내부점 선형계획법에서의 사후처리†

Postsolving in interior-point methods

이상욱* 임성묵** 성명기*** 박순달*

leesw@orlab.snu.ac.kr, sungmook@nca.or.kr, mkseong71@yahoo.com, soondal@snu.ac.kr

* 서울대학교 산업공학과, ** 한국전산원, *** LG-CNS

Abstract

It is often that a large-scale linear programming(LP) problem may contain many constraints which are redundant or cause infeasibility on account of inefficient formulation or some errors in data input. Presolving or preprocessing is a series of operations which removes the underlying redundancy or detects infeasibility in the given LP problem. It is essential for the speedup of an LP system solving large-scale problems to implement presolving techniques. For the recovery of an optimal solution for the original problem from an optimal solution for the presolved problem, a special procedure, so called postsolving, must be applied. In this paper, we present how a postsolving procedure is constructed and implemented in LPABO, a interior-point based LP system. Briefly, all presolving processes are logged in a data structure in LPABO, and after the end of the solution method an optimal solution for the original problem is obtained by tracing the logs. In each stage of the postsolving procedure, the optimality of intermediate solutions is maintained. We tested our postsolving procedure on Netlib, Gondzio and Kennington LP data sets, and concluded that the computational burden of the procedure is relatively negligible compared with the total solving time.

1. 서론

대형 선형계획법 문제는 희소행렬로 이루어진 경우가 많다. 이러한 대형 선형계획법 문제를 저장하기 위해서는 많은 기억용량을 필요로 하게 되며 이로 인해 선형계획법 프로그램의 효율성이 저하되게 된다. 저장공간의 효율성을 위해서는 행렬의 비영요소만을 저장하는 방식을 취함으로써 효율성을 증대시킬 수 있다. 비영요소의 저장방식외에, 대형 선형계획법 문제는 문제의 입력과정에서 생긴 중복행, 또는 중복열을 포함하는 경우나 비가능 문제의 요소를 포함한 경우가 발생하게 된다. 전자의 경우는 필요없는 부가적 제약식 또는 열을 제거함으로써 문제의 크기를 줄이는 것이 효율적이게 된다. 후자의 경우는 문제를 풀기전에 미리 비가능성을 판단함으로써 사전에 문제의 비가능성을 판단하게 된다. 이러한 일련의 과정을 사전처리라고 부른다. 사전처리를 수행하게 되면 원래의 주어진 문제와는 다른 축소된 문제가 된다. 따라서, 사전처리 후의 문제를 선형계획법으로 풀고 난 후의 최적해를 바로 사용할 수 없게 되고 다시 원래의 최적해로 복원하는 과정이 필요하게 된다. 이와 같이 원래의 주어진 문제의 최적해로 복원하는 과정을 사후처리라고 한다.

사전처리에 관한 연구는 Brearly et al.[3]에 의해 처음 제시되었다. 단체법에 관한 사전처리는 Brearly [3], Bradley [2], Williams [14], Tomlin [11] [12] [13], Seong[9], Lim[6]에 의한 연구가 이루어져 왔으며, 내부점에 관한 사전처리는 Lustig [7], Fourer [4], Anderson [1], Gondzio [5], Seong [9]에 의해 연구되어 왔다. Brearly[3]에 의해 사전처리가 제시되었지만 사후처리에 관한 언급은 없다.

사후처리에 관한 연구는 Fourer [4], Anderson [1], Gondzio [5]에 의해 내부점 선형계획법을 이용한 사후처리 방법을 제안하였다. Fourer[4]는 사전처리된 문제의 최적해로부터 원 문제의 최적해를 복원하는 방법을 제안하였는데, 최적 쌍대변수가 가져야 할 성질을 이용하여 제거된 제약식에 대해서 쌍대변수의 값을 찾는 방법을 제안하였다. Fourer [4]가 제안한 사전처리 방법은 비교적 간단하기 때문에 원최적해는 쉽게 복원할 수 있다. 여러 가지 사전 처리 방법을 적용하게 되면 사후처리는 복잡하게 된다. 쌍대 최적조건을 만족하는 해를 구하는 것은 선형 가능성의 문제를 푸는것과 같다. Anderson [1]은 최적 쌍대변수가 가져야 할 성질을 이용하여 제거된 제약식에 대한 쌍대변수의 값을 찾는 방법을 연구하였는데, Fourer [4]와 달리 각각의 사전처리 방법에 대해서 원변수 및 쌍대변수의 값을 구하는 방법을 자세히 설명하였다. Gondzio [5]는 사전처리 과정을 원변수와 쌍대변수에 관한 사전처리로 구별하여 각 사전처리의 각 단계를 차례로 저장한 다음, 사전처리된 문제의 최적해에서 출발하여 사전처리 과정을 역으로 따라 가면서 원변수와 쌍대변수의 최적값을 복원하는 방법을 제시하였다. 이 방법은 매 사후처리 과정에서 최적성을 유지하게끔 하는 전략이다. 위에 열거된 연구들은 사후처리에 대하여 단순히 사전처리의 역으로 사후처리를 수행하는 것을 제안하였기 때문에 사후처리를 구현하는데 있어 어려움이 되고 있다.

본 연구에서는 Anderson [1]과 Gondzio [5]가 제안한 방법들의 필요성 및 사후처리의 과정을 본 연구진에서 개발한 내부점 프로그램인 LPABO에 구현한 내용을 바탕으로 기술하고 사후처리 시간이 전체 수행속도에 비해 걸리는 시간이 실험을 통하여 보이는 것

† 본 연구는 한국과학재단의 특정기초연구과제(과제번호 R01-2002-000-00168-0)의 지원을 받았음.

을 목적으로 한다.

2. 선형계획법에서의 사전처리

본 장에서는 사후처리의 설명을 위해 사전처리에 대한 간략한 설명을 한다. 앞으로 설명할 선형계획 문제는 다음과 같다.

$$(P): \min c^T x \quad s.t \quad Ax = b, \quad l \le x \le u$$

$$(D): \max b^T y + u^T w - l^T z \quad s.t \quad A^T y + w - z = c, \quad y \text{ unrestricted}, \ z, w \ge 0$$

위의 식에서 $c, z, w \in R^n$, $b.m \in R^m$, $A \in R^{m \times n}$ 이고 T는 전치행렬을 의미하기로 한다.

먼저 사전처리의 목적은 다음과 같다.
- 주어진 문제의 비가능성 및 무한해 판별
- 문제의 축소

첫 번째 목적을 위한 사전처리는 다음과 같이 분류할 수 있다.
- 공백행을 이용한 비가능성 판정
- 단일 요소행을 이용한 비가능성 판정
- 제약식의 한계를 이용한 비가능성 판정
- 공백열을 이용한 무한해의 판정
- 변수들의 한계를 이용한 무한해의 판정

다음으로 문제의 축소와 관련한 사전처리 방법은 다음과 같이 분류할 수 있다.
- 공백행 및 공백열의 제거
- 단일요소행 및 자유 제약식(free rows)의 제거
- 단일요소열 및 고정변수(fixed columns)의 제거
- 등식 제약식을 갖는 이중 요소행의 제거
- (암시적) 자유변수를 갖는 희소행의 제거
- 할인가를 이용한 변수값의 고정(단체법)
- 중복행의 제거
- 변수의 한계 강화

실제 프로그램에서는 위의 사전처리 방법들을 순차적으로 적용하게 된다.

앞으로의 전개를 위해 다음과 같이 용어를 정의한다.

정의 1. 제약식의 최대 허용값(최소 허용값) : 제약식 i의 최대 허용값(최소 허용값) $\bar{b}_i(\underline{b}_i)$은 다음과 같이 정의된다.

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

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

정의 2. 암시적 한계 : 제약식 i로부터 유도된 변수 x_j의 암시적 한계 l_j', u_j'는 다음과 같이 정의된다.

l_j와 u_j가 모두 유한인 경우는

$$x_j \ge l_j' = u_j + \frac{b_i - \underline{b}_i}{a_{ij}}, \ \forall j \in N_i$$

$$x_j \le u_j' = l_j + \frac{b_i - \bar{b}_j}{a_{ij}}, \ \forall j \in P_i$$

l_j또는 u_j가 무한인 경우는

$$x_j \ge l_j' = \frac{b_i - \sum_{j \in P_i} a_{ij} l_j - \sum_{j \in N_i - \{k\}} a_{ij} u_j}{a_{ik}}, \ k \in N_i$$

$$x_j \le u_j' = \frac{b_i - \sum_{j \in P_i - \{k\}} a_{ij} l_j - \sum_{j \in N_i} a_{ij} u_j}{a_{ik}}, \ k \in P_i$$

이 된다.

정의 3. 암시적 자유변수 : 변수 x_j의 하한 l_j와 상한 u_j에 대하여 $l_j \le l_j'$이고 $u_j \ge u_j'$이면 x_j는 암시적 자유변수라고 정의한다.

3. 사후처리의 구현

3.1 사후처리의 특성

사전처리가 총 n회 수행되었다고 할 경우 i번째의 사전처리를 수행하고 난 뒤의 문제를 P_i라고 하자. 사전처리와 사후처리는 다음의 과정을 거치게 된다.

$$P_1 \rightarrow P_2 \rightarrow \cdots \rightarrow P_{n-1} \rightarrow P_n \rightarrow P_{n-1} \rightarrow \cdots \rightarrow P_2 \rightarrow P_1$$

위의 그림에서 P_1에서 P_n까지는 사전처리의 과정이고 P_n부터 P_1까지는 사후처리의 과정을 나타낸다. 사후처리를 위해 사전처리에서는 각 사전처리 방법이 일어날 때마다 사전처리 방법이 일어난 순서와 변경된 정보를 기록하여 두어야 사후처리가 수행되어짐을 알 수 있다.

3.2 사후처리의 구현

사전처리의 각 단계에서 일어나는 연산을 기록하기 위해서 'HISTORY_FILE'을 만들어 정보를 기록하였다. 'HISTORY_FILE'의 구조체를 살펴보면 다음과 같다.

```
struct HISTORY_FILE {
        int type;
        int row1, row2;
        int col;
        double value1, value2;
};
```

위의 구조체에서 'type'는 어떤 사전처리 연산이 적용되었는가를 나타내고, 'row1','row2'는 행번호를 'col'은 열 번호를 저장하게 된다. 'value1'과 'value2'는 변수의 최적값, 변수의 한계의 변경 전후의 값, 선회 승수들을 저장하게 된다. 따라서 HISTORY_FILE의 변수를 배열로 사용하게 되면 매 회 적용되는 사전처리의 정보를 저장할 수 있게 된다.

4. 내부점 방법에서의 사후처리법

본 장에서는 3장에서 설명한 방법으로 사후처리를 적용하기 위해 각 사전처리 방법에 대한 사후처리 방법에 대하여 알아본다.

4.1 중복행의 제거

중복행은 비영요소가 없거나 중복행을 이루는 변수들의 한계이내의 값이 모두 해당 제약식을 항상 만족시키게 되어 원문제에서 제거되는 행을 말한다. 제거된 중복행에 대한 최적 쌍대변수만 복원하면 되는데, 중복행은 항상 만족되므로 중복행에 해당하는 쌍대변수의 값은 $y^* = 0$이 된다.

4.2 변수의 한계강화

x_k의 상한이 제약식 r에 의해 u_k에서 $u_k'(< u_k)$로 강화되었다고 가정하자. 원변수는 영향을 받지 않고 쌍대변수만 영향을 받으므로 '중복행의 제거'와 같이 쌍대변수의 값만 복원시키면 된다. x_k^*의 값이 u_k'보다 작으면 상보여유조건에 의해 $w_k = 0$이 된다. $x_k^* = u_k'$인 경우는 다음과 같이 쌍대해를 복원한다.

$$w_k^* = 0,\ y_r^* = \frac{c_k - \sum_{i \neq r} a_{ik} y_i^* + z_k^*}{a_{rk}}.$$

$a_{rj} \neq 0 (1 \leq j \leq n, j \neq k)$에 대해서 다음을 수행한다.

$$\alpha = c_j - \sum_i a_{ij} y_i^*.$$

만약 $\alpha \geq 0$이면, $w_j^* = \alpha,\ z_j^* = 0$

만약 $\alpha < 0$이면, $w_j^* = 0,\ z_j^* = -\alpha$

하한이 강화된 경우도 상한이 강화된 경우와 같이 쌍대해를 복원할 수 있다.

4.3 단일요소행의 제거

단일요소행은 제약식의 형태가 등식이거나 부등식인 경우중의 하나이다. 두 경우 모두 원변수에는 영향이 없으므로 쌍대변수의 복원만이 필요하게 된다. 부등식의 형태에 따라 다음과 같이 복원한다.

i). 단일요소행이 등식일 경우

$$w_k^* = z_k^* = 0,\ y_r^* = \frac{c_k}{a_{rk}}$$

ii) 단일요소행이 부등식일 경우

$w_k^* = 0$, u_k가 강화된 경우.

$z_k^* = 0$, l_k가 강화된 경우.

$$y_r^* = \frac{c_k + w_k^* - z_k^*}{a_{rk}}$$

4.4 치환 및 선회연산에 의한 방법

제약식 중 (암시적)자유변수를 가진 등식이 있을 경우 이 제약식은 치환에 의해 제거될 수 있다. 치환에 의한 사전처리가 일어나기 전에 제거되는 자유변수의 열을 단일요소열로 만들기 위한 선회연산이 일어나는데 선회연산은 원변수와 w, z에는 영향을 주지 않지만 y에는 영향을 주므로 y에 대한 복원이 필요하게 된다. 사전처리는 선회연산이 수행된 후 치환이 일어나므로 사후처리에서는 치환에 대환 복원연산을 수행한 후 선회연산에 대한 사후처리를 수행하게 된다. 치환에 의해서 제약식 r과 변수 x_k가 제거되었다고 가정할 경우 복원은

$$x_k^* = \frac{b_r - \sum_{j \neq k} a_{rj} x_j^*}{a_{rk}},\ y_r^* = \frac{c_k - \sum_{i \neq r} a_{ik} y_i^*}{a_{rk}},\ w_k^* = z_k^* = 0$$

이 된다. 선회연산이 두 행 r과 k에서 r을 선회행, v를 선회승수로 하여 일어났다고 가정하면 선회연산 전후의 쌍대목적함수의 값이 같음을 이용하여

$$y_r^* = y_r^{*\prime} + v y_k^*,\ y_k^* = y_k^{*\prime}$$

와 같이 쌍대해를 복원한다. k행이 목적식이면 $y_r^* = y_r^{*\prime} + v$가 된다.

4.5 변수의 고정

변수의 상한과 하한의 값이 같아서 변수의 값이 고정된 경우는 쌍대해만 복원하면 된다. 고정된 변수가 x_k라고 가정하면 다음과 같이 쌍대해를 복원한다.

$$\alpha = c_k - \sum_i a_{ij} y_i^*$$

만약, $\alpha \geq 0$이면, $w_k^* = \alpha,\ z_k^* = 0$

만약, $\alpha < 0$이면, $w_k^* = 0,\ z_k^* = -\alpha$

4.6 강제제약식의 제거

강제제약식은 해당 제약식을 만족시키는 유일한 방법이 제약식을 이루는 변수들을 상한 또는 하한에 고정시키는 것이 유일한 제약식을 말한다. 제약식 r이 강제제약식일 경우 쌍대해의 복원은 다음과 같다. 제약식 r의 형태에 따라서 y_l과 y_u를 초기화한다.

만약 형태가 '≤'이면, $y_l = -\infty$, $y_u = 0$,

만약 형태가 '≥'이면, $y_l = 0$, $y_u = \infty$,

만약 형태가 '='이면, $y_l = -\infty$, $y_u = \infty$.

제약식 r에서 고정된 모든 변수, x_k에 대해서 다음을 수행한다.

$$\alpha = \frac{c_k - \sum_{i \neq r} a_{ik} y_i^*}{a_{rk}}.$$

만약 $x_k^* = l_k$, 즉 하한으로 고정이 되었으면,

$a_{ik} > 0$이고 $\alpha > y_l$이면, $y_l = \alpha$,

$a_{ik} < 0$이고 $\alpha < y_u$이면, $y_u = \alpha$.

만약 $x_k^* = u_k$, 즉 상한으로 고정이 되었으면,

$a_{ik} < 0$이고 $\alpha > y_l$이면, $y_l = \alpha$,

$a_{ik} > 0$이고 $\alpha < y_u$이면, $y_u = \alpha$.

$$y_r^* = \min\{|y_l|, |y_u|\}.$$

제약식 r에서 고정된 모든 변수, x_k에 대해서 다음을 수행한다.

$$\alpha = \frac{c_k - \sum_i a_{ik} y_i^*}{a_{rk}}.$$

만약 $x_k^* = l_k$, 즉 하한으로 고정이 되었으면,

$w_k^* = 0$, $z_k^* = -\alpha$,

만약 $x_k^* = u_k$, 즉 상한으로 고정이 되었으면,

$w_k^* = \alpha$, $z_k^* = 0$.

4.7 이중요소행의 제거

이중요소행의 제거는 '선회연산'과 '중복행의 제거'로 이루어지므로 4.1과 4.4에서 설명한 방식으로 쌍대해를 복원할 수 있다.

4.8 이중열의 제거

열 k와 j사이에 $A_{.k}=\alpha A_{.j},\ c_k=\alpha c_j$인 관계가 성립하여 두 열이 $\bar{k}$로 합쳐진다고 가정하면 사후처리는 다음과 같이 이루어진다.

$$c_k=\frac{\bar{c}_k}{\alpha+1},\ c_j=\bar{c}_k-c_k$$

$$\eta=c_k-\sum_i a_{ik}y_i^*$$

만약 $\eta\geq 0$ 이면, $w_k^*=\eta, z_k^*=0$

만약 $\eta<0$ 이면, $w_k^*=0, z_k^*=-\eta$

$$\eta=c_j-\sum_i a_{ij}y_i^*$$

만약 $\eta\geq 0$ 이면, $w_j^*=\eta, z_j^*=0$

만약 $\eta<0$ 이면, $w_j^*=0, z_j^*=-\eta$

5. 실험결과

본 연구진에 의하여 개발된 내부점 선형계획법 프로그램인 LPABO를 사용하여 위에서 설명한 사후처리 방법을 구현하여 실험을 수행하였다. 대상 문제는 비영요소의 수가 10,000개 이상인 NETLIB문제 및 Kennington문제와 Gondzio계열의 문제를 대상으로 하였다. 실험결과 사후처리에 걸리는 시간은 아래의 Table 1과 같이 실제 내부점 알고리즘을 수행하는 시간에 비해 상당히 작고 사전처리 시간에 비하여도 상당히 적게 걸림을 알 수 있다. 실험환경은 ALPHA 워크스테이션, RAM 128MB이고 컴파일러는 gcc v2.95.2를 사용하였다.

Table 1. 사후처리 수행속도

name	postsolving time	solving time	postsolving /presolving
25fv47	0.01	1.66	0.17
cycle	0.01	2.73	0.08
czprob	0.01	0.83	0.06
d2q06c	0.02	12.33	0.07
d6cube	0.02	6.66	0.05
degen3	0.02	12.74	0.17
dfl001	0.07	671.98	0.11
greenbeb	0.04	3.88	0.13
nesm	0.01	1.81	0.13
pilotja	0.01	3.61	0.11
sctap3	0.01	0.94	0.14
ship12l	0.01	1.40	0.02
stocfor3	0.13	17.25	0.25
woodlp	0.03	4.29	0.14
cre-d	0.24	55.41	0.28
ken-13	0.37	40.24	0.22
pds-20	0.34	8230.06	0.02
osa-30	0.32	279.66	0.14
ch	0.02	9.83	0.09
cq9	0.08	73.94	0.08
ge	0.08	20.76	0.09
nl	0.04	36.81	0.08

6. 결론

본 연구에서는 사전처리를 한 선형계획문제의 최적해를 복원하기 위한 사후처리 방법들에 대하여 알아보고 사후처리를 위한 전략에 대해 살펴보았다. 복원되는 해의 최적성을 유지하기 위해 일련의 사전처리 정보를 저장하는 로그파일을 사용하였으며 각 사전처리 방법에 따른 복원 방식을 설명하였다. 실험결과 사전처리 시간에 비하여 0.1배의 시간이 걸리고 전체 해법 시간에 비해 상당히 적은 수행시간이 소요됨을 알 수 있다.

7. 참고문헌

[1]. Andersen, E.D. and K.D. Andersen, Presolving in linear programming, Mathematical Programming, 1995, 71, 221-245.

[2] Bradley, G.H., G.G. Brown and G.W. Graves, Structural redundancy in large-scale optimization models, in: Redundancy in Mathematical Programming, ed. M.H. Karwan et al., Springer, Berlin, 1983.

[3] Brearley, A.L., G. Mitra and H.P. Williams, Analysis of mathematical programming problems prior to applying the simplex algorithm, Mathematical Programming, 1975, 8, 54-83.

[4] Fourer, R. and D.M. Gay, Experience with a primal presolve algorithm, in: Large Scale Optimization: State of the Art, ed. W.W. Hager et al., Kluwer, 1994.

[5] Gondzio, J., Presolve anlaysis of linear programs piror to applying an interior point method, INFORMS Journal on Computing, 1997, 9, 73-91.

[6] Lim, S., M. Seong and S. Park, An implementation of preprocessing for the simplex method, Journal of the Korean Institute of Industrial Engineers, 1999, 25, 233-239.

[7] Lustig, I.J., R.E. Marsten and D.F. Shanno, Interior point methods for linear programming: computational state of the art, INFORMS Journal of Computing, 1994, 6, 1-14.

[8] Park, S., W.J. Kim, T. Seol, M. Seong and C.K. Park, LPABO: A Programming for interior point methods for linear programming, Asia-Pacific Journal of Operations Research, 2000, 17, 81-100.

[9] Seong, M., Presolving and Postsolving in Linear Programming, Ph.D. Thesis, Seoul National University, Seoul, Korea, 2000.

[10] Seong, M., S. Lim and S. Park, An implementation of preprocessing for interior point method for linear programming, Journal of the Korean Operations Research and Management Science Society, 1999, 24, 1-11.

[11] Tomlin, J.A. and J.S. Welch, Formal optimization of some reduced linear programming problems, Mathematical Programming, 1983, 27, 232-240.

[12] Tomlin, J.A. and J.S. Welch, A pathological case in the reduction of linear programs, Operations Research Letters, 1983, 2, 53-57.

[13] Tomlin, J.A., J.S. Welch, Finding duplicate rows in a linear programming model, Operations Research Letters, 1986, 5, 7-11.

[14] Williams, H.P., A reduction procedure for linear and integer programming models, in: Redundancy in Mathematical Programming, ed. M.H. Karwan et al., Springer, Berlin, 1983.