Journal of the Korean Institute of Industrial Engineers
Vol. 25, No. 2, pp. 217-225, June 1999.

단체법을 위한 사전처리의 구현†

임성묵 · 성명기 · 박순달

서울대학교 산업공학과

An Implementation of Preprocessing for the Simplex Method

Sungmook Lim · Myeongki Seong · Soondal Park

Preprocessing is the essential technique in the implementation of the simplex method for large scale linear programming problems. In this research, we explained the effect of preprocessing in the simplex method, classified the techniques into four categories, and compared our results with those of HOPDM, CPLEX and Soplex by computational experiments. We also noted some implementing issues of preprocessing for the simplex method and the recovery of the optimal basis of the original problem from that of the preprocessed problem.

1. 서 론

선형계획법은 많은 응용분야를 가지고 있어 그 활용도가 대단히 높다(박순달, 1992). 따라서 선형계획법을 위한 많은 해법들이 개발되고 효율적인 구현방법들이 많은 연구자들에 의해 시도되었다. 선형계획법의 해법으로 Dantzig에 의한 단체법과 Karmarkar의 사영법 이후의 내부점방법이 대표적인데(Gondzio, 1997), 본 연구에서는 단체법에 대한 사전처리(Preprocessing)에 대해 다루고자 한다.

사전처리의 역할은 첫째, 주어진 문제를 검사하여 문제 속에 내재되어 있는 비가능성(Infeasibility)이나, 무한해(Unboundedness)의 존재 여부 등을 미리 파악하는 것이다. 단체법에서는 문제의 비가능성을 알아내기 위해서 또다른 선형계획법 문제를 푸는 과정인 국면 1을 수행하여야 한다. 따라서, 문제를 풀기 전에 미리 그 비가능성을 파악할 수 있다면, 그 효과는 크다 할 수 있다(성명기, 박순달, 1996). 또한, 사전처리를 통한 비가능성의 판별은 문제의 어떤 부분에서 비가능이 발생했는지 알 수 있으므로 사용자의 입장에서도 이점을 가지고 있다. 한편, 사전처리 없이 단체법으로 주어진 문제가 무한해를 가짐을 알아내는 시점은 국면 2에서이다. 따라서, 이에 대한 효과도 크다. 사전처리의 역할로 두 번째는, 문제 축소기능이다. 즉, 문제 속에 들어 있는 의미없는 제약식, 변수 등을 제거하여 사전처리 전의 원래 문제와 동일한 의미를 지니지만 문제의 크기가 줄어든 문제를 생성하는 것이다. 이러한 기능은 대형의 선형계획법 문제를 고속으로 해결하는 것이 요구되는 상황에서 필수적인 기능이라 할 수 있다. 물론, 축소된 문제를 풀 때 단체법의 수행속도가 빨라진다는 보장은 하지 못하지만 일반적으로 성능향상을 기대할 수 있는 것이다. 사전처리는 많은 상용코드, 공개코드들에서 구현되고 있고, 그 효과를 확인할 수 있는데(Mészáros, 1998; Andersen, 1995; Andersen, 1996; Andersen and Andersen, 1995; Gondzio, 1997; Lustig, 1994), 본 연구에서는 단체법 프로그램에 적용될 수 있는 사전처리와 이의 효율적 구현에 관해 연구하고, 그 결과를 다른 단체법 프로그램들과 비교했다.

2. 사전처리의 영향

주어진 선형계획법 문제를 단체법으로 풀기 전에 사전처리를 수행하게 되면, 단체법의 수행에 많은 영향을 끼치게 된다. 우선, 사전처리를 통해 비가능성이나 무한대 해를 판별하는 경우를 제외하고 문제가 축소되는 경우만 살펴보기로 한다.

희소행렬을 다루는 선형계획법 문제에서 일반적으로 문제의 크기라는 것은 제약식의 수, 변수의 수, 문제 속의 비영요소의 수를 의미한다. 사전처리를 통해 문제가 축소된다는 것은

† 본 연구는 정보통신부의 97년도 대학기초연구지원사업 (97-G-0683)의 지원을 받았음.

그 제약식과 변수의 수가 축소된다는 것을 의미하기로 한다. 즉, 사전처리를 통해 문제의 제약식과 변수의 수가 줄어든다고 하더라도 문제의 비영요소수가 줄어든다는 것을 보장하지 못하고, 오히려 증가될 수도 있다는 것을 뜻한다. 우선, 사전처리를 통해 제약식의 수가 줄어들게 되면 단체법에서 아주 중요한 영향을 미치는 기저행렬의 크기가 줄어들어 기저역행렬 관련 연산의 속도향상을 기대할 수 있다. 물론, 기저역행렬 관련 연산의 성능을 결정짓는 것으로 기저행렬의 크기 이외에 기저행렬의 희소도도 큰 작용을 하게 되는데, 이것은 입력자료의 비영요소의 수에 영향을 받게 된다(박순달, 김우제, 박찬규, 임성묵, 1998; 박찬규, 임성묵, 김우제, 박순달, 1997). 사전처리에서 제약식을 제거하는 기법들 중에는 한 제약식이 제거되면서 다른 제약식에 아무런 영향을 미치지 못하는 기법도 있지만 다른 제약식에 영향을 준 다음 제거되는 기법들도 있다. 즉, 다른 제약식의 비영요소수에 영향을 주면서 제약식을 제거시키는 기법의 경우에는 기저행렬의 크기를 줄이는 반면 기저행렬의 비영요소수를 증가시킬 우려도 있게 된다. 사전처리를 통해 변수의 수를 줄이는 경우에는 단체법에서의 진입변수 선정과 퇴화가능성에 영향을 주게 된다. 단체법의 계산과정 중 가장 중요한 비중을 차지하는 것은 앞에서 설명한 바와 같이 기저역행렬 관련 연산들이다. 이를 제외하고 단체법의 계산량을 좌우하게 되는 것은 진입변수 선정이라고 할 수 있다. 즉, 진입가능한 비기저변수를 선정하기 위해 주어진 문제의 비기저 변수들을 조사하여야 하는데 그 과정에 소요되는 비용은 변수의 수에 좌우된다. 따라서, 변수의 수가 작을수록 그 비용이 감소하는 것은 자명하다. 따라서, 사전처리를 통해 변수의 수를 줄일 때 그 효과는 진입변수 선정에서 나타나게 된다. 그러나, 제약식을 줄이는 경우와 마찬가지로 변수를 제거할 때에도 다른 변수의 비영요소수에 영향을 미치는 경우가 있는데, 이러한 경우에는 기저행렬의 비영요소수를 증가시켜 기저역행렬 관련 연산을 느리게 할 수도 있다. 마지막으로 사전처리를 통해 변수의 상하한을 줄일 수도 있는데, 이에 대한 효과는 단체법의 경우 부정적이라고 할 수 있다. 그 이유는 변수의 상하한 폭이 줄어들게 되면 그 변수의 개선폭이 줄어들게 되어 퇴화 가능성이 높아지기 때문이다. 따라서 이러한 기법은 단체법 프로그램에서의 사전처리에서는 부적절하다고 할 수 있다.

3. 사전처리 방법

일반적인 선형계획문제는 (P)와 같은 일반한계문제이다.

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

여기서 A는 $m \times n$ 행렬이고, $\mathbf{x}$, $\mathbf{l}$, $\mathbf{u}$, $\mathbf{c}$는 n차 벡터이고, $\mathbf{d}$, $\mathbf{b}$는 m차 벡터이다. 그리고, $\mathbf{l}$, $\mathbf{u}$, $\mathbf{d}$, $\mathbf{b}$ 는 무한대의 값을 허용한다.

본 논문에서 구현한 사전처리는 크게 문제의 비가능성, 무한해의 발견기법과 문제 축소기법으로 나눌 수 있고, 문제 축소기법들은 다음과 같이 분류될 수 있다.

- 기본적 방법
- 선회연산을 통한 제거방법
- 내재적 중복성의 제거방법
- 변수의 상하한 축소방법

그림 1. 문제 축소기법의 분류.

우선, 문제의 비가능성과 무한해의 발견기법을 설명하면 다음과 같다.

3.1 비가능 판정 및 무한대 해의 발견

문제에 내포된 비가능성의 판별과 무한대 해의 발견은 사전에 해법적용을 방지하여 문제해결 능력을 증가시킨다.

우선, 제약식의 비영요소가 하나도 없는 공백행(Empty row)의 경우를 살펴보자. 이러한 경우는 사용자의 입력착오로 인해 발생할 수 있는 경우이다. 공백행의 경우 공백행 제약식의 좌변이 양의 값을 가지거나 우변이 음의 값을 가지면 이 제약식은 만족되지 못하여 비가능성을 유발시키게 된다. 다음으로,

비영요소가 하나뿐인 단일요소행(Singleton row)의 경우를 생각해 보자. 이러한 경우에는 단일요소행 제약식의 변수의 한계로 처리할 수 있는데, 이러한 처리로 생성되는 변수의 한계가 원 문제의 변수한계를 어기게 되는 경우에 비가능성을 발생하게 된다. 예를 들어 $d_i \le a_j x_j \le b_i$ 라는 단일요소행이 있을 때, 이 제약식은 a_j가 양일 경우 $d_i/a_j \le x_j \le b_i/a_j$라는 변수의 한계로 처리할 수 있다. 그런데 원 문제에서 x_j의 한계가 $l_j \le x_j \le u_j$로 주어졌을 때, $d_i/a_j > u_j$이거나 $b_i/a_j < l_j$ 일 경우에 주어진 제약식은 비가능성을 유발시키 는 것이다.

비영요소를 하나도 가지지 않는 공백열(Empty column)의 경우에 목적함수 계수의 부호와 그 변수의 상하한을 이용해 무한대 해의 존재를 판별할 수 있다. 예를 들어 목적함수의 계수가 양인 공백열 변수의 상한이 무한대일 경우에 그 변수는 제약식들에 아무런 영향을 미치지 않고 무한대의 값을 가질 수 있고, 이러한 경우에 목적함수의 값은 무한대가 되므로 주어진 문제는 무한대의 해를 가지게 되는 것이다.

변수의 한계가 주어졌을 경우 한 제약식이 취할 수 있는 값은 그 범위가 정해지게 되는데 원 문제에 주어진 제약식의 영역이 그 범위를 벗어나게 되면 그 제약식은 만족되어질 수 없

어 비가능성을 유발시킨다. 앞에서 설명한 단일요소행에 대한 비가능성 판정의 경우도 이 경우의 특수한 형태라 할 수 있다.

한 변수가 영향을 미치는 제약식은 그 변수열이 비영요소를 가지는 행에 해당되는 제약식들이다. 이때, 한 변수가 영향을 미치는 제약식의 가능성에 아무런 제약을 가하지 않고 목적함수에 기여할 수 있다면 이 변수는 그 한계를 만족시키는 한도 내에서 최대한 목적함수를 개선시킬 수 있는데, 만약, 목적함수를 개선시키는 방향으로 변수의 한계가 무한대라면 이 문제는 무한대의 목적함수값을 가질 수 있다. 예를 들어 다음의 변수 x_j가 존재할 경우에 주어진 문제는 무한대 해를 가지게 된다.

i) $c_j > 0$, $u_j = \infty$
ii) 모든 i에 대해,
$a_{ij} > 0$ 이면 $b_i = \infty$
$a_{ij} < 0$ 이면 $d_i = -\infty$

그림 2. 무한대 판정의 예.

즉, 윗그림과 같은 경우에는 ii)에 의해 변수 x_j가 무한히 커져도 제약식의 가능성을 해치지 않고, i)에 의해 목적함수는 무한대의 값을 가지게 된다.

이상의 방법을 정리하면 비가능 판정과 무한대 해의 판정방법은 다음과 같다.

- 공백행에서의 비가능 판정
- 단일요소행에서의 비가능 판정
- 제약식의 범위를 이용한 비가능 판정
- 공백열에서의 무한대 판정
- 변수의 변화가능범위를 이용한 무한대 판정

그림 3. 비가능성과 무한대 해의 판정.

3.2 기본적 방법

사전처리에서 기본적 방법은 문제에 대한 별다른 분석없이 표면적인 면에서 드러나는 문제 축소가능성을 포착하는 방법이다. 이러한 기법들로는 공백행·공백열의 제거, 자유행의 제거, 단일요소행의 처리, 고정변수의 제거, 그리고 목적함수 계수가 영인 단일요소열의 처리 등이 있다.

(1) 공백행 및 공백열 제거

공백행의 경우에는 앞에서 설명한 비가능 판정방법에서 비가능으로 판정되지 않는 경우에 제거할 수 있다. 즉, 항상 만족되는 제약식으로 판정할 수 있다. 또한, 공백열의 경우 무한대 해의 존재로 판정되지 않는다면 일정값으로 고정시킬 수 있다.즉, x_j의 열의 공백열일 경우 다음과 같이 x_j의 값을 고정시킬 수 있다.

$c_j > 0$ 이면 $x_j := u_j (\neq \infty)$
$c_j < 0$ 이면 $x_j := l_j (\neq -\infty)$

그림 4. 공백열의 고정.

(2) 단일요소행과 자유행의 제거

단일요소행인 제약식의 경우 단일요소에 해당하는 변수에 대한 한계로 처리할 수 있다.

한편 제약식의 상한과 하한이 모두 무한대인 경우 그 제약식은 어떠한 값을 취해도 무방하다. 이러한 경우 이 제약식은 언제나 만족되는 제약식으로 제거되어도 해공간을 축소시키지 않는다.

(3) 단일요소열과 고정변수의 처리

목적함수의 계수가 영인 변수열이 단일요소열일 경우에 이 변수는 비영요소가 존재하는 행에 대해 상하한을 지니는 여유변수의 역할을 하고 있다고 할 수 있다. 일반적으로 문제 (P)에서와 같은 영역제약식을 기저크기를 m으로 유지한 채 처리하기 위해서 그 영역제약식에 한계를 가지는 여유변수를 추가하게 된다. 이러한 과정을 역으로 취하는 것이 단일요소열 처리 방법이다. 그 과정은 x_j에 대한 열이 단일요소열이고, $a_{ij} \neq 0$이라고 할 때 다음과 같다.

$a_{ij} > 0$ 이면 $b_i := b_i - l_j \cdot a_{ij}$, $d_i := d_i - u_j \cdot a_{ij}$
$a_{ij} < 0$ 이면 $b_i := b_i - u_j \cdot a_{ij}$, $d_i := d_i - l_j \cdot a_{ij}$

그림 5. 단일요소열의 처리.

변수의 상한과 하한이 일치하는 변수를 고정변수라고 하는데 이러한 변수들은 그 값을 미리 고정해 둘 수 있다. 이러한 변수를 일정값으로 고정할 때에는 그 영향을 우변상수와 좌변상수에 반영하는 과정이 필요하다.

3.3 선회연산을 통한 제거방법

선회연산을 통한 사전처리는 변수치환을 통해 등식 제약식과 변수를 제거하는 방법을 일컫는다. 변수치환방법은 크게 이중요소행(Doubleton row)에 대한 치환과 자유변수에 대한 치환방법으로 나눌 수 있다. 또한 자유변수에 대한 치환방법은

원 자유변수(Original free variable)에 대한 처리와 암시적 자유변수(Implied free variable)에 대한 처리로 분류될 수 있다.

(1) 이중요소행에 대한 치환연산

이중요소행이란 변수를 두 개만 가지는 동식제약식을 지칭하는데 이러한 제약식은 치환에 의해 제거될 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 i행과 x_j를 제거할 수 있다. 이때 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 행하는 것을 의미한다.

(2) 자유변수에 대한 치환연산

자유변수란 변수의 상하한이 무한대인 변수를 말한다. 이러한 자유변수는 두 가지로 분류될 수 있다. 즉, 원 문제에서 주어지는 상하한이 이미 무한대인 원 자유변수와 제약식에 의해 암시되는 상하한이 원 문제에서 주어지는 상하한보다 더 강하여 자유변수로 취급되어도 문제에 아무런 영향을 미치지 못하는 암시적 자유변수가 있다. 이러한 자유변수들은 치환에 의해 제거될 수 있다. 예를 들어 다음과 같은 제약식이 있다고 할 때,

$$a_1x_1 + \cdots + a_nx_n = b$$

x_n가 자유변수라고 하자. 그러면, 치환식

$$x_n = (b - a_1x_1 - \cdots - a_{n-1}x_{n-1})/a_n$$

를 이용하여 x_n을 제거할 수 있다. 물론 x_n에 대한 상하한 제약이 없으므로 추가되는 제약식은 없다. 암시적 자유변수를 찾기 위해서는 우선, 각 제약식이 취할 수 있는 범위를 구해야 한다. 그 범위는 다음과 같이 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 위의 범위를 이용하여 다음 식을 유도할 수 있다.

$$\begin{aligned}
&\forall k \in P_i,\\
&\quad \underline{b}_i + a_{ik}(x_k - l_k) \le \sum_j a_{ij}x_j \le b_i\\
&\forall k \in P_i,\\
&\quad \underline{b}_i + a_{ik}(x_k - l_k) \le \sum_j a_{ij}x_j \le b_i\\
&\forall k \in N_i,\\
&\quad \underline{b}_i + a_{ik}(x_k - u_k) \le \sum_j a_{ij}x_j \le b_i
\end{aligned} \qquad (3.1)$$

그리고, (3.1)에서 다음과 같이 암시적 한계를 유도할 수 있다.

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

위에서 새로 구해진 변수 x_k에 대한 상하한이 원래 주어진 상하한보다 강한 한계이면 x_k에 대한 한계가 i번째 제약식에 의해 암시되게 되어 자유변수로 처리될 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 (3.1)은 (3.2)로 교체된다.

$$\begin{aligned}
&a_{ik}x_k + \sum_{j\in P_i-\{k\}} a_{ij}l_j + \sum_{j\in N_i} a_{ij}u_j\\
&\quad \le \sum_j a_{ij}x_j \le b_i, \quad \text{if } k \in P_i\\
&a_{ik}x_k + \sum_{j\in P_i} a_{ij}l_j + \sum_{j\in N_i-\{k\}} a_{ij}u_j\\
&\quad \le \sum_j a_{ij}x_j \le b_i, \quad \text{if } k \in N_i
\end{aligned} \qquad (3.2)$$

(3.2)는 다음과 같은 암시적 한계를 유도한다.

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

위의 새로운 한계도 암시적 자유변수를 찾아내는 데 사용될 수 있다.

3.4 내재적 중복성의 제거방법

내재적 중복성이란 명시적으로 표현된 문제의 제약이 문제 내에서 또다시 암시되고 있는 경우를 말한다. 이러한 제약들은 제거될 수 있다. 이러한 형태로 분류될 수 있는 사전처리로는 할인가를 이용한 변수값의 고정기법, 내재적 중복제약식

제거기법, 변수의 상하한 강화기법 등이 있다.

(1) 할인가를 이용한 변수값의 고정기법

문제 (P)를 단체법으로 풀어 최적해를 얻었을 때, 하한 비기저변수의 할인가는 비음이고 상한 비기저변수의 할인가는 비양이다. 그런데 문제의 형태를 보아서 할인가의 부호를 미리 결정할 수 있는 변수들이 있다. 그러한 변수들은 할인가의 부호를 계산하여 그 값을 하한이나 상한으로 고정할 수 있다.

기저행렬이 B라고 할 때, 변수 x_j의 할인가는 다음과 같이 계산된다.

$$\bar{c}_j = \pi A_{\cdot j} - c_j, \quad \pi = c_B^T B^{-1}$$

그리고 π_i의 부호는 제약식의 형태를 이용해 다음과 같이 결정될 수 있다.

$$d_i = -\infty,\ b_i < \infty \text{ 일 때 } \pi_i \geq 0$$
$$d_i > -\infty,\ b_i = \infty \text{ 일 때 } \pi_i \leq 0$$

그러므로 다음과 같은 조건을 만족시키는 변수들은 그 한계값으로 고정될 수 있다.

$$\begin{aligned} &c_j > 0, \quad a_{ij} > 0 \ \forall i \in F \\ &\qquad\quad a_{ij} < 0 \ \forall i \in G \qquad \text{또는} \\ &c_j < 0, \quad a_{ij} > 0 \ \forall i \in G \\ &\qquad\quad a_{ij} < 0 \ \forall i \in F \end{aligned}$$
$$(\text{단, } F = \{i : d_i > -\infty,\ b_i = \infty\})$$
$$G = \{i : d_i = -\infty,\ b_i < \infty\})$$

그림 6. 할인가를 이용한 변수의 고정.

위에서 첫째 경우에는 할인가 $\bar{c}_j$가 항상 음이므로 x_j는 상한으로 고정될 수 있고, 둘째 경우에는 할인가가 항상 양이므로 x_j는 하한으로 고정될 수 있다.

(2) 내재적 중복제약식의 제거

암시적 자유변수를 파악하기 위한 기법에서 설명했듯이 각 변수의 상한과 하한값을 이용하여 한 제약식이 가질 수 있는 최대값과 최소값을 구할 수 있다. 그러면 다음의 관계가 성립한다.

$$\underline{b}_i \leq \sum_j a_{ij} x_j \leq \bar{b}_i$$

그러면 다음과 같은 경우에 위 i번째 제약식은 변수들의 한계들에 의해 그 중복성이 암시되는 제약식이 되어 제거가능하다.

$$d_i \leq \underline{b}_i, \quad b_i \geq \bar{b}_i$$

(3) 변수의 상하한 강화 기법

변수의 상하한을 강화하는 기법은 암시적 자유변수 처리기법에서 설명했듯이 식 (3.1)과 식 (3.2)에서 유도되는 암시적 한계를 이용해서 변수의 상하한을 좀더 강화시키는 기법이다. 이러한 기법은 변수의 상하한을 이용하는 다른 사전처리의 효과를 높이는 기능을 하게 된다.

4. 사전처리의 구현방법

단체법 프로그램을 위한 사전처리의 구현을 위해 고려해야 할 사항으로는 앞에서 설명되었던 여러 사전처리의 적용방법과 효과적인 구현방법이다.

4.1 사전처리의 적용방법

기본적 처리는 별다른 고려사항 없이 쉽게 구현될 수 있다. 하지만 선회연산을 이용한 처리방법에서는 두 가지 고려사항이 있다. 첫째는 입력자료에 대한 희소도의 유지이고, 둘째는 입력자료의 수치적 안정성 문제이다. 일반적으로 주어진 행렬에 선회연산을 수행하게 되면 비영요소의 추가도입이 불가피하고, 또한 행렬의 수치적 안정성도 영향을 받게 된다.

입력자료의 희소도를 유지하기 위해서는 과도한 치환연산을 지양해야 한다. 즉, 추가 비영요소의 수를 예측하여 그 수가 너무 크다면 치환연산을 포기해야 한다. 추가 비영요소의 수를 정확하게 계산하는 것은 계산상 부담이 되므로 이에 대한 상한을 구하여 사용한다. 즉, 치환대상이 되는 행의 비영요소수와 치환대상이 되는 열의 비영요소수를 곱한 값을 그 상한으로 사용한다. 본 연구에서는 이 상한값이 10을 넘을 경우 치환연산을 포기하도록 하였다.

입력자료의 수치적 안정성을 고려하기 위해서는 선회연산 시 선회요소의 적절한 선택이 필요하다. 이중요소행에 대한 치환연산의 경우, 더 큰 절대값 계수를 가지는 열이 선회열이 되도록 한다. 자유변수에 대한 치환연산의 경우에는 threshold pivoting의 개념을 이용하였다. 즉, 선회행의 최대절대값의 일정배(약 0.1) 이상의 비영요소를 가지는 열을 선회열로 선택하였다.

내재적 중복성 제거기법에서 변수의 상하한 강화기법은 적용하지 않았다. 기존의 알려진 바와 같이 변수의 상하한이 강화되면 단체법 수행과정에서 퇴화 가능성이 높아져 전체 수행성능이 저하되었기 때문이다.

4.2 사전처리의 효과적 구현

사전처리의 효과적 구현을 위해서 고려해야 할 것으로는 사전처리 적용의 적절한 순서와 효과적인 자료구조의 구현이다.

(1) 사전처리의 적용순서

사전처리의 적용순서로는 기본적 처리, 선회연산을 통한 처리, 내재적 중복성 처리의 순서로 행하였다. 그리고 이 세가지 기법을 반복적으로 수행하였고, 제거율이 1~2% 이상일 때까지 계속 수행하였다.

(2) 자료 구조

일반적으로 단체법 프로그램에서 입력자료를 보관하는데는 자료의 연속성이 보장되어 효율적인 열압축구조(Column packed form)와 같은 정적 자료구조를 사용한다. 그러나 사전처리과정 중에는 비영요소의 삽입·삭제가 빈번히 일어나기 때문에 동적 자료구조의 사용이 적절하다. 이를 위해 본 연구에서는 행·열 단방향 연결 리스트구조를 사용하였다. 특히 연결 리스트와 같은 동적 자료구조는 선회연산을 통한 처리에서 필수적이다.

행과 열이 삭제될 때에는 자료구조에 그 내용을 곧바로 적용하지 않고 삭제되었다는 표시만 하여 사전처리 수행속도를 높였다. 또한, 사전처리과정 중에는 한 행이나 열에 존재하는 비영요소의 수를 자주 알아야 하는데 이를 위해 본 연구에서는 그 수를 계속 갱신하여 보관하였다.

사용자의 입장에서 보면 자신이 입력한 문제에 대한 해의 출력을 원하므로 원 문제에 대한 해를 도출하기 위한 문제 복구과정이 필요하게 되는데 이를 위해 사전처리 전의 문제를 보관할 필요가 있다. 이를 위한 방법으로 적당한 배열을 할당받아 원 문제를 보관했다가 해법 종료 후 다시 복원하는 방법과 원 문제를 파일로 저장했다가 다시 읽어들이는 방법이 있다. 첫째 방법은 메모리의 낭비가 단점이지만 신속히 원 문제 자료를 복원할 수 있다는 장점이 있다. 반면, 둘째 방법은 메모리의 낭비는 없지만 복원시 다소 지연시간이 있다는 것이 단점이다.

5. 원 문제 최적기저의 복원

원 문제 (P)에 사전처리를 적용하여 변형된 문제를 (P')이라고 하자. 단체법 프로그램에서는 이제 (P')을 풀게 되고 그 결과로 (P')에 대한 최적기저가 생성되게 된다. 그러나 사용자의 입장에서는 원 문제 (P)에 대한 최적기저를 얻기 원한다. 따라서, 해법종료 후에 변형된 문제의 최적기저로부터 원 문제의 최적기저를 복원하는 과정이 필요하게 된다. 본 절에서는 이에 대해 설명한다.

5.1 제약식이 제거되는 경우

사전처리에 의해 제약식이 하나 제거되는 경우에는 기저행렬의 크기가 하나 줄게 된다. 따라서, 원 문제를 복원할 때에는 기저행렬의 크기를 하나 증가시키기 위해 기저변수를 하나 추가하여야 한다. 따라서 사전처리과정에서 제약식이 제거된다면 나중의 복원과정에서 기저변수가 될 수 있는 변수에 대한 정보를 보관하여야 한다. 그 방법들은 다음 정리들과 같다.

[정리 1]

공백행이 제거되는 경우에는 그 공백행에 대한 여유변수 또는 인공변수가 기저변수가 된다.

(증명) 사전처리 후의 문제를 (P')이라고 하고, (P')에 대한 최적기저를 B' 이라고 하자. 그러면, (P')에 제거된 공백행 제약식이 추가된 문제 P에 대한 최적기저가

$$B=\begin{pmatrix} B' & 0 \\ 0 & 1 \end{pmatrix}$$

가 됨을 보이면 된다.

(P')에 대한 최적기저해를 $\mathbf{x}_{B'}$, $\mathbf{x}_{N'}$ 이라고 하면 (P)의 기저 B에 대한 기저해는 다음과 같다.

$$\mathbf{x}_B = B^{-1}(\mathbf{b} - N\mathbf{x}_N)$$
$$= \begin{pmatrix} (B')^{-1} & 0 \\ 0 & 1 \end{pmatrix}\mathbf{b} - \begin{pmatrix} (B')^{-1} & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} N' \\ 0 \end{pmatrix}\mathbf{x}_N$$

$$\begin{pmatrix} \mathbf{x}_{B'} \\ x_0 \end{pmatrix} = \begin{pmatrix} (B')^{-1}\mathbf{b}' \\ b_0 \end{pmatrix} - \begin{pmatrix} (B')^{-1}N' \\ 0 \end{pmatrix}\mathbf{x}_{N'}$$

위 식에서 $\mathbf{x}_{B'}$ 은 가능 기저해이고 제거된 공백제약식은 가능제약식이므로 $\mathbf{x}_B$는 가능 기저해이다.

다음으로 쌍대가능성에 대해 알아보면,

$$\bar{c}_j = c_j - \mathbf{c}_B^T\begin{pmatrix} (B')^{-1} & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} A_{\cdot j} \\ 0 \end{pmatrix}$$

$$\bar{c}_j' = c_j - \mathbf{c}_B^T(B')^{-1}A_{\cdot j}', \quad \bar{c}_0 = 0$$

윗식을 통해 $\mathbf{x}_{B'}$는 쌍대가능해이므로 $\mathbf{x}_B$도 쌍대가능해임을 알수 있다. 따라서 B는 (P)의 최적기저이다. ■

[정리 2]

단일요소행이 제거되어 변수의 한계로 처리될 때, 그 변수가 기저변수이면 그 행의 여유변수가 기저변수가 된다.

(증명) 사전처리 후의 문제를 (P')이라고 하고, (P')에 대한 최적기저를 B' 이라고 하자. 그리면, (P')에 제거된 단일요소행 제약식이 추가된 문제 P에 대한 최적기저가

표 1. 실험문제의 정보

문제이름	제약식	변수	비영요소	문제이름	제약식	변수	비영요소
25fv47	821	1571	10400	perold	625	1376	6018
80bau3b	2262	9799	21002	pilot	1441	3652	43167
agg3	516	302	4300	pilot.ja	2030	4883	73152
bnl1	643	1175	5121	pilot87	940	1988	14698
bnl2	2324	3489	13999	pilotnov	975	2172	13057
cycle	1903	2857	20720	scfxm3	990	1371	7777
czprob	929	3523	10669	sctap3	1480	2480	8874
d2q06c	2171	5167	32417	shell	536	1775	3556
degen3	1503	1818	24646	ship12l	1151	5427	16170
fffff800	524	854	6227	ship12s	1151	2763	8178
ganges	1309	1681	6912	stair	356	467	3856
greenbea	2392	5405	30877	stocfor3	16675	15695	64875
maros	846	1443	9614	tuff	333	587	4520
nesm	662	2923	13288	woodw	1098	8405	37474

$$B=\begin{pmatrix} B' & 0 \\ 1\cdots 0 & 1 \end{pmatrix}$$

가 됨을 보이면 된다. 이에 대한 증명은 [정리 1]의 방식과 유사하다. ■

[정리 3]

치환에 의해 제거되는 제약식에 대한 기저변수는 치환연산시에 제거되는 변수가 된다.

(증명) 치환연산에 의해 제거되는 변수는 (암시적) 자유변수이다. 따라서 영구적 기저변수가 된다. ■

[정리 4]

내재적 중복제약식이 제거될 때, 그 제약식의 기저변수는 여유변수가 된다.

(증명) 내재적 중복제약식의 여유변수는 항상 양의 값을 지니게 되므로 항상 기저변수가 된다. 그리고, 여유변수열은 단일요소열이므로 기저행렬에서 선회점이 될 수 있다. 따라서, 내재적 중복제약식의 기저변수는 여유변수가 된다. ■

5.2 변수가 제거되는 경우

앞에서 설명한 기저변수 지정방법만을 이용하여서는 원 문제의 최적기저를 복원하는 데 불충분하다. 그 이유는 (P)에 대한 최적기저를 복원한 후, 나머지 비기저변수의 상태 즉, 각 비기저변수가 상한 비기저변수인지 하한 비기저변수인지를 정할 때 원비가능성을 유발시킬 우려가 있다. 예를 들어 복원 최적기저에 대해 한 비기저변수의 할인가를 계산해 볼 때, 그 값이 영이 나왔다면 쌍대가능성만을 위해서는 그 변수의 상태를 상한으로 하든 하한으로 하든 상관이 없지만 원가능성에는 영향을 미칠 수 있다. 따라서, 최적기저의 복원작업 이외에 최적에서의 변수값도 복원하는 과정이 필요하다. 이를 위해 변수의 값이 고정되는 경우에 그 고정되는 값을 보관하고 있어야 하고, 치환연산에 의해 제거되는 변수들은 그 치환식을 보관하고 있어 나중에 그 변수값을 다른 변수값들을 이용하여 복원하여야 한다.

5.3 변수의 한계가 변하는 경우

변수의 한계가 변하는 경우에는 그 변수의 한계를 최종적으로 변하게 만드는 제약식에 대한 정보를 보관하여야 한다.

6. 실험결과

Netlib 문제 중 대형문제 28개에 대해서 제거되는 제약식과 행의 개수 및 사전처리 후 A의 비영요소 개수 등을 다른 코드들과 비교하였다. 비교대상은 공개 단체법 프로그램인 Soplex v1.0, 내부점 선형계획법 코드인 HOPDM v2.13 그리고 상용 프로그램인 CPLEX v3.0 등이다. 또 사전처리에 걸리는 시간도 비교하였다. 본 논문의 방법은 LPAKO에 구현되었다.

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

- LPAKO: gcc -O3
- Soplex : gcc -O3
- HOPDM : f77 -fast -O3 -libmil -native -N150
- CPLEX: 알려지지 않음.

실험문제의 정보는 <표 1>과 같다.

표 2. 사전처리 결과비교

문제이름	LPAKO			SOPLEX			HOPDM			CPLEX		
	(1)	(2)	(3)	(1)	(2)	(3)	(1)	(2)	(3)	(1)	(2)	(3)
25fv47	136	122	9986	141	127	-	52	36	9959	137	122	9903
80bau3b	277	765	19748	246	720	-	297	1063	19048	297	1119	18981
agg3	38	4	4242	36	4	-	61	8	3972	234	53	2298
bnl1	188	185	4604	180	129	-	85	66	4619	192	180	4632
bnl2	1368	1338	10297	1354	1312	-	476	482	12458	1381	1394	10252
cycle	827	928	13416	763	769	-	503	454	14111	974	1066	12993
czprob	454	968	5172	406	634	-	268	835	5376	465	1032	4982
d2q06c	291	485	30846	305	496	-	159	203	30263	296	550	30600
degen3	93	93	24460	93	93	-	0	0	24363	96	96	24427
fffff800	206	195	4917	226	215	-	211	191	4388	229	214	4804
ganges	810	1034	4035	829	1000	-	469	509	5487	733	878	4187
greenbea	1183	2041	23624	943	1110	-	520	1494	23161	1372	2347	23028
maros	291	541	6279	199	245	-	220	491	5743	307	600	5788
nesm	56	273	12941	56	273	-	16	247	12926	40	216	12933
perold	102	265	5317	98	254	-	45	143	5381	118	280	5359
pilot	142	370	40677	109	316	-	91	323	40506	166	409	40467
pilot.ja	210	585	10979	201	565	-	62	288	70361	140	372	70370
pilot87	116	347	70412	123	340	-	145	442	10931	195	568	10985
pilotnov	202	449	11523	215	461	-	145	311	11466	190	435	11528
scfxm3	246	213	6652	228	183	-	186	105	6487	275	218	6494
sctap3	134	713	7640	134	713	-	134	124	8229	136	713	7630
shell	284	540	2476	292	548	-	49	327	2902	286	571	2414
ship12l	541	1280	12507	318	204	-	541	1280	9230	542	1280	9222
ship12s	883	916	5679	786	696	-	811	854	4273	884	916	4121
stair	107	189	3482	108	190	-	0	83	3664	114	197	3520
stocfor3	6096	4957	45212			-	1339	9159	55088	5935	13755	52492
tuff	184	192	3984	178	173	-	87	64	3707	191	199	4041
woodw	395	3211	19574	9	173	-	395	3058	19727	543	4395	14536

(1) 제거된 행의수 (2) 제거된 열의 수 (3) 사전처리 후 A의 비영요소 수

(-: 프로그램에서 출력하지 않는 내용임)

<표 1>의 문제를 실험한 결과는 <표 2>, <표 3>과 같다. <표 2>는 각 코드들의 사전처리 수행결과를 나타내고 있는데 LPAKO의 사전처리가 HOPDM의 사전처리에 비해 우수한 성능을 나타내고 있고, Soplex에 비해서는 비슷한 성능을 보였다. 한편 CPLEX에 비해서는 열등하였다.

단체법 프로그램에서 사전처리의 효과를 검증하기 위하여 공백행과 공백열만을 제거한 경우와 본 연구에서 제시한 모든 방법이 구현된 경우를 실험 비교하였는데, 이는 다음 <표 3>과 같다.

이 실험결과를 살펴보면 우선 사전처리가 대형의 선형계획법 문제에 대한 단체법 프로그램의 성능향상에 큰 영향을 미친다는 것을 알 수 있다.

예를 들어, pilot87, stocfor3, greenbea 등의 대형문제에서 사전처리의 효과는 상당하다고 할 수 있다. 그러나 agg3, degen3 등

표 3. 사전처리의 영향

문제이름	사전처리 안함		사전처리함		문제이름	사전처리 안함		사전처리함	
	반복횟수	시간	반복횟수	시간		반복횟수	시간	반복횟수	시간
25fv47	1579	8.64	1594	8.08	perold	1569	6.19	1345	5.39
80bau3b	7675	59.10	7213	51.27	pilot	4817	100.34	4320	98.22
agg3	146	0.27	163	0.28	pilot.ja	2391	13.52	2470	14.36
bnl1	891	2.53	688	1.64	pilot87	6873	321.24	5891	250.38
bnl2	2248	18.96	1374	6.56	pilotnov	1482	8.30	1142	6.14
cycle	1502	8.44	1587	6.73	scfxm3	1013	3.32	908	2.49
czprob	859	4.27	580	2.26	sctap3	575	2.27	581	1.99
d2q06c	6111	91.71	5611	78.65	shell	442	1.24	337	0.65
degen3	1759	14.46	1962	17.89	ship12l	817	7.04	737	3.35
fffff800	175	0.39	132	0.34	ship12s	408	2.21	370	0.99
ganges	640	3.36	502	1.34	stair	450	1.00	258	0.62
greenbea	6295	69.10	4028	33.08	stocfor3	6706	334.15	5162	166.88
maros	931	3.37	713	2.02	tuff	182	0.37	203	0.37
nesm	2411	8.20	2134	7.65	woodw	1364	12.69	754	5.05

과 같은 문제에서처럼 사전처리의 효과가 부정적으로 나타나는 경우도 있는데 이는 문제의 축소가 단체법 반복횟수를 증가시키거나 비영요소의 추가도입으로 인한 연산의 속도저하 등을 초래한 경우이다.

그러나 전체적인 실험결과를 볼 때 사전처리의 효과는 긍정적이라고 볼 수 있다.

7. 결 론

본 연구에서 단체법 프로그램에 적용될 수 있는 사전처리 방법들은 다음과 같이 분류하였다.

- 기본적 방법
- 선회연산을 통한 제거방법
- 내재적 중복성의 제거방법
- 변수의 상하한 축소방법

그리고 단체법 프로그램을 위한 사전처리를 구현할 때 고려되어야 할 사항으로, 비영요소의 추가도입 문제, 수치적 안정성, 사전처리의 적용순서, 빠른 처리를 위한 자료구조 등에 대해 알아보았다. 또한, 사전처리된 문제의 최적기저로부터 원 문제의 최적기저를 복원하는 문제에 대해서 해법을 제시하였다.

위 방법들에 대한 실험결과를 통해 사전처리는 대형의 선형계획법 문제를 단체법으로 효과적으로 풀기 위해 필수적이라는 것을 알 수 있었다. 또한, 본 연구의 실험결과를 토대로 구현된 LPAKO의 사전처리는 타 공개 단체법 프로그램에 비해 우수하거나 비슷한 성능을 나타내었으나, 상용 프로그램인 CPLEX에 비해서는 열등함을 보였다.

참고문헌

박순달 (1992), *선형계획법(3정판)*, 민영사.

박순달, 김우제, 박찬규, 임성묵 (1998), 단체법 프로그램 LPAKO 개발에 관한 연구, *경영과학*, 15(1).

박찬규, 임성묵, 김우제, 박순달 (1997), 상하분해를 이용한 단체법의 구현에 관한 연구, *대한산업공학회'97 추계학술대회 논문집*.

성명기, 박순달 (1996), 대형선형계획법문제의 사전처리, *한국경영과학회'96 추계학술대회논문집*, 285-288.

Andersen, E. D. (1995), Finding all linearly dependent rows in large-scale linear programming, *Optimization Methods and Software*, 6, 219-227.

Andersen, E. D., Gondzio, J., Mészáros, C. and Xu, X. (1996), Implementation of interior point methods for large scale linear programming, *Interior Point Methods in Mathematical Programming*, T. Terlaky (Eds.), Chapter 6, 189-252, Kluwer Academic Publisher.

Andersen, E. D. and Andersen, K. D. (1995), Presolving in linear programming, *Mathematical Programming*, 71, 221-245.

Fang, S. C. and Puthenpura, S. (1993), *Linear Optimization and Extensions: Theory and Algorithms*, Prentice-Hall, Inc.

Gondzio, J. (1997), Presolve analysis of linear programs prior to applying an interior point method, *INFORMS Journal on Computing*, 9(1), 73-91.

Lustig, I. J., Marsten, R. E. and Shanno, D. F. (1994), Interior point methods for linear programming: computational state of the art, *ORSA Journal on Computing*, 6(1), 1-14.

Mészáros, C. (1998), On free variables in interior point methods, *Optimization Methods and Software*, 9, 121-139.