대한산업공학회/한국경영과학회 2002 춘계공동학술대회
한국과학기술원(KAIST) 2002년 5월 3일-4일

혼합정수계획법을 위한 분지한계법의 효율적인 구현†

An efficient implementation of branch-and-cut algorithm for mixed integer programming

도승용*, 이상욱*, 임성묵**, 박순달*

Abstract

A Branch-and-Cut algorithm is a branch-and-bound algorithm in which cutting planes are generated throughout the branch-and-bound tree. It is now one of the most widespread and successful methods for solving mixed integer programming problems. In this paper we presents efficient implementation techniques of branch-and-cut algorithm for mixed integer programming problems

1. 서론

혼합정수계획법(mixed integer program)은 목적함수와 제약식이 일차식이고, 변수들이 정수 또는 실수를 가지는 수리계획법을 의미한다[1]. 혼합정수계획법은 매우 많은 분야에 응용되어지고 있는 수리계획법의 한 분야이다.

혼합 정수계획법 해법으로는 다음과 같은 두 가지 방법이 있다.

- 분지한계법
- 절단평면법

절단평면법은 일련의 완화된 선형계획법(Linear Programming)문제를 풀어 혼합정수계획법의 해를 구하고자 하는 방법으로서 Gomory에 의해 1960년대에 제시되었다[1]. 이후 절단평면을 생성하는 여러 가지 방법이 제시되었으나 최근 들어 Gomory에 의해 제시된 절단제약식을 이용한 절단평면법이 다시 주목을 받고 있다.

절단평면법은 이론적인 면에서는 매우 높이 평가되는 방법이지만 실제 구현을 통한 실험에서는 그 효율성 면에서 부정적인 것으로 보고되었다. 특히, 절단평면을 첨가하더라도 완화된 선형계획법 문제의 최적 목적함수 값이 감소되는 폭이 작아지는 현상이 발생하는 것으로 알려져 있다. 이는 상당한 개수의 절단평면을 첨가하더라도 목적함수의 크기가 크게 줄어들지 않는 것을 의미하여 현실적으로 절단평면법만을 사용하여 혼합정수계획법의 해를 구하는 것이 어렵다.

최근 들어 분지한계법과 절단평면법을 결합한 분지절단법(branch-and-cut)이 제시되면서 다시 절단평면법이 주목을 받고 있다. 이는 분지한계법을 기반으로 각 분지된 문제에서 절단평면을 생성함으로써, 절단평면만을 사용하는 절단평면법에 비해 완화된 선형계획법 문제의 목적함수 값을 효과적으로 줄일 가능성이 많아졌기 때문이다.

현재 분지절단법은 혼합정수계획법을 푸는 가장 효율적인 해법들 중의 하나로 간주되어진다. 이러한 분지절단법을 구현한 소프트웨어

† 한국과학재단의 특정기초연구과제(과제번호 2000-1-31500-001-02)의 지원을 받음.
* 서울대학교 산업공학과 ** 한국 전산원

로는 CPLEX[5], bc-opt[3], ABACUS[7], MINTO[4] 등이 있다. 이 소프트웨어들은 모두 국외에서 개발되어져 있고 아직 국내에서 대형 혼합정수계획법 문제를 풀기 위한 소프트웨어는 전무한 실정이다.

따라서 본 연구에서는 분지절단법을 효율적으로 구현하기 위해 고려해야 할 사항들을 살펴보고자 한다. 분지절단법은 분지한계법을 기반으로 하여 절단평면을 첨가하는 방법이다. 분지절단법이 효율적으로 구현되기 위해서는 먼저 분지한계법이 효율적으로 구현되어져 있어야 한다. 본 논문의 3장에서는 분지절단법의 기반이 되는 분지한계법을 효율적으로 구현하기 위해 고려해야할 사항을 살펴본다. 그리고 4장에서는 분지절단법의 구현시에 중요한 고려사항인 절단평면의 생성 및 관리방법들에 대해 살펴보고자 한다.

2.분지절단법

먼저 다음의 정수계획법 문제를 고려하자.

$$\begin{aligned} &max\ \boldsymbol{c}^T\boldsymbol{x} \\ &s.t\ A\boldsymbol{x} \le \boldsymbol{b} \end{aligned} \qquad (2.1)$$
$$x_i = \text{정수 또는 실수}\ \forall i$$

분지절단법은 분지한계법에 절단평면법을 결합한 방법이다. 이 방법은 분지한계법을 기반으로 하면서 탐색나무의 각 부문제에서 절단평면을 생성하여 각 부문제의 쌍대한계(dual bound)를 강화시킨다.

분지절단법은 먼저 식 (2.1)의 이완된 문제를 푼다. 여기서는 문제의 이완은 선형계획법으로 이완시키는 것을 의미한다. 여기서 $S=\{\boldsymbol{x}:\ A\boldsymbol{x}\le\boldsymbol{b},\ x_i=\text{정수 또는 실수}\}$라고 하면 S의 선형계획법 이완은 $P=\{\boldsymbol{x}:\ A\boldsymbol{x}\le\boldsymbol{b},\ \boldsymbol{x}\in R^n\}$이다.

(2.1)의 선형계획법 이완문제를 푼 결과 최적해 $\bar{\boldsymbol{x}}\in S$이면 (2.1)의 최적해를 구한 것이 된다. 만약 $\bar{\boldsymbol{x}}\notin S$이면 $\bar{\boldsymbol{x}}$를 제거할 수 있는 절단제약식을 도입하게 된다. 절단제약식을 만들 수 없을 때까지 이 과정을 반복한다. 그런 다음 정수변수를 중에서 실수변수를 가지는 변수 x_i에 대해 분지를 수행하게 된다. 즉 $x_i \ge \lceil \bar{x}_i \rceil$와 $x_i \le \lfloor \bar{x}_i \rfloor$의 범위로 나누어진 두 개의 부문제로 나누어진다.

다음은 분지절단법의 일반적인 흐름이다.

[알고리즘 : 분지한계법]

단계 1 초기화

L을 풀어야 하는 문제목록이라고 하자.

L에 식 (2.1)의 문제를 첨가한다.

$\underline{z}=-\infty$라고 둔다.

단계 2 부문제 선택

L이 비어있으면 멈춘다.

목록 L에서 부문제 P선택한다.

단계 3 한계

문제 P의 선형계획법 이완문제를 푼다.

비가능이면 단계 2로 간다.

$\boldsymbol{c}^T\bar{\boldsymbol{x}} \le \underline{z}$이면 단계 2로 간다.

$\bar{\boldsymbol{x}}$가 정수제약을 만족하면 $\underline{z}=\boldsymbol{c}^T\bar{\boldsymbol{x}}$라고 두고 단계 2로 간다.

단계 4. 절단평면의 생성

문제 P에서 절단평면을 생성한 다음, 문제 P에 이를 첨가한 다음 단계 3으로 간다.

만약 절단평면을 생성할 수 없다면 단계 5로 간다.

단계 5 분지

문제 P를 분리하여 부문제들로 나누어 목록 L에 첨가한다. 그리고 단계 2로 간다.

위의 분지절단법의 알고리즘을 보면 일반적으로 분지한계법과 유사하다. 단지 단계 4의 절단평면을 생성하는 단계가 추가되었을 뿐이다.

3. 분지한계법의 효율적인 구현

분지절단법은 분지한계법을 기반으로 하여 절단평면을 추가하는 것이다. 따라서 분지절단법을 효율적으로 구현하기 위해서는 먼저 분지한계법이 효율적으로 구현되어져 있어야 한다. 분지절단법의 단계 2의 부문제 선택전략이나, 단계 3의 한계, 그리고 단계 5의 분지는 모두 분지절단법의 전략들이다. 이 장에서는 분지한계법을 구현시에 고려할 사항들을 살펴보자.

3.1 쌍대단체법의 이용

분지한계법은 매 회 풀어야 하는 부문제들이 쌍대가능성은 유지된 채 원비가능성만 발생하게 되고, 절단평면법 또한 추가되는 절단제약식으로 인해 원비가능성만 발생한다. 그러므로, 쌍대가능성을 효과적으로 이용할 수 있는 쌍대단체법이 원단체법에 비해 훨씬 유리하게 된다.

3.2 부문제의 저장

분지한계법에서는 매 회 정수가 아닌 변수를 분지를 통해 새로운 부문제를 생성하고, 다음에 풀어야 할 부문제를 선택해야 한다. 분지한계법이 진행될 수록 생성되어지는 부문제의 수는 기하급수적으로 증가하게 된다. 이러한 부문제들을 모두 저장하기에는 상당한 저장공간이 필요하다. 이러한 부문제들 에는 이미 선형계획법을 적용해서 푼 것과 아직 풀지 않은 것으로 나눌 수 있다. 이미 선형계획법을 푼 문제들 중에는 분지끝(fathomed) 부문제와 그렇지 않은 부문제로 나누어진다. 이때 분지 끝 된 부문제는 더 이상 저장할 필요가 없다. 따라서 아직까지 풀지 않은 부문제들과 선형계획법을 적용한 부문제들 중에서 분지끝이 되지 않은 문제만 저장하면 된다. 선형계획법 이완을 적용하여 푼 문제들 중에서 분지끝이 되지 않은 문제를 저장하는 것은 분지된 변수들의 정보를 보관하기 위해서이다.

3.3 부문제의 선택

부문제 선택방법은 분지한계법 프로그램의 성능에 매우 중요한 역할을 한다. 부문제를 선택하는 방법으로 흔히 다음과 같은 방법들이 많이 사용되어진다.

- 깊이우선 탐색방법(depth first search)
- Best First 탐색방법
- 이 단계법(two phase method)
- Best Projection 방법

가. 깊이우선 탐색방법

깊이 우선 탐색은 문제 목록에서 가장 최근에 생성된 문제를 선택하는 방법이다. 이 방법의 장점은 탐색에 필요한 기억공간의 요구량이 적으면서 정수해를 보다 빨리 찾을 가능성이 많은 방법이다. 왜냐하면 깊이가 늘어날수록 제약이 많아져 정수해에 근접할 가능성이 루트노드보다 많기 때문이다. 또한 부모문제와 자식문제사이의 차이점은 분지된 변수 하나만이 상하한이 다를 뿐이다. 이는 부모문제의 최적해를 초기해로 하여 쌍대단체법을 적용하면 자식문제의 최적해를 쉽게 구할 수 있는 장점이 있다. 이 방법의 단점으로는 나무의 크기가 커지는 경향이 있다.

나. Best First 탐색방법

Best-First탐색방법은 문제 목록에서 문제를 선택할 때, 목적 함수 값의 한계가 가장 좋은 문제를 우선적으로 선택하는 방법이다. 이 방법은 정수해를 빨리 찾을 가능성이 높다고 말할 수는 없지만 정수해를 발견했을 경우 최적해에 근접할 가능성이 높다. 따라서 정수가능해를 찾은 경우 많은 문제를 보다 빨리 분지끝 시킬 수 있고 문제 목록에서 탐색 대상 문제의 수를 줄일 수 있다.

이 방법의 단점으로는 좋은 정수 가능해가 빨리 발견되지 않는 경우 분지 끝 대상이 되는 문제수가 적게 된다. 따라서 풀어야 하는 부문제의 수가 크게 증가한다. 이것은 많은 메모리를 요구하게 되는 문제점이 발생하게 된다. 그리고 Best-First 탐색방법으로 문제를 선택을 하면 이전에 풀었던 문제와 다음에 푸는 문제가 서로 연관성이 많이 떨어진다. 즉 부모 문제

와 자식 문제의 관계가 되지 않는다. 따라서 부문제를 푸는 경우 많은 계산량을 요구하게 된다. 그러나 이러한 단점은 문제목록의 문제들의 부모문제의 기저정보를 저장한 후 이 정보를 이용함으로서 극복할 수 있다.

다. 이 단계법

탐색전략의 목적은 정수가능해를 찾고, 더 좋은 목적 함수값이 존재하지 않음을 증명하는데 있다. 따라서 문제를 푸는 중간에 탐색 전략의 목적에 따른 방법의 전환을 고려할 수 있다. 즉, 첫 번째 단계에서는 빠르게 좋은 정수해를 찾고 두 번째 단계에서는 첫 단계에서 얻은 해를 이용하여 많은 문제를 분지끝 시킬 필요가 있다. 이 단계법의 가장 간단한 방법으로는 정수가능해를 찾기 전까지는 깊이 우선 탐색방법을 이용하고 정수가능해를 찾은 이후에는 Best-First 탐색방법을 이용하는 것이다.

라. Best Projection 탐색방법

Best Projection탐색방법은 문제 목록의 문제가 정수해를 가졌을 때의 목적함수값을 추정하여 추정값이 가장 좋은 문제를 선택하는 방법이다. 추정값은 아래와 같이 구하며, 이 값이 가장 좋은 문제를 선택하게 된다.

$$E_i = \bar{z}_i + \left(\frac{z^t - \bar{z}_0}{s_0} \right) s_i , \quad i \in N \ (3.1)$$

여기서 $s_i = \sum_{j \in I} \min(f_j, 1-f_j)$이며 부문제 i의 정수비가능(integer infeasibility)의 합을 나타낸다. 그리고 $\bar{z}_i$는 부문제 i의 상한, $\bar{z}_0$는 루트노드의 상한, s_0는 루트노드의 정수비가능의 합을 나타낸다. 이 방법은 문제를 풀기 전에 정수 가능해를 가졌을 때의 목적 함수값을 예측할 수 있게 한다.

정수가능해를 찾기 전에는 깊이 우선 탐색으로 문제를 선택하고 정수가능해를 찾은 경우 (3.1)을 이용하여 탐색 대상 부문제들의 목적함수값의 추정하여 이 중 추정값이 가장 좋은 문제를 선택한다. 추정값이 좋은 문제를 선택함으로써 그 만큼 좋은 정수 가능해를 찾을 가능성을 높이며 좋은 정수해를 찾은 경우 그 만큼 많은 문제들을 분지끝이 가능하게 됨으로써 전체 수행속도에 효과를 발휘할 수 있게 된다.

3.4 분지전략

혼합정수계획법에서 선형계획법으로 이완된 부 문제의 변수 값이 실수인 정수변수 중에서 변수 하나를 선택하여 새로운 한계를 주어서 문제의 공간을 나누게 된다. 이완문제에서 많은 실수값을 가지는 정수변수가 존재하게 되고 그 중에서 하나를 선택해야된다. 분지한계법를 효과적으로 수행하기 위해서는 분지 변수를 선택 시 목적함수값의 상하한을 빠르게 수렴시키는 변수의 선택에 대하여 관심을 가지게 된다. 따라서 부모문제와 자식문제의 목적함수값의 차이를 크게 하는 변수를 선택하는 게 유리하게 된다. 분지전략에는 다음과 같은 방법들이 사용된다.

가. 목적함수의 계수를 이용하는 방법

분지 대상이 되는 정수변수 중에서 목적함수의 계수가 가장 큰 변수를 선택하는 방법으로 만일 변수간의 연관성이 없다면 목적함수계수가 벌금(penalty cost)의 역할을 하게 된다.

나. 최대 정수비가능(maximum integer infeasibility)을 이용하는 방법

흔히 알려진 방법으로는 $\max_{i \in I} \min \{f_i, 1-f_i\}$인 변수 x_i를 분지할 변수로 선택하는 방법이 있다. 이는 정수의 정도가 가장 안 좋은 변수를 분지변수로 선택하는 것이다. 단 $f_i = \bar{b}_i - \lfloor \bar{b}_i \rfloor$

다. 벌금을 이용하는 방법

최적해의 상하한의 수렴을 위하여 일반적으로 부모문제와 자식문제간의 목적함수값 차이를 크게 하는 변수를 선택하는 것이 유리하다. 실수값을 갖는 정수변수에 상하한을 주면

원비가능이 발생하게 된다. 이 때 원가능을 유지하기 위해서는 선회연산을 실시해야 하며, 1회 선회연산을 실시했을 때의 목적함수값의 감소량을 이용하여 분지변수를 선택하는 방법이다.

변수 x_i를 실수값을 갖고 있는 정수변수라고 가정하면

$$x_i = \overline{b_i} + \sum_j \overline{a_{ij}}(-x_j) \qquad (3.2)$$

x_i를 분지하기 위해서 $x_i \le \lfloor \overline{b_i} \rfloor$과 $x_i \ge \lceil \overline{b_i} \rceil$를 제약식에 추가한 후, 표준형으로 만들면 다음과 같다.

$$\begin{aligned} x_i + x_i^D &= \lfloor \overline{b_i} \rfloor \\ x_i - x_i^U &= \lceil \overline{b_i} \rceil \end{aligned} \qquad (3.3)$$

식 (3.2)을 식 (3.3)에 대입하면

$$\begin{aligned} x_i^D &= (\lfloor \overline{b_i} \rfloor - \overline{b_i}) + \sum_j \overline{a_{ij}} x_j \\ x_i^U &= (\overline{b_i} - \lceil \overline{b_i} \rceil) - \sum_j \overline{a_{ij}} x_j \end{aligned} \qquad (3.4)$$

이 된다. 여기서 x_i^D, x_i^U가 음수이므로 원가능성이 깨지게 되고, 쌍대단체법에서 x_i^D, x_i^U이 1회 선회연산에서 탈락변수가 된다.

$x_i \le \lfloor \overline{b_i} \rfloor$을 추가하고 1회 선회연산을 실시했을 때, 목적 함수값의 감소량을 다운 패널티(down penalty)라 한다. $x_i \ge \lceil \overline{b_i} \rceil$인 경우 업 패널티(up penalty)라 하며 아래와 같이 계산이 된다.

$$P_D = \min \left\{ -\frac{(\lfloor \overline{b_i} \rfloor - \overline{b_i})}{\overline{a_{ij}}} \overline{c_j} : (\overline{a_{ij}} > 0,\ j \in R) \text{ 또는 } (\overline{a_{ij}} < 0,\ j \in R') \right\}$$

$$P_U = \min \left\{ \frac{(\overline{b_i} - \lceil \overline{b_i} \rceil)}{\overline{a_{ij}}} \overline{c_j} : (\overline{a_{ij}} < 0,\ j \in R) \text{ 또는 } (\overline{a_{ij}} > 0, j \in R') \right\}$$

(단, R : 하한 비기저 지수, R': 상한 비기저 지수)

3.5 기저정보의 이용

문제목록에서 부문제 하나가 선택이 되면 선형계획법 문제로 이완을 하여 부문제를 풀게 된다. 이 때 매회 푸는 부문제는 부모문제와 비교하면 쌍대가능성은 유지된 채 원비가능성만 발생하게 된다. 따라서 부모문제의 기저 상태를 저장한 후, 문제를 풀 때 이 정보를 초기 기저로 잡는데 이용하여 문제를 푸는 것이 유리하다. 하지만 모든 부문제들에 대해 기저정보를 보관하게 되면 매우 많은 저장공간이 필요하다. 보통 부모문제와 자식문제의 기정정보는 많은 차이가 나지 않는다. 따라서 부모문제와 자식문제 사이의 변화된 기저의 정보만 탐색나무에 보관하고 있으면 모든 문제에 대해 기저정보를 보관하는 것보다 저장공간을 효율적으로 사용할 수 있다.

4. 분지절단법의 구현시 고려사항

3장에서는 분지절단법의 기반이 되는 분지한계법을 효율적으로 구현할 때 고려해야 하는 사항을 살펴보았다. 분지절단법은 분지한계법에 절단평면법을 결합한 방법이다. 여기서는 분지절단시에 고려해야 하는 절단평면의 생성, 절단평면의 리프팅(Cut Liffting) 그리고 절단평면들을 효율적으로 관리하기 위한 절단평면 풀(Cut-Pool)의 구현방법에 대해 살펴보기로 하자.

4.1 절단평면의 생성

가. 고모리 절단평면의 생성방법

혼합정수계획법에서 완화된 선형계획법의 부문제의 해가 정수제약을 만족하지 않는 경우 다음과 같이 고모리 절단제약식을 생성한다. 정수조건을 만족하지 않은 변수 x_i로부터 절단평면을 만든다고 하자.

$$x_i + \sum_{j \in H_l} \overline{a_{ij}} x_j + \sum_{j \in H_R} \overline{a_{ij}} x_j = \overline{b_i}$$

$$\Rightarrow \sum_{\{j \in H_l:\, f_{ij} < f_i\}} f_{ij} x_j + \frac{f_i}{1-f_i} \sum_{\{j \in H_l:\, f_{ij} > f_i\}} (1-f_{ij}) x_j + \sum_{\{j \in H_R:\, a_{ij} > 0\}} \overline{a_{ij}} x_j - \frac{f_i}{1-f_i} \sum_{\{j \in H_R:\, a_{ij} < 0\}} \overline{a_{ij}} x_j \ge f_i$$

단, $f_i = b_i - [b_i]$, $f_{ij} = \overline{a_{ij}} - [\overline{a_{ij}}]$

H_I : 정수를 가지는 비기저 변수의 집합

H_R : 실수를 가지는 비기저 변수의 집합

나. 커버 부등식

다음과 같은 집합 S가 있다고 하자.

$$S = \left\{ x \in B^n : \sum_j a_j x_j \leq b \right\}$$

여기서 a_j와 b는 양수를 가정하며, $a_j < 0$ 인 경우에는 $\overline{x_j} = 1 - x_j$ 로 치환하여 a_j 를 양수로 유지한다. $C \subset N$이 $\sum_{j \in C} a_j > b$을 만족하면 C을 커버(cover)라 한다. 그리고 C가 C의 진부분집합이면서 커버인 부분집합을 포함하고 있지 않으면 최소커버(minimal cover)라 한다. 임의의 커버 C에 대해, 다음의 부등식을 커버부등식(cover inequality)이라 하고 집합 S에 대하여 유효하다.

$$\sum_{j \in C} x_j \leq |C| - 1$$

커버부등식은 가능공간에 대한 유효한 부등식으로서 배낭제약식에서 유도가 되며 커버 집합에 속하는 모든 변수가 1이 아님을 의미한다. 이 커버부등식을 문제에 추가함으로써 얻을 수 있는 효과는 정수간격(integrarity gap)을 줄일 수 있다는데 있다. 그러나 원문제에 제약식을 추가하는 경우 중복적인 제약식이 추가가 되어 문제의 크기가 커지게 되어 불리하게 될 수도 있다.

다. 절단제약식의 리프팅(Cut Lifting)

절단제약식의 리프팅은 분지한계법의 한 부문제에서 생성된 절단제약식이 다른 모든 부문제에서 유효할 수 있도록 만드는 절차를 말한다. 이는 분지절단법에서 매우 중요한 절차인데 만약 한 부문제에서 생성된 절단제약식이 그 부문제의 자식 노드에 해당하는 문제들에서만 성립한다면 생성된 절단제약식을 관리한느 일이 매우 복잡해질 뿐만 아니라 절단제약식에 의해 제거되는 비정수해가 일부문제에만 적용되어 그 효과가 크지 않을 수 있다.

Balas는 혼합0-1정수계획법에 적용될 수 리프트 앤 프로젝트(Lift-and-Project)방법을 통해 모든 부문제에 적용될 수 있는 절단제약식을 유도하였다. 그리고 Nemhauser et al. 등은 커버부등식을 보다 높은 차원의 유효부등식(Valid Inequality)으로 순차적 리프팅 방법을 도입하였다. 이와 관련된 자세한 이론은 관련문헌을 참조하기 바란다[2, 7].

라. 절단평면의 질 평가기준

생성된 절단 평면 $\boldsymbol{a}^T \boldsymbol{x} \geq \beta$이 얼마나 좋은가를 판단하는 기준으로서 두 가지 방법이 제시되어 있다. 첫 번째 방법은 최적해 $\overline{\boldsymbol{x}}$와 절단평면 $\boldsymbol{a}^T \boldsymbol{x} \geq \beta$와의 직교거리를 사용하는 방법이다. 이 때 절단평면은 $\| \boldsymbol{a} \| = 1$ 또는 $\beta = 1$로 정규화되어 있다고 가정한다. 전자의 경우를 α-정규화라 부르고 후자의 경우를 β-정규화라 부른다. 거리가 멀수록 좋은 절단제약식이라 볼 수 있으나 꼭 그렇지 않는 경우도 있어 모든 경우에 잘 들어맞는 척도는 아니다. 두 번째는 $\boldsymbol{a}^T \overline{\boldsymbol{x}} - \beta$ 값을 척도로 이용하는 방법이다. 물론 이 척도가 항상 잘 들어맞는 것은 아니다.

4.2 절단평면의 관리

가. 생성되는 절단평면의 개수

분지절단법에서는 모든 부문제에서 절단평면을 만들지 않고 일부 부문제에서만 만드는 방식을 취할 수도 있다. 예를 들어 탐색나무에서 k개의 노드를 탐색할 때마다 r개의 절단평면을 마드는 방식이 있다. 이 때 k를 생략인자(skip factor)라고 한다. 이는 절단평면을 생성한 부문제로 부터 k번째 탐색하는 부문제에서만 절단평면을 생성하는 방법이다. 생략인자 k를 고정해서 사용하는 고정전략(fixed strategy), 문제마다 다르게 사용하는 자동전략(automatic strategy), 탐색나무의 각 노드마다 k를 조정해주는 적응전략(adaptive strategy)이

있다.

나. 절단제약식 풀(Cut-Pool)의 관리

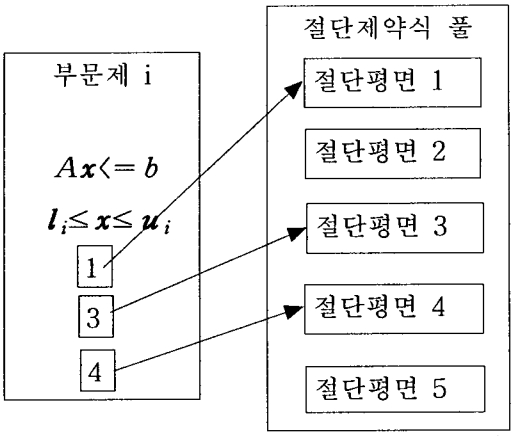
분지절단법에서는 각 부문제에서 절단평면을 생성하는데 생성된 절단평면의 수가 많아지면 완화된 부문제의 크기가 매우 커지고 복잡해지게 된다. 따라서 생성된 모든 절단제약식들을 완화된 문제에 포함시킬 것이 아니라, 그 중의 일부만을 부문제에 포함시키는 방법을 생각할 수 있다. 여기서 분지절단법에서 고려중인 부문제에 포함시킨 절단평면의 집합을 활성절단(active cut)집합이라 하고, 고려되지 않는 나머지 생성된 절단평면의 집합을 절단제약식 풀(cut-pool)이라고 부른다. 활성절단집합과 절단풀을 관리하는 기본절차는 다음과 같다.

단계 1. 고려중인 부문제를 푼다.
단계 2. 비활성 절단제약식(inactive cut)을 찾아 활성절단제약식 집합에서 제거하고 이를 절단제약식 풀에 넣는다.
단계 3. 절단평면 풀에 있는 절단평면들 중에서 위반절단(violated cut)제약식을 찾아 일부 또는 전부를 절단평면 풀에서 제거하여 활성절단평면 집합에 포함시키고 단계 1로 간다.
단계 4. 절단제약식을 생성한다. 이를 절단제약식 풀에 포함시키고 단계 3으로 간다.

여기서 가능한 한 절단제약식을 관리하는데 드는 오버헤드를 줄일 필요가 있다. 따라서 활성절단제약식 집합으로부터 절단제약식 풀로의 잦은 이동은 피해야 하므로 비활성 절단제약식을 찾는데 있어서도 신중을 기해야 한다.
비활성 절단제약식을 찾는 방법으로는 해당 절단제약식의 쌍대변수값이 일정회수 이상 연속적으로 0이 되면 비활성 절단제약식으로 선정하는 방법을 Johnson et al등이 제시하였다. 보통 10번이상 연속으로 쌍대변수의 값이 0이 되면 비활성 제약식으로 간주한다.
절단평면 풀의 크기가 계속 커지게 되면 위반절단제약식을 찾는데 필요한 계산시간도 같이 증가하게 된다. 또한 기억공간의 한계면에서도 절단평면 풀에 저장되는 절단제약식의 수를 제한할 필요가 있다. 보통 절단평면 풀의 크기는 많은 경우에도 500개 이내로 제한하는 것이 계산시간면에서 효율적이다.
단계 4에서 생성되는 절단평면들을 모두 포함시키기 보다는 생성된 절단평면의 질을 따져 절단평면 풀에 추가하는 것이 보다 바람직하다. 예를 들어 기존에 생성된 절단평면과 새로 생성된 절단평면이 이루는 각이 매우 적은 경우에는 새로 생성된 절단평면과 기존의 절단평면이 중복적일 가능성이 많으므로 새로 생성된 절단평면을 절단평면 풀에 포함시키지 않은 방법을 사용할 수도 있다.

4.2 절단제약식 풀의 구현

분지한계법에서는 절단제약식 풀에 있는 절단제약식들 중에서 현재의 정수해가 아닌 최적해를 제거할 수 있는 절단제약식들은 현재 부문제의 제약식에 첨가되고, 활성절단제약식 중에서 현재의 최적해를 제거할 수 없으면 절단제약식 풀로 옮겨지게 된다. 이는 매 회 절단제약식들을 부문제의 제약식에 첨가하거나 삭제해야 하는 연산을 반복해야 한다. 이와 같은 연산을 효율적으로 구현하기 위해 각 부문제에서 첨가되는 절단제약식을 가리키는 포인터를 도입하였다. 즉 절단제약식들은 일단 절단제약식 풀에 저장되어져 있고 부문제에서는 필요로 하는 제약식들의 절단제약식 풀1에서의 위치만 보관하고 있으면 된다. 따라서 절단제약식 풀에 저장되어져 있는 절단평면을 현재의 부문제에 첨가하고 싶으면 이 포인터만 절단평면이 저장되어져 있는 위치만 가리키도록 하면 된다. 그리고 절단평면을 삭제하고 싶으면 이 절단평면을 가리키고 있는 포인터를 해제하면 된다. 다음은 절단제약식 풀과 부문제 사이의 관계를 나타낸다.



[그림 1] 부문제와 절단제약식 풀과의 관계

위의 그림을 보면 부문제 i는 필요로 하는 절단평면들의 절단제약식 풀에서의 위치를 저장하고 있다. 만약 이 절단제약식들이 더 이상 필요 없으면 이 절단평면에 대한 포인터를 해제시켜주면 된다.

5. 결론

혼합정수계획법은 매우 많은 응용분야를 가지고 있는 수리계획법이다. 혼합정수계획법 문제를 푸는 데에 분지한계법을 기반으로 하여 절단평면을 첨가하는 분지절단법이 매우 효율적인 해법으로 알려져 있다. 본 연구에서는 분지절단법의 바탕이 되는 분지한계법을 효율적으로 구현하기 위해 쌍대단체법의 이용, 부문제의 저장방법, 분지전략, 탐색전략, 기저정보를 이용하는 방법들에 대해 살펴보았고, 분지절단법을 효율적으로 구현하기 위해 고려해야할 절단평면의 생성, 커버 부등식, 절단제약식 리프팅, 그리고 절단제약식의 관리를 위한 절단제약식 풀의 구현방법에 대해 살펴보았다.

참고문헌

[1] 박순달, 경영과학, 제4판, 민영사, 2000.

[2] E. Balas, S. Ceria, G. Cornuejols, "Mixed 0-1 Programming by Lift-and-Project in a Branch-and-Cut Framework", Management Science, Vol. 42, No. 9(1996), 1229-1246

[3] Cercile, Cordier, Hugues Marchand, Richard Laundy, Laurence A. Wolesy, "bc-opt : a branch-and-cut code for mixed integer programs", Math. Prog. Ser. A 86(1999), 335-353

[4] G. L. Nemhauser and M. W. P. Savelsbergh, and G. C. Sigismondi. MINTO, a Mixed INTeger Optimizer, Operations Research Letters, 15(1994), 47-58

[5] http ://www.cplex.com

[6] Laurence Wolsey, Integer Programming, John Wiley& Sons, 1998

[7] Stefan Thienel. ABACUS A Branch - and-Cut System, PhD thesis, Universitat zu koln, 1995