다 우수한 사전처리를 구현하였다. 그러나 CPLEX에 비해서는 그렇지 않았다.

다른 프로그램들의 사전처리와 비교해 보면, 본 논문에서 제시한 방법에서는 특히 열에 대해 처리하는 부분이 부족하다는 것을 알 수 있다.

현재까지 CPLEX에서의 사전처리가 가장 우수하다고 알려져 있지만 그 방법들은 공개가 되지 않고 있다. 본 논문에서의 방법은 그 동안 발표되어 온 사전처리 방법들 및 본 논문에서 부분적으로 추가된 방법들을 구현하였는데 아직은 CPLEX의 사전처리를 따라가지 못하고 있다. 따라서 추가적인 방법에 대한 연구가 필요하다.

한편, 해법을 수행한 후 해를 복원할 때 원변수에 대한 해의 복원은 쉽게 되나 쌍대해에 대한 해의 복원은 그렇지 않다. 특히 사전처리 과정 중에 변수의 한계가 변경되게 되면 더욱 그러하다.

참 고 문 헌

[1] 박순달, 「선형계획법」, 3판, 민영사, 1993.

[2] 성명기, 박순달, "대형선형계획법문제의 사전처리", 「한국경영과학회 '96추계학술대회논문집」 (1996), pp.285-288.

[3] Andersen, E.D., "Finding all Linearly Dependent Rows in Large-Scale Linear Programming," *Optimization Methods and Software*, Vol.6(1995), pp.219-227.

[4] Andersen, E.D., J. Gondzio, C. Mészáros, X. Xu, "Implementation of Interior Point Methods for Large Scale Linear Programming," in : *Interior Point Methods in Mathematical Programming*, T. Terlaky (ed.), Chapter 6, pp.189-252, Kluwer Academic Publisher, 1996.

[5] Andersen, E.D., K.D. Andersen, "Presolving in Linear Programming," *Mathematical Programming*, Vol.71(1995), pp.221-245.

[6] Gondzio, J., "Presolve Analysis of Linear Programs Prior to Applying an Interior Point Method," *INFORMS Journal on Computing*, Vol.9, No.1(1997), pp.73-91.

[7] Lustig, I.J., R.E. Marsten, D.F. Shanno, "Interior Point Methods for Linear Programming : Computational State of the Art," *ORSA Journal on Computing*, Vol.6, No.1(1994), pp.1-14.

[8] Mészáros, C., "On Free Variables in Interior Point Methods," *Optimization Methods and Software*, Vol.9(1998), pp.121-139.

[9] Wright, S., "Modified Cholesky Factorizations in Interior-Point Algorithm for Linear Programming," Technical Report, Mathematics and Computer Science Division, Argonne National Lab., Argonne, IL, USA (1998).